UNIVERSITY OF NAIROBI

A TWO LAYER MIXED INTEGER PROGRAMMING MODEL FOR DYNAMIC COMPOSITE WEBSERVICE SELECTION IN VIRTUAL ORGANIZATIONS INSPIRED BY LAYERING AS OPTIMIZATION DECOMPOSITION

BY

ABIUD WAKHANU MULONGO

THESIS SUBMITTED IN PARTIAL FULFILMENT OF THE REQUIREMENTS OF THE DEGREE OF DOCTOR OF PHILOSOPHY IN COMPUTER SCIENCE OF UNIVERSITY OF NAIROBI

MAY 2016

DECLARATION

This thesis is my original work. It has not been presented for any other degree award in any other university.

Signature _____ Date_____

Abiud Wakhanu Mulongo

P80/93241/2013

The thesis has been submitted for examination with our approval as the University Supervisors.

Signature _____ Date_____

***Dr. Elisha T. O Opiyo* Omulo**

School of Computing and Informatics

Signature _____ Date_____

Dr. Elisha Odira Abade

School of Computing and Informatics

DEDICATION

To my wife Jael Netondo :- You know best the PhD *journey I have walked &. You stood by me in this enduring journey*

My sons Berring Fusho and Liam Muya: You bring incomparable joy in my life.

To my Dad Enos Wakhanu and mum Catherine Namisoo: I am because you are- you are my heroes.

To *my late grandfather Mulongo wa Muya: I miss your great secrets of life*

ACKNOWLEDGEMENT

When I enrolled for my PhD studies back in October 2013, during the PhD induction session, I curiously asked whether I could be allowed to finish my PhD in less than 2 years! In reply, to this, Prof. Wagacha Waiganjo , the PhD coordinator said, “Sure, you could do it in say 18 months as the minimum”. And Prof. Okello Odongo, the director, School of Computing & Informatics, added, “In that case, the school would give you a trophy”. It’s three years on, and now I think, the two Profs were just being modest and kind – I had no idea what PhD was, now I do. My PhD journey was long and tough. In the course of it, I fainted not once, not twice, and admitted the same number of times. This is how I realized the “Damaging effects of “PhD”. And therefore, I would like to profoundly appreciate the following who made it possible to successfully finish the race :- *To my supervisors* Dr. Elisha T. O Opiyo and Dr. Elisha Abade. You were always available for me at the shortest notice, despite your many other official commitments. You gave me the best guidance and advice I could ever imagine as far my *PhD* research was concerned. When I felt like am chasing the wind and was about to throw in the towel, you made what seemed impossible possible to me. I felt rejuvenated to maintain course. And when I felt too confident about this “*big idea*, you posed very challenging questions, which at the end of it, have made me the philosopher I am today. I hold you in the highest of esteem. To my colleagues at SCI who at one time or the other, were my companions through the journey. Your support and inspiration was pivotal in getting this work to its current position. To all the administrative, technical and support staff at the School of Computing. Your support and cooperation was invaluable. To the management of the University of Nairobi, and the School of computing and informatics. Thank you for providing a conducive environment that promotes freedom of thought and independence in scholarly work.

ABSTRACT

The key motivation for virtual organizations (VOs) is the need for *business agility* against a highly volatile and globally competitive market. The agility includes the ability to dynamically and efficiently package and deliver *highly customized services that maximally satisfy the utility of service consumer demands over the Internet.* Dynamic webservice composition (DWSC) is an essential Information Communication Technology (ICT) enabler of this form of agility in VOs. However, dynamic webservice composition remains a multiple criteria decision making (MCDM) nondeterministic polynomial (NP) hard optimization problem despite more than 10 years of extensive research. This makes the applicability of DWSC to problems of industrial relevance currently limited. Mixed Integer Programming (MIP) is the most widely used technique in efficiently modelling the problem. There are two MIP models for the DWSC problem: a *local planning strategy*, herein L-MIP and a *global planning strategy* hereafter S-MIP. L-MIP is provably polynomial time and practically multiple times faster than S-MIP. However L-MIP lacks the ability to capture global inter workflow task webservice Quality of Service (QoS) constraints and generally is less optimal relative to S-MIP. It has been demonstrated that L-MIP generates composite webservices that are 20% to 30% worse in quality with respect to S-MIP. S-MIP on the other hand guarantees global optimality but is susceptible to exponential state space explosion, making the strategy limited to problems in which the number of service providers per business workflow task *n* is small.

This thesis aimed to design a DWSC MIP global planning strategy that is *more efficient* than S-MIP. The second objective was to evaluate the performance of the proposed strategy versus S-MIP and L-MIP in terms of runtime efficiency and solution quality. The study proposed a two layer MIP model dubbed *SLUM: Service Layered Utility Maximization*. SLUM is inspired by the theory of *Layering as Optimization Decomposition*. Unlike all the existing DWSC MIP models that formulate and solve a single MIP model, in SLUM there are two hierarchically layered MIP models. One layer attempts to maximize the utility of service consumers and the other attempts to maximize the utility of virtual enterprise brokers. The DWSC is then solved sequentially. Efficiency gains from SLUM over S-MIP are hypothesized due to space reduction.

The study used both *theoretical* and *empirical* methodologies to evaluate the performance of SLUM against S-MIP and L-MIP in terms of two metrics: *running time* and *relative solution quality* (*RSQ*). Our first main contribution is that we derive a theoretic running time model of SLUM and using L-Hospital's Law, show that the theoretic speedup of SLUM with respect to S-MIP is given by the function $\frac{(\mathbf{2})^{\boldsymbol{k}}}{\mathbf{1}+\ \rho}$, $where\ \rho = (\prod_1^k(\mathrm{n}-\epsilon_i))/(\boldsymbol{n^k})$, *n* is the number of service providers per business workflow task, *k* is the number of sequential workflow tasks, ϵ_i is the number of service providers against the i[th] workflow task who fail to satisfy the webservice QoS requirements during the optimization process at the first layer. The study defines the parameter ρ as the *Composite Service Phase Transition Rate*. ρ lies on the interval [0,1]. The significance of the model is that at any one time instance, as $\rho \to 0$, implying very few service providers proceed for phase two optimization process, a virtual enterprise broker could expect relative speedup of up to $(\mathbf{2})^{\boldsymbol{k}}$, so that when k=2, SLUM is bound to be nearly 4 times faster than S-MIP. On the other hand when $\rho \to 1$, meaning that very few service providers get eliminated during phase one, the virtual enterprise broker could expect average speedups of up to $(\mathbf{2})^{\boldsymbol{k-1}}$. Therefore at k=2, SLUM is expected to be nearly 2 times faster than S-MIP on average. Thus, we show that for other values of ρ, the *expected speedup of SLUM with respect to S-MIP is bound to be on the* $interval\ \ [(\mathbf{2})^{\boldsymbol{k-1}}, (\mathbf{2})^{\boldsymbol{k}}]$.

Our other major contributions were through experimentation. In the first set of experiments on the running time performance was investigated. Seven different setups were designed with each experiment having a unique ρ value. The value of k was fixed at 2. The following methods were used for data analysis: *statistical regression analysis , scalability curves (speed up vs number of service providers), differential calculus using L-Hospital's Law, empirical relative complexity analysis, speedup vs* ρ curves. Our second major contribution is that from the empirical results we show that the $\frac{(\mathbf{2})^{\boldsymbol{k}}}{\mathbf{1}+\ \rho}$ approximately holds in practice. This was verified using L-Hospital's Rule and polynomial regression curve fitting. We found that the empirical expected speedup values at each of the *ρ values were all below but in close range with the theoretical values.* For example at ρ =0.0296, an expected speedup of 3.6 was obtained against the theoretical 3.885. Further, the

speedup vs ρ plot confirmed the inverse relation between speedup and ρ in $\frac{(\mathbf{2})^{\boldsymbol{k}}}{\mathbf{1}+\ \rho}$. The empirical relative complexity coefficients β_1 obtained for the various ρ values, were between 0.783 for the lowest ρ value and 0.96 for the highest ρ value. Moreover the β_1 values were generally proportional to ρ . The deductions here are that for all ρ SLUM is asymptotically faster than S-MIP. The second deduction is that asymptotic speedup of SLUM with respect to S-MIP is inversely proportional to ρ. This further verifies the model $\frac{(\mathbf{2})^{\boldsymbol{k}}}{\mathbf{1}+\ \rho}$. On the other hand, the initial relative performance parameter β_0 obtained via empirical relative complexity analysis generally showed that S-MIP is 1.3 times faster than SLUM initially. Using L-Hospital's Law, exponential regression functions as well as the scalability curves, we found that at a constant ρ value, the speedup of SLUM with respect to S-MIP grows as the number of service providers grow larger but eventually reaches a limit. Further, that below ten service providers per task, S-MIP is generally faster than SLUM, beyond which SLUM is faster. The study also established L-MIP is several orders faster than both S-MIP and SLUM, and that the running time of L-MIP has a polynomial upperbound. On the other hand, the study also established that even though SLUM is on average and asymptotically faster than S-MIP, both of them have an empirical running time model bound between polynomial and exponential growth. Thus both models are superpolynomial, further confirming the NP hardness of the DWSC problem. On the other hand, our empirical results on RSQ, show that SLUM has an average RSQ of 93%, which is 7% less optimal compared to S-MIP, while L-MIP has an RSQ value of 87% which is 6% less optimal than SLUM.

We conclude that if there is no need for global constraints at all, L-MIP is recommended over S-MIP and SLUM. However, if end user global constraints is a critical concern, optimality is a critical concern and the number of service providers per task is generally below 10, S-MIP should be used. SLUM is preferred over the two models if global constraints are critically needed and the number of service providers per task is above 10. Therefore virtual enterprise brokers could mix the three models in order to maximize their value and the value of their service consumers.

Keywords: *Dynamic Composite Webservice Selection, Mixed Integer Programming, Layering, Optimization, Decomposition, Virtual Organizations.*

Table of Contents

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

AI	Artificial Intelligence
BPEL	Business Process Execution Language
BPM	Business Process Modeling
BPMN	Business Process Modeling Notation
CP	Constraint Programming
DDL	Descriptive Disjunctive Logic
DWSC	Dynamic Webservice Composition- the subject of this thesis.
HMSCM	Hierarchical Service Composition Model
ICT	Information and Communication Technology
IP	Internet Protocol
JOM	Java Optimization Modeler
JSON	Java String Object Notation
L-MIP	Mixed Integer Programming using Local Planning Strategy for service composition
LOD	Layering as Optimization Decomposition
LP	Linear Programming
MAS	Multi-Agent System
MCDM	Multiple Criteria Decision Making
MIP	Mixed Integer Programming
REST	Representational State Transfer
SAW	Simple Additive Weighting

SCUM	Service Consumer Utility Maximization
SPUM	Service Provider Utility Maximization
SLUM	Service Layered Utility Maximization - the proposed model in this study
S-MIP	Single Layered (Standard) Mixed Integer Programming using global planning strategy
SOA	Service Oriented Architecture
SOC	Service Oriented Computing
TCP	Transport Control Protocol
SAT	Satisfiability – as used in Artificial Intelligence
SOAP	Simple Object Access Protocol
VE	Virtual Enterprise
VEB	Virtual Enterprise Broker
VIC	Virtual Industry Cluster
VO	Virtual Organization
WFC	Workflow Consortium – An international body that defines process modelling standards
WS-BPEL	Webservice Business Process Execution Language
WSC	Webservice Composition
WSDL	Webservice Description Language

1 CHAPTER 1: INTRODUCTION

The key motivation for virtual organizations (VOs) is the need for *business agility* against a highly volatile and globally competitive market (Molina & Flores, 1999). The agility includes the ability to dynamically and efficiently package and deliver *highly customized services that maximally satisfy the utility of service consumer demands over the Internet (*(Molina & Flores, 1999)*; (*Rabelo et al, 2007;2008;2009) . Dynamic webservice composition (DWSC) is an essential ICT enabler of this form of agility in VOs (Rabelo et al, 2007; 2008,2009).

However, webservice composition in dynamic environments such as virtual organizations remains a non-deterministic polynomial hard multiple criteria decision making optimization problem (Moghaddam et al , 2014; Xu et al, 2012);Mahboobeh & Joseph, 2011; Xu et al, 2011; Bartalos & Bieliková,, 2011; Singh, 2012 ; Mulongo et al, 2015;2016a, 2016b) despite a decade of extensive research on the topic. This means that in some cases, it's not feasible to find an optimal solution to the webservice composition problem within practically acceptable time. This limits its viability for problems of industrial relevance. The very nature of the problem lends itself to a mathematical programming solution. In the literature, Mixed Integer Programming (MIP) is the most widely used mathematical programming technique to tackle the webservice composition problem in situations requiring dynamic decisions. However, existing MIP techniques suffer from one or a combination of the following problems: - 1) exponential state space explosion implying that in some cases, MIP may not yield a solution in practically acceptable time, 2) some MIP models lack the ability to capture and take into account the global optimization constraints meaning that for this class of MIP solutions, service consumers are denied the chance to specify some critical global constraints, and consequently the quality of solutions produced are sometimes suboptimal and 3) all existing methods require service consumers to specify preferences and weight ratings over the whole set of webservice QoS attributes including low level performance attributes. This requirement can be tedious to the end user (Benatallah, 2004) and at the same time, while all the QoS attributes might be essential, some of the attributes may be too technical to be discernable by an average user. Therefore a pressing question is: *How can we design a more efficient Mixed Integer Programming* composite *webservice selection strategy that can produce* high quality composite webservice *solutions without: - denying service consumers an opportunity to specify all*

their critical local and global constraints ? This question is worthwhile answering albeit a difficult one. The main purpose of thesis is to address this research question.

In order to appreciate the significance and the intricacies involved in addressing the foregoing research question, in section 1, we first provide some background to Webservice Composition distinguishing between workflow based and Artificial Intelligence planning based service composition, and dynamic workflow based from static workflow based service composition in section 1.1.1. This is followed by a brief introduction to the Virtual Organizations (VOs) and how dynamic webservice composition addresses the challenges in VOs in section 1.1.2. In section 1.1.3 we highlight the key issues that make dynamic webservice composition challenging. In section 1.1.4, we expound on the current MIP models to the problem and discuss their weaknesses and thus provide a case for why further research is needed. We then outline a statement of the problem in section 1.2. In section 1.3, the overall research goal is highlighted. Two specific research objectives are stated in section 1.4. A list of research questions are posed in section 1.5. We give an overview of our proposed solution in 1.6, whose formal details can be found in section 2.10. The scope of the thesis is given section 1.7. Sections 1.8, 1.9 and 1.10 respectively discuss the justification for the study, an overview of how the rest of the thesis is organized and a summary of this chapter.

1.1 Background

1.1.1 Webservices and WebService Composition

A web service is a distributed software component that enables machine to machine interaction over the network using standard network protocols such as the Simple Object Access Protocol and *REST*. Webservice composition on the other hand, is a process that involves the discovery, selection, linking and execution of a set of atomic distributed webservices in a specified logical sequence in order to service complex customer request that none of the services could fulfill singularly (Rao, 2004), (Rao & Su, 2004; Schahram & Wolfgang, 2005; (Cammarimha and Arfsamanesh, 2007;Arfsamanesh et al, 2012). Webservice composition can be achieved through the use of workflows or through the use of Artificial Intelligence (AI) planning (Rao & Su, 2004). Workflow based webservice composition involves defining a business process detailing the logical sequence of tasks that should be performed in order respond to some consumer need. The business process is then automated into a computer executable workflow using standards such as the

Business Process Execution Language, and business process execution engines such as the Oracle, IBM or Activiti. The workflow is also referred to as an abstract composite service (Rao Jinghai, 2004), (Rao & Su, 2004). Once an abstract composite service is defined, a concrete service composition is required to link each of the abstract tasks within the workflow to some concrete executable webservices. The result of a concrete webservice composition process is a concrete composite service (Rao & Su, 2004). The execution of an abstract workflow, causes the concrete composite service to be executed to produce the outcome desired by a service requestor. On the other hand, Artificial Planning approaches aim at fully automated web service composition- no human intervenes; both the logical sequence of tasks to be performed and the web services to be linked through the sequence are unknown a priori; they have to be established only at runtime automatically based on the knowledge inferred from a service request (Rao, 2004). However, despite many years of scientific research on the subject, the AI techniques are still far from real and are yet to find their way into industry (Mahboobeh & Joseph, 2011). Therefore currently, owing to its worldwide adoption and strong industry support, workflow based service composition remains the only viable option for VOs. Hereafter, our focus in this thesis therefore is on workflow based webservice composition and therefore unless otherwise, workflow based webservice composition and webservice composition shall be used interchangeably.

Workflow based webservice composition is further classified into *static webservice composition* and *dynamic webservice composition (*Schahram & Wolfgang, 2005). In dynamic workflow based web service composition, the web service that is to execute a workflow task is unknown a priori until the workflow is executed in response to an external service request (Zeng et al, 2004). In this case, when the workflow is invoked, the set of web services that best answer the demands of the request at a point in time has to be first discovered ,selected from a service repository and invoked in the logical order enforced by the workflow. Dynamic workflow based service composition is contrasted from *static workflow based webservice composition,* in that in the latter, each workflow task is bound to a known web service in advance at design time and the binding can only be altered manually. Static workflow based webservice composition is easier to implement than dynamic webservice composition. Moreover, static webservice composition suffices in business cases where there is no need for customizing service responses in line with consumer specific QoS requirements. Further, if there was negligible variance in the QoS attributes of each component webservice all the time, then the use of static webservice composition would still produce the best

known composite webservice, whichever combination of webservices is defined at design time. However, in dynamic environments, customer QoS needs will vary from time to time and from customer to customer. Secondly, in real service environments, the QoS attributes of individual webservices will vary both at design time and at runtime. Considering the two scenarios, static service composition are severely limited. Thus, today due to its potential benefits, dynamic webservice composition has become an active area of research. This thesis explores dynamic workflow based webservice composition.

1.1.2 Dynamic Webservice Composition and Virtual Organizations

A virtual organization (VO) is a dynamic, temporary, and strategic alliance of many independent and heterogeneous firms (that have their unique core business competencies), that are logically interconnected using Information and Communication Technology networks (Molina & Flores , 1999; Rabelo & Gusmeroli., 2008; Amit et al , 2010; Arfsamanesh et al,2012). Globalization, sophistication of product and service development, fast shifting consumer demands, coupled with stringent time to market constraints are the driving forces behind the emergence of VOs (Molina & Flores, 1999). Due to the these constraints, no single firm, even large ones have the internal capacity and the time required to develop and deliver a complex composite product (The need for business agility, value added services, efficiency in service delivery of value added services and customer centricity are distinguishing survival strategies for global virtual organizations (Molina & Flores, 1999), (Rabelo & Gusmeroli, 2008), (Amit et al , 2010), (Arfsamanesh et al ,2012). In VOs, firms (usually small scale) share competencies, business processes and resources to fulfill a specific market need where none of them can independently deliver the need Mulongo & Flores, 1999). According to Molina & Flores (1999) the VOs are characterized by:

i. Heavy reliance on innovation and information technology and customer centricity.
ii. Independence and short lived relationships among the enterprises. New service providers or suppliers with better products or services can be substituted, removed, added by the virtual enterprise broker when needed in order to respond to customer requests. This exemplifies the agility and dynamic nature of VEs.
iii. Business agility.
iv. Customer centricity.

In order to support the dynamic nature and business objectives of VOs, the ideal information technology framework for virtual organizations should at minimum provide mechanisms for collaboration, negotiation, interoperability and integration of business processes (Picard W. et al, 2010). The authors in (Rabelo & Gusmeroli., 2008) identify Service Oriented Architecture (SOA) as the appropriate information technology framework for VEs. SOA supports the agility of VEs through web service composition (Picard W. et al, 2009). In their ICT-Infrastructure reference framework for collaborative networked organizations Rabelo & Gusmeroli (2008) identifies *webservices* and *webservice composition* as essential ICT services required to facilitate inter enterprise business process integration and coordination towards fulfilling a complex consumer request. In the context of VOs, the different distributed webservices are each owned by geographically disperse entities called virtual enterprises (VE), where a VE is formally defined according to (Molina and Flores, 1999). The webservices are the software components that produce the data required to execute one of the business tasks required to fulfill a particular business process e.g. an online purchase order process. By leveraging the core competencies of each VE exposed via webservices in the VO, VOs can quickly generate a more value added composite service that meets a complex market demand using the concept of *webservice composition*.

Under the VO reference architectures by (Molina and Flores, 1999), (Rabelo & Gusmeroli, 2008), and (Picard, 2009), among many other things, the responsibility of implementing webservice composition lies with a business entity called *Virtual Enterprise Broker (*VEB). From the point of view of the service consumer, a VEB is the service provider. By exploiting workflow based webservice composition, Virtual Enterprise Brokers, can quickly define a new business process, automate the workflow and assemble already existing webservices owned by different virtual enterprises, to execute the workflow to produce a more value added composite service that satisfy a certain market demand. The reuse of already existing business capabilities and already existing software technology components promotes the business agility differentiating factor of VOs and reduces time to market.

However, the degree of business agility of a VEB does not only depend on how fast the VEB is able to generate a composite service from already existing atomic webservices, but also on how well the generated composite service satisfies specific quality of service (QoS) requirements

demanded by different service requestors from time to time (Mulongo et al, 2016a). The implication is that the workflow based service composition strategy chosen by the VEB needs to be highly adaptive and sensitive to the webservice QoS requirements of each and every service consumer at all times. If the VEB is able to achieve this requirement, then the *customer centricity* distinguishing element of VOs would be a reality. Due to the time varying nature of consumer demands, the volatility of the VOs and the large number of service providers within a VO, as seen in section 1.1.1, static service composition falls short in adapting to changing external needs of service consumers and internal changes within the VO service environment. On the other hand, even though dynamic webservice composition has many challenges but inhibit its utility, it's a more promising technology solution to the business challenges of VOs.

The successful implementation of dynamic webservice composition would offer the VEBs the following benefits. (1) Improved likelihood of the service consumer obtaining high quality solutions because the best composite service is selected from a pool of many potential solutions. Even in the event that no suitable solution is found that satisfies the consumer, the user can be provided with the list of feasible solutions and choose whether or not one of them nearly satisfies them, (2) Through re-planning strategies, workflows that are dynamically bound to webservices at runtime are more likely to survive failures through selection of different execution paths hence boosting system reliability and customer experience.

In spite of the benefits dynamic webservice composition has to VEBs, a number of factors make the technique a formidable challenge. Some of the key challenges are highlighted in section 1.3

1.1.3 Key Challenges of Dynamic Webservice Composition in Virtual Organizations

1.1.3.1 Large Size of Virtual Enterprise Providers with Similar Services

In a global virtual organization operating within a particular business domain, there are potentially hundreds to thousands of small to medium virtual enterprises offering competing functionally similar simple services (Abiud et al, 2015). The total number of service providers summed from each category of services is even larger (Abiud et al, 2015). Although in each cluster, the services may be functionally similar, they may be differentiated on some quality of service (QoS) criteria. Even when the differentiating factor is a single QoS parameter, the sheer numbers of services make the selection of the best composite service a challenge. To put this into perspective, consider a

composite travel reservation product that contains four simple services: flight service, hotel service, insurance package and a taxi service. Assume further that for each of the simple services, there are 10 service providers. When a virtual enterprise broker is faced with a customer request enquiring for a trip, the VEB is required to select the best combination of four services, 1 from a pool of 10 candidate services. It's easy to show that there are 10^4 or 10,000 possible composite services from which to select the best service. A marginal change from 10 to 20 services per category exponentially escalates the solution space to 160,000 and 100000000 for 100 services per task.

1.1.3.2 Large Number of Webservice Quality of Service Attributes

Functionally equivalent webservices (each webservice provided by a different enterprise) can exhibit significant variations in quality of service along dozens of QoS parameters (Zeng et al, 2004) . A close examination of the number of papers on webservice QoS such as Zeng et al (2004), (Rajendran and Balasubramanie 2009), (Xu et al, 2011), (Mahboobeh & Joseph, 2011), (Kuyoro Shade et al, 2012), reveal a wide range of important QoS parameters associated with webservices. From these studies and others, the most common webservice QoS attributes are *reliability, availability, response time, reputation, security, cost and throughput*. The combination of the dimensionality of QoS attributes with even a small number of services exponentially increases the combinatorial complexity of the service selection problem. Intuitively the problem is expected to worsen as the both the number of QoS attributes and the number of candidate services grows larger. The challenge to the virtual enterprise broker transforms from just *how to select the best composite service from a large set services based on a single criterion* to *how to efficiently select the best combination service* from a huge *set of services on multiple criteria*. Further, in this case, the selection should factor in constraints and preferences that are either explicitly stated by the service consumer or implied by user needs

1.1.3.3 Large QoS Constraints by Service Consumers

From a fixed set of webservice QoS attributes, different service consumers could enforce varying number of QoS constraints from time to time. The larger the number of QoS constraints the more the complex it becomes to solve the dynamic webservice composition problem.

1.1.3.4 Volatile Service Environment

In virtual organizations, new entrants (VEs) with more quality services could join the VO or already existing VEs could exit the VO in the middle of a service composition process. Alternatively, some webservices may become temporarily unavailable, timeout or workflows may develop internal errors during the composition process. These challenges necessitate embedding transaction management and fault handling and replanning strategies within the composition process to ensure that workflow execution is sustained in the presence of faults or composition decision are re-adjusted in the middle to factor in potentially high quality webservices that have just joined. Including fault handling and replanning mechanisms even though desirable only escalates the computational effort required to solve the dynamic webservice composition problem.

1.1.3.5 Complex Workflow Patterns

The fundamental structure of a workflow is the sequential pattern. But more complex workflows can take parallel patterns, XOR patterns and a combination thereof. Workflows with more complex patterns can cause the composition process to be harder in computational effort and even more prone to faults (Bartalos & Bieliková,2011). For example, workflows containing parallel flows inherently have the same challenges of parallel programs such as synchronization, deadlocks and data inconsistencies if not well handled.

1.1.4 Overview of Mixed Integer Programming

As stated earlier, dynamic webservice composition is a multiple criteria non deterministic polynomial hard optimization problem. There are two main classes of multiple criteria decision making algorithmic solutions to the dynamic webservice composition problem: - *local planning optimization* algorithms and *global planning optimization* algorithms (Zeng et al, 2004). In each of these approaches, the objective is to maximize some utility function over a set of decision variables that are constrained. The utilities are computed using the Simple Additive Weighting model (Hwang & Yoon, 1981). In local planning approach, for each workflow task, the webservice with the highest aggregate utility value and that also satisfies the QoS constraints is selected (locally) without regard to other tasks within the workflow (Zeng et al,2004). The combination of the best service for each task forms the best composite webservice. Suppose there are *k* sequential workflow tasks and each has *n* candidate webservices per task, the solution space

using a local planning approach is *nk*. Thus, the most naive local planning strategy that evaluates each and every candidate webservice is still polynomial time.

Although local planning algorithms to the webservice composition problem are provably polynomial time and hence more suitable for real time or near real time e-Commerce applications, they lack support for global constraints. The result is that local planning denies a service requestor the chance to express critical global webservice QoS e.g in a situation where a service consumer requires that the total service execution (or access) cost should not exceed a particular budget and/or the total execution duration of tasks should be less than some threshold value (Abiud et al 2015; 2016). Further, because of its inability to capture global constraints, local planning algorithms have a high probability of yielding suboptimal solutions (Zeng et al, 2004).

Global planning based algorithms on the other hand overcome the limitations of local planning models by considering global constraints across workflow tasks. Given sufficient time, global planning is guaranteed to yield an optimal solution. For a business workflow having *k* sequential tasks with each task having *n* possible, the solution space is n^k possible candidate composite webservices when using the global planning strategy. A naïve global planning algorithm would have to compute the utilities of all the n^k composites and then evaluate each of the composites against specified QoS constraints and select the best composite webservice subject to the set of QoS constraints (Zeng et al, 2004). The limitation of naïve global planning strategies is that when the variables *n* or *k* or both grow larger, the algorithms severely suffer exponential state space explosion, hence an optimal solution within reasonable for large enough *n* and *k* is computationally intractable (Zeng et al, 2004).

An alternative to the naive local planning and naïve global planning algorithms is to apply Mixed Integer Programming, MIP (Byod et al, 2003) for optimization of composite service selection. MIP is an efficient technique for modeling and solving many real world optimization problems in which some variables take on integer values while other variables are continuous (Zhu, 2006), (Kitching, 2010) , (Tramontani , 2008) , (Mancini et al , 2009), (Kitching, 2010), . As such, today MIP is the most widely used method to address the webservice composition problem. Unlike naïve approaches to the problem, MIP does not attempt to exhaustively search the entire solution space, but instead relies on intelligent enumeration techniques such as the branch and bound to efficiently arrive at an optimal solution (Zeng et al 2004; Mancini et al , 2009; Ed Klotz, & Alexandra

Newman, 2012). In the literature, there are thus two complementary MIP models exist for the webservice composition problem: (1) MIP based on *local planning optimization* strategy, hereafter *L-MIP* and 2) MIP based on global planning optimization strategy, hereafter, *S-MIP*. Like other local planning and global planning techniques, both L-MIP and S-MIP are formulated guided by the Simple Additive Weighting (SAW) model by HC-L and K. Yoon (1981). Although L-MIP is more efficient than the naïve local planning method, it still shares the same limitations of local planning methods - inability to capture global constraints, prohibiting the service consumer from expressing critical constraints that span workflow tasks, and thus high likelihood yielding suboptimal solutions. For instance the study in (Ardagna & Pernici, 2005) experimentally compared the solution quality of L-MIP and S-MIP and established that on average, L-MIP yields solutions that are 20% to 30% worse in quality compared to global planning approaches. Thus L-MIP and other local planning strategies are limited to application areas where inter workflow task constraints is not a requirement.

S-MIP on the other hand inherits the major strength of global planning algorithms- the ability to capture global workflow QoS constraints and produce solutions that are more optimal compared to L-MIP. Although S-MIP is generally far more efficient than naïve global planning optimization methods for the webservice composition, S-MIP still suffers exponential state space explosion as the complexity of the problem grows larger in terms of number of service providers per task, number of workflow tasks, number of constraints and complexity of workflow patterns. This constrains the applicability of MIP to small scale problems only (Zeng et al, 2004). Zeng et al (2004) pioneered the formulation of a global planning algorithm for dynamic webservice composition based on MIP, hereafter we will refer to this algorithm as S-MIP. As an example, Zeng et al (2004) shows that the runtime performance of S-MIP starts to severely sore on workflows having more than 40 webservices per task. In general, finding an optimal or near optimal solution for large complex optimization problems using MIP in some cases may be intractable in practice (Toni et al, 2009), (Ed Klotz, & Alexandra. Newman, 2012).

One would argue that with the current state of the art computing hardware and high performance computing technologies, it should be possible to solve MIP problems very fast. On the contrary, up to date research shows that even combing the most sophisticated MIP libraries such as the IBM CPLEX with the fastest computing infrastructure would still take a couple of minutes, through

hours to a couple of weeks to solve a given optimization problem, depending on many factors such as the structure and formulation of the problem, the number of constraints, the coefficients of constraint inequalities, the bounds of the on the right hand side of the constraint inequalities (Ed Klotz, & Alexandra Newman ,2012).

Another way of solving complex service composition problems is to cast the problem as a Satisfiability (SAT) Problem. In SAT, a problem is specified in form of propositional logic and derivative modelling formalisms such as Descriptive Disjunctive Logics (DDL). Although SAT problems are NP complete (Cook, 1971), many very efficient SAT algorithms exist today such as SATPlan (Kautz & Selman., 1992), WalkSAT (Kautz & Selman., 2004), and GraphPlan (Blum & Merrick , 1997). These algorithms are applicable to a large spectrum of practical problems. For instance within webservice composition research, SATPlan and SATPlan are recommended for complex operator large scale service selection (Seog & Soundar, 2006). Other closely related service selection optimization algorithms include A* and its variants, genetic algorithms, Answer Set Programming (ASP). Answer Set Programming (Lifschitz ,2002; Yu, 2005a; 2005b), is based on DDL and has been proven to be very efficient as exemplified by the work (Rainer & Dorn, 2009). However, as a downside, SAT and other Artificial Intelligence Planning based approaches to web service composition are limited in their scope of application in the following ways (Abiud W.M et al, 2015): - First, for most complex problems, it's always difficult to model some problems efficiently as SAT problems (Kitching, 2010) . Second, Artificial Intelligence Planning and SAT solutions are more naturally suited to semantic webservices composition. The reason for this is because; semantic webservices are semantically annotated using Artificial Intelligence like languages easily allowing for automated reasoning. But to date, semantic webservice composition is yet to bear any fruits in commercial use. On the contrary, workflow based service composition based on WSDL services continue to enjoy strong industry support as they permeate many business applications. Third, generally, SAT and Constraint Satisfiability Problems are plagued by the same inadequacy seen in mathematical programming techniques such as MIP- the plague is exponential state space explosion (Mancini et al , 2009).

In addition to efficiency and optimality considerations, all the existing algorithmic solutions to the webservice composition have one shared deficiency: they require the service consumer to specify relative ratings and preferences in terms of weights on the interval [0, 1] on all the webservice QoS

attributes (Mulongo et al, 2015). It becomes tedious for the end user to capture the weights especially when the number of QoS parameters is huge (Zeng et al, 2004). Moreover, a good number of the QoS parameters even though very relevant, are too technical to make sense to an average user (Abiud et al, 2015). A strategy that allows the possibility of capturing local and global constraints over all available QoS attributes, while shielding the end user from the complexity of having to specify weight ratings on too low level QoS attributes is needed (Mulongo et al, 2015).

Thus, from the foregoing, research gaps are glaring and the need for further research into better webservice composition architecture and algorithms that address all or a subset of these issues is evident.

1.1.5 Overview of Layering as Optimization Decomposition

Layering is not a new terminology in computer science. From a software engineering viewpoint, layering is one of the architectural decomposition techniques of partitioning a large complex software system into simpler components called layers (Bachmann, 2000). The components are strictly ordered with one layer *A (the lower layer)* providing services to another layer *B* (the upper layer) (Bachmann, 2000) .The immediate advantage of layering is modularity, modifiability and portability ((Bachmann, 2000). The second advantage of layering is that it hides technical complexity of the computational details of lower layers, with the topmost layer having the least technical details. Eventually layering boosts usability of a system.

Counter intuitively, as a third benefit, layering can be used as an efficient algorithmic method in solving hard optimization problems found in complex computing and communication systems. In order for layering to be used in as an optimization tool, a formal theory is required. But while layering is an old decomposition technique, formal theories on layering as an optimization technique can only be traced in the communications and computer networks community. *Layering as Optimization Decomposition* (Mung, 2006), Mung. *et al. 2007*) and (Low, 2012) is such a theory. The theory provides a framework for rigorous and quantitative formulation of layering as a divide and conquer algorithm towards solving complex cross layer hard network resource allocation and scheduling optimization problems. *Layering as Optimization Decomposition* theory perceives the task of solving a complex cross layer communication network design optimization problem as the solution to multiple well-coordinated subproblems, in which each layer is treated as a subproblem. Each layer aims at maximizing its local utility but together all

layers aim towards maximizing some global utility. The theory has led to the modularized and layered reformulation of the Network Utility Maximization problem. The reformulation of the NUM (Kelly et al, 1998) problem based on the theory has been applied to re-engineer the TCP/IP protocol stack with appreciable performance improvements. We refer the reader to Chapter 2 for more details on the theory of Layering as Optimization Decomposition.

Whereas Layering as Optimization Decomposition is well formalized in the networking community and proven to lead to more efficient, optimized, modular and transparent layered networks, a similar formalization in the webservice composition lacks and equally to the best of our knowledge, there exists no method that exploits or is inspired by the inherent advantages of the theory. In this thesis, we argue that although Layering as Optimization Decomposition formalism is rooted in the Network Utility Maximization problem, the complexity of issues involved in the web service selection problem, as discussed in section 1.3 closely resemble the NUM (Kelly, Maulloh and Tan, 1998) problem and therefore Layering as Optimization Decomposition as used in the networking should inspire a reformulation of existing webservice composition optimization strategies.

A primary objective this thesis is in understanding how Layering as Optimization Decomposition could inspire the improvement in efficiency of the current MIP algorithmic solutions to the problem of the service composition problem. A secondary objective is to understand how the technique could be applied to reduce the burden on the user in dynamic webservice composition.

1.2 Statement of the Problem

Consider a Virtual Organization in which the Virtual Enterprise Broker offers a composite service through several virtual enterprises that are interconnected via the Internet. On a service consumer requesting the composite service, a business workflow with k sequential tasks has to be executed to effect the service. Each task is executable by an appropriate webservice exposed on the Internet by a virtual enterprise within the virtual organization. For each workflow task, there are n alternative webservices. Each webservice has a total of q_t webservice quality of service attributes including but not limited to service availability, service reliability, service throughput, service access cost, service reputation, security, service response time etc. At any one time, the QoS values of the q_t attributes vary from webservice to webservice. Further, the service consumer has minimum QoS constraints expectations about the composite webservice being requested for. The

problem is: - *efficiently autogenerate a sequence of k webservices that when executed, maximally satisfies the QoS requirements of the service consumer*. This is the *dynamic composite webservice selection problem*. The problem is an essential ICT capability for collaborative virtual organizations (Rabelo et al, 2007) as it affords virtual enterprise brokers and virtual enterprises the business agility required to adaptively respond to complex time varying online consumer service requests in a globally competitive market Mulongo & Flores, 1999). Despite its potential benefits, over the last 10 years, due to the coupling of the factors discussed in 1.3, dynamic webservice composition remains a multiple criteria decision making (MCDM) nondeterministic polynomial hard optimization problem (Zeng et al, 2004; Seog et al, 2006; Mahboobeh and Joseph , 2011). This limits the range of industrially relevant problems for which DWSC can find a high quality solution within practically acceptable time (Mulongo et al, 2015; 2016).

Mixed Integer Programming, MIP is a well-known method for efficiently modeling most complex optimization and decision problems in which the variables can take on a combination integer, real or binary values. As such the technique has been widely applied to model and solve the DWSC problem. There are two complementary MIP models in the literature for the dynamic webservice composition problem. MIP exploiting a local planning optimization strategy, hereafter dubbed *L-MIP* and MIP exploiting global planning strategy, hereafter *S-MIP*. The L-MIP technique is provably polynomial time but lacks support for global inter workflow task constraints on webservice quality of service, and therefore in some cases denies the service consumer an opportunity to specify critical QoS constraint that span more than one business workflow task. In addition, due to its local scope, the quality of L-MIP solutions is highly probable to be suboptimal. For instance, Ardagna & Pernici (2007) show that on average, L-MIP yields solutions that are 20% to 30% worse in quality compared to MIP using a global planning strategy. S-MIP on the other hand is capable of generating optimal solutions, and is more efficient than naïve global planning algorithms. However, when the service composition problem grows in complexity in the number of webservices per task, or number of QoS constraints or a combination of these, existing S-MIP is susceptible to exponential state space explosion and therefore practically constrained to small scale webservice composition problems (Zeng et al, 2004). For example, Zeng et al (2004), experimentally shows that beyond 40 webservices per workflow task, the run time performance of S-MIP starts to severely dip. In addition, all existing multiple criteria decision making algorithms to the problem require the service consumer to specify weight preferences and QoS constraints on

the entire range of relevant QoS attributes. This requirement can be very tedious to an end user (Zeng et al, 2004), (Mulongo et al, 2015). Thus, dynamic webservice composition algorithms that are more efficient and provide minimum guarantee on the solution optimality quality, without sacrificing the ability of service consumers to capture global QoS constraints, and without requiring the end user to interact with all QoS attributes and their constraints are urgently needed.

1.3 Research Goal

As explained in the preceding sections, the global planning mixed integer programming approach, S-MIP suffers *exponential state space explosion* even though it supports both *local and global constraints* and guarantees *global optimality*. On the other hand, the local planning mixed integer programming models, L-MIP *supports only local constraints* and produces solutions that are *less quality* compared to S-MIP, even though L-MIP is practically several orders faster than S-MIP.

Our main aim was to design a *more efficient Mixed Integer Programming* dynamic composite webservice selection strategy that does not *deny service consumers an opportunity to specify all their critical local and global webservice QoS constraints.* Our approach takes the S-MIP model and converts it into a two layered model inspired by the concept of *Layering as Optimization Decomposition*. The main output of the research is a new architectural model called "Service Layered Utility Maximization", *SLUM* model together with the associated mathematical models for the dynamic composite webservice selection problem (see section 1.7 for an overview of SLUM and section 2.10 for details). See also (Abiud W. M. et al, 2015; 2016a; 2016b) for associated publications.

To know whether we reached our reach goal, our proposed architecture, *SLUM* was evaluated in terms of two metrics: *run time execution efficiency* and *solution quality* against the S-MIP and L-MIP models. Running time and solution quality are the main two metrics of evaluating optimization models (Eitan, 1981). Therefore our research goal will be said to have been achieved if the proposed architectural model, SLUM, satisfies the two requirements below:-

Requirement 1: The proposed two layer model, SLUM is faster than the (single layered) global planning mixed integer programming model, SLUM and;

Requirement 2: The solution quality of the proposed two layer model, SLUM is at least as good as that of the local planning mixed integer programming model, L-MIP.

1.4 Specific Research Objectives

Given the research goal stated in section 1.3, the specific research objectives that were pursued were:-

1. Design a layered hierarchical mixed integer programming model for the composite webservice selection problem following the concepts from the theory of Layering as Optimization Decomposition
2. Evaluate the performance of the SLUM model against the single layered global planning technique (S-MIP) and the local planning method (L-MIP) in terms of two metrics:
   i. Running time (performance efficiency) and;
   ii. Solution quality.

The two metrics, *running time* and *Solution quality* are the most common performance measures for optimization models (Eitan, 1981), (Hoos et al, 2003). For details on the performance evaluation methodology see chapter 3.

1.5 Research Questions

As explained in sections 1.1, 1.1.4 and section 1.2, it's known that the running time of the S-MIP is non deterministic polynomial and can be exponential in some cases especially when the number of webservices grows larger. It's also known that S-MIP produces more optimal solutions than L-MIP. On the other hand, it's also known that the running of L-MIP has a polynomial upper bound and thus as empirically demonstrated in (Zeng et al, 2004), L-MIP is many orders of magnitude faster than S-MIP. Similarly, it's also known that L-MIP on average produces less optimal solutions whose quality is on average in the range 70% to 80% relative to S-MIP. What is unknown is how SLUM could perform in terms of runtime efficiency and solution quality relative to S-MIP and L-MIP. This leads us to the research questions stated in section 1.5.1 and section 1.5.2.

1.5.1 Running Time

The overall research question concerning performance efficiency is:

RQ1: For a composite webservice selection problem having a workflow with k tasks and v alternative webservices per task, how does the runtime efficiency of *SLUM compare with that of S-MIP and L-MIP* when each is used to solve the problem*?* The specific research questions arising from this question are:

RQ1.1: How does the running time of SLUM grow as the number of service providers per task increase?

RQ1.2: How does the running time growth of SLUM compare with that of S-MIP and L-MIP?

RQ1.3: How much speedup is achievable when using SLUM over S-MIP to autogenerate composite webservices given a business workflow having n webservices per task?

RQ1.4: What is the minimum number of service providers per workflow task that a virtual enterprise broker needs to have in order to benefit from the relative efficiency of SLUM when compared to S-MIP?

The significance of RQ1.1 is to derive a mathematical model that characterizes the performance efficiency of SLUM as a function of the number of service providers per task. From the model, we could infer the running time complexity either as linear, polynomial, exponential or a combination of the same. Given that the composite webservice selection problem is still non deterministic polynomial, the second rationale for RQ1.1 is that by deriving the running time model(s) of SLUM, we are interested to investigate whether or not SLUM can guarantee polynomial time solution to the problem. In case it does not, the problem then remains NP hard and our results will further empirical evidence of the NP hardness of the composite webservice selection problem. If otherwise, then we could conclude that the two layered approach proposed herein constitutes a polynomial time solution to the problem that has been known to be NP hard. RQ1.1 is also a pre-requisite to answering research question RQ1.2.

Using the model obtained after answering RQ1.1, we are mainly interested in benchmarking the performance efficiency of SLUM against S-MIP. We will also gauge the performance efficiency of SLUM against L-MIP. This seeks to establish whether SLUM is worse than S-MIP or better and under what circumstances. This is the goal of RQ1.2.

In case our hypothesis that SLUM is faster than S-MIP turns out to be true, RQ3 aims to establish how much faster on average should a virtual enterprise broker expect SLUM to be when compared to S-MIP.

Sometimes, an algorithm can be faster on average than another algorithm only when the problem size is very large enough, say in thousands or millions. Since the problem size in our case is the number of virtual enterprises per task, SLUM wouldn't be practically useful if its improved speedup is realizable only for huge *n* values, since it's unlikely that the number of virtual enterprises per task is infinitely large. The goal of RQ1.4 is to determine the number of virtual enterprises per task that a virtual enterprise broker should have in order to gain the minimum possible speedup from SLUM. If RQ1.4 can be answered, then a virtual enterprise broker could also estimate the speedup of SLUM at any number of virtual enterprises per service.

1.5.2 Solution Quality

The overall research question concerning solution quality is:-

RQ2: How does the average solution quality of SLUM compare with that of L-MIP and S-MIP?

This leads us to the following specific research questions?

> **RQ2.1**: What is the average relative solution quality (percentage accuracy of quality) of the composite webservices generated by SLUM relative to S-MIP?
>
> **RQ2.2**: What is the percentage difference in the relative solution quality by SLUM relative to L-MIP?

In order to provide answers to the foregoing research questions, a system methodology for analyzing and comparing the performance of the three algorithms is required. With respect to efficiency/running time, algorithms can analyzed using two main approaches: *theoretical (mathematical) or empirical.* In the theoretical approach, a mathematical model is developed that characterizes the performance behaviour of the algorithm, and the algorithm is analyzed within the model. The empirical approach involves running an algorithm and testing its performance against specific problem instances and collecting performance data (Hoos, 2003), (Seogewick & Flajolet, 2009). Empirical evaluation of algorithms complements theoretical/mathematical approach (Coffin & Saltzman, 2000).

In section 2.14, we attempt to answer research questions *RQ1.1* and *RQ1.2* using theoretical/mathematical analysis. The theoretical results could pre-empt some analytic performance efficiency properties of SLUM independent of specific machine implementation

details – the basis of theoretical algorithm analysis in computer science. These results could provide a benchmark on running time properties of the SLUM model against the benchmark algorithms during empirical evaluation.

1.6 Overview of Our Proposed Approach

Towards our research goal stated in 1.3 and the two research objectives in section 1.4, this thesis is about converting the well-known (single layered) global planning MIP based dynamic webservice composition model originally formulated by Zeng et al (2004) which is the basis for present MIP based webservice selection models that are based on MIP, into a multilayered MIP model. Inspired by the formal theory of *Layering as Optimization Decomposition* (see section 1.1.5 and chapter 2), we propose a hierarchical two-layer Mixed Integer Programming model dubbed *SLUM: Service Layered Utility Maximization* as in (Abiud et al, 2016); also synonymously referred to as Hierarchical Multilayer Service Composition Model, HMSCM as in (Abiud et al, 2015).

Instead of viewing dynamic composite service selection as a one shot monolithic complex problem as it's the case with all the existing strategies, motivated by the theory of *Layering as Optimization Decomposition*, we view the service composition problem as a *network with two hierarchical layers* in which one of the layers is a mixed integer optimization subproblem whose objective is to maximize the utility of a service consumer on a subset of the QoS attributes. The other layer is a mixed integer optimization subproblem whose objective is to maximize the utility of the service provider (in this case, the virtual enterprise broker) on the remaining subset of QoS constraints. Both layers exploit global planning allowing users to specify both global and local constraints. The layer concerned with maximization of the utility of the service consumer is termed as the *Service Consumer Utility Maximization (SCUM) layer*, while that one concerned with maximization of the utility of the *Service Provider as the Service Provider Utility Maximization Layer (SPUM)* (Abiud et al, 2015). The formulation of the SCUM subproblem is in terms of QoS attributes that are a direct concern of the service consumer and are less technical in nature. The SPUM subproblem is formulated in terms of low level technical attributes that would only be a direct concern of the service provider. The layering is done in a manner that the optimization at the SPUM layer is completely transparent to the service consumer i.e the end user is not required

to specify weight preferences and constraints explicitly on low level technical– this is instead handled by the virtual enterprise broker, even though optimization of such QoS attributes indirectly benefits the service consumer without their explicit knowledge. The two subproblem are then sequentially solved beginning with the lower layer of the two layers such that the output solutions of the lower layer becomes the inputs to the upper layer, and the output of the upper layer constitutes the best composite service that meets the needs of the end user (and implicitly the needs of the virtual enterprise broker. In chapter 2 and chapter 3, we show that together, the two layers attempt to solve the global optimization objective and that our approach is more efficient and less tedious to the end user than the state of the art. Two questions that arise are: 1) exactly what kind of webservice QoS attributes belong to the SPUM layer and which ones belong to SCUM layer and why? And 2). In which order are the subproblem solved – SCUM then SPUM or SPUM then SCUM. Why? These questions are problems in themselves. We provide detailed answers to these questions in chapter 2.

1.7 . Scope and Limitations of the Study

In section 1.1.3, we saw that there are a number of key issues that make the dynamic webservice composition a complex multi-dimensional problem. In this section, we discuss what the study focused on and what the study did not address and the reasons justifying the decisions.

1.7.1 Nature and Scale of the Virtual Organization

SLUM targets virtual enterprise brokers operating within global virtual organizations as described in (Molina & Flores, 1999). The envisaged global virtual organization framework is that one in

1.7.2 Nature and Pattern of Business Workflows

Although the proposed model is generic enough, in this thesis, we assume workflow based service composition only as defined in Rao et al (2004). To simplify analysis without loss of generality, this thesis will only focus on sequential workflows even though it should not be hard to extend the model to other complex workflow patterns. By focusing on sequential workflows only, we avoided extraneous factors such as parallel performance issues that would arise when dealing with workflows containing parallel gates. In any case, parallel tasks could be abstracted as a high level sequential tasks. Moreover, in selecting the best composite service where a workflow contains a parallel task, the task with the largest execution time is usually used which is essentially sequential.

1.7.3 Nature of the Service Environment

The study does not address issues related to a changing service environment due to events such as occurrence faults, new service exits and service entries in the middle of an ongoing webservice composition – doing so would require replanning mechanisms. This is beyond the scope of the study. Such issues have been substantively addressed in studies such as (Urban et al, 2011). Further, the method proposed in this study could be used in conjunction with existing fault aware replanning strategies, for example the one proposed in (Zeng et al, 2004).

1.8 Significance of the Study

1.8.1 Significance to Industry and Practitioners

Dynamic webservice composition is an essential ICT infrastructure support service for collaborative virtual organizations (Rabelo et al, 2007), Mulongo & Flores, 1999). In a globally competitive market, DWSC affords virtual enterprise brokers and virtual enterprises the business agility required to adaptively respond to complex time varying online consumer service requests Mulongo & Flores, 1999). On the other hand, dynamic webservice composition gives the service consumer all the benefits highlighted in section 1.1.2.

The study introduces more efficient and near optimal service composition strategy that could boost applicability of dynamic webservice composition. Note that as explained in all the preceding sections, the applicability of dynamic webservice composition is severely limited to virtual organizations that operate a relatively small network of service providers due to computational complexity of the problem. As virtual organizations, lead by virtual enterprise brokers span across the globe, the need for efficient DWSC strategies will grow. The successful adoption of dynamic webservice composition that is also efficient and near optimal would offer the benefits to various stakeholders as outlined in subsections 1.8.1.1, 1.8.1.2 and 1.8.1.3.

1.8.1.1 Significance to Service Consumers

i. Improved likelihood of the service consumer obtaining highly customized quality solutions because the best composite service is selected from a pool of many potential solutions (Mulongo et al, 2016). Even in the event that no suitable solution is found that satisfies the consumer, the user can be provided with the list of feasible solutions and choose whether or not one of them nearly satisfies them.

ii. Through re-planning strategies, workflows that are dynamically bound to webservices at runtime are more likely to survive failures through selection of different execution paths hence boosting system reliability and customer experience.

iii. Enhanced convenience resulting from shorter turnaround times in online services.

1.8.1.2 Significance to Virtual Enterprise Brokers & Virtual Enterprises

This study proposed a more efficient dynamic webservice composition strategy that could boost adoption of dynamic webservice composition. In some real time Internet business applications such as ultra-low latency trading platforms, response time is as good as the quality of service delivered. Nielsen et al (1993; 2010), (Nah , 2004), (Akamai , 2009) and Nngroup (2014) note that tolerable waiting limit of web application users is typically 4 seconds with 10 seconds considered as annoying. Broadwell (2004) describes response time as a critical user centric performance factor for online services alongside data quality. Nielsen et al (1993; 2010; Nah, 2004; Akamai, 2009; Nngroup 2014) agree that response time is a critical determinant of customer retention as well customer churn.

Further, (AgileLoad, 2012) demonstrates that the computational logic within the server side of the web applications accounts for the largest percentage of performance efficiency delays – 76%. Thus improved runtime performance of dynamic webservice composition could enhance user experience.

Current methods of user centric dynamic webservice composition require that the user express preferences and constraints on the entire set of available webservice QoS constraints. As stated earlier, some QoS attributes could be too technical to be comprehensible by an average user. The method proposed in this study, SLUM, in addition to efficiency gains and optimality, could

improve usability of dynamic webservice composition as SLUM does not require end users to directly specify QoS constraints and weight preferences on low level technical QoS attributes.

1.8.2 Significance to the Research Community

The study introduces a new architectural thinking about the structure of the dynamic composite webservice selection problem. The new thinking borrows the idea of Layering as Optimization Decomposition and its combination with mixed integer programming– an optimization approach that has been applied successfully in the communication systems field. The study thus pioneers one of the possibly many layering schemes and layering models that could be used to tackle the dynamic webservice composition problem and structurally related problems. The following are some of the different types of researchers that would find this work relevant.

i. Service computing researchers who may be interested in exploring improved layering models or layering schemes based on this study.
ii. Other computer science researchers. Dynamic webservice composition as explained in section 1.1 can be viewed as planning problem. Researchers from other areas of computer science research could find relevance in exploring new applications of the layered mixed integer programming approach introduced in the study.
iii. Experts in decision theory, optimization theory, management science and operations research would find this work of interest either with a view to extending it in solving related problems that require multiple criteria decision making.

1.9 Operational Definitions

There are key terms used throughout the rest of the thesis that:

i. Have interchangeable meanings and or ;
ii. Have overloaded meaning i.e could mean more than one thing or;
iii. Have different meanings in other domains.

These words are: virtual organization (VO), collaborative networked organization (CNO), virtual enterprise broker, virtual enterprise, service provider, provider, service requestor, service consumer, and consumer.

To resolve ambiguity in the usage of the above terms, in this thesis, the foregoing terms will be used as explained in the following subsections.

1.9.1 Service Provider and Provider

In Service Oriented Architecture, from a business perspective, a service provider is the business entity that offers a particular service to service consumers (IBM, 2004). From a technology view point, a service provider is the software application component that enables the business service being offered by the business entity by providing appropriate service responses to the service consumer (IBM, 2004). For the purposes of this thesis, the term *service provider* shall be strictly used from the business view point as explained here. This view is also consistent with the recent definition of a *service provider* according to (Terlouw & Albani, 2013). Terlouw (2013) defines service provider as used in service oriented architecture as *the party that offers a service to consumers.*

In addition, when viewed from a business angle, the word *service provider* and the word *provider* can be used interchangeably (Terlouw & Albani, 2013). Thus in this thesis, the two terms are used interchangeably.

1.9.2 Service Consumer, Consumer and Service Requestor

In Service Oriented Architecture, from a business perspective, a service consumer is the party that requests a service offered by as a service provider (IBM, 2004). From a technology view point, a service consumer is the client side software component that mediates the user requesting for the service and the service provider (IBM, 2004). According to the webservice architecture framework by IBM (2004), term *service consumer* and *service requestor* are used interchangeably and from the business perspective as opposed to technology perspective. Further, the two terms *service consumer* and *service requestor* are used interchangeable with the term *consumer*. This is consistent with the definition of the word *consumer* by (Terlouw & Albani, 2013) i.e a consumer is the party that requests for a service offered by a provider.

1.9.3 Virtual Organization, Collaborative Virtual Organization and Collaborative Networked Organization

In the open literature, the term *virtual organization* is often used rather liberally for example to imply an organization where employees can telecommute. This is not the meaning ascribed to the term in this thesis. The context of usage of this term is within the domain of electronic commerce involving business to business collaborations. Precisely, in this thesis, the term *virtual organization* used as defined in (Molina & Flores, 1999). Refer to section 1.1.2 for a detailed background on virtual organization. The term virtual organization is also referred to as *collaborative virtual organization* or *collaborative networked organization* (Rabelo et al, 2007; 2008). Unless explicitly stated otherwise, the meaning and usage of these three terms in this words remains as such.

1.9.4 Virtual Enterprise Broker

Molina & Flores (1999) identifies, defines and describes the concept of *virtual enterprise broker* as a fundamental component of the architecture framework for virtual organizations. As explained earlier in section 1.1, according to the architecture framework for virtual organizations by Molina (1999), *a virtual enterprise broker* is the consumer facing business entity, typically with:-

i. the expertise in analyzing market demands and complex consumer needs ;
ii. the expertise and responsibility in identifying business opportunities arising from the market demands;
iii. the ability and responsibility to design complex value added services/products to meet the demand;
iv. the ability and responsibility of identifying a set of industry specific expert business entities called *virtual enterprises* (see section 1.9.5 for definition of the term *virtual enterprise)* to contribute to the production and delivery of the complex product or service.
v. The responsibility to broker and manage the delivery of the service/product to the consumer. This also involves setting up the requisite computing infrastructure required for consumers to access the service, order and purchase the service and managing the quality of the service being delivered to the consumer.

The thesis adopts the definition of the term as described in Molina & Flores (1999). Moreover, since in the context of a virtual organization, the virtual enterprise broker is the business that provides services to the consumer, from a consumer perspective, the definition of a virtual enterprise broker then coincides with the definition and usage of the term *service provider* as used in this thesis. For this reason, in this thesis, *virtual enterprise broker* and *service provider* are used interchangeably.

1.9.5 Virtual Enterprise.

A virtual enterprise is another fundamental component within a virtual organization according to Molina & Flores (1999). In the framework in (Molina & Flores, 1999), a virtual enterprise is a business entity specialized in a particular industry domain e.g insurance, education, aviation, information technology and so on. Typically, a single virtual enterprise might not have the competencies to wholly deliver a complex product/service identified by the virtual enterprise broker ((Molina & Flores, 1999). The virtual enterprise would then need a combination of virtual enterprises typically from different industry domains to deliver the service (Molina & Flores, 1999).

1.10 Organization of this thesis

The rest of this thesis is organized as follows:-

Chapter two contains a review and analysis of literature related to *dynamic webservice composition*, local planning, and global planning methods to dynamic webservice composition, with a particular emphasis on the mixed integer programming solution to the problem. The theory of Layering as Optimization Decomposition and how it has been used in the communication networks community is discussed elaborately. Gaps in related literature are stated. Chapter two also contains our proposed solution dubbed "SLUM: Service Layered Utility Maximization" which uses a two layer mixed integer global planning to the dynamic webservice composition problem. We first give a qualitative description of how the formulation of SLUM maps onto the Layering as Optimization Decomposition conceptual framework justifying every design decision made. This is followed a detailed description of the mathematical model underpinning SLUM. We then describe how our proposed framework differs from the state of the art, discussing both its relative strengths and relative limitations.

Based on our research objectives and research questions stated in section 1.4 and 1.5 respectively, we go ahead in chapter two to derive some theoretical performance efficiency models related to the running time of SLUM and derive a mathematical model that can be used to estimate the speedup of SLUM in relation to S-MIP. The mathematical performance models can be found in section 2.12. Later this model is verified experimentally in chapter four.

Chapter three, *Methodology* contains a detailed description of the experimental methods that were used to verify and validate the proposed model and the corresponding theoretical performance models. In part, the experimental methodology complements the theoretical analysis of chapter

two and on the other side, provides an empirical validation tool of the claims made under the theoretical analysis approach. Under the experimental method, we begin by explaining the performance metrics - *CPU running time* and *Relative Solution Quality* used to benchmark our proposed solution in more details. This is followed by a detailed description of the *experimental protocol*- the procedure that was followed to achieve implausible experimental results. It's in this section that we discuss how we addressed the various issues that could threaten validity of our experiments. We then identify and discuss a set of complementary as well alternative methods of analysis in the section titled "*Performance Data Analysis Methodology".* The analysis methods and tools discussed include: - *Statistical Regression Analysis (linear, polynomial and exponential regression),* the *L-Hospital's Rule from differential calculus,* the *empirical relative complexity and empirical relative complexity coefficients, parametric and nonparametric statistical tests* , use of central measures of *tendency and lastly scaling curves* . We justify the use of one or a combination of these approaches over the other and the contribution of each method towards the understanding of the performance behaviour and performance differences between our proposed method against the baseline (S-MIP) and against the alternative algorithm (L-MIP). We also discuss the basis for interpretation of various results.

Chapter four contains details on the specific experiments carried out and the results obtained. The results are generally captured using tables and visually represented using scatter plots and where appropriate bar graphs to show the performance of SLUM, S-MIP and L-MIP algorithms. Where appropriate, a series of regression functions are obtained from the scatter plots, their goodness of fit computed and their statistical significance tested, and results interpreted as per the basis provided in the preceding section, "*Performance Data Analysis and Interpretation methodology".* Moreover, using the obtained regression functions the following are parameters are determined: - expected growth of the function; this is based on the L-Hospital's rule, the empirical relative complexity and empirical relative complexity coefficients. From these parameters, a couple of other quantities are derived. A series of equations are used appropriately to capture these parameters. From the raw data that is tabulated, we also compute a number of descriptive statistics based on arithmetic mean and show how they enrich the understanding of performances differences among the three algorithms. Finally, a discussion of the empirical results follows. The discussion is presented in a form that is designed to show the answers to the research questions. Moreover, in our discussions we relate our empirical findings to the following: - the analytic

arguments, the theory and mathematical analysis presented in chapter two, theory from the rest of computer science body of knowledge, and the results obtained from previous studies. We also discuss any peculiar findings that cannot be immediately linked to any known theory.

Chapter five contains the conclusions obtained from the results. Here, we reflect on the dynamic webservice composition problem as stated in chapter 1, and then highlight our major contributions towards solving the problem. As we discuss our contributions, we also explain the limitations of our approach informed by the results and our analytic considerations in chapter two, clearly describing the conditions which our method would be preferred over mixed integer programming using a local planning strategy and mixed integer programming using a global planning strategy. We then give recommendations that help the virtual enterprise broker tackle the *algorithm selection problem* as first described by John (1976), which in this case is *"when should L-MIP, S-MIP and SLUM be used* for dynamic webservice selection problem *?"*, This section also contains highlights of future work given that dynamic webservice selection problem remains an active area of research due to its significance, albeit with many issues that remain unresolved – this study tackles just a tiny portion of the issues.

1.11 Chapter Summary

In this chapter, we introduced the concepts of *dynamic webservice composition* and *virtual organizations.* We identified the main problem faced in solving the dynamic webservice composition problem within virtual organizations. We then identified the two complementary approaches that are currently used to tackle the problem*: Local Planning using Mixed Integer Programming and global planning using Mixed Integer Programming*. The overall strengths and weaknesses of these methods were discussed and gaps that necessitate further research were identified. We also introduced the concept of *Layering as Optimization Decomposition* and highlighted how the theory has been used to efficiently solve related optimization problems in the communication networks field. We then articulated the problem statement followed a statement of our research goal, which is *Design a more efficient Mixed Integer Programming model that can dynamically generate the best composite service taking into account both local and global QoS constraints, and without requiring the service consumer to specify all QoS constraints.* From the research goal, two specific objectives were identified in section 1.4 and two main research questions stated in section 1.5.We gave an overview of the proposed solution SLUM, which is

inspired by the theory of Layering as Optimization Decomposition. We then discussed the scope of this study and explained the justification for the study. We gave operational definitions of key terminologies that are used throughout the thesis. Finally, we gave an overview of how the rest of the thesis is organized.

2 CHAPTER 2: LITERATURE REVIEW

As stated in chapter one, dynamic webservice composition is a non-deterministic polynomial hard multiple criteria decision making problem. The goal of this study is to explore the use of a two layer mixed integer programming model (SLUM) to tackle the problem. This chapter presents in detail the literature that contributed to the formulation of the proposed model. In the section (2.1) titled *"Introduction to Service Oriented Architecture and Computing"*, key concepts and models that shall be referenced throughout this dissertation are discussed. Section 2.2 revisits the problem of *workflow based dynamic webservice composition* in more elaborate terms. The *structural model* and the *process model* underpinning webservice composition are discussed. In addition, the design space complexity of DWSC is expounded. The rationale for this topic is to set the context of our specific work within a larger body of existing issues related to DWSC.

In chapter one, we indicated that a majority of the existing algorithms (including MIP algorithms) that combat the DWSC problem are based on the multiple criteria decision making, Simple Additive Weight, SAW in (HC-L & K.Yoon, 1981). In section 2.3, we present and discuss in details the naïve *local planning utility maximization* and the naive *global planning utility maximization* mathematical models related to DWSC. The standard (single layer) mixed integer programming model (S-MIP) for DWSC based on the work of Zeng et al (2004) is elaborated in section 2.4 . While as will be later seen, the models in section 2 and section 2.5 contribute to the mathematical foundation of our proposed solution, the work in section 2.5 contributes to the structural and process models our solution, helping answer the question: *how should the layering be structured* and in what order should the *utility maximization process be done*.

In section 2.6, the theory of Layering as Optimization Decomposition and how it has been used in the communication networks field is presented. The general concepts will then be applied into the proposed model.

In section 2.7, a review of the related work is presented. The related work reviewed is grouped as follows. The focus is on those studies generally following the SAW method by Hwang & Yoon, (1981) which is the basis for a majority of multiple criteria decision techniques for the service composition problem. As explained in (Mulongo et al, 2015; 2016a), to the best of our knowledge, no previous work has dealt with *Layered Mixed Integer Programming Model* for the webservice

composition problem before. Therefore other than our work in (Mulongo et al, 2015; 2016a), there is no any previous related work.

In section 2.7, a summary of the gaps in literature are discussed. In section 2.8, similarities and differences between our proposed solution and existing work is given.

In section 2.8, we highlight a summary of the gaps in the literature.

In section 2.9, we present our proposed model. We begin by presenting a *qualitative model of SLUM* in section 2.9.1. Section 2.9.1 discusses the rationale for major architectural decisions e.g what webservice QoS attributes should be placed within the SCUM layer and which ones should be placed under the SPUM layer. It also attempts to answer the question, which of the layer serves the other i.e should solve the SCUM subproblem first then SPUM second or vice versa and why? Using, *Layering as Optimization Decomposition theory* discussed earlier, these questions are answered. In summary, section 2.9.1 is about " the *structural view* and the *process view* (without the internal mathematical model details) of the *dynamic webservice composition problem* inspired by the Network Utility Maximization Model , NUM problem and *Layering as Optimization Decomposition theory* discussed in section 2.7. In section 2.10.2, formal mathematical models underlying SLUM are presented. A Mathematical optimization model at the SCUM layer and another one at the SPUM layer are detailed. Both models are based on the S-MIP. Finally, the optimization process given the two mixed integer programming models is described. A summary of how our proposed model addresses the research question: *Design* a *more efficient Mixed Integer Programming* webservice composition strategy that can *produce high quality solutions that are on average near global optimal* without: *denying service consumers an opportunity to specify all their critical local and global webservice QoS constraints.*

In section 2.10, we present the man differences between our approach and the state of the art. Section 2.11 outlines the two benchmark algorithms – one baseline – the S-MIP and the other as an alternative – the L-MIP.

In section 2.12 we present mathematical performance models for the proposed model. This formal model on one hand is verified experimentally in chapter 3. Conversely, the results that will be obtained in chapter three can be checked against the formal models.

Table 1: Literature Review Road Map

Concept, Theory, Framework, Model, Method	Section	Contribution
Introduction to Service Oriented Computing & Service Oriented Architectures	2.2	SLUM
Dynamic Workflow based Webservice Composition	2.3	*SLUM*
Local Planning Strategy for Webservice Selection	*2.4*	*SLUM*
Global Planning (general) based Service Selection	*2.5*	*SLUM*
Integer Programming Model for Service Composition	*2.6*	*SLUM*
Layering as Optimization Decomposition	*2.7*	*SLUM*
Related Work	2.8	
Summary of the Gaps in the Literature	*2.9*	*SPUM*
Summary of Gaps in the Literature	*2.9*	*SLUM*
Proposed Systems Model (SLUM)	*2.10*	*SLUM*
Qualitative and Architectural Model	*2.10.1*	*SLUM*
Mathematical Models	*2.10.2*	*SLUM*
Differences between our Proposed Model and the rest	*2.11*	*SLUM*
Benchmark Algorithms	*2.12*	
Research Questions	*2.13*	*SLUM*
Theoretical Running time performance results	*2.14*	*SLUM, Phase Transition* Rates

2.1 Introduction to Service Oriented Architecture

Service Oriented Architecture (SOA) is paradigm for organizing and utilizing distributed capabilities that may be under the ownership of different domains. Therefore a basic element of SOA is the notion of a *service (*Picard et al, 2010). The notion of a service has multifaceted meanings and definitions. As such there exists no precise definition and mutual understanding of the term service (Terlouw & Albani, 2013). However, the debate about what a service is or is not is beyond the scope of this work. Instead we will pick on working definitions that are commonly used in literature.

A service is an interaction between a requesting party called a consumer and an offering party called a provider or service provider or supplier (Terlouw & Albani 2013).

According to the Open Group (2007) and Picard et al (2010), Service Oriented Architectures facilitate business agility in the following ways: - Service discoverability enables business to expose functionalities that can be quickly discovered by service consumers. In virtual organizations and virtual enterprises, this means that virtual enterprise brokers can easily locate the partners given a business opportunity (Picard. et al, 2009). Another important benefit is service reuse through *service composition*. Within enterprises, a business can easily combine a subset of existing atomic services to create a new composite service with less software effort (Khan .H.L et al, 2010).

Webservices are the most widely used technology to implement service oriented architectures. A webservice is a software system designed to support interoperable machine to machine interaction over a network. Discoverability is one of the properties of a webservice. This means that webservice can be located and identified (usually via a URL) by other software systems over Internet (Schahram & Wolfgang, 2005).

XML is the most widely used format for describing a web service although other formats such as JSON can be used for REST webservices. For XML based services, The WSDL (W3C, 2001) is the widely adopted language for describing the grammar and syntax of a webservice. With the advent of the semantic web and semantic webservices, other description languages such as OWL, OWL-S 1, WSML2, USDL, and WSDL-S 3 and WSMO have emerged. These languages seek to enable fully automated discovery of webservices by adding semantic annotations to service descriptions. Most of them are based on Artificial Intelligence concepts especially description logics.

Figure 1 below is the webservice architectural model. The architecture shows the main actors, components and operations supported by a service oriented architecture based on webservices (IBM, 2004). This model is also the same the one described in (Dustdar& Schreiner, 2005). The actors/components are service provider, service requestor /consumer and the components are the service registry, the operations are found, publish, and bind.

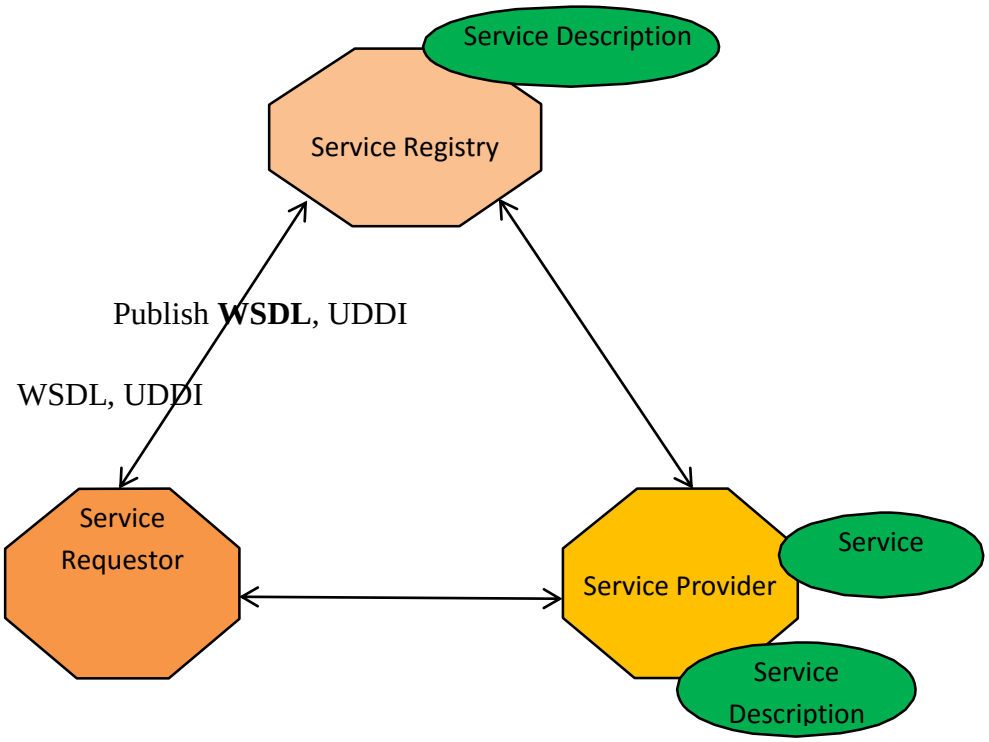


Figure 1 : A basic Webservices Architecture, Source (IBM, 2004)

2.1.1 Actors and Components of the Service Oriented Architecture

2.1.1.1 Service Requestor/Service Consumer

Service requestor or service consume from a business perspective, is the business or user that requires certain functions to be satisfied. From an architectural perspective, this is the application that is looking for and invoking or initiating an interaction with a service. The service requestor role can be played by a browser driven by a person or a program without a user interface, for example another Web service

2.1.1.2 Service Registry/Service Repository

Is a searchable registry of service descriptions where service providers publish their service descriptions? Service consumers find services and obtain binding information (in the service descriptions) for services during development for static binding or during execution for dynamic binding. For statically bound service requestors, the service registry is an optional role in the architecture, because a service provider can send the description directly to service requestors. There exist other ways in which service consumers can obtain information about a service

description: , a local file, FTP site, Web site, Advertisement and Discovery of Services (ADS) or Discovery of Web Services

2.1.2 Basic Computational Operations in a Webservices Model

2.1.2.1 Publish Operation

A publish operation enables services to be accessible to service requestors through search accessible, a service description needs to be published so that the service requestor can find it. Where the service is published can vary depending upon the requirements of the application

2.1.2.2 Find Operation

Enables a service requestor to retrieve a service description directly or by querying the service registry for the type of service required. Thus the find operation facilitates the process of *service discovery.* The find operation can be involved in two different lifecycle phases for the service requestor: at design time to retrieve the services interface description for program development, and at runtime to retrieve the service's binding and location description for invocation (IBM, 2004).

2.1.2.3 Bind Operation

Enables a service request to invoke a *service* at run time by use the binding details provided in the service description.

A *service* and a *service description* constitute artefacts of a web service. Whereas a web service is an interface described by a service description, the concrete implementation of the interface is called a *service* **(**IBM, 2004**)**. Extending this definition, a service is a software module deployed on network accessible platforms, provided by the service provider. *Service description* contains the details of the interface and implementation of the service. This includes its data types, operations, binding information and network location. It could also include categorization and other metadata to facilitate discovery and utilization by service requestors, for example in semantic web services.

2.1.3 Webservice Publication

The publication of Web Services includes the production of the service descriptions and the subsequent publishing. Publishing can use a variety of mechanisms (IBM, 2004). The service description can be generated, hand-coded, or pieced together based on existing service interface

definitions. Developers can hand-code the entire service description, including the UDDI entry. Tools exist to generate parts of the WSDL and potentially parts of the UDDI entry from meta-data artifacts from the programming model and the deployment of the Web service executable.

A service description can be published using a variety of methods. These various methods provide different capabilities depending on how dynamic the application using the service is intended to be. The service description can be published to multiple service registries using several different approaches. The simplest case is a *direct publishes*. A direct publish means the service provider sends the service description directly to the service requestor. Direct publish can occur after two business partners have agreed on terms of doing e-business over the Web, or after fees have been paid by the service requestor for access to the service. In this case, the service requestor can maintain a local copy of the service description but will need to occasionally update the service whenever changes in the service description occur on the side of the service provider.

Slightly more dynamic publication uses DISCO or ADS. Both DISCO and ADS define a simple

HTTP GET mechanism to retrieve Web Services descriptions from a given URL. An enhanced m service description repository would provide a local cache of service descriptions, but with additional search capabilities. For service description repositories that span hosts within an enterprise, a service provider would publish to a private UDDI server. There are several types of private UDDI nodes that can be used depending on the scope of the domain of Web Services published to it. Internal Enterprise Application UDDI node: Web Services for use within a company for internal enterprise applications integration should be published to a UDDI node of this kind. The scope of this UDDI node can be single application, departmental or corporate. These UDDI nodes sit behind the firewall and allow the service publishers more control over their service registry and its accessibility, availability and publication requirements. Portal UDDI node: Web Services published by a company for external partners to find and use can use a portal UDDI node. A portal UDDI node runs outside the service provider's firewall or between firewalls. This kind of private UDDI node contains only those service descriptions that a company wishes to provide to service requestors from external partners. This allows companies to retain control of their service descriptions, access to the UDDI node and quality of service for the UDDI nodes. Partner Catalog UDDI node: Web Services to be used by a particular company can be published to a partner catalog UDDI node. A partner catalog UDDI node sits behind the firewall. This kind of

private UDDI node contains only approved, tested and valid Web service descriptions from legitimate business partners. The business context and meta-data for these Web Services can be targeted to the specific requestor. E-Marketplace UDDI node: For Web Services that the service provider intends to compete for requestors' business with other Web Services, the service description should be published to an e-marketplace UDDI node or the UDDI operator node. E-marketplace UDDI nodes are hosted by an industry standards organization or consortium and contain service descriptions from businesses in a particular industry.

2.1.4 Webservice Discovery

The discovery of Web Services includes the acquiring of the service descriptions and the consuming of the descriptions. Acquiring can use a variety of mechanisms. Like publishing Web service descriptions, acquiring Web service descriptions will vary depending on how the service description is published and how dynamic the Web service application is meant to be. Service requestors will find Web Services during two different phases of an application lifecycle design time and runtime.

At design time, service requestors search for Web service descriptions by the type of interface they support. At runtime, service requestors search for a Web service based on how they communicate or qualities of service advertised. With the direct publish approach; the service requestor caches the service description at design time for use at runtime. The service description can be statically represented in the program logic, stored in a file or in a simple, local service description repository. Service requestors can retrieve a service description at design time or runtime from a service description repository, a simple service registry or a UDDI server. The look-up mechanism needs to support a query mechanism that provides find by type of interface (based on a WSDL template), the binding information (that is, protocols), properties (such as QOS parameters), the types of intermediaries required, the taxonomy of the service, business information, and so on. The various types of UDDI servers have implications on the number of runtime binding Web Services to choose from, the policy for choosing one among many, or the amount of prescreening that must be done by the requestor before invoking the service.

UDDI servers can classified as either internal, partner catalog or e-Market place (IBM, 2004). Internal UDDI repositories are those used to publish services within an enterprise, partner catalog

UDDI repositories are those shared among trusted business partners, e-Market place UDDI repositories are accessible to any service requestor via the Internet.

Internal enterprise application UDDI servers and partner catalog UDDI servers will require no prescreening to establish trust of the service. Service selection can be based on binding support, historical performance, and quality of service classification, proximity, or load balancing.

E-marketplace UDDI nodes will have more runtime services to choose from. Some prescreening must be done to verify that the Web service provider is a worthy partner. A service can be chosen based on price promises, cost, presence on approved partners list, as well as binding support, historical performance, quality of service classifications and proximity.

After a service description is acquired, the service requestor needs to process it to invoke the service. The service requestor uses the service description to generate SOAP requests or programming language-specific proxies to the Web service. This generation can be done at design time or at runtime to format an invocation to the Web service. Various tools can be used at design time or runtime to generate programming language bindings from WSDL documents. These bindings present an API to the application program and encapsulate the details of the XML messaging from the application.

2.2 Dynamic Workflow Based Webservice Composition

Web service composition involves combining a set of individual web services to respond to a service request that cannot otherwise be achieved using any single service (Dustdar & Schreiner, 2005), (Bartalos & Bieliková,2011).Web service composition is a critical research challenge in the realization of business agility (Rabelo et al, 2007).

Webservices composition takes different approaches and forms. In chapter 1, we saw that there are two paradigms of webservice composition – one following the workflow based service composition and the other following the Artificial Intelligence Planning approach. Further we saw that workflow based composition can be static or dynamic. This thesis is dedicated to dynamic workflow based service composition. In subsection 2.3.1.1, we formally define the notion of a *business process*, a *workflow* and *task*. We then discuss the types of workflow operations. In subsection 2.3.1.2, we go ahead to elaborate on the activities involved in dynamic workflow based composition. To contrast dynamic webservice composition from static composition, we also

illustrate static workflow based service composition. A recap of the issues that make dynamic workflow based service composition a hard optimization problem are discussed in subsection 2.3.1.3

2.2.1 Business Process, Workflows, Tasks and Workflow Patterns

As stated in chapter one, the scope of this work is limited to workflow oriented web service compositions mainly because virtual enterprises as elaborated in chapter 1 have specific products and services that are delivered to the public through well-defined and known business processes. According to the Workflow Management Coalition[1], a process is as a representation of a business process in a form that supports automated manipulation such as modelling or enactment by a workflow management system. A process is composed of a set of activities/tasks, each task corresponds to the execution of given operations. A workflow is an automated business process. According to the Business Process Modelling notation (BPMN), the execution of workflow tasks can follow different logical patterns e.g sequential, parallel, exclusive OR and so on. BPMN also defines different types of tasks based on the agent that executes them. Hence we have human tasks that require human intervention, service tasks that can be executed by computer programs automatically etc. This study is dedicated to workflows that are fully automated via webservices. Figure 2 shows a sequential workflow whose tasks are executed by webservices. Figure 3 illustrates a workflow in which some tasks are performed in parallel, while figure 4 captures a workflow in which at one of the steps one and only of the two tasks is executed based on some business logic. Hence forth, our discussions are within the context of purely sequential workflow.

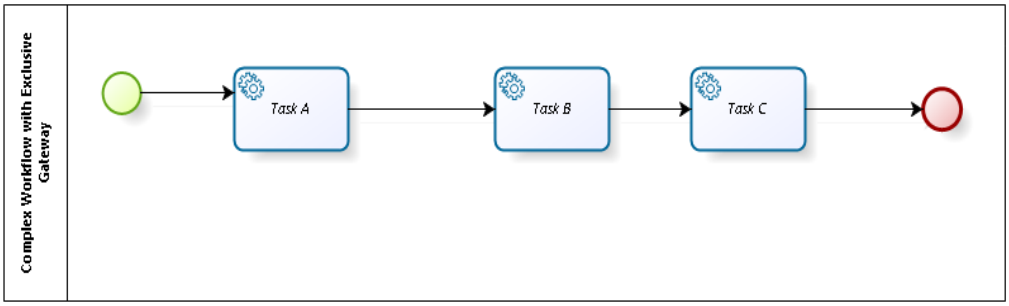


Figure 2 : Example Sequential Workflow with webservice tasks

[1] http://www.wfmc.org/

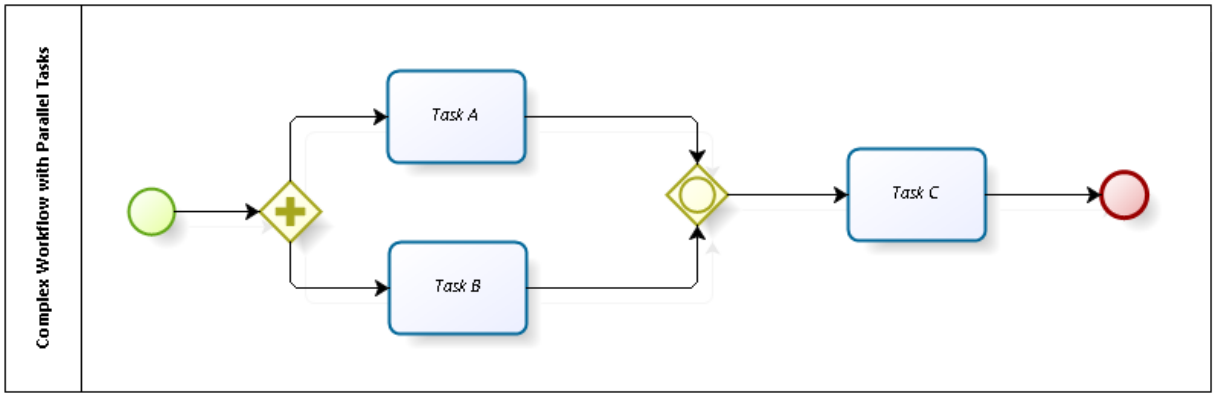


Figure 3 Example Complex Workflow with Parallel Webservice Tasks

Figure 3 shows a workflow with two parallel tasks and one sequential task.

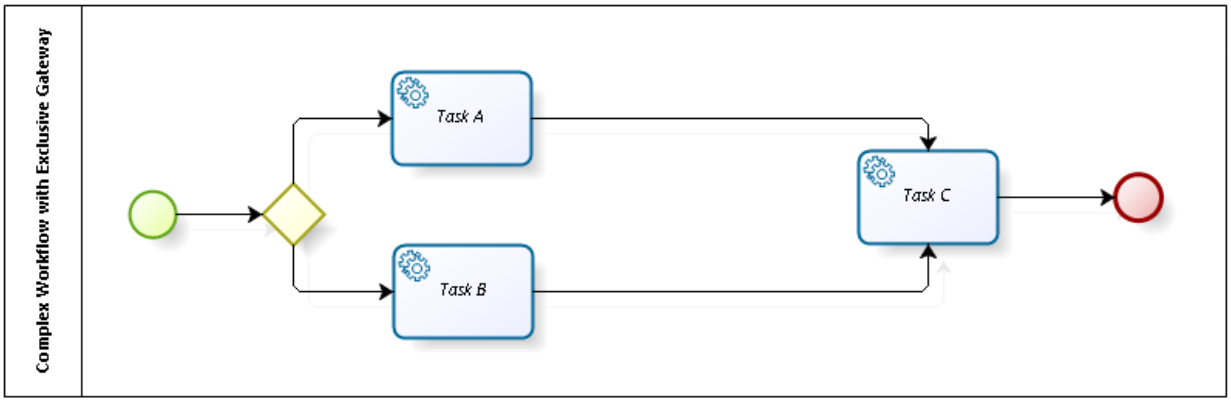


Figure 4 Example Complex Workflow with Exclusive OR Gateway

In figure 4, either task A or task B will be executed based on some business rules then followed by the execution of task C.

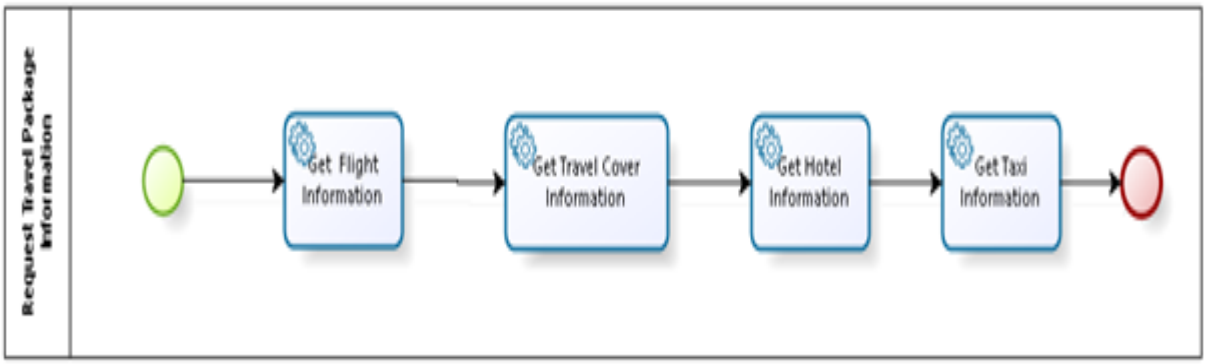


Figure 5 Example Travel Planning Sequential Workflow

In the example in figure 5, we illustrate a pure sequential workflow based on a simplified version of the well-known travel planning and reservation webservice composition problem. The business process involves four sequential tasks. A customer makes an enquiry concerning available flights and their details. If flight enquiry task is successfully executed, then the customer queries travel insurance cover information that is related to the destination of the flight and then once satisfied, enquires about availability of hotels in the destination area and finally once the first three tasks satisfy the user, he makes enquiry about local taxis in the destination area. Eventually the process of booking flight, cover and so on (not included here) might follow.

On the basis of workflow patterns, a distinction is made between *simple operator* and *complex operator* webservice composition (Seog et al, 2005). In the context of workflow based compositions, simple operator service composition is the one in which the workflow tasks are connected by the sequential flow pattern only such as the one shown in figure 2, while complex operator service composition involves workflows in which there are flow patterns other than the sequential flow pattern. These patterns include the parallel gateway, the exclusive OR gateway etc (Seog et al, 2005).

2.2.2 Dynamic Workflow Based Service Composition Process

To better understand dynamic workflow based webservice composition, we first illustrate *static service composition*. In static service composition, for each workflow task, a corresponding service component is linked to the workflow task and design time and finally the workflow is deployed Schahram & Wolfgang, 2005). This means that at runtime, the service component associated with a particular task cannot be automatically changed. While this is an easier

composition strategy, it's severely limiting. For example failure of any one of the component services automatically implies failure of the workflow and therefore the service consumer does not get desired outcomes (Mulongo et al, 2016a). Secondly, the technique is insensitive to specific needs of a service consumer (Mulongo et al, 2016a). Figure 6 illustrates static workflow based webservice composition.

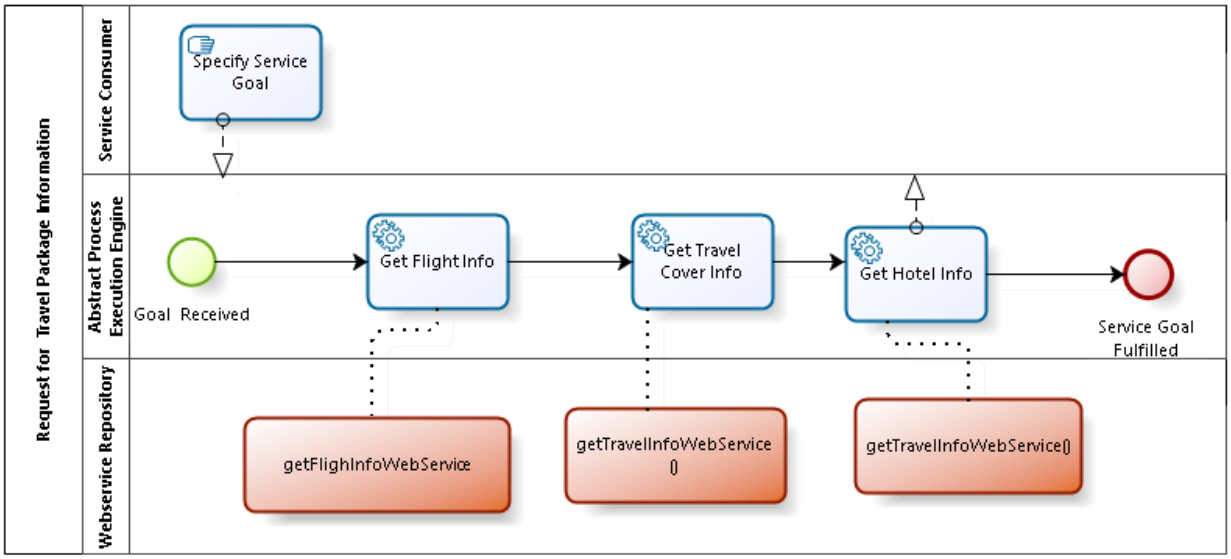


Figure 6 Static Webservice Composition in Workflow Based Service Composition.

The dotted line without an arrow shows an association between an abstract task and a concrete webservice. In static service composition as can be seen, each abstract workflow is associated with exactly one sequence of concrete webservices. i.e, before execution of the workflow, there is one to one mapping between the workflow and the composite service. Thus prior to workflow execution, the sequence of webservices to be known in known in advance.

To address the deficiencies of static webservice composition, dynamic composition is needed to automatically adapt to unpredictable changes in the service environment and to adapt to customer requirements with minimal or no user intervention (Schahram & Wolfgang, 2005). In dynamic service composition based on workflows, the services that bind to a workflow task are not known in advance. For each workflow service task, a corresponding concrete webservice need to be determined at run time. This flexibility means that there could be more than one concrete service for each abstract task that can fulfill the task. The result is a set of possible sequences of concrete webservices that can be used to realize a single abstract workflow. Usually then in dynamic webservice composition, the problem is reduced to service selection problem. How do we select

the best composite service from the available sets to fulfill the user requirements specification? Figure 7 shows dynamic webservice composition involved in the travel planning problem.

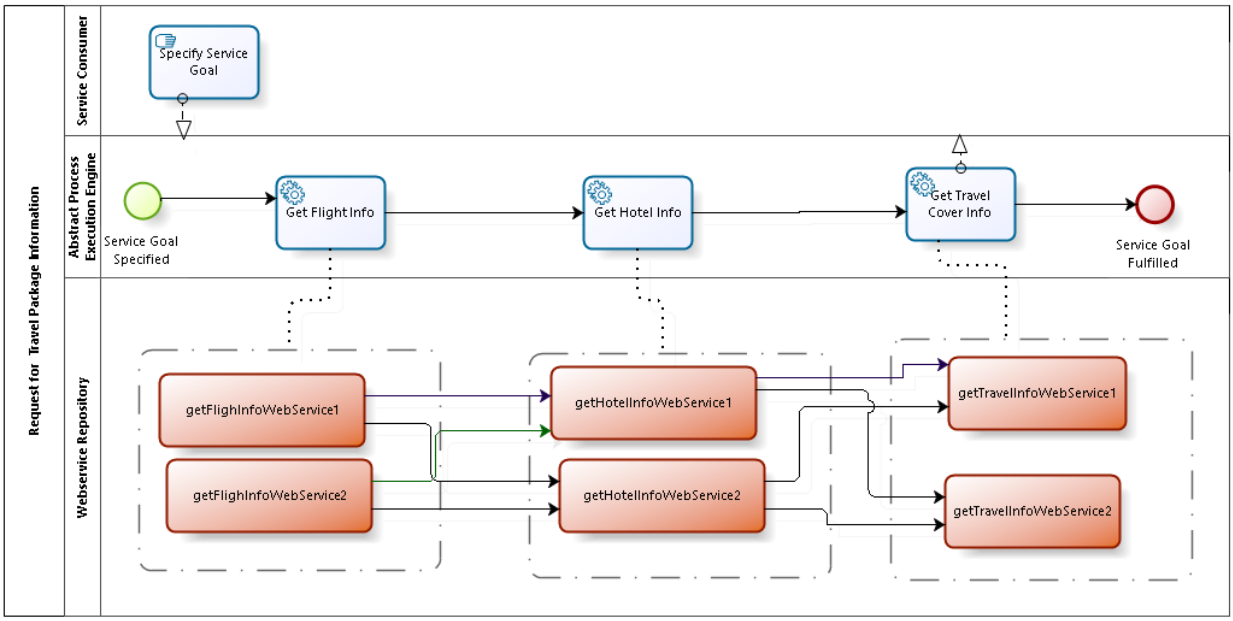


Figure 7 Illustration of Workflow based Dynamic Webservice Composition.

In this diagram, each workflow task is associated with a pool or a community of webservices capable of performing the workflow task. In this example we have chosen n=3 (number of tasks in the business process) and m=2 (number of webservices in each service pool). It turns out that 8 possible sequences of concrete webservices can be selected to execute the workflow. Each sequence constitutes a concrete composite service, composed be joining one service from the first pool to the service in the second pool, then a flow from the service in the second pool to a service in the third pool etc

A generic process framework for the dynamic webservice composition is shown in figure 8.

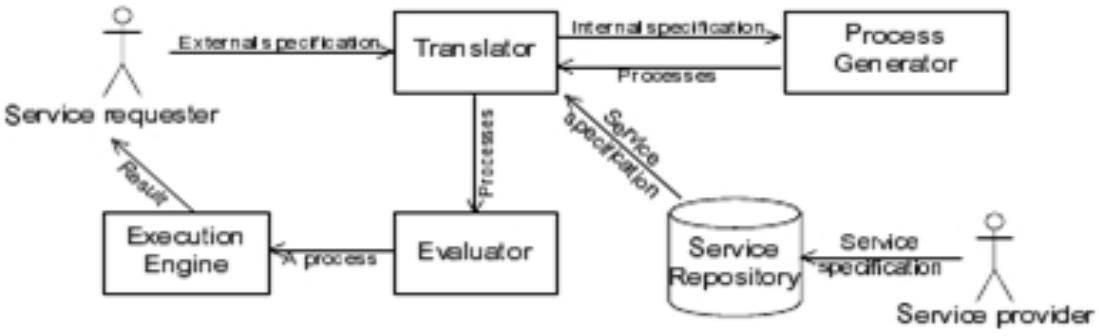


Figure 8: A generic Reference Architecture for Dynamic Webservice Composition: Source (Rao et al, 2005)

From figure 8, the framework consists of actors, components and processes. The service requestor performs the task of *external specification*. The service provide performs *service specification.* The service repository contains a list of published webservices. Then there is a translator, evaluator and execution engine. The role of the service composition system is to accept an *external specification,* translate the specification into an internal specification, generate a process, and search the repositories for the sequence of webservices that meet the external specification, evaluate the best sequence of services and execute the sequence to produce results that meet the external specification. This steps are detailed below.

2.2.2.1 External Goal Specification

During composition, the first step involves the service requestor specifying their requirements. The requirements contain both the functional requirements and preferences. The preferences are further constraints over the outcome the services. Together, the functional requirements and user preferences form a goal. The goal is then linked to an existing business process. The goal is decomposed into requests that can are then associated with a task within a business process. Each business process task defines a unique functionality.

2.2.2.2 ***Service Discovery***

Concrete webservices that match the given task within the business process are searched within the service repository. One or more services may be found that match the functionality. In case of more than one are found, a service selection strategy is needed to pick the best service**.**

2.2.2.3 Service Selection

Through the use of the evaluator, the best composite service is selected based on some selection strategy

2.2.2.4 ***Process Execution***

Each of the webservices within the composite service is bound to the corresponding service task within the process/workflow. The task is then executed by the workflow/process engine causing the transition to the next task within the workflow until the entire process is completed.

2.2.3 Dimensions of Optimization Complexity in Dynamic Webservice Selection

2.2.3.1 High Dimensionality of Webservices Decision Variables and Constraints

Many strata of decision attributes or decision variables and the constraints enforced on these variables have to be considered in order to select the best composite service that fulfills the user's functional and nonfunctional requirements. According to the classification given in (Mahboobeh et al, 2011), the categories of quality (nonfunctional) attributes include technical domain independent attributes such as response time , availability ; non-technical domain independent attributes such as service execution cost and service reputation, domain dependent quality attributes such as refresh time for a traffic monitoring service . In addition to these, there are domain dependent functional attributes. For example in the travel reservation problem involving multiple component services, flight cost is a decision attribute specific to a flight service, sum assured and premium charged applies to a travel insurance component service only and hotel daily rate to a hotel service only. Different users will express different preferences over the functional attributes of a service. The multidimensionality of decisions involved in service composition turns the service composition problem into a Multiple Criteria Decision NP hard problem. The problem becomes more complicated as the number of decision variables and constraints becomes larger as this leads to combinatorial space explosion. Finding an optimal solution in polynomial time becomes harder and computationally time consuming and may be infeasible altogether.

2.2.3.2 The large scale of candidate webservices with similar functionality

In virtual enterprises, several service providers are available that offer services with the same functionality. As mentioned in chapter 1, different providers can be differentiated in terms of the different attributes. Even when only a single attribute is the basis of service selection optimization, that is, the service selection problem is a single attribute decision problem, finding the best composite service from a large set of candidate services are available per task is nontrivial. Generally, for an abstract service with n tasks with m candidate services per task, yields a bipartite graph with m^n different candidate execution paths (Benatallah, 2004). To put this in perspective, a composition problem involving 4 tasks with 10 candidate services per task, results into 10,000 alternative composite services. Adjusting $m=100$, gives 100000000 different composite services. Thus the search state

space exponentially explodes with increase in the size of *m.* Just like in the case of Multiple Criteria Decision Making problem above, determining the best composite service in polynomial time with increasing *m* for a single decision problem in polynomial time remains a challenge.

2.2.3.3 Non Deterministic Nature of Component Webservices.

Component webservices do not operation in isolation. The context, which is the circumstances or the facts surrounding the invocation of a service operation, can significantly lead to multimodal time varying distributions of the software/service operation [cit. Such facts include network link performance, CPU and memory utilization, the service load etc at the time of service invocation. Service invocation time significantly contributes to overall service composition time . How to accurately predict the most efficient services so as to subsequently lead to efficient service composition environments under non deterministic is a challenge.

2.2.3.4 The Scale of Services

The complexity of service selection increases with the number of services involved in the composition process consequently increasing the search space. The issue is further aggravated when considering several decision variables against each webservice. This explodes the space further. For instance consider a 5 task business process and a service repository containing 100 functionally similar webservices per task. If the services per task are ranked against 1 variable only e.g service execution cost, then using a local optimization strategy yields 500 searches + 500 comparisons =1000 computations using the most naïve algorithm . Using global optimization would yield 1000 searches + 100^5 = 10000000000 comparisons!

2.3 Local Planning Optimization Solution to Dynamic Webservice Selection

As pointed out in chapter 1, the optimization scope of dynamic webservice composition can be local or global. Algorithms whose optimization scope is local are commonly referred to as local planning algorithms within the webservices community. In local planning, the selection of the most optimal composite service is performed at task level such the best composite service is the sequence of the best atomic webservice selected from each workflow task (Zeng et al, 2004). On the other hand, composite service selection algorithms that consider

optimization constraints across workflow tasks are called global planning algorithms (Zeng et al, 2004). In general local planning algorithms for webservice composition are faster than their global planning counterparts. On the other hand, the global planning algorithms generate more quality solutions than their local planning counterparts. Because, webservice is multiple criteria decision making problem, both existing local planning and global planning algorithms for the problem make use of the Simple Additive Weighting method, SAW (MCDM) (Hwang & Yoon,, 1981) in computing the utilities of each possible candidate solution.

In this section, we provide the mathematical formulation of the naïve local planning optimization based on SAW. Two mains steps are described: Normalization and weighting.

2.3.1 Webservice Quality Attribute Vectors and Matrices

The input to the optimization problem is a set of *M* by Q matrices. Each matrix is a set of quality attribute values of all candidate webservices capable of executing a given workflow task. For a *k* length workflow, there are *k* such matrices. Thus *M* is the number of candidate webservices for a given workflow task. Q is the number of quality attributes associated with each webservice. Each row in the matrix is a quality vector *V,* against a single candidate web service. Each V_i has Q elements. The jth element of V_i is the QoS value of the jth quality attribute against a given webservice. The table below illustrates this.

Table 2: Local Planning Optimization Service Quality Matrix for a Single Workflow Task

Service	Quality of Service Parameters (Decision Variables) (columns identified by *j)*					
	A	*P*	*T*	*R*	*C*	*D*
W1	0.95	0.99	500	4	10	200
W2	0.9	0.98	550	5	9	100
W3	0.92	0.89	600	2	10	300
W4	0.99	0.97	450	3	8	150

In table 3, there are four candidate webservices that can execute some workflow task. For each webservice, there are six associated webservice QoS attributes denoted by A, P, T, R, C and Where;

A = Average availability of a component webservice

P = probability of success execution/execution success rate

T= Expected response time

C= cost of execution of a component service

R = reputation of a component service

D= standard deviation in response time of a component webservice

The four quality row vectors are :- V_1 = (0.95,0.99, 500, 4,10,200) , V_2 = (0.9,0.98,550,5,9,100), V_3 =(0.92,0.89, 600,2 ,10, 300), V_4= (0.99, 0.97, 450, 3, 8,150).

Thus, V_1 shows that the webservice W_1 has an expected availability of 95%, probability of successful execution 99%, expected response time of 500 ms, execution cost of 10 units, a

reputation of 4 (of 5) and deviation in response time of 200ms. On the other hand, the column vector V_a = (0.95, 0.9, 0.92, 0.99) contains availability quality values of the four services: W1, W2, W3 and W4 respectively, V_p = (0.99, 0.98, 0.89,0.97) holds the reliability quality values of W1, W2, W3 and W4, and so on and so forth.

2.3.2 Normalization/Scaling of Quality of Service Column Vectors

For some webservice quality attributes, increasing values are desirable. Such quality attributes are termed as positive quality attributes (Zeng et al, 2004), (Abiud W.M et al, 2015). For example, reliability, availability, throughput and reputation are *positive quality attributes*. On the other hand, for some quality attributes such as response time, service access cost, decreasing values of the attributes is desirable. Thus, this type of attributes is called *negative quality attributes*.

Since each webservice is associated with a mixture of positive and negative quality attributes, a method of computing an aggregate utility value of each webservice on the set of quality attributes is required. The Simple Additive Weight method mentioned earlier has been widely used. The first step when using SAW, is the normalization phase and the second phase is weighting. In normalization phase, every webservice quality attribute value is normalized such that the resultant normalized value lies on the continuous interval (0,1).

Normalization works as follows. Quality attribute values are normalized column by column, one column at a time. Negative web service quality variable scaled according to equation (2.1) and positive web service quality variable scaled according to equation (2.2)

$$F1 = V_{ij=} \begin{cases} (Q_j^{\max} - Q_{ij})/(Q_j^{max} - Q_j^{min}) & \text{If } Q_j^{max} \neq Q_j^{min}, 1 \text{ otherwise} \end{cases} \quad (2.1)$$

$$F2 = V_{ij=} \begin{cases} Q_{ij-\ (Q_j^{\min})}/(Q_j^{max} - Q_j^{min}) & \text{If } Q_j^{max} \neq Q_j^{min}, 1 \text{ otherwise} \end{cases} \quad (2.2)$$

Referring to the example given in table 3, section 2.4.1, availability, probability of success and reputation are positive quality attributes and hence the vectors A, P and R would be scaled according to (2.2) while response time, execution cost and standard deviation are negative quality attributes and consequently the values in the vector T, C and D would be scaled according to (1).

As an example, the scaling of the vector V_a = (0.95, 0.9, 0.92, 0.99) yields V'_a = (0.56, 0, 0.22, 1) where V'_a is the image of V_a after scaling.

2.3.3 Weighting of Normalized QoS Row Vectors

During this phase, a weight value is assigned to each quality attribute such that the sum of the weights is 1. All existing algorithms require that the service consumer specify the weights (Mulongo et al, 2015; 2016). The weight assigned by a service consumer on a given quality attribute is a measure of their degree of preference for that quality attribute. The larger the weight value the more preferred the quality attribute.

Define the column vector W according to equation (2.3), such that (2.4) holds.

$$\boldsymbol{W} = \begin{bmatrix} \boldsymbol{w1} \\ \boldsymbol{w2} \\ \boldsymbol{w3} \\ \boldsymbol{...} \\ \boldsymbol{wQ} \end{bmatrix} \tag{2.3}$$

$$\sum_{j=1}^{j=Q} w_j = 1 \tag{2.4}$$

Where w_j is the weight assigned to the jth attribute in the quality matrix and Q is the number of quality attributes.

For each task, for each row vector $V_{i,}$ the utility value, U, of a webservice is then computed as per equation (2.5)

$$U_i = \sum_{j=1}^{j=Q} V_j w_j \tag{2.5}$$

2.3.4 Selection of the Best Composite Webservice

In the addition to specifying the preferences, the service consumer also specifies a set of local Qos constraints *C*, such that each constraint is associated with one of the quality attributes in *Q*. In naïve local planning, for each task, the quality vector of each webservice $V_{i,}$ is check against the relevant constraints in C. The webservice with the highest utility and that also satisfies all the constraints in C is selected to execute the ith task in the workflow.

2.3.5 Analytic Runtime Performance Analysis of Naïve Local Planning

There are three main operations involved in the naïve local planning. First, the determination of the utility values of each webservice which takes *nk* operations. Second, for each webservice, checking whether each quality attribute in Q respects the corresponding constraint in C. Thus for one webservice, Q comparisons are required for establishing compliance to the established constraints. Accordingly, for one workflow task with *n* candidate services, we have *nC* operations and thus *nkC* for the entire workflow. Assume the worst case scenario where each and every webservice per task obeys all the constraints in *C*. The third step is to select the service with the highest utility for each task. This takes *nk* operations. In total, the major operations are $2nk + nkC = nk(2 + C) = O(nkc)$. This is polynomial time. Hence as said in chapter 1, local planning guarantees a solution within polynomial time. However, as noted before, local planning composition schemes deny the user an opportunity to express inter task constraints such as the total budget limit on service execution cost. Further local planning optimization is suboptimal. In section 2.5, we discuss how the naïve global planning algorithm overcomes these limitations.

2.4 Global Planning Optimization Solution to Dynamic Webservice Selection

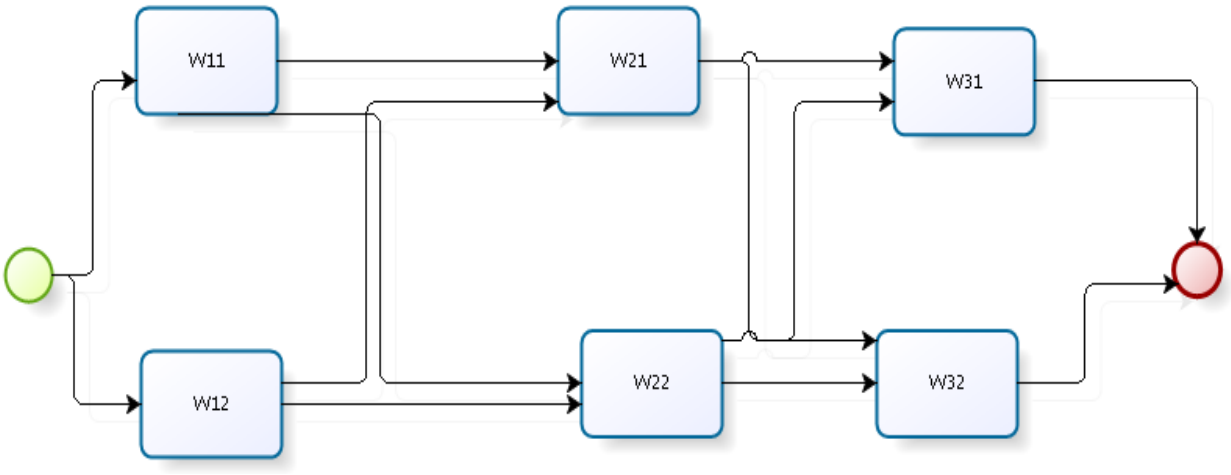


Figure 8 Illustration of Global Planning strategy for webservice Composition using a Bipertite Graph

Consider the webservice Bipertite graph in figure 8 above. The first vertex set V_1 has two webservices W_{11} and W_{12} each capable of executing the first task of a workflow. The second vertex set V_2 contains two candidate webservices W_{21} and W_{22} each capable of executing the second task in a workflow. The third vertex set V_3 contains two webservices W_{31} and W_{32} each capable of

executing the third (and last) task of a three task workflow. The edge or arrow emanating from one webservice in a vertex set V_i to another webservice in a vertex set V_{i+1} is a possible execution path (Zeng et al, 2004). The set of execution paths joining W_{11} ,W_{21} and W_{31} or W_{11} , W_{22} and W_{32} are possible execution plans. In figure 9, there are such 8 execution plans which is the same as n^k where k is the number of workflow tasks and n is the number of candidate webservices per task. Each execution plan constitutes a possible composite webservice.

In global planning optimization, utility maximization is subject to both local constraints and global constraints across the entire workflow. To achieve this, the first step in global planning is to compute aggregate QoS values of each execution path. Like in local planning optimization, what follows is normalization, weighting, and then selection of the best composite webservice subject to constraints set by the service consumer.

2.4.1 Composite Webservice QoS Aggregation Functions

Webservice QoS aggregation functions compute a single joint QoS value across the set of atomic webservices within an execution plan or composite webservice (Zeng et al, 2004). Depending on the nature of webservice QoS attribute, the aggregate QoS value of a composite may be additive, multiplicative, arithmetic mean or the minimum of the QoS attribute values of the individual webservices.

Consider the commonly used webservice QoS attributes as in (Shade et al 2012; Mahboobeh & Joseph, 2011; Rajendran and Balasubramanie, 2009; Zeng. et al 2004) QoS attributes are captured in table 2.4. The symbols associated with atomic service QoS attributes as well as composite service QoS attributes are shown.

Table 3 . The set of webservice QoS Attributes and their Symbols: Source: Mulongo et al (2015)

QoS Name	Absolute Symbol	Atomic Service Symbol	Composite Service Symbol
Reliability	R	r^s	r^c
Availability	A	a^s	a^c
Throughput	H	h^s	h^c
Execution Duration	D	d^s	d^c
Execution Cost	C	c^s	c^c
Reputation	U	u^s	u^c
Security	Z	z^s	z^c

Using the symbols in table 2.4, for pure sequential workflows, the aggregate QoS values r^c, a^c, h^c, d^c, c^c, u^c and z^c are computed as per the aggregation functions in table 2.5.

Table 4: Composite Service QoS Aggregation Functions –: Source: Mulongo et al (2015)

QoS Name	Aggregation Function
Reliability	$r^c = \prod_{I=1}^{i=N} r^s$
Availability	$a^c = \prod_{I=1}^{i=N} a^s$
Throughput	$h^c = 1/N(\sum_{I=1}^{i=N} h^s)$
Execution Duration	$d^c = \sum_{I=1}^{i=N} d^s$
Execution Cost	$c^c = \sum_{I=1}^{i=N} c^s$
Reputation	$u^c = 1/N\left(\sum_{I=N}^{i=N} u^s\right)$
Security	$z^c = min(\sum_{i=1}^{i=N} z^s$

2.4.2 Composite Webservice Quality Attribute Vectors and Matrices

Like is the case with local planning, a *M* by *Q* matrix is generated. The number of rows M is the number of composite webservices that are n^k in number. Thus each row vector V_i is a composite webservice in which the jth value is the jth aggregate QoS value associated with the composite webservice.

2.4.3 Normalization of Composite Webservice QoS Vectors

Normalization of the aggregate QoS values is performed according to equations 2.1 and 2.2 above.

2.4.4 Weighting of Composite Webservice QoS Attribute Values

Equation 2.5 is used to compute the weighted utility value of each composite webservice.

2.4.5 Selection of the Best Composite Webservice

The naïve exhaustive global planning search algorithm evaluates each and every candidate composite service against the constraints using some rule based logic (IF ELSE statements). Although the strategy is bound to find an optimal solution, the search space can be exponential as the number of atomic candidate webservices increase in size (Zeng et al, 2004), (Ardagna & Penci , 2007). The problem is compounded with simultaneous growth in the number of sequential workflow tasks, the number of QoS attributes and number of constraints. At the very least, an effort of n^k is required which can be exponential for large enough *n.*

2.5 Mixed Integer Programming Solution to Dynamic Webservice Selection

Linear Programming (LP) and Mixed Integer Programming (MIP) are the most important optimization techniques to efficiently model and solve real world optimization problems (Berthold Timo et al , 2012), (Ed Klotz, & Alexandra M. Newman, 2012). A MIP problem is defined by a maximization or minimization objective function, a set of integer and non-integer decision variables, a set of constraints.

In the area of webservices composition, Zeng et al (2004) & Ardagna and Pernici (2007) modelled the webservices composition problem as an MIP problem and demonstrated that the MIP is far more efficient than the naïve global planning optimization described in section 2.5. Similarly, local planning technique can also be modelled as an MIP problem, in which case the resultant optimization model can be solved faster than the naïve local planning.

The general procedure for modelling the webservice composition using MIP is to formulate an objective function of the form in equation 2.6.

$$Max \ \sum_{i=1}^{i=n} \ V_j \ \sum_{j=1}^{j=Q} x_i^j \qquad (2.6)$$

Where in equation (2.6) *n* is the number of candidate webservices, and V_j as earlier defined is the QoS vector containing Q quality attribute values. x_i^j is a binary decision variable denoting the selection or no selection of a webservice for a given task and is defined as per equation 2.7.

$$x_i^j = \begin{cases} 1, & S_{ji} \rightarrow T_j \\ 0, & otherwise \end{cases} \qquad (2.7)$$

Where in equation 2.7:-

S_j :- Is the set of candidate webservices against the jth workflow task

S_{ji} is the ith webservice with the set S_j,

T, is the set of workflow tasks

T_j is the jth workflow task

$S_i \rightarrow T_j$ means that S_i is assigned to , T_j

When the MIP is formulated based on a global planning approach, the vector V_j is determined by applying the aggregation functions described in subsection 2.5.1 and the normalization procedure described in 2.5.2. When the MIP is formulated based on local planning strategy, no aggregation is done, and normalization is performed as described earlier.

The next step is formulating the constraints. One constraint to be enforced that only one and one webservice (from the pool of candidate service can be assigned to a workflow task. This constraint is captured in equation 2.8.

$$\sum_{i=1}^{n_j} x_i^j = 1, \quad \forall j \in T \tag{2.8}$$

Let R, A and H be the reliability, availability and throughput thresholds set by the service consumer on every an atomic webservice (in the case local planning) or composite webservice, in the case of global planning. The following constraints are enforced. We have:

$$\prod_{i=1}^{i=N} r \quad \geq R \tag{2.9}$$

Since the constraint in equation 2.9 is nonlinear, we linearize it by taking the logarithms on both the L.H.S and R.H.S of 2.9 to get equation 2.10.

$$\sum_{i=1}^{i=N} \log(r) \geq log\ R \tag{2.10}$$

Similarly, the availability constraint on composite service availability is expressed according to (2.11).

$$\sum_{i=1}^{i=N} \log(a) \geq log\ A \tag{2.11}$$

The constraint on composite service throughput at the SPUM layer is captured in equation 2.12

$$h^c = 1/N(\sum_{i=1}^{i=N} h) \geq H \tag{2.12}$$

Further, let D, C, U and Z be the maximum execution duration, maximum total cost, minimum reputation and minimum security values respectively. The constraints in equations 2.13 to 2.16 hold.

$$\left(\sum_{i=1}^{i=N} d\right) \leq D \tag{2.13}$$

$$\left(\sum_{i=1}^{i=N} c\right) \leq C \tag{2.14}$$

$$\left(\sum_{i=1}^{i=N} u\right) \geq U \tag{2.15}$$

$$\left(\sum_{i=1}^{i=N} z\right) \geq Z \qquad (2.16)$$

2.6 Layering as Optimization Decomposition

2.6.1 Decomposition as an Optimization Method in Engineering and Computer Science

In engineering and Computer Science, decomposition, is breaking a complex problem or complex system into parts or components that are easier to conceive, understand, solve and program (Songqing & Gary, 2009). Based on the goal of decomposition, (Kusiak and Larson 1995) identifies three categories of decomposition: 1) *product decomposition*, 2) *process decomposition* and 3) *problem decomposition.* In product decomposition, a system or product is decomposed into its parts or components (Kusiak and Larson 1995). Such a decomposition promotes understanding of a product or systems structure (Songqing & Gary, 2009). Process decomposition involves problems in which there is flow of *elements* or *information* (Songqing Shan · G. Gary Wang, 2009). Taking the process of webservice composition as an example, it's a multistep process involving conveyance of information or elements from one stage to another. Problem decomposition entails breaking a complex problem into subproblems. In Computer Science, this is commonly referred to as *divide and conquer*. Problem decomposition is at the heart of intensive research on the multidisciplinary design optimization (Kodiyalam and Sobieszczanski-Sobieski 2000; Simpson et al. 2004).

The overall benefits of decomposition, whether product, process or problem decomposition are (Songqing & Gary, 2009):-

i) **Maintainability and Modularity** :-In the case of software systems, product decomposition leads to reduced programming/debugging effort

ii) **Efficiency**:- Problem decomposition means solving computational subproblems is more efficient using techniques such as parallel/concurrent computing and distributed computing

iii) Enable multi-criteria analysis :- Where a single or multiple decision makers are involved

iv) **Improved Coordination** :- improved coordination and communication between the decomposed sub-problems

v) **Enhance reliability and Robustness** of optimization problems.

The goal of this thesis targets the second and third benefits, specifically exploiting the concept of Layering as Optimization Decomposition. It's worth noting that although the natural approach to obtaining efficiency gains from decomposed subproblems in computer science is to use parallel or distributed computing as stated in (ii) above, decomposition can also benefit from efficiency even when the resultant subproblems are solved sequentially. This is possible because performance gains arises from the observation that problem complexity grows more than linearly (Byod et al, 2003), and therefore the growth in complexity of the original problem is generally larger than the sum of the parts.

2.6.2 Layering as Software Architecture Decomposition

From a software systems engineering viewpoint, *layering* is one of the software architecture (structure) decomposition techniques in which a large complex software system into simpler components called layers (Bachmann , 2000). The components are strictly ordered with one layer *A (the lower layer)* providing services to another layer *B* (the upper layer) (Bachmann , 2000) .The immediate advantage of layering is modularity, modifiability and portability (Bachmann , 2000). The second advantage of layering is that it hides technical complexity of the computational details of lower layers, with the topmost layer having the least technical details. Layering also promotes loose coupling of components or functionality. Eventually layering boosts usability of a system. Well known layering techniques in software engineering include the three layer *Model View Controller* (MVC) pattern governing the design of interactive and database intensive applications. Usability for instance emanates from the fact at the topmost layer, users are shielded from the low level mechanics and jargon at the controller and model levels. Thus, the complex computational details are hidden beneath. The second widely known exemplar of layering as an architectural /structural decomposition style comes from the communication networks field – layered networks based on the OSI model. In the OSI model, the lowest level issues such as bit rate control are handled at the lowest layer and are far hidden from the end user. At the application layer, the user can enjoy the benefits of the lower layers without being aware of their existence or understanding the details under them. A fundamental question in layering is *which layer should provide which services and how should the layers be interconnected?*

2.6.3 Layering as Optimization Decomposition

If *optimization* is the primary goal of problem decomposition (as seen in section 2.7.2), layering could be used as a *problem (or optimization) decomposition* strategy. If this is the case, then the third benefit of layering is to achieve efficiency in solving hard optimization problems. *Layering as Optimization Decomposition (*Mung, 2006; *Mung. et al., 2007;* Low, 2013) is an emerging formal theory that attempts to provide architectural as well as quantitative tools for understanding and solving hard optimization problems in layered communication networks. *Layering as Optimization Decomposition* theory perceives the task of solving a complex layered communication network design optimization problem as the solution to multiple well-coordinated sub problems, in which each layer is treated as a subproblem. Viewed this way, in addition to being a product/system decomposition technique, layering is also a *problem decomposition* method. This thesis does not aim at understanding the algorithmic and mathematical details of the framework but rather the conceptual details of Layering as Optimization Decomposition, and how such concepts could inspire the design of a more efficient mixed integer programming global planning model for dynamic webservice composition. Conceptually, layering as optimization is defined by two core ideas (Mung, 2006): The *communication network is viewed as an optimizer* and, the *communication protocols are viewed as* distributed *solutions to some global optimization problem*. The global optimization problem in this case is the Network Utility Maximization (NUM) problem described by (Kelly et al, 1998). The technique works as follows (Mung, 2006), (Low, 2013):- A set of many optimization decision variables are decomposed into subsets of decision variables. Then each network layer being viewed as a subproblem iterates over a subset of the decision variables, pursuing to optimize its local utility. Within each of the layers, the subproblem is formulated a NUM problem. Interfaces between layers are modelled as functions of primal or dual variables. When two optimization problems are such that the there is a common variable, *y* in the objective functions of the two subproblems as in *f(x,y) and f(x,z*), then *y* is a primal or interface or complicating variable (Byod et al , 2003). The problem: *minimize* $f_1(x) = f1(u_1,y_1) + f_2(u_2,y_2)$ *s.t* $y_1=y_2$ could be decomposed into two separate functions that are coupled by the constraint $y_1=y_2$. y_1, y_2 are the Lagrange dual constraints. Each layer below serves the layer above. Together, the individual layers strive towards global optimality.

An important question in Layering as Optimization Decomposition is which layer performs what functions or which layer pursues what objectives (Mung, 2006). This question can further be

rephrased as how layering should be done? Any solution to this question is referred to as a *layering scheme* (Mung, 2006). As will be discussed in section 2.9, this question is at the core of this thesis. The starting point to this question is to recognize the fact that the formulation of the optimization problems is in the form of the NUM problem. The NUM problem involves two main objective functions, depending on who is concerned with the outcomes of the network optimization process (Mung, 2006):- sum of utilities by end users. The end user utilities could be functions of the following variables: - rate, reliability, delay, jitter, power level and so on and so on. The second objective function is the network wide cost function - functions of congestion level, energy efficiency, network life time, joint error estimation etc. A second feature of NUM is that the end users' needs are at the forefront of network design (Mung, 2006). As much as optimization objectives are pursued both from a user's perspective and network operator perspective, the optimization goal from the point of view of the network operator has to indirectly aim at maximizing the end user utility, albeit without the direct knowledge of the user. For example, the benefits of improved information coding and modulation techniques at the physical layer do not only lead to reduced bit error rates (BER) but also enhanced reliable applications. While the improvements at the lower levels benefit the end user, the user does not need to be aware of them. This emphasizes the other advantage of layering which had discussed earlier –abstraction – shielding the end user from technical complexity. Thus NUM can act as a benchmark of possible layering schemes (Mung, 2006). But because NUM does not enforce any predetermine layering scheme or layered network architecture, the choice of a layering scheme is left to human judgment and skill (Mung, 2006). Hence different layering schemes to Layering as Optimization Decomposition may lead to different layered network architectures that in turn may yield varying levels of efficiency (Mung, 2006).

Table 5 : Multi-Layer Optimization Objectives in TCP/IP Layered Network, Source: (Steve Low (2013)

Layer	Optimization Objective	Solution
Application	Minimize response time	Various Application Protocols e.g HTTP,SFTP
Transport	Maximize Utility	TCP
Network	Minimize Path Cost	IP
Link	Reliability, Channel Access,	Various MAC protocols
Physical	Minimize Signal to Noise Ratio, Maximize Capacity etc	Various Physical Layer protocols

Whereas Layering as Optimization Decomposition is well formalized in the networking community and proven to lead to more efficient, optimized, modular and transparent layered networks, a similar formalization in the webservice composition lacks and equally to the best of our knowledge, there exists no method that exploits or is inspired by the inherent advantages of the theory. In this thesis, we argue that although Layering as Optimization Decomposition formalism is rooted in the Network Utility Maximization problem, the complexity of issues involved in the web service selection problem, as discussed in section 1.3 closely resemble the NUM (Kelly et al, 1998) problem and therefore Layering as Optimization Decomposition as used in the networking could inspire a more efficient reformulation of the existing (one layer) mixed integer programming webservice composition optimization strategies.

2.7 Related Work

The problem of *dynamic composite webservice selection* has been studied extensively over the last decade. The strategy taken in (Pan & Mao., 2013) uses a multi-agent model for automatic dynamic webservices composition based on Artificial Intelligence Planning based on the OWLSPlan tool, specifically targeting semantically annoted webservices based on the OWL-S service description

language. Similar to the approach in (Benatallah B. et al, 2004), a vector of quality attributes are captured about a target service, and then the service composition problem is defined by specifying an initial state and goal state using OWL. The plans the agents follow are captured using the PDD language. OWL-S plan is then used to generate all possible execution path where each path is possible solution (composite service) satisfying the service composition goal. Finally the weighting formula used in (Zeng et al, 2004) is also used in (Pan & Mao., 2013) to compute the overall score on each attribute and the service with the maximum score is chosen. A key benefit of composition strategies based on OWLS-Plan is that OWL-S Plan allows online reactive replanning in case of service failure during an ongoing composition process. However, in this method it's not apparent how externally end users express their constraints. In addition this, the limitations related to the work in (Zeng et al, 2004) still remain in (Pan & Mao ,2013) i.e the lack of separation of quality parameters into what could be considered standard quality attributes across different consumers within the same application domain and consequently combinatorial complexity arises. Note also that while in (Pan & Mao ., 2013). only OWL-S webservices are targeted, our proposed method is a generic model independent of a specific service description language – our approach makes the assumption that agents searching over webservices descripted in a specific service language would need to implement a translator component whose function maps to the translation module as defined in (Rao Jinghai and Xiaomeng Su ,2004).

Even though the approach taken in (Mahdi B. et al, 2012) follows a Fuzzy logic approach to service composition involving multiple user defined quality attributes, the deficiencies related to (Benatallah B. et al, 2004) can also be observed in (Bakhshi & Hashemi, 2012) in regard to search space explosion and also in regard to the practicality of the method by which user express their constraints. Regarding the method of expressing constraints, in (Bakhshi & Hashemi, 2012), users express constraints as Fuzzy rules. A weighting approach is also used where a user assigns what the authors call a Confidence Factor (CF) on the range [0, 1] where the CF denotes the importance of each fuzzy rule. Thus rules that are more important from a user's point of view are assigned a higher CF value than the less important ones. We still maintain that in real life applications, not many end users would firstly be able to specify their nonfunctional constraints using Fuzzy rules. Moreover, if they did, it is difficult for end users to assign weights to rules and make sense out of them. Further, since according to the approach suggested in this work, a fuzzy rule is constructed by combining one or more QoS attributes ,e ach assigned a value such as low, high, very high etc

and then assigning a single value (the rank). Let *A* be the number of quality attributes and *L* be the number of discrete values that are assignable to each of the attributes. Considering, the simplest case where a fuzzy rule consists of only one attribute yields *AL rules.* To put this in perspective if A =5 and L =4, this yields 20 rules. Obviously, this would be a taunting task to an average user to assign CF values to 20 rules. As noted, the method suffers scalability due to exhaustive generation of plans which is not efficient for a large number of constraints and services.

The authors in (Alifarai et al, 2010) propose a QoS based optimization on Mixed Integer Programming. Although Zeng et al, (2004) earlier demonstrated that MIP is more efficient than exhaustive search, we saw in section 1 that MIP is still susceptible to exponential explosion. The deficiency in (Alifarai et al, 2010) can be said of the MIP techniques in (Gabrel. et al, 2013), (Yan, 2012) and (Ngoko et al, 2013). Moreover, Yan (2012) does not provide any justification for the use of an MIP method based on Taylor expansion.

A Fuzzy logic based multiple criteria method is presented in (Yan, 2012). The method involves a user expressing preferences by assigning a Confidence Factor (CF) on the range [0, 1] to Fuzzy rules. A CF denotes the importance of each fuzzy rule. The higher the CF value the more important the Fuzzy rule is relative to another rule. Two shortcomings observed on other approaches are inherent in this approach. Firstly, the fact that a user has to express their preferences by specifying confidence Factors, imply that the approach is limited to technical audience only. Lastly, the complexity of the problem exponentially expands with an increase in the rule base.

A decomposition method for service selection based on Mixed Integer Programming is presented in (Singh, 2012). The method works as follows: Global constraints are converted into local constraints. Even though (Singh, 2012) claims reduced MIP model that can be solved in linear time, neither details on how the decomposition method works nor are empirical results supporting the hypothesis provided.

The two step Mixed Integer Programming algorithm by Alrifai et al (Virginie G. et al, 2013) is based on decomposition of global constraints into local constraints. After the decomposition, local optimization using local constraints follows. Firstly, each web service QoS attribute value is partitioned into quantized levels for each web service in each service community. The goal is to find the best combination of quantized values that will be used as upper bound constraints within

the second step. MIP is applied to find the best combination of values that satisfy the constraints. In the second stage, a local planning selection algorithm is used to select the best web services. The challenge with this approach is that expressing global constraints that will not be violated by local constraints is a challenge. Further, the performance of the model is affected by the number of quantity levels *d*. The larger the *d* the less efficient the model becomes and vice versa. The value of *d* or range of *d* for which the model can perform better than conventional MIP remains unknown. Thirdly, like all other techniques, the model is still too complex from an end user perspective and therefore its utility is equally limited to technically sophisticated users.

Liu (2012) proposes a genetic algorithm for composite service selection. The advantage this has over MIP based models is that Genetic Algorithms (GA) are more efficient for ultra large problem sizes. However, GAs require configuration and tuning of extra parameters such as the population size (Ngoko Y., Goldman A, and Milojicic D, 2013).

The main innovation by Ngoko Y., Goldman A, and Milojicic D. (2013) is a MIP global optimization model for workflow based service compositions involving multiple cooperating abstract composite service services. Moreover, the optimization model takes into account service level agreement constraints. Generally the authors empirically show that their MIP global planning model is more desirable than local planning in terms of optimality of solutions. However, only two QoS attributes are considered during optimization so it remains unknown how the model can behave as the number of QoS factor s grow larger.

Similar to (Singh, 2012) and (Gabrel et al , 2013)., our work in (Mulongo et al, 2015), also described in section 2 of this paper, in general follows Mixed Integer programming with a decomposition strategy to optimize composite web service selection. But unlike the rest, the work in (Mulongo et al, 2015), , combines the well-known MIP global planning technique for web service selection in (Zeng et al, 2004) with the emerging theory of Layering as Optimization Decomposition (Mung, 2006; Steve Low, 2013). The distinctive features of this approach are: 1) two distinct objective functions, one addressing the concerns of the end user and the other addressing the concerns of the service provider. Although, in (Abiud W. M. et al, 2015), each layer (sub problem) can viewed as a local optimization problem as within the scope of the entire "network", from a web service composition perspective, each of the local sub problem employs a self-contained global planning method using the S-MIP technique in (Zeng et al, 2004) although

on a subset of constraints and subset of QoS attributes. This brings the second unique feature: 2) that we are still able to express global constraints, and that these global user and service provider constraints alike, are guaranteed not to be violated, 3) By using layering as a design time architectural style, separating the concerns of the users from those of the provider, we not only achieve more efficiency, but also derive simplicity that shields the average end user from the complexity of technical jargon, and tediousness of having to capture constraints and weight preferences on low level QoS attributes. All the other methods lack this feature.

There are dynamic service selection and composition strategies which do not necessarily consider user defined constraints but strive to optimize operational performance quality of service attributes such as availability, response time, throughput etc to ensure efficient service composition. Such strategies confirm our argument that operational performance oriented QoS attributes should not be explicitly specified by an end user of services but rather an implicit given. The methods embedded in ADULA frameworks (Monsincat et al, 2010) fall under this category. ADULA is a self-regulating webservices composition framework. ADULA (Monsincat et al, 2010) uses a simple yet effective randomized probabilistic algorithm that aims to improve system throughput by balancing service requests. The framework does this by distributing requests to those services that recently performed better and are functionally equivalent to those that performed poorly. QoS of services is measured primarily based on service response time thus faster *services* are selected in favour of *slower services.* Slower services are put in quarantine for some time *t.* To ensure fairness based on the on the observation that once slower services may become faster in the near future and vice versa, the framework uses an adaptation method that is based on *change of rate (with respect to response time).* A maximum time delta value is fixed say at 500 ms such that if the difference between a service's current response time and the previous response time exceeds the delta value, then a change in the service's state is reported and its state is updated accordingly either as *slow or fast.* A weakness of this technique is however that the quarantine if the quarantine period is too long, then currently slow services that could otherwise have become faster are starved. The converse is true if the quarantine period is too small. Our proposed QoS model at layer 3 is similar to this approach where a service with the highest performance is selected to participate in the composition. However, one main difference exists between our work and the work in (Adina et al, 2010). Our model jointly considers reliability and response time of a service whereas in (Adina M et al, 2010) only response time is considered. The benefit our model has over the one in

ADULA is that we do not only select a service that is most likely to execute fastest but also the one that has the highest chance of executing successfully.

2.8 A Summary of the Gaps in the State of the Art

From the foregoing discussions it's irrefutable that all the existing dynamic webservice composition algorithms have the following deficiencies:-

i. **End user burden**:- In all the existing multiple criteria utility based approaches (both local and global planning), service consumers are required to specify their preferences by supplying weight values for all the set of available webservice QoS parameters. When the dimension of such variables is large, it not only becomes too tedious for the user but the weight assignment process becomes less objective (Mulongo et al, 2015). For example it's too tempting to ask the end user to specify relative weights on QoS attributes like throughput, reliability and availability etc ,first because any Internet user would always expect that their service request is going to be successfully responded (100% expected reliability), by implication 100% expected availability. Secondly, even if hypothetically, users were willing to trade off reliability or availability for instance, the nuances of these technical QoS terminologies can be too blurry to an end user for them to objectively assign relative weights accordingly (Mulongo et al, 2015). An exception to this is the method described in ADULA (Adina et al, 2010) which focuses on high throughput service composition without involving the end user. The advantage of the model in ADULA (Adina et al, 2010) is that end users are by implication shielded from the technical details of the service composition while enjoying the efficiency of the underlying model. The disadvantage is that the optimization strategy in (Adina et al, 2010) and related models is purely system oriented and ignores the element of preferences specific to a particular service consumer such as cost, reputation etc. There is a need for a framework of service composition that allows the user to obtain near optimal solutions efficiently without having to directly interact with all the quality attributes of the webservices.

ii. **Exponential Explosion**: - All the existing global planning strategies are flat structured. Thus, as the number of candidate webservices grows larger, the algorithms suffer from exponential state space explosion making them severely constrained when it comes to large scale industrial scale service based applications. Zeng et al (2004) recommends that for

large scale service composition problems, composite webservice selection should be done in hierarchical layered manner, applying global planning at each layer. However, to date, to the best of our knowledge, such a framework lacks.

2.9 Proposed Solution: Service Layered Utility Maximization Model (SLUM)

The proposed method is based on the mixed programming global planning model described in section 2.6. However, the deviation from the current practice is our fundamental rethinking about the structure of the dynamic webservice composition problem. Instead of viewing the problem as flat structure as is the case with all the state of the art, we view the problem as "a layered network" with two layers. One of the layers strives to maximize end user utility on a subset of webservice QoS variables, while the other layer attempts to maximize the service provider utility on another subset of webservice QoS attributes. We show that together, the two layers strive towards a global objective, which is to generate within the shortest time possible, the composite webservice that maximally satisfies the utility of the service consumer. The union of the subsets constitutes the entire range of QoS attributes and the range of QoS constraints over this range. At each of the two layers, a mixed integer program is formulated in terms of the subset of QoS variables. The subproblems are then solved sequentially in a layered fashion. Analytically, this formulation is more likely to yield more efficient solutions due to space reduction. Space reduction arises in two ways (see section 2.11), first the sum of the search space at the two layers grows much smaller compared to the original problem as the input size grows larger. Secondly, since the subproblems are solved sequentially in a layered manner, some services at the first layer could be eliminated, further shrinking the overall search space across the two layers. In addition, the design inherently shields end users from specifying their expectations on all the QoS constraints. This architectural rethinking is inspired by the theory of *Layering as Optimization Decomposition* as described in section 2.7. In section 2.10.1, we qualitatively describe how our proposed design maps on the concept of Layering as Optimization Decomposition. In section 2.10.2, we describe the mathematical models underlying the proposed design.

2.9.1 Qualitative Description of the SLUM Model

We cast the service selection problem onto the network utility maximization problem (NUM) based on the formalism of Layering as Optimization Decomposition as follows. First, we view the

composite service composition as “a multi layered network” with each layer trying to achieve some local optimality towards to global optimization objective. In the case of network design, the global optimization problem is formulated as the basic NUM, generalized NUM or the stochastic NUM problem. Then based on the NUM global optimization problem, layered variants of the NUM (Kelly et al, 1998) problem are formulated and solved. In the case of service composition, there is no universally agreed formulation of global “Service *Utility Maximization”* optimization model. However, as stated in chapter 1, the MIP global optimization model by Zeng et al (2004) has been widely adopted in the formulation of MIP solutions to service selection problems. In the place of NUM therefore we have what we dub here as “basic Service Utility Maximization (SUM)” model, referring to the MIP model in (2004). Then, we adapt the basic SUM model to fit the proposed layered architecture leading to SLUM for Service Layered Utility Maximization Model. SLUM is quantitatively described in section 2.11.2.

Secondly, we have to identify the “layers” in our “network”. Unfortunately, unlike in network design where there are well established network models such as OSI and TCP/IP, no network model or layered network formulation of the service selection problem exists today. Luckily we can draw some analogies from the NUM problem. Based on existing work on QoS aware webservice selection, we work backwards to identify a minimum number of “layers” in the network. Here goes the analogy. The generalized NUM problem puts the end user at the forefront leading to two types of optimization objective functions (Mung, 2006). 1) maximizing end users sum of utility functions over variables like rate, reliability, delay , jitter and 2) a network wide cost function determined by the network operator that can be functions of congestion, power efficiency etc. Putting the service consumer at the forefront, we can see at least two similar objective functions naturally arising in webservice composition problem. The following objectives can be identified: The first objective is that the service consumer would like to get access to the composite service at the minimum possible cost within the shortest possible time. Therefore from a consumer perspective minimization of *financial burden (which* includes minimizing actual cost of accessing the service and minimizing the financial risk) and *minimization of service response time are key concerns.* Financial risk is associated with QoS factors like reputation and security .Thus from this perspective, we have that the end user objective function is a utility function over the following webservice QoS attributes: service execution cost, reputation, security and response time. On the other hand, the most important performance parameter from a business perspective is throughput

–how many customers can be served in unit time. By implication, this extends to response time, reliability, availability etc. Therefore in the global virtual organization case, the virtual enterprise broker key objective is maximizing webservice total utility over throughput and other performance factors that affect throughput including response time, reliability, and availability. From these two objectives, we work backwards to formulate the two "layers" (subproblems) of our "network": SCUM and SPUM. The objectives of these two layers were introduced in section 1 and here we summarize them in Table 6.

Table 6 : Multi-layer Webservice Optimization Objectives in the Proposed Model: *SLUM, Source: Mulongo et al (2015)*

Layer	Optimization Objective	Solution
Service Consumer Utility Maximization (SCUM)	Maximize the utility function over composite webservice execution cost, reputation, security and response time	SCUM MIP model
Service Provider Utility Maximization (SPUM)	Maximize the utility function over response time, service execution success, throughput ,availability	SPUM MIP model

Third, we need to establish which of the two layers "serves" the other. This is the same as asking the question: should the optimization process start at SPUM layer then SCUM layer (bottom up service selection optimization) or from SCUM then SPUM (top-down service selection optimization), does it matter which way? Starting with the last question, the answer is yes, the flow of information during the optimization process using the layered approach matters. Assume a top down approach is chosen. There is a possibility of selecting services with the lowest costs, lowest financial risk and lowest response time at the SCUM layer but that have the worst reliability, reliability and or throughput when evaluated at the SPUM layer. There are two possibilities. First, if none of the composites meets the constraints at SPUM layer, then no solution is found. Second, a subset or all the webservices may meet the threshold constraints on reliability and availability

but only marginally. The result is that such webservices will have a higher probability of failure during execution whereas potentially more reliable but more costly and less efficient services were "prematurely" dropped at SPUM (Mulongo et al, 2015). Conversely, if a bottom up optimization approach is followed, there is a possibility that composite services with the highest throughput, availability, reliability are chosen but may fail to meet the test at SCUM layer i.e either they do not meet cost, financial risk or response time constraints. If they did meet only marginally, the execution of the composite service may result in one of the following. Non responsiveness (web service takes too long to respond), a higher cost burdens to user, or potential loss of cash due to less trusted services.

The point is that *bottom up* and *top down* optimization approaches, each constitute a possible *layering scheme* such that each layering scheme may yield different values to global optimization objective resulting into different optimality values. So whether to follow the bottom up or top down optimization layering scheme is a problem itself. This thesis does not address the problem of which of the two schemes is better in efficiency or optimality. Instead, this the study took a top down up approach –solve the SCUM problem first then afterwards the SPUM problem. The reason is that end users objectives remain at the core. i.e reducing financial burden, financial risk and reducing the time taken to access a service. The worry that less costly and more efficient but less reliable and low throughput services that are more likely to fail during execution can resolved by making the following observations. First, webservice reliability is a function of availability among factors. A service that often fails during execution due to unavailability is less reliable. Fortunately, availability is a QoS factor that can be captured as part of the service level agreements (SLAs) between the virtual enterprise broker and the various virtual enterprises within the global virtual firm. The SLAs will ensure that variability in service availability across virtual enterprises is within acceptable bounds, in case the virtual enterprises were to remain within the global virtual market. Secondly, webservices that are less responsive (large response times) have double negative impact. One is that potential timeouts definitely ruin the reliability of the service- failure to execute successfully. Second, the delay negatively impacts the overall system throughput. However, the top down optimization approach automatically mitigates these drawbacks – since the utility function accepts response time as part of the inputs and its output value is also restricted by the constraints on response time, it means that resultant services are not only of low financial burden but of high efficiency thus leading to overall increased throughput and reliability of the

composition system. Even more, our proposed MIP optimization algorithm at the SCUM layer attempts to find all feasible solutions that are then promoted to the SPUM layer. This is done so as to avoid early elimination of otherwise candidate webservices with higher reliability, availability and throughput values. Thus, in our proposed layered scheme, the *SCUM layer serves the SPUM layer.*

Fourth, we need to identify primal or Langrage dual variables between the SCUM and SPUM layers, if at all there are. We observe that response time is a "coupling or primal or interfacing variable" connecting SCUM and SPUM layer. We can eliminate the primal variable by maintaining this variable at only one of the two layers. Since optimization is done top down, we have that this variable is maintained at the SCUM layer only. There are two main reasons motivating this decision. The first reason is to maintain the validity of our choice of the top down optimization approach as explained in the preceding paragraph. The other and perhaps the most important reason is premised on the empirical evidence the response time for distributed software components exhibits time varying multimodal statistical distributions. We make allusions to (Kounev S. , Gorton I & Sachas K, 2008). By implication, the distribution of response time of webservices is a stochastic process. Therefore, to increase chances of more efficient services being promoted from the SCUM layer to SPUM layer, response time must be one of the decision variables to be considered in the first cycle of optimization, which happens by choice to be at the SCUM layer. In the end, SCUM and SPUM layers are decoupled in decision variables but coupled by data dependencies. Data dependency arises from the fact that the SPUM layer has to wait for the outputs from the SCUM layer, which are then used as the input solution space of SPUM.

Fifth, we model the flow of data or information from one network layer to the next layer as the flow of the webservice composition Bipertite graph. The original graph contains all candidate webservices. As the graph flows through Layer 2 and Layer 1, some services are eliminated.

Table 7: Mapping the Concepts in Layering as Optimization Decomposition to the Proposed SLUM model

Concept in Layering as Optimization Decomposition	Concept Realization in SLUM Model
Layers: Layers in the OSI Model	Two layers :- SCUM and SPUM
Relation between Layers	SCUM layer serves SPUM layer i.e the output of SCUM layer optimization is the input solution space to the SPUM subproblem
Decision Variable Subsets	• Integer and real Variables at SCUM layer related to cost, reputation, and security and execution duration. • Integer and real variables at SPUM are in terms of reliability, availability, throughput and related low level performance parameters. • In addition, at layer, Boolean decision variables are defined at each layer.
Dual/Langrage/Interface Variables	None
Objective Functions :-	At SCUM: Maximize the utility of the service consumer where the utility is function of the subset of variables mentioned above At SPUM : Maximize the utility of the service provider
NUM : Network Utility Maximization Model ((Kelly , et al, 1998)	The Mixed Integer Programming global planning model herein S-MIP described in Zeng et al (2004)

Layering Scheme	Topdown – SCUM then SPUM optimization (as opposed to the other way round)

2.9.2 Mathematical Formulation of the SLUM Model

2.9.2.1 Introduction

We formally restate the composite service selection problem as follows:

Given the tuple,$\langle R, F, G \rangle$

Find: $P^b \in G$ that can execute F *to* satisfy *R*

Where;

i. *R* is the complex service request such that $R = \langle r_1, r_2, \ldots, r_n \rangle$ where r_k is an atomic service request within *R.*

ii. *F* is a sequential abstract workflow such that $F = \langle t_1, t_2 .., t_n \rangle$ where t_k is a workflow task within *F* such that the execution of t_k leads to the fulfillment of $r_{k.}$ The tasks are sequentially ordered as $t1 \rightarrow t_2 \rightarrow_{,..,} \rightarrow t_n$,

iii. *G* is the webservice composition Bipertite graph such that *G* is the N-tuple $\langle V_1, V_2, \ldots, V_n \rangle$ where V_k is a vertex set containing a list of functionally similar concrete webservices that can execute the task t_k . Therefore V_k is the data structure $List<W_{kj}>$ where W_{kj} is the j^{th} service in $V_{k..}$ Each W_{kj} can be defined by the tuple $\langle I, O, Q \rangle$ where *I* is the set of input parameters, O is the set of output parameters and Q is the set of QoS values associated with W_{kj}. Any complete path constituted by a service drawn from V_1, and another service from V_2 , ..., and finally another service from V_n consistutes a candidate solution. If *m* is the number of services in every $V_{k..}$ then as shown earlier in section 1, there exists m^n such candidate solutions or candidate composite services. Therefore we need to find $P^{b,}$ the best path (composite service) that satisfies *R.*

This thesis is about solving for P^b. In section 3.2.2 we elaborate how our SLUM model finds P^b using the Service Layered Utility Maximization (SLUM) algorithm.

The two layered mixed integer solution, SLUM, to the problem above, involves the following steps. Decomposing the set of webservice attributes into subsets such that one subset is assigned to the SPUM layer and the other is assigned to the SCUM layer. Secondly, formulating the SPUM optimization subproblem by specifying its objective function and constraints over the decision variables at this layer. Thirdly, formulating the SCUM subproblem by specifying its objective function and constraints in terms of its subset of decision variables. Lastly, solving the two subproblems in a sequential (layered) manner, beginning with the SCUM subproblem then winding in SPUM subproblem.

2.9.2.2 Webservice Quality of Service Decomposition

The original set of webservice QoS attributes, Q is divided initially into two disjoint partially layered sets of QoS attributes, Q_1 and Q_2, such that Q_1 is in SPUM layer and Q_2 is assigned to SCUM layer. Q_2 contains all webservice QoS parameters related to the financial burden and financial risk to be borne by the service consumer and one performance QoS parameter – response time. Q_1 contains the set of all performance parameters except response time. In practical SOA applications, the quality of service attributes decomposition step should be performed by the Virtual Enterprise Broker.

1.2.1.3 The SPUM layer Subproblem

2.9.2.3.1 SPUM Weight Assignment to QoS Parameters

As a first step, the Virtual Enterprise Broker should define a weight vector W_1 in which the *i*th element corresponds to a weight assigned to the *ith* QoS element in Q_1 such that $\sum_{j=1}^{j=n} W_1^j = 1$. A weight value assigned to a Qos parameter in Q_1 indicates the relative priority of that QoS attribute from a service providers point of view. Suppose W_1 = [0.5, 0.2, and 0.3] for reliability, availability and throughput respectively, then it means that the service provider is concerned about service reliability more than any other QoS attribute. From the same example, the virtual enterprise broker prefers services with a higher throughput than service which may have a higher availability with

smaller throughput values. The weights can be adjusted as service performance statistics evolve over time.

2.9.2.2.1 SPUM Layer Objective Function Definition

At SPUM layer , the objective function of the SUM problem, F_1 is to maximize the utility function U_1 over the set Q1, given the initial webservice graph G, the weight vector W_1, the set decision variables X_1 subject to a set of constraints C^1. X_1 contains the set of decision variables at Layer 1, while C is the set of constraints on X_1. The objective is captured according (2.17) and refined according to (2.18).

$$\boldsymbol{F_1 = maximize[\![U_1(M_1^c, W_1)\ ]\!]} \quad (2.17)$$

The objective function F_1 in (1) is translated as: maximize the value of the utility function U_1 which takes as input, the QoS matrix M_1^c and the weight vector W_1. M_1^c is the matrix containing normalized aggregate QoS values for each candidate composite service (plan) on every QoS attribute in Q1. i.e by adopting a notation similar the one used in (Benatallah, 2004), the rows represent a candidate execution plan and the columns represent the jth QoS attribute and M_1^{cij} is the raw aggregate j^{th} QoS value of the i^{th} execution plan . To compute M_k^{cij}, the aggregation functions given in Table IV are used accordingly.

Note that some QoS parameters can be positive while others negative. The QoS of positive parameters increase with increasing values of the parameter. The QoS of a negative parameter decline with increasing value of the attribute. For example in Table 2.4, execution duration and execution cost are both negative QoS attributes and the rest are positive parameters. For this reason, the matrix M_1^c needs to be normalized. If M_k^{cij} is a positive parameter, we denote the normalized image of M_k^{cij} by M_k^{cij+} or M_k^{cij-} otherwise. M_k^{cij+} and M_k^{cij-} are computed according to the scaling functions given in (2.18) and (2.19) respectively.

$$\boldsymbol{M_k^{cij+} = [\![M_k^{cij} - M_k^{cjmin}]\!] / [\![M_k^{cjmax} - M_k^{cjmin}]\!]} \quad \textbf{(2.18)}$$

$$\boldsymbol{M_k^{cij-} = [\![M_k^{cjmax} - M_k^{cij}]\!] / [\![M_k^{cjmax} - M_k^{cjmin}]\!]} \quad (2.19)$$

In both (2.18) and (2.19) :

i. If $M_k^{cjmax} - M_k^{cjmax} = 0$,1 is returned.

ii. M_k^{cjmax} is the maximum value in the jth column

iii. M_k^{cjmin} is the minimum value in the jth column

iv. k, as usual is the SCUM optimization layer or SPUM optimization layer

We will denote the resultant matrix after scaling the matrix M_k^c by $M_k^{c\prime}$.Thus the optimization objective function at SPUM is revised to (2.20).

$$\boldsymbol{F_1 = maximize[\![U_1(M_1^{c\prime}, W_1)\ ]\!]} \tag{2.20}$$

By applying the Simple Additive Weighting, SAW (Hwang & Yoon,, 1981) to (2.20) as our utility function, equation (2.21) holds.

$$\boldsymbol{F_1 = maximize[\![M_1^{c\prime} * W_1]\!]} \tag{2.21}$$

Equation 2.21 can be expanded to (2.22). (2.22) holds because in our case all SPUM layer QoS variables are positive.

$$\boldsymbol{F_1 = maximize\left[\!\left[\sum_{j=1}^{j=3}[\ M_1^{cij+} * W_1^j]\right]\!\right]} \tag{2.22}$$

2.9.2.2.2 Definition of SPUM Optimization Constraints

Let *R, A* and *H* be the reliability, availability and throughput thresholds set by the virtual enterprise broker on every execution plan. We use the notation C^{ki} to denote the i^{th} constraint at the k^{th} layer. When *k=1*, the following constraints are enforced. We have:

$$\boldsymbol{C^{11}}:\ \boldsymbol{r^c \geq R}\ \ \textbf{or}\ \boldsymbol{\prod_{I=1}^{i=N} r^s\ \ \geq R} \tag{2.23}$$

SinceC^{11}: is nonlinear, we linearize it by taking the logarithms on both the L.H.S and R.H.S of (2.23) to get (2.24).

$$\boldsymbol{C^{11}}:\ \boldsymbol{\log r^c = \sum_{i=1}^{i=N}\ \log(r^s) \geq log\ R} \tag{2.24}$$

C^{11}, as represented in (2.24) is the constraint on composite service reliability.

Similar to C^{11}, C^{12}, the availability constraint on composite service availability is expressed according to (2.25).

$$\boldsymbol{C^{12}: \log a^c = \sum_{i=1}^{i=N} \log(a^s) \geq log\ A} \quad (2.25)$$

The constraint on composite service throughput at the SPUM layer is captured in (2.26).

$$\boldsymbol{C^{13}: h^c = 1/N(\sum_{i=1}^{i=N} h^s) \geq H} \quad (2.26)$$

We need a binary variable to indicate whether or not a webservice WS_{ji} is selected from the vertex set $V_i \in G$ to execute a workflow task, t_i. Conventionally this variable is represented as y_{ij} . In this work, we will represent this variable as y_k^{ij} to reflect our layered architecture, where *k* is the layer number. At *k=1*, constraints C^{14} and C^{15} hold. C^{14} indicates that a service can assume y_1^{ij} value of 1 or a y_1^{ij} value of zero. In (2.28), C^{15} dictates that only one service can be selected from each vertex set V_i to execute a task t_i in the set F of workflow tasks.

$$\boldsymbol{C^{14}: 0 \leq y_1^{ij} \leq 1} \quad (2.27)$$

$$\boldsymbol{C^{15}: \sum y_1^{ij} = 1, i \in V_i, \forall i \in F} \quad (2.28)$$

In addition to the above constraints, at layer 2, we introduce the binary variable l_2^{ij}. l_2^{ij} indicates whether or not the service WS^{ij} was selected during layer 2 SCUM optimization process. We enforce the constraint in (2.29) to imply that only services previously selected during layer 2 optimization should be selected.

$$\boldsymbol{C^{16}: 0 \leq e_2^{ij} = 1} \quad (2.29)$$

Thus the set of optimization constraints C^1 at layer 1 contains $C^{11}, C^{12}, C^{13}, C^{14}, C^{15}, C^{16}$:

2.9.2.3 The SCUM Layer Subproblem

2.9.2.3.1 Weight Assignment to QoS Parameters

As a first step, the service consumer should define a weight vector W_2 in which the *i*th element corresponds to a weight assigned to the *ith* QoS element in Q_2 such that $\sum_{j=1}^{j=n} W_2^j = \mathbf{1}$. A weight value assigned to a Qos parameter in Q_2 indicates the relative priority of that QoS attribute from a service consumer point of view. Suppose W_2 = [0.1, 0.4, 0.3, and 0.2] for execution duration, execution Cost, reputation and security respectively, then it means that the service consumer cares about cost more than any other QoS attribute. Recall that this differs from the state of the art where the end user is always assumed to be responsible for specifying weight preferences over all QoS attributes. With our approach, the end user can benefit from the optimization of parameters such as throughput, reliability and availability without necessarily being aware of the optimization process surrounding these parameters, just in the same way in the NUM problem, the end user can benefit from improved physical layer forward error correcting codes while such details are abstracted from them. After all, all service consumers always expect that whenever they access a service it's available and that it will execute successfully all the time. Consequently with our methodology, end users have fewer QoS attributes over which to specify weights.

2.9.2.3.2 SCUM Layer Objective Function Definition

At layer 2, the objective function of the SCUM problem, F_2 is to maximize the utility function U_2 over the set Q2, given the webservice graph G^1, the set decision variables X_2 subject to a set of constraints C^2. X_2 contains the set of decision variables at Layer 2 and C_2 is the set of constraints on X_2. $G^1 \sqsubseteq G$ i.e G^1 is the set of feasible solutions from Layer 1 or the set of candidate solutions at Layer 2. G^1 may contain all or just a subset of paths from the original graph, *G*. This objective function is stated according to (2.30).

$$\boldsymbol{F_2 = maximize[\![U_2(M_2^c, W_2)]\!]} \qquad 2.30)$$

By applying (2.17) and (2.18) and using the conventions adopted in this paper, (2.30) transforms to (2.31). The objective function in (2.31) holds since at SCUM layer duration and cost are negative parameters while reputation and security are positive parameters.

$$\boldsymbol{F_2 = maximize}\left[\left[\sum_{j=1}^{j=2}[\ \boldsymbol{M_2^{cij-}} * \boldsymbol{W_2^j}] + \sum_{j=3}^{j=4}[\ \boldsymbol{M_2^{cij+}} * \boldsymbol{W_2^j}]\right]\right] \quad (2.31)$$

2.9.2.3.3 SCUM Layer Optimization Constraints

Let *D, C, U* and *Z* be the extreme values set by the service consumer on composite service execution response time, execution cost, reputation and security in that order. Here we define the constraints on composite service execution duration, execution cost, reputation and security in (2.32), (2.33), (2.34) and (2.35) respectively.

$$\boldsymbol{C^{21}: d^c = \left(\sum_{i=1}^{i=N}\ d^s\right) \leq D} \quad (2.32)$$

$$\boldsymbol{C^{22}: c^c = \left(\sum_{i=1}^{i=N}\ c^s\right) \leq C} \quad (2.33)$$

$$\boldsymbol{C^{23}: u^c = \left(\sum_{i=1}^{i=N}\ u^s\right) \geq U} \quad (2.34)$$

$$\boldsymbol{C^{24}: z^c = \left(\sum_{i=1}^{i=N}\ z^s\right) \geq Z} \quad (2.35)$$

In (2.32), (2.33, (2.34) and (2.35) the service consumer expects the best composite service:-

i. Not to take more than *D* seconds before the consumer gets the final results to their service request as conveyed by C^{21}.
ii. To cost them not more than *C* units of money to access the business service provided by the technical composite service as captured by C^{22}.
iii. To have an average reputation of at least U on the interval [1, 5].
iv. To have a security rating of not less *Z* on the average. The security associated with accessing the business service in this case is the average of the each service provided by each virtual enterprise.

Just like with SPUM Layer , constraint on the allocation constraint y_{ij} are defined. Adopting our notation, we have (2.36) and (2.37) with the usual meanings.

$$\boldsymbol{C^{16}: 0 \leq y_2^{ij} \leq 1} \quad (2.36)$$

$$\boldsymbol{C^{17}: \sum y_2^{ij} = 1, i \in V_i\ , \forall i \in F} \quad (2.37)$$

2.9.2.4 SLUM Optimization Solution Process

At SCUM layer, all feasible solutions are determined i.e all combination of services that can fulfill the objective function F_2 subject to the constraints set C^2 are returned in a solution pool. The reason for obtaining all feasible solutions as opposed to the optimal solution is so as to prevent possibility of prematurely dropping a webservice which would have otherwise scored better than a majority of the selected services.

We define a Webservice to Task Assignment Matrix (STAM). At SCUM layer , we will denote this matrix by $L_{1.}$As an example, consider a two task workflow. Suppose initially before selection there were 3 candidate services per task.

Before SCUM layer evaluation, this matrix is represented in tabular form as in table 8 and table 9 and after SCUM layer Optimization the matrix L_1 is represented as shown in table 8 below.

Table 8: An Example Webservice to Task Assignment Matrix before SCUM Optimization, Source: Mulongo et al (2015)

Workflow Task, *i*	**Candidate Web service, *j***		
	1	**2**	**3**
1	0	0	0
2	*0*	0	0

Table 9: An example Webservice to Task Assignment Matrix after SCUM Optimization, Source (Mulongo et al, 2015)

Workflow Task, *i*	**Web service, *j***		
	1	**2**	**3**
1	1	1	0
2	1	0	1

During optimization at SCUM layer, for each service s_{ij} that is selected and assigned to a task *i* , *yij* is updated to *1*.Suppose the resultant webservice to task assignment matrix after SCUM optimization is as shown in table 9. The Webservice to Task Assignment matrix, L1 in table 9 indicates that:-

i. That services $S_{11,}$ S_{12} were selected for *task 1* while service S_{13} was not selected for task 1 after SCUM layer optimization.
ii. Service S_{21} and S_{23} were selected for *task 2* while service S_{22} was eliminated.
iii. Out of the 9 candidate solutions, only 4 feasible solutions were found. In this case only the paths $< S_{11,}, S_{21} >$, $< S_{11,} S_{23} >$, $< S_{12,} S_{21} >$ and $< S_{12,} S_{23} >$ will be evaluated for performance at the SPUM layer.

Thus during SPUM optimization process , the e_2^{ij} values of $S_{11,}$, $S_{12,}$ $S_{21,}$ S_{23} will be 1 and only these services will be evaluated at the SPUM layer.

Having selected webservices whose combination maximizes the utility of user preferences on service execution cost, reputation etc and that meet the constraints defined on cost, reputation , at SPUM layer, the goal is to select the service combination that maximizes utility on performance related QoS subject to constraints defined on the performance QoS variables. The output of the SPUM optimization process is therefore a set of service combinations that fulfill requirements of both SCUM and SPUM layers. The solution at SPUM layer therefore consistutes P_b .

2.10 A Summary of How our Proposed Model Differs from the State of the Art

Our proposed architecture (SLUM), unlike all the existing approaches, employs a hierarchical layered approach to the dynamic webservice composition problem. *Our approach is inspired by the theory of Layering as Optimization Decomposition*. There are two layers, one attempting to maximize service consumer utility, while the other layer strives to maximize the service provider utility. The original set of quality attributes is split into two disjoint subsets. One subset of QoS attributes is assigned to one of the layers while the other is assigned to the other. At each layer, a global planning mixed integer program is formulated in terms of their corresponding subsets of QoS attributes. The two subproblems are then solved sequentially so that the output of one of the layers becomes the solution space to the remaining layer. Together, both layers attempt to solve a global optimization problem- which is "*Efficiently dynamically select the best composite service from a pool of alternative webservices that are differentiated on a wide range of QoS attributes".*

Our proposed design and model improves on the existing approaches by filling the two gaps in section 2.9 through the relative strengths outlined in section 2.10.1. On the other hand, the relative limitations of our design are outlined in section

2.10.1 Relative Strengths

2.10.1.1 Reduced burden on the end user

In our strategy SLUM, service consumers specify weight preferences and QoS constraints on a smaller set of QoS variables as opposed to the entire range of QoS attributes. Thus, this is less laborious compared to the state of the art.

2.10.1.2 Reduced Problem Complexity Due to Space Reduction

As mentioned, exponential explosion in the search space with respect to number of webservices *n*, means that the running time of MIP strategies and other global planning strategies is non deterministic polynomial. Our proposed model combats the problem of space explosion inherently as follows:-

i. Space Reduction through Decomposition (without Service Elimination)

Because of the two layer decomposition of webservice QoS attributes, each webservice QoS matrix at each of the layers is nearly half the original webservice QoS matrix. We saw that for global planning, given a fixed workflow of size *k*, the computational effort is proportional to n^k. But with the layered approach, the size of computational effort is reduced to $2((½)^k n^k)$. It's intuitive that as $n \rightarrow \infty$, $2((½)^k n^k) << n^k$. This supports the theory that decomposition, even when done sequentially, improved efficiency results from the fact that growth in the complexity of a computational problem as a function of the input size is more than linear. In chapter 3, we investigate and analyze in details the runtime performance of SLUM relative to state of the art on problem instances of varying sizes in *n.*

ii. Space Reduction through Decomposition with Webservice Elimination

The efficiency gains in (i) above are not due to service elimination but due to superlinear growth of computational problem complexity. However, in real scenarios, some webservices are likely to be eliminated as described in section 2.10. This means that the overall computational effort while using SLUM may be smaller than $\mathbf{2((½)^k n^k)}$. Now, at the SCUM layer, the initial search space would be n^k composite webservices while at the SPUM layer, the search space given that some services eliminated at SCUM layer is generally less than n^k. This is results into further gains in efficiency of our strategy compared to the state of the art. Let g_1 be the search space at SCUM layer. The computational effort at SCUM layer is proportional to n^k. Let g_2 be the search space at the SPUM problem. Generally, $g_2 \leq g_1$. When $g_2 = g_1$, it means no webservice was eliminated at the end of SCUM and when $g_2 < g_1$, it means some webservices were eliminated during SCUM optimization process. In this study we will define *Composite Service Phase Transition Rate (CSPTR),* denoted by ρ, as the ratio g_2 / g_1. In chapter 3, this study also investigates the relative runtime performance of SLUM under different ρ values.

2.10.2 Relative Limitations

Although the MIP formulation at each of the layers is global in scope and therefore SLUM yields globally optimal solutions within the scope of the layer, the optimization scope at each layer does not take into account the constraints at the other layer (Mulongo et al , 2015; 2016). As explained in section 2.10 and in (Mulongo et al, 2015; 2016a), due to this limitation , depending on the structure of the problem instance, in some cases, like local planning algorithms, the overall solution quality of SLUM could be suboptimal when compared to flat structured (single layer) global planning algorithms. However, note that unlike local planning algorithms, SLUM is able to capture global constraints at each layer. Moreover, analytically, we envisage that although SLUM might yield suboptimal solutions relative to state of the art single layer global planning MIP algorithms, the solution quality of SLUM is on average better than that of local planning algorithms. The theoretical explanation for this is that under local planning, optimization process considers only n options (at a task level only) and thus ignores the other $n^k - n$ composite service options (Mulongo et al, 2016). On the other hand, SLUM at each layer considers $(n/2)^k$ *maximum* options against a maximum possible $(n)^k$ and thus at the very least ignores $(n)^k - (n/2)^k$ composite service options (Mulongo et al, 2016). Since, in general $(n)^k - (n/2)^k < n^k - n$, $n>4$ at a fixed k, then local planning algorithms are more likely to yield less optimal solutions on average than SLUM (Mulongo et al , 2016). Thus, the study also investigated the solution quality of SLUM against the mixed integer programming local planning approach to establish their performance differences in terms of optimality.

In general, the optimality limitation of our solution is not a major issue especially if turns out to be near optimal. This is so because, for problems of industrial relevance where optimal solutions cannot be found in acceptable amount of time, suboptimal but more efficient approaches are often sought (Tonci et al, 2009).

2.11 Benchmark Algorithms

Based on the research objectives as stated in section 1.4, the baseline algorithm in this study is the single layer Mixed Integer Programming global planning model described in section 2.6. This algorithm is dubbed *S-MIP*. By comparing our proposed two layer MIP solution SLUM, we benchmark the efficiency as well as the solution quality of our solution. Since the solution quality

of S-MIP is conjectured to dominate the one for SLUM for all problem instances, the main motivation for comparing SLUM against S-MIP on the solution quality parameter, is to determine how close the solution quality of SLUM is on average to the global optimum. The alternative algorithm chosen for our benchmark is the local planning algorithm described in 2.4 whose formulation is based on mixed integer programming too. This algorithm is hereafter abbreviated *L-MIP*. We already have an idea that the runtime efficiency of L-MIP like any other exact local planning algorithms is theoretically proportional to $nk(2 + C)$ (see section 2.4.5). Analytically this is super faster than any conceivable global planning algorithm including SLUM. Hence, the main reason for benchmarking SLUM against L-MIP is not to compare their relative runtime performance efficiency but to establish their relative performance in terms solution quality, given that as explained in section 2.11.2, both the two share the same limitation; they are both suboptimal relative to the single layer global planning alternatives. But what is unknown is by how much is one better than other in terms of solution quality.

Through these benchmarks, a major contribution of this work (see the conclusion chapter) is given single layered global planning MIP algorithms, local planning MIP algorithm and two layered (or layered) global planning MIP algorithms, which one is better under what conditions ? This question is what (John, 1976) calls the *algorithm selection problem?*

2.12 2.12 Theoretical Performance Efficiency Assessment of SLUM Model

In order to provide answers to the research questions that were stated in section 1.5, a system methodology for analyzing and comparing the performance of the three algorithms is required. With respect to efficiency/running time, algorithms can analyzed using two main approaches: *theoretical (mathematical) or empirical.* In the theoretical approach, a mathematical model is developed that characterizes the performance behaviour of the algorithm, and the algorithm is analyzed within the model. The empirical approach involves running an algorithm and testing its performance against specific problem instances and collecting performance data (Hoos, 2003), (Seogewick & Flajolet, 2009). Empirical evaluation of algorithms complements theoretical/mathematical approach (Coffin & Saltzman, 2000).

In this section, we attempt to answer research questions *RQ1.1* and *RQ1.2* using theoretical/mathematical analysis. The theoretical results could pre-empt some analytic performance efficiency properties of SLUM independent of specific machine implementation

details – the basis of theoretical algorithm analysis in computer science. These results could provide a benchmark on running time properties of the SLUM model against the benchmark algorithms during empirical evaluation.

We begin by deriving mathematical function that describes the worst case performance of SLUM under two special cases and then we obtain a generalized model. Suppose for some webservice composition problem instance, the QoS constraints are such that all candidate webservices at the beginning of SCUM optimization transit to the SPUM, layer. This represents the special case for which $\rho = 1$. This case could happen (though rarely expected). This case deserves special attention as it represents the most computationally expensive case to solve for our proposed model, and generally depicts the performance property of SLUM as ρ approaches 100%. In this scenario, the performance gain due to SLUM over S-MIP, if any, is due to space reduction purely due to decomposition (and not due to elimination) as explained in section 2.10. We present this special case in section 12.12.1.

The second special case is when for some problem instance, the QoS constraints are such that all webservices at the SCUM layer are eliminated i.e no feasible solution is found at the SCUM layer and by implication no feasible solution found globally. In this case, $\rho = 0$. This case is worth attention because, it not only signifies infeasibility but of practical importance, it gives us an idea of the performance behaviour of SLUM for a particular problem instance of size *n* when the rate ρ approaches zero. The case is explained in section 12.12.2.

We then obtain a generalized mathematical function representing the efficiency of SLUM under realistic scenarios, where some webservices that do not meet the QoS constraints at the SCUM layer are eliminated, and some may be promoted for further optimization at the SPUM layer. In the generalized case, the composite service phase transition rate ρ is such that $0 \leq \rho \leq 1$. The generalized case is presented in section 2.12.3.

In each of the three cases, we also obtain the relative speedup of SLUM with respect to SLUM and with respect to L-MIP. When the speedup is in relation to SLUM.

The notations used in this section and the rest of the sections are given in the table 10 below.

Table 10 : Notations used in the Theoretical Performance Analysis of the Proposed Model (SLUM)

Notation	Meaning
q_1	The number of webservice QoS attributes at the Service Consumer Utility Maximization Layer of the SLUM model
q_2	The number of webservice QoS attributes at the Service Provider Utility Maximization Layer of the SLUM model
q_t	The total number of webservice quality attributes such that $q_t = q_1 + q_2$.
k	The number of business workflow tasks
N	The number of functionally similar webservices per workflow task.
Ω_s or Ω	Theoretical relative speedup of SLUM with respect to S-MIP
Ω_l	Theoretical relative speedup of SLUM with respect to L-MIP
g_1	The total number of composite webservices available to SLUM before the start of optimization at the SUM layer
g_2	The total number of composite webservices available to SLUM at the end of optimization at the SUM layer (or beginning of SPUM optimization process).
ρ	Composite Service Transition Rate: $\rho = g_{2/} g_1$
ϵ_i	The number of webservices that were eliminated against the i^{th} workflow task at the SCUM layer
t_B	The theoretical running time of SLUM

2.12.1 Special Case: Composite Service Phase Transition Rate $\rho = 1$

Conceptually, the initial set of possible composite webservices to be optimized using a global planning strategy such as S-MIP is n^k.The time taken to solve the optimization problem in this case is proportional to n^k. The time taken to solve a similar problem by L-MIP is proportional to

nk operations Consider that according to (Zeng et al, 2004) and so (Mulongo et al, 2015), a webservice is modelled as a vector whose elements are the values of the various QoS attributes. In S-MIP the size of each vector is $\boldsymbol{q_t}$ **.** Thus, we have *n* candidate QoS vectors per task to be optimized. In SLUM, at the SCUM layer, the size of each vector is $\boldsymbol{q_1}$ in length, and at the SPUM layer, the size of each (sub) vector is $\boldsymbol{q_2}$. The size of each (sub) vector at the SCUM layer as a proportion of the original vector having $\boldsymbol{q_t}$ elements is $[q_1/(q_1+q_2)]$. Similarly, the size of each (sub) vector at the SPUM layer as a proportion of the original vector is $[q_2/(q_1+q_2)]$. Respectively, the number of (complete) vectors at the SCUM and SPUM layers are $[q_1/(q_1+q_2)\,]*n$ and $[q_2/(q_1+q_2)\,]*n$. Assuming, no webservice is eliminated at the SCUM layer, the theoretical running time taken by SLUM to solve the two sequentially decomposed subproblems is given by *equation 2.38*

$$t_B = \left[\left[[q_1/(q_1+q_2)\,]*n\right]\right]^k + \left[\left[[q_2/(q_1+q_2)\,]*n\right]\right]^k\right]. \qquad (2.38)$$

Let $[q_1/(q_1+q_2)\,] = \acute{\omega}_1$ and $[q_2/(q_1+q_2)\,] = \acute{\omega}_2$, equation 2.38 could be re-written as equation 2.39

$$t_B = \left[[\acute{\omega}_1\, n]]^k + [\acute{\omega}_2 n]]^k\right] = (\acute{\omega}_1{}^k + \acute{\omega}_2{}^k)\, n^k \qquad (2.39)$$

Considering that $\ll n$, the constant terms could be ignored, so that equation 2.39 could be generalized according to equation 2.40.

$$t_B = O(n^k) \qquad (2.40)$$

Thus from equation 2.40, the conclusion is that theoretically, SLUM could be asymptotically as worse as the single layered MIP solution, and for that matter SLUM could be non polynomial deterministic. Hence, this compared to the running time of L-MIP which in the order of *nk*, means L-MIP still outperforms SLUM by several orders of magnitude without further proof.

2.12.1.1 Deriving *Relative Speedup with respect to S-MIP,* Ω_s Using L-Hospital's Rule

Even if two algorithms have the same worst case complexity class, one algorithm might be better than the other on average. By applying L-Hospital's rule, we can compare the relative growth of two functions. Let, $f(x)$ and $g(x)$ be two functions . L-Hospital's rule states that $n \rightarrow \infty \frac{f(x)}{g(x)} =$

$\lim x \to \infty \frac{\partial f(x)}{\partial g(x)}$. Thus by applying the rule to equation 2.39, we obtain an expression for Ω_s as per equation 2.41

$$(\Omega_s) = \frac{n^k}{((\acute{\omega}_1)^k + (\acute{\omega}_2)^k)(n^k)} = \frac{1}{((\acute{\omega}_1)^k + (\acute{\omega}_2)^k)} \quad (2.41)$$

Given that $[q_1/(q_1 + q_2)\,] = \acute{\omega}_1$ and $[q_2/(q_1 + q_2)\,] = \acute{\omega}_2$, and $(q_1 + q_2) = (q_t)$, equation 2.41 can be expanded and simplified to equation 2.42.

$$(\Omega_s) = \frac{(q_t)^k}{((q_1)^k + (q_2)^k)} = \frac{(q_1 + q_2)^k}{((q_1)^k + (q_2)^k)} \quad \mathbf{(2.42)}$$

When $q_1 \approx q_2$, from equation 2.42 , then $(q_1 + q_2)^k > \{[\boldsymbol{q_1}]^k + [\boldsymbol{q_2}]^k\}$. Hence $\Omega_s > \mathbf{1}$. Thus, the complexity of the sum of the parts of two the sequentially decomposed SLUM subproblems grows much slower than the complexity of the whole. Thus, even without elimination of any webservice at the SCUM layer, SLUM would theoretically perform faster than S-MIP.

When the number of QoS attributes at the SCUM layer equals the number webservice QoS attributes at the SPUM layer (the ideal case), i.e $q_1 = q_2$, then, it's easy to show that equation 2.43 holds.

The ideal case is when the ratio $\boldsymbol{q_1 : q_2 = 1}$ **i.e** $\boldsymbol{q_1 = q_2}$ **,** so that the original webservice composition problem is sequentially decomposed into two equal layers in the number of QoS attributes. In this case, equation (3) is further simplified to (4) and finally (5).

$$(\Omega_s) = (\mathbf{2})^{\boldsymbol{k-1}}$$

(2.43)

Thus from equation 2.43, we see that provided $\boldsymbol{q_1 = q_2}$ **,** the (Ω_s) is only dependent on the length of the business workflow and the speedup is a power function of *k.*

Where $\boldsymbol{q_1 \neq q_2}$ **,** we show through the examples below that $(\Omega_s) < (\mathbf{2})^{\boldsymbol{k-1}}$ and (Ω_s) gets much smaller than $(\mathbf{2})^{\boldsymbol{k-1}}$**,**tending towards 1 as the ratio $\boldsymbol{q_1 : q_2}$ $\boldsymbol{or}$ $\boldsymbol{q_2 : q_1}$ gets much larger than 1. Note that the practical maximum limit of $\boldsymbol{q_1/q_2}$ is $\boldsymbol{q_t - 1}$**,** in which case we have $\boldsymbol{q_t - 1}$ QoS attributes at layer 1 and 1 QoS attributes at layer 2. Thus, more formally, let $r =$ $\boldsymbol{q_1/q_2}$ **or** $=$ $\boldsymbol{q_2/q_1}$ **,** whichever is larger. We show that as $r \to 1, (\Omega_s) \to (\mathbf{2})^{\boldsymbol{k-1}}$ and $r \to (\boldsymbol{q_t - 1}), \Omega \to \mathbf{1}$ **.**

Example 1: k =2, $q_1 = q_2 = 4$

This example considers a business workflow with two sequential tasks in which the number of QoS attributes at layer 1 and layer is equal to 4 and therefore q_t =8. Applying the generalized equation in (4) we have $(\Omega_s) = [(\mathbf{8})^2] / \{[\mathbf{4}]^2 + [\mathbf{4}]^2\}$=**2.** Since $q_1 = q_2$ **,** we could also use (7) directly to have $(\mathbf{2})^{2-1} = \mathbf{2}$

Example 2: k =2, $q_1 = 4, q_2 = 3$

This example is motivated by the practical considerations of the SLUM model described in section 2.10 and also in (Abiud W . M et al, 2015), where the number of QoS at the layer 1 (the Service Consumer Utility Maximization layer) is 4 i.e reputation, security, service execution duration and service access cost, and three QoS attributes at the layer 2 (Service Provider Utility Maximization layer) i.e reliability, availability and throughput. Note that according to the SLUM model [24], the number of QoS attributes at either layer could be varied based on the practical guidelines in [24]. Since $q_1 > q_2$ **,** $r = \frac{q_1}{q_2} = \mathbf{1.333}$ and $\Omega = [(\mathbf{7})^2] / \{[\mathbf{4}]^2 + [\mathbf{3}]^2\}$=**49/25 = 1.96.** Notice that $r = \mathbf{1.333} \approx \mathbf{1}$ *and* $\Omega = \mathbf{1.96} \approx \mathbf{2}$.

Example 3: k =2, $q_1 = 6, q_2 = 1$

This is a hypothetical example where layer 1 has six QoS attributes and layer 2 has only one QoS attribute – it demonstrates the effect of a high degree of imbalance between the number of QoS attributes in the two layers on the magnitude of theoretical speedup of SLUM. Here we have $r = q_t - 1 = 6$ and $\Omega = [(\mathbf{7})^2] / \{[\mathbf{6}]^2 + [\mathbf{1}]^2\}$=**49/43 = 1.13.** Notice that three QoS attributes at the layer 2 (Service Provider Utility Maximization layer) i.e reliability, availability and throughput. Note that according to the SLUM model [24], the number of QoS attributes at either layer could be varied based on the practical guidelines in [24]. Since $q_1 > q_2$ **,** $r = \frac{q_1}{q_2} = \mathbf{1.333}$ and $\Omega = [(\mathbf{7})^2] / \{[\mathbf{4}]^2 + [\mathbf{3}]^2\}$=**49/25 = 1.96.** Notice r has reached the limiting value and $\Omega = 1.333$ is much closer to 1 than to $(\mathbf{2})^{k-1}$ or 2.

2.13.1.2 Deriving Relative Speedup with respect to L-MIP, Ω_s Using L-Hospital's Rule

Following the procedure of the previous section, from equation 2.41, (Ω_l) is given by equation 2.44.

$$(\Omega_l) = \frac{nk}{((\acute{\omega}_1)^k + (\acute{\omega}_2)^k)(n^k)} \quad (2.44).$$

In equation 2.44, we see that the denominator high order term (n^k) dominates the numerator term n as k→∞. Hence $(\Omega_l) < \mathbf{1}$ **.** The conclusion is the L-MIP is much faster than SLUM under $\rho = 1$

2.12.2 Special Case: Composite Service Phase Transition Rate $\rho = 0$

When $\rho = 0$, it implies that in equation 2.41, $\acute{\omega}_2 = 0$. Equation 2.41 then transforms to equation 2.45

$$(\Omega_s) = \frac{\boldsymbol{n^k}}{((\acute{\omega}_1)^k)(\boldsymbol{n^k})} = \frac{\mathbf{1}}{((\acute{\omega}_1)^k)} = \frac{(q_1 + q_2)^k}{((q_1)^k)} \quad (2.45)$$

Since, $^{(q_1 + q_2)^k} \gg \left((q_1)^k\right)$, then $(\Omega_s) > 1$ **.** For $q_1 = q_2$ equation 2.46 follows.

$$(\Omega_s) = \frac{(q_1 + q_2)^k}{((q_1)^k)} = \frac{(2q_1)^k}{((q_1)^k)} = (\mathbf{2})^k \quad (2.46)$$

Recall from equation 2.43 that when $\boldsymbol{\rho = 1}$**,** $(\Omega_s) = (2)^{k-1} < (2)^k$. Thus the conclusion is that as the transition rate approaches 0, the speedup of SLUM relative **to** *SMIP* tends to two times larger than when the transition rate tends towards 1. For instance, when k=2 and $= 1$ **,** $(\Omega_s) = 2$, while when k=2 and $\boldsymbol{\rho = 0}$ **,** $(\Omega_s) = 4$. Remember that $\boldsymbol{\rho = 0}$ means infeasibility and therefore hitting the ceiling of $(\mathbf{2})^k$ in practice might not be possible.

Similarly, when $= 0$ **,** Ω_l is given by 2.47.

$$(\Omega_s) = \frac{\boldsymbol{nk}}{((\acute{\omega}_1)^k)(\boldsymbol{n^k})} \quad (2.47)$$

Again from equation 2.47, we see that SLUM is much slower in performance compared to L-MIP.

2.12.3 Generalized Case: Composite Service Phase Transition Rate, $0 \leq \rho \leq 1$

Before the start of SCUM optimization, there are $\boldsymbol{n^k}$ candidate composite webservices, and as usual the computational effort is upper bounded by $\boldsymbol{n^k}$. At the end of SCUM optimization, for each workflow task, some webservices might be eliminated early. As defined earlier ϵ_I is the number of webservices eliminated early against the i[th] task at the SCUM layer ,where $(\epsilon_i) \geq 0$. Thus, at the end of SCUM layer optimization, for each task, $(n - \epsilon_i)$ candidate webservices got

promoted for a second round optimization at layer 2. Therefore the total number of composite webservices promoted to the SCUM layer is $(\mathrm{n}-\epsilon_1) * (\mathrm{n}-\epsilon_2) * \ldots, * (\mathrm{n}-\epsilon_k) = \prod_1^k(\mathrm{n}-\epsilon_i)$. It follows that the total computational effort with elimination in consideration is given by equation 2.48 $(\boldsymbol{n^k(q_1/q_t)^k} + \prod_1^k(\mathrm{n}-\epsilon_i)\,(\boldsymbol{q_2/q_t})^k)$ Therefore the generalized function of the speedup of SLUM with respect to S-MIP under $0 \leq \rho \leq 1$ is captured in equation 2.49.

$$(\Omega_s) = \frac{\boldsymbol{n^k}}{(\boldsymbol{n^k(q_1/q_t)^k} + \prod_1^k(\mathbf{n}-\epsilon_i)(\boldsymbol{q_2/q_t})^k)} \tag{2.48}$$

By dividing the numerator and the denominator of the R.H.S of equation 2.48 by $\boldsymbol{n^k}$ we get equation 2.49

$$(\Omega_s) = \frac{\mathbf{1}}{(\boldsymbol{q_1/q_t})^k + (\boldsymbol{q_2/q_t})^k\ (\prod_1^k(\mathbf{n}-\epsilon_i))/(\boldsymbol{n^k})} \tag{2.49}$$

In equation 2.49, the ratio $(\prod_1^k(\mathrm{n}-\epsilon_i))/(\boldsymbol{n^k})$ happens to be the composite service phase transition rate as defined in thesis in table 2.11. Hence in a more compact form, equation 2.49 could be re-written as equation 2.50.

$$(\Omega_s) = \frac{\mathbf{1}}{(\boldsymbol{q_1/q_t})^k + \boldsymbol{\rho}(\boldsymbol{q_2/q_t})^k} = \frac{(\boldsymbol{q_1+q_2})^k}{(\boldsymbol{q_1})^k + \boldsymbol{\rho}(\boldsymbol{q_2})^k} \tag{2.50}$$

When $q_1 \approx q_2$, equation 2.50 transforms to equation 2.51.

$$(\Omega_s) = \frac{(\mathbf{2})^k(\boldsymbol{q_1})^k}{(\boldsymbol{q_1})^k + \boldsymbol{\rho}(\boldsymbol{q_1})^k} = \frac{(\mathbf{2})^k}{\mathbf{1} + \boldsymbol{\rho}} \tag{2.51}$$

From equation 2.51, we can deduce the following:

i. Substituting $\rho = 1$ in the equation yields $(\Omega_s) = (\mathbf{2})^{\boldsymbol{k-1}}$ **,** which confirms equation 2.43

ii. Substituting $\rho = 0$ in the equation yields $(\Omega_s) = (\mathbf{2})^{\boldsymbol{k}}$ **,** which confirms equation 2.46

iii. (Ω_s) has an inverse relationship with ρ and therefore for known *k, a* plot of the graph (Ω_s) vs ρ is predicted to be a decreasing function.

2.13 Chapter Summary

This chapter reviewed the pertinent literature relevant to our research problem as defined in chapter one. Through the review, we established the key concepts and theories involved in dynamic webservice composition. Mathematical models for the two well-known approaches to dynamic webservice composition: *local planning* and *global planning* were presented and discussed. We then went into the details of the Mixed Integer programming model formulation for the problem. The theory of Layering as Optimization Decomposition was presented. A survey of related work was discussed and the gaps identified. We presented our proposed model SLUM that is based on mixed integer programming and Layering as Optimization Decomposition. We gave an account of the philosophy of the design behind it. This was followed by details of the mathematical formulations of SLUM including a series of equations. We then gave a summary of the strengths and limitations our approach in qualitative terms. A list of research questions worth investigation were posed. Finally, theoretical results related to some of the research questions were established.

The key highlights of this chapter are that:-

i. Dynamic webservice composition remains NP hard multiple criteria decision problem.
ii. There exists gaps in the state of the art: - either a strategy is very efficient but does not support global constraints or can support global constraints but very inefficient. All strategies require the user to capture all the critical QoS constraints.
iii. The proposed model SLUM bridges the above gaps. It does not require a user to specify all critical QoS constraints.
iv. From the theoretical results of section 2.14, SLUM is predicted to be much faster than the standard global planning mixed integer programming models on average. The average speedup of SLUM with respect to S-MIP could be on the interval $[(\mathbf{2})^{\boldsymbol{k-1}}, (\mathbf{2})^{\boldsymbol{k}}]$ where *k* is the number of sequential workflow tasks.
v. From the theoretical results in section 2.14, SLUM could be much slower than the local planning strategy
vi. From the analytic considerations of section 2.10 and 2.11, SLUM could be less optimal on average than S-MIP. At the same time, SLUM could be more optimal than L-MIP.

In the next chapter, we describe the experimental methodology that was used to validate our analytic and theoretical claims about our model and to answer the research questions stated in this chapter.

3 CHAPTER 3: METHODOLOGY

In this chapter, in section 3.1, we first give a summary of the research process that was followed to pursue the research objectives and to answer the research questions that were outlined in chapter one. In section 3.2, we detail research design followed. Section 3.3 outlines the tools that were used to implement the three mixed integer programming optimization algorithms as well as the tools that were used to conduct simulations. In section 3.4, we discuss the methods of data analysis and interpretation. A summary of the chapter is given in section 3.5.

3.1 The Research Process

In line with the research objectives and research questions in chapter one, the study involved the design of a new of optimization model called SLUM and the evaluation of performance of the model against two related alternative optimization algorithms – S-MIP and L-MIP. According to Hoos, 2003), (Bartz-Beielstein & Preub, 2014), (Barr, 2001), a general process model recommended for the performance evaluation of optimization algorithms is shown in figure 9. This process model has been widely adopted in studies such as (Coffin & Saltzman. 2000), (Hoos, 2003), (Hoos, 2009), (Hoos et al, 2014), (Mu & Hoos, 2015and Goldsmith in (Goldsmith Fredrick Simon, 2009), (Levitin, 2011), (Nudelman, 2005). This study also adopted the process model in pursuit of the research objectives and the research questions.

This study adopted both theoretical and empirical methods in the analysis of the runtime efficiency and solution quality of S-MIP and L-MIP.

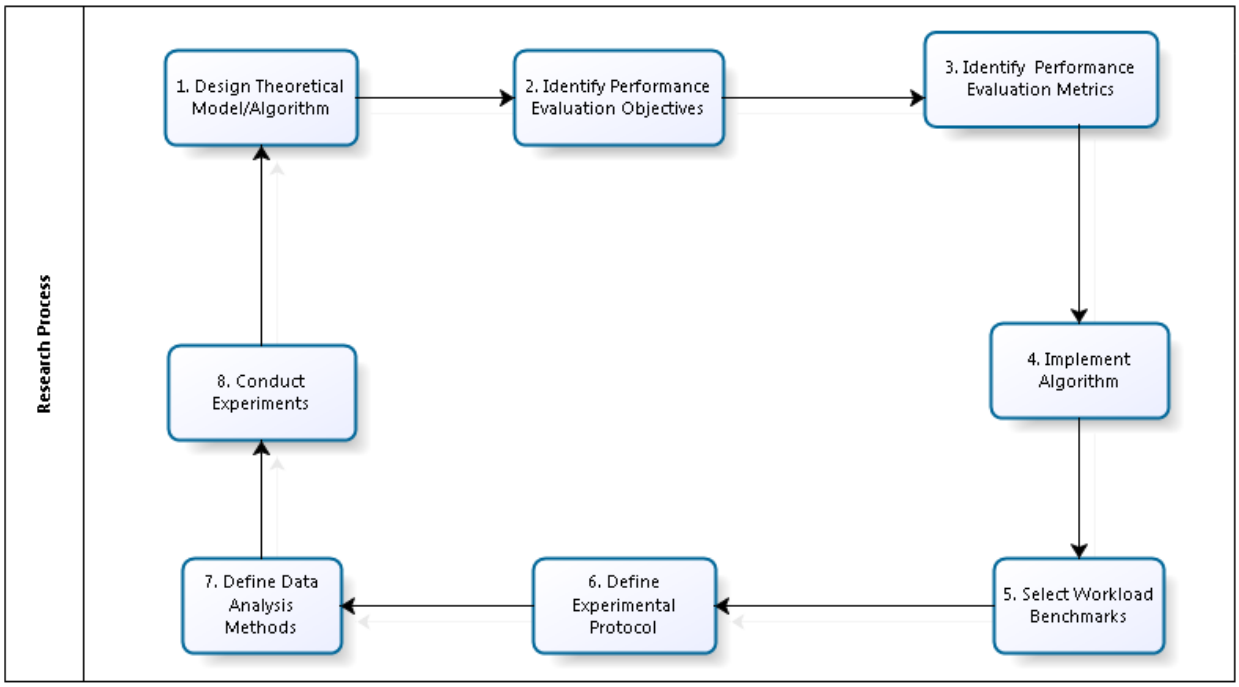


Figure 9: Research Process Framework, Source (Hoos , 2003 ; Seogewic, 2009)

In table 3 below, we highlight in summary, how the tasks undertaken in this study map onto the overall research process framework in figure 9 above.

Table 11: Outline of the Overall Research Process adopted by the Study. Source (Hoos 2003 ; Seogewic, 2009)

	Process Step	**Approach in this study**
1	Design a theoretic model/Algorithm	In line with our first research objective, through literature review, the study designed a two layer MIP model. The output of this phase was the SLUM model. See section 2.9
2	Identify performance evaluation objectives	The performance evaluation objectives are directly related to our second research objective and to the six research questions RQ1.1, RQ1.2, RQ 1.3, RQ1.4, RQ 2.1 & RQ 2.2. See section 3.1.1 for more details
3	Identify Performance Metrics	With respect to our second research objective, two metrics were identified – runtime efficiency and solution quality. The choice for of these metrics is explained in more detailed in section 3.2 & section 3.3 for details
4	Implement Algorithm	

		Involved choosing an appropriate optimization modelling tool. In the study, we chose the Java Optimization Modeler (JOM). See details in section 3.4.
5	Selection of Benchmarks	This involved selecting sample problem instances that are used as subjects to aid in the evaluation. Our study used synthetically generated instances. See section 3.2 & 3.4 for details
6	Define Experimental Protocol	For experimental evaluation, this involves the procedure followed in collecting performance data e.g the system environment, how measures were recorded and so on. We describe this under the section 3.5 titled "experiment setup"
7	Define performance data analysis methods	The study adopted both theoretic performance analysis (given in section rigorous inferential statistical analysis , numerical & empirical algorithmic complexity analysis based on regression analysis,
8	Conduct Experiments	Involved performing experiments to collect the performance data and analysis of the results, interpreting the results and comparing the empirical results with the theoretic performance models and or previous results. See chapter four.

3.2 Research Design

This study sought to answer the research question below.

i. Research Questions Related to Runtime Efficiency

RQ1: For a composite webservice selection problem having a workflow with k tasks and v alternative webservices per task, how does the runtime efficiency of *SLUM compare with that of S-MIP and L-MIP* when each is used to solve the problem*?* The specific research questions arising from this question are:

RQ1.1: How does the running time of SLUM grow as the number of service providers per task increase?

RQ1.2: How does the running time growth of SLUM compare with that of S-MIP and L-MIP?

RQ1.3: How much speedup is achievable when using SLUM over S-MIP to autogenerate composite webservices given a business workflow having n webservices per task?

RQ1.4: What is the minimum number of service providers per workflow task that a virtual enterprise broker needs to have in order to benefit from the relative efficiency of SLUM when compared to S-MIP?

ii. **Research Questions Related to Solution Quality**

RQ2: How does the average solution quality of SLUM compare with that of L-MIP and S-MIP? This leads us to the following specific research questions?

> **RQ2.1**: What is the average relative solution quality (percentage accuracy of quality) of the composite webservices generated by SLUM relative to S-MIP?

RQ2.2: What is the percentage difference in the relative solution quality by SLUM relative to L-MIP?

The researcher adopted a scientific research approach to answer research question RQ1 which relates to runtime efficiency and an empirical approach to answer research question RQ2 which relates to solution quality.

Scientific approach to performance evaluation of computational algorithms combines both theoretical models and experimental evaluation (Coffin & Saltzman, 2000; Hoos, 2003; Seogewick & Flajolet, 2009). In scientific approach, a theoretical model is developed and then experiments are performed to verify whether the theoretical model holds in practice (Coffin & Saltzman, 2000; Hoos, 2003; Seogewick & Flajolet,, 2009). Scientific approach has the advantages of both theoretical and empirical approach in that the empirical results can be checked for consistency with the theoretical model and on the other hand, the practical performance limitations that cannot be detected theoretically can be detected through real problem instances (Coffin & Saltzman, 2000; Hoos, 2003; Seogewick & Flajolet,, 2009).

In the theoretical approach, a mathematical model is developed that characterizes the performance behaviour of the algorithm in terms of the problem input size, and the algorithm is analyzed within the model (Seogewick & Flajolet,, 2009). The analysis could make use of asymptotic worst case analysis, average case analysis, and differential calculus using L-Hospital's Rule. The advantage of theoretical analysis approach is its rigor and its independence of implementation platform details. Secondly, theoretical models are not susceptible to experimental bias and experimental

errors. The challenge of theoretical approach is how to choose a good model that is realistic, and the need for mathematical details (Seogewick & Flajolet,, 2009). A second weakness of the method is that some performance properties of an algorithm can be difficult to model mathematically.

Based on the rigor of theoretical analysis, this study conducted a theoretic performance asssement of the runtime efficiency of SLUM, S-MIP and L-MIP and attempted to compare the efficiency of the three models. From the theoretical perspective, the study attempted to answer research questions RQ1.1, RQ 1.2 and RQ 1.3 as per section 2.12. The key results were:

i. The generic runtime efficiency model for SLUM is given by $\left(\boldsymbol{n^k}(\frac{\mathbf{1}}{\mathbf{2}})^{\boldsymbol{k}} + \prod_1^k(\mathrm{n}-\epsilon_i)\,(\frac{\mathbf{1}}{\mathbf{2}})^k\right)$ and that of S-MIP is given by $\boldsymbol{n^k}$ and that of L-MIP is $\boldsymbol{nk}$. Here, we see that L-MIP is polynomial time and many times faster than both SLUM and S-MIP. Both S-MIP and SLUM are likely to be exponential in k. However, since $\left(\boldsymbol{n^k}(\frac{\mathbf{1}}{\mathbf{2}})^{\boldsymbol{k}} + \prod_1^k(\mathrm{n}-\epsilon_i)\left(\frac{\mathbf{1}}{\mathbf{2}}\right)^k \ll \boldsymbol{n^k}\right.$ for large enough n.

SLUM is generally faster than S-MIP by $(\Omega_s) = \frac{(\mathbf{2})^{\boldsymbol{k}}}{\mathbf{1}+\ (\boldsymbol{n^k}/(\prod_{\mathbf{1}}^{\boldsymbol{k}}(\mathbf{n}-\epsilon_{\boldsymbol{i}})))}$ or $(\Omega_s) = \frac{(\mathbf{2})^{\boldsymbol{k}}}{\mathbf{1}+\ \boldsymbol{\rho}}$ on average. Where $\boldsymbol{\rho} = (\boldsymbol{n^k}/(\prod_{\mathbf{1}}^{\boldsymbol{k}}(\mathbf{n}-\epsilon_{\boldsymbol{i}}))).$ And since $\boldsymbol{\rho}$ is on the interval [0,1], the function $(\Omega_s) = \frac{(\mathbf{2})^{\boldsymbol{k}}}{\mathbf{1}+\ (\boldsymbol{n^k}/(\prod_{\mathbf{1}}^{\boldsymbol{k}}(\mathbf{n}-\epsilon_{\boldsymbol{i}})))}$ has values on the interval $[(\mathbf{2})^{\boldsymbol{k-1}}, (\mathbf{2})^{\boldsymbol{k}}]$.

These theoretical results show that the speedup (Ω_s) attainable by SLUM relative to S-MIP on average lies on the interval $[\ \mathbf{2^{k-1}}\ ,\ \mathbf{2^k}]$ where k is the number of sequential workflow tasks. The theoretical results also reveal that speedup in between the interval depends on the parameter ρ, which we defined as the "composite service phase transition rate". So for instance when k=2, the average speedup of SLUM with respect to S-MIP is expected to be in the range [2, 4], which is a significant performance gain. These theoretical results are useful and could provide a baseline for benchmarking the results obtained through experimentation.

However, the theoretical analysis was not a sufficient means of evaluating the runtime performance of SLUM and the benchmark algorithms because of the earlier mentioned limitations of theoretical analysis. For this reason, we sought the empirical approach as a complementary method because :

i. In attempting to answer RQ2, the study sought to investigate how the performance of SLUM and S-MIP compare initially for small values of *n.* Theoretical analysis cannot detect some performance characteristics of an algorithm that could very useful in practice. Empirical approach provide some details that highly generalized mathematical models cannot. For example, an algorithm *A* that analytically is idealized to be asymptotically better than another algorithm *B* could be worse in performance initially or even on average when tested on real problem instances. Even better, the two algorithms *A* and *B* could be equally bad or good asymptotically, but in practice, one of them could be better than the other. Because asymptotic analysis might be irrelevant for problems of practical essence (Hoos et al, 2003; Seogewick, & Flajolet, 2009), (Goldsmith Fredrick Simon, 2009), it's important to analyze initial as well average case performance of an algorithm empirically. By adopting an empirical approach, we wanted to understand the conditions under which SLUM scales better than the baseline algorithm or performs worse than the baseline.

ii. The theoretical approach could not address the research question *RQ1.4:* What is the minimum number of service providers per workflow task that a virtual enterprise broker needs to have in order to benefit from the relative efficiency of SLUM when compared to S-MIP?.

iii. Empirical analysis reflects real life implementations taking into platform dependent factors, something that theoretical analysis falls short of (Hoos et al, 2003). Because of the possibility of overgeneralizing when using theoretical analysis, the empirical approach was followed as a tool of validating the theoretical/mathematical model. In this case the two approaches become complementary. Coffin & Saltzman (200) notes that theoretical analysis and empirical analysis can be used to complement each other. Coffin & Saltzman (2000) for example notes that empirical performance models (EPM) for an algorithm's running time could be developed statistically to serve as an average case equivalent of the theoretical running model.

Despite the foregoing justification for the use of empirical methodology, the empirical method has a number of shortcomings that threaten validity and reproducibility of experiments, especially where performance efficiency is concerned. Ahuja & Orlin (1996) for instance, remarks that empirical studies involving CPU running time as a measure of algorithm performance suffer from multiple sources of variability: programming language used, compiler, computer hardware, the approach used to encode the algorithms into computer programs (which depends on the skill of the programmer), problem input size parameter combinations and whether or not at the time of measurements, other users were engaging the computing hardware that is used to execute the experiments.

The study did not use theoretical analysis despite its advantages, to evaluate and analyze performance relating to solution quality because it was too complex if not impossible to model the solution quality mathematically as a function problem input size. This is because the solution quality of an optimization algorithm depends on various other parameters such as the statistical structure of the problem instance, the number of constraints, the left and right hand side expressions of the constraints among others. Therefore, we were unable to answer research question two using this approach. A majority of existing research on optimization algorithms in general and composite webservice selection algorithms in particular use an empirical approach to measuring the solution quality of an algorithm (see for example Zeng et al, 2004), (Ardagna & Penci, 2005), (Mulongo et al, 2016).

In section 3.2.1 we illuminate more on the research design concerned with runtime performance efficiency evaluation and in section 3.2.2 we shade more light on the research design relating to solution quality. Issues discussed under each of the subsections are the sampling, sample size, number of runs, dependent and independent variables.

3.2.1 Runtime Efficiency Study

3.2.1.1 Algorithms to be Evaluated.

The algorithms that were compared were S-MIP, L-MIP and S-LUM. Recall that all the three algorithms use Mixed Integer Programming (MIP). Note also that both S-MIP and L-MIP are both flat structured except that S-MIP uses global planning while L-MIP uses local planning. SLUM on the other hand, uses global planning like S-MIP unlike the rest is two layered.

3.2.1.2 Dependent and Independent variables

The dependent variable in respect of runtime efficiency was CPU running time. It's the most widely used variable for measuring efficiency of algorithms empirically. See section 3.3 for a justification on the choice.

The independent variables can be inferred from the theoretical performance models derived theoretically. Based on the theoretical model $(\boldsymbol{n^k}(\frac{\mathbf{1}}{\mathbf{2}})^{\boldsymbol{k}} + \prod_1^k(\mathrm{n}-\epsilon_i)\,(\frac{\mathbf{1}}{\mathbf{2}})^k)$, there are three variables that could be varied while the rest are fixed and the impact on the performance of SLUM relative to the other two investigated. i.e $\boldsymbol{, k,}\ \mathrm{n},\ \epsilon_i$. However, there are two issues to be deal with :

i. The variable ϵ_i is also a variable. That's, the number of service providers eliminated at layer might not be the same for each task. Thus for $\boldsymbol{k}$ the number of eliminated service providers are $\epsilon_{1.}, \epsilon_{2,} \ \ldots, \epsilon_k$. Unfortunately, ϵ_i is not a variable you can directly set a priori since the number of services to be eliminated is further dependent on the problem instances at hand and the optimization constraint expressions and the value of the left hand side (L.H.S) of the constraint expression.

ii. And even if these parameters remained unchanged, it's not still possible to know beforehand what the values of $\epsilon_{1.}, \epsilon_{2,} \ \ldots, \epsilon_k$ would be until the experiments are

run.

The first issue was overcome by having to consider the joint effect of $\in_{1.}, \in_{2,} \ \ldots, \in_{k}$ as implied in the $(\boldsymbol{n^k}(\frac{\mathbf{1}}{\mathbf{2}})^{\boldsymbol{k}} + \prod_1^k(\mathrm{n}-\in_i)\,(\frac{\mathbf{1}}{\mathbf{2}})^k\ )$. i.e we can investigate the effect $\prod_1^k(\mathrm{n}-\in_i)$ on the runtime performance. However, recall that the ratio $\boldsymbol{n^k}/\prod_1^k(\mathrm{n}-\in_i)$ is the composite phase transition rate $\boldsymbol{\rho}$. Thus in this study, the effect of the $\boldsymbol{\rho}$ at a fixed *n* and *k* was investigated as opposed to $\in_{i.}$

This study will restricted the value of *k* at 2 and hence maintained two independent variables for two reasons. Firstly, $k = 2$ is the smallest size of any business workflow required for webservice composition. Given that theoretically, we have an idea that both SLUM and S-MIP could have an exponential runtime. By maintain k at the smallest possible value, and investigating the effect of varying the number of service providers per task, if turns out that SLUM and S-MIP are empirically exponential in runtime growth on a two task workflow, then their performance for larger k values cannot be any better. This observation lets us to comprehensively test the impact of change in the number of service providers on the runtime efficiency. Note that this research is about dynamic webservice composition within virtual organizations, Virtual Organizations would typically span the global (Rabelo et al, 2008) and likely to be operating tens to thousands of service providers per task (Mulongo et al, 2015;2016a). On the other hand, the number of workflow tasks *k* are likely to be much smaller the number of service providers per task, especially because virtual organizations by their real motivation of existence have very lean business processes in their service delivery (Molina & Flores, 1999). The third reason is that in order to answer the research questions, sufficient samples of each of the three variables would be needed in order to sufficiently make a conclusion about the effect each of the variables on the response time while the other two are fixed. For example, based on existing studies such as (Zeng et al, 2004) at least eight samples of *n* would be required. This study introduces another dimension, the $\boldsymbol{\rho}$ value which lies on the range [0,1]. Although there are no previous studies that investigated the effect of $\boldsymbol{\rho}$ on runtime efficiency , we theoretically already know that when this parameter is set at the extreme values have the speedup as $[(\mathbf{2})^{\boldsymbol{k-1}}, (\mathbf{2})^{\boldsymbol{k}}]$ at $\boldsymbol{\rho = 1}$ and $\boldsymbol{\rho = 0}$ respectively. We would have to have other $\boldsymbol{\rho}$

values chosen between 0 and 1 fairly across the interval in order to investigate the response time behaviour of SLUM when $\boldsymbol{\rho}$ tends to 1, tends to 0, is somewhere midway between zero and 0.5 and midway between 0.5 and 1 and near 0.5. These are 5 additional values in addition to 0 and 1 bringing this number to 7 different values of $\boldsymbol{\rho}$.

This kind of setup involving only two variables (**n** and $\boldsymbol{\rho}$) would require 8 by 7 = 56 experiments on one treatment (algorithm). Since we are comparing 3 algorithms, it would require 56 by 3 =168 experiments at the very minimum. If we were to further vary k say to 3, 4 and 5, this would result to 504 experiments. The problem is not yet over. Since we analytically see that two of the algorithms have an exponential scalability in *k,* it would have been time consuming to perform the experiments on standard computers. As will be seen shortly, this study in fact performed 336 experiments 112 experiments for each algorithm.

It could be argued that from $\frac{(\mathbf{2})^{\boldsymbol{k}}}{\mathbf{1+}\ \boldsymbol{\rho}}$, the contribution of *k* to the relative speedup of SLUM is much larger than the contribution $\boldsymbol{\rho}$, and therefore fix p throughout and vary k. This is true. However, in addition to the considerations in the foregoing paragraphs about why we chose to fix k at 2, this study the first one to attempt to define the concept of composite service transition rate and its impact on the runtime efficiency. Although a generic theoretic speedup model $\frac{(\mathbf{2})^{\boldsymbol{k}}}{\mathbf{1+}\ \boldsymbol{\rho}}$ was derived in this study, it was desirable to investigate experimentally how $\boldsymbol{\rho}$ impacts runtime efficiency. In any case, at a fixed k, the impact of varying $\boldsymbol{\rho}$ between 0 and 1 has a significant performance gain over S-MIP, ranging between 200% to 400%.

3.2.1.3 Setting and Computing the Value of ρ Experimentally

The issue that we cannot fix $\boldsymbol{\rho}$ .directly before experiment run as identified in section 3.2.1.1 remains. We don't know how many services per task will be eliminated and hence we can't fix $\boldsymbol{\rho}$ upfront. The study overcame this problem in the following way. At a fixed *n* and *k,* through a trial and error process, the R.H.S values of the one of the SCUM optimization constraints was adjusted,

then experiments were performed. Recall from chapter two section 2.9, that as an example, the following was one of the constraint expression for the total response time a service consumer would be willing to wait. $\left(\sum_{i=1}^{i=N} \boldsymbol{d^s}\right) \leq \boldsymbol{D}$ (see equation 2.32). While holding the rest of the constraints **R.H.S,** *D* is adjusted, experiments run, then the number of service providers eliminated for task 1, task 2 and so on is observed and recorded. Consequently, , $\boldsymbol{\rho = n^k / \prod_1^k (\mathrm{n} - \epsilon_i)}$ is then computed. Note that, we could adjust the R.H.s values of more than one constraint simultaneously, provided we get the desired $\boldsymbol{\rho}$ value at a given. Thus for example, let *n=10*, *k=2* for some problem instance. Therefore, initially, there are 10 webservices per task. Say by setting $\left(\sum_{i=1}^{i=N} \boldsymbol{d^s}\right) \leq \mathbf{10}$, after phase one optimization process we have that $\epsilon_{1.=}\ 4$, $\epsilon_{1.=}\ 2$, $\prod_1^k (\mathrm{n} - \epsilon_i) = (10-4) * (10-2) = 48$, thus $\rho = \frac{48}{100} = 0.48$. Thus, the performance of SLUM at n=10, k=2 , $\rho = 0.48$ is reported. Having determined the combination of constraint values that yielded the $\rho = 0.48$, the experiments would be repeated with *n=10, k=2* , $\left(\sum_{i=1}^{i=N} \boldsymbol{d^s}\right) \leq \boldsymbol{D}$ for a number of runs for SLUM and then repeated while n=10, k=2 and *D* is fixed for S-MIP and L-MIP.

3.2.1.4 Sampling the values for the phase transition rate $\boldsymbol{\rho}$

As said earlier, no previous study examines the effect of **ρ** empirically on the runtime efficiency of SLUM except our study in (Mulongo et al, 2016a). Thus the choice of the number of values of $\boldsymbol{\rho}$ and which values of ρ should be sampled from the interval [0,1] was entirely guided by the researcher. Nevertheless, a guiding principle adopted by the researcher as explained in section 3.2.1.1 is that, through the trial and error process described in section 3.2.1.4, for at a fixed n and k=2, a $\boldsymbol{\rho}$ tending to zero was sought. Note that while theoretically $\boldsymbol{\rho}$ can assume zero, in practice a $\boldsymbol{\rho}$ value of zero is not possible as this would be signify lack of feasibility in the first phase of optimization and the optimization process would in this case terminate prematurely. This value helped investigate whether in practice, as $\boldsymbol{\rho \to 0}$, at k=2, the relative speedup tends to $\boldsymbol{(2)^k = 4}$. Similarly, the researcher through trial and error, set a value of $\boldsymbol{\rho = 1}$, so that we investigate

the convergence of the speedup of SLUM to $(2)^{k-1} = 2$. Other values were set as follows. Some values above 0.5, a value closer to or equal to 0.5, and some values below 0.5. This was done through an exploratory process. Through this trial and error process, the followed ρ values of 0.0296,0.064, 0.13, 0.36, 0.45 and 0.6 were obtained. Note that since this is a two task workflow, getting the square root of each of the ρ values, would give the average number of atomic webservices per task that were promoted per task from phase one to phase two. So that at $\rho =$ 0.0296, 0.13, 0.36, 0.45 and 0.6, 1the average number service providers promoted to layer are respectively 17%, 25%, 36%, 60%,67%, 77%, 100% respectively. The researcher considered this sample fairly representative and sufficient to infer implausible performance trend attributable to ρ.

3.2.1.5 Sampling the values of the number of service providers per workflow task n

When benchmarking algorithms in terms of running time, the three issues to be considered are:- 1) instance hardness- the researcher should focus on hard instances, 2) instance size. A range should be provided for scaling studies and, 3) instance type. The researcher ought to provide a variety of problem instances (Hoos, 2003; Barr, 2001). Problem instance type or variety can be achieved through the use of real application instances or ensembles of instances from random distributions Hoos, 2003).

The researcher chose 16 problem instances ranging from the simplest having five candidate webservices per workflow task, to the hardest having 80 candidate webservices per workflow. In between the two were instances whose size was a multiple of five. Therefore problem instances of size *n=5, 10, 15,… 80* were considered. The sixteen problem instances provide an adequate variety, given that in a similar study in (Zeng et al, 2004), four problem instances were used. Similarly, the hardness of the problems ranging from 5 to 80 is sufficient since in related such as (Zeng et al, 2004), the problem hardness in terms of number of webservices per task is varied from 10 to 40. However, it's worth noting, that while the instance type (statistical structure) affects the running time of randomized algorithms, it does not affect the running time of exact algorithms (Mulongo et al, 2016). Since all the three algorithms are exact algorithms, our focus was on how variation

in the problem instance size affects the running time of each of the three algorithms. In addition to the considerations of previous related studies, the sample size chosen is sufficient for analysis given that as explained in section 3.3, the study adopted rigorous inferential statistical analysis techniques including empirical relative complexity analysis. These methods are powerful tools for inferring runtime performance differences without requiring large samples (Coffin & Saltzman, 2016). This so because, for runtime growth, it's the size of the problem (in this case the number of service providers per task) that matters more than the number of samples i.e few samples with reasonable empirical hardness are more desirable than many sample with low empirical hardness.

3.2.1.6 Problem Instances – the Subjects.

Problem instances variety can be achieved through the use of real application instances or ensembles of instances from random distributions (Hoos, 2003). In this study, a problem instance is the graph $\boldsymbol{G_1}=$ $\left\{\begin{bmatrix}\langle \mathbf{0.99, 0.95, 1000, 5, 4, 10, 20}\rangle, \\ \langle \mathbf{0.91, 0.85, 800, 3, 3, 20, 50}\rangle \end{bmatrix}, \begin{bmatrix}\langle \mathbf{0.90, 0.88, 100, 3, 4, 40, 20}\rangle, \\ \langle \mathbf{0.95, 0.90, 800, 5, 5\ 15, 10}\rangle, \end{bmatrix}\right\}$ in which the first matrix is a pool of functionally similar webservices assigned for task 1 and the second matrix is a pool of functionally similar webservices assigned to task 2. Therefore since we fixed the number of tasks at 2, each problem instance had exactly two matrices.

Each vector within a matrix represent a sequence of different QoS values attached to a single webservice. Each vector has seven QoS values. The values are respectively reliability, availability, throughput, security, reputation, response time and service cost. The seven parameters as explained in chapter one, chapter two and specifically section 2.9, are the most widely used QoS parameters in the webservices research community due to their significance. The number of vectors per webservice is equivalent to *n*, the number of webservices per task.

Therefore, each of the 16 problem instances mentioned in preceding section took the structure above, having 5 by 7 matrix, 10 by 7 matrix, 15 by 7 matrix …, 80 by 7 matrix.

To generate a problem instance with the above structure, two webservices were programmed in Java, each performing a different task in a composite workflow. Using SOAPUI, each of the service was replicated using the SOAPUI mock services feature. Each replica of the webservice would then generate a functionally similar real webservice. So for each webservice, 5 mock webservices, 10 mock webservices , …, 80 mock webservices were generated. Each mock webservice generated had a unique service endpoint. Using groovy scripts within each mock service, the QoS properties were simulated through randomization. For reliability and availability, each of the webservices was programmed to throw faults at random times and timeout, for response time, the webservices were programmed to delay for a random number of seconds. For security, random default values were assigned and so to reputation. Each webservice was invoked only once at ago to generate and report their vector of QoS values. The QoS vectors of the first 5 webservices for each task was saved to a unique file in the structure like the one above, the first 10, the 15 and so on. Therefore 80, different files generated.

3.2.1.6 Runtime Performance Efficiency Metrics

When measured empirically, the running time of an algorithm can be captured using CPU time or through the use of operations counts. The use of CPU running time often raises validity threats. Ahuja & Orlin (1996) for instance, remarks that empirical studies involving CPU running time as a measure of algorithm performance suffer from multiple sources of variability: programming language used, compiler, computer hardware, the approach used to encode the algorithms into computer programs (which depends on the skill of the programmer), problem input size parameter combinations and whether or not at the time of measurements, other users were engaging the computing hardware that is used to execute the experiments. Nevertheless, CPU running time still remains the most widely used method for measuring algorithm running time (Coffin & Saltzman, 2000). On the other hand, operations count entails automatically counting the number of times an algorithm executes major operations (*Ravindra et al, 1996)*. Specific methods of how to implement the operations count method empirically include the *representative (bottleneck) operations counts* method proposed in (Ahuja & Orlin (1996) and the *trend-prof* tool utilizing *execution operation counts* (Goldsmith Fredrick Simon, 2009). This method ideally targets to curtail the challenges of experimentations based on CPU running time. However, the *execution operation counts* approach is not devoid of limitations. In practice it might be difficult to establish which operations critically

affect performance and which ones are insignificant and should therefore be excluded (Mulongo et al, 2016). Moreover, because the counting of such operations is automated through a profiler such as the one in Goldsmith Fredrick Simon, 2009) the impact of the profiler on the program (system) under test may be another source of variability that would be equally elusive to isolate.

This study adopted the CPU running time since its straight forward and popular. The mentioned validity issues due to CPU running time are addressed in the experiment design approach and in the choice of the analysis methods in section and respectively. The measurement units of time were "seconds". The choice for "second" as a unit of measure is motivated by two reasons: 1) the second is the SI unit of time and 2) MIP algorithms take seconds to weeks to solve given problem instances (Ed Klotz, and Alexandra M. Newman , 2012) therefore using the second is adequate enough to ensure high precision and resolution of time measurements. For each problem instance, the CPU running time was automatically tracked and recorded on successful termination of the optimization process.

3.2.1.7 Experimental Protocol in Measuring CPU Time

In this section, we address several issues that need to be addressed during the actual processing of running the experiments in order to ensure validity of results.

The time taken to find an optimal solution can be affected by the number of constraints. During experimentation, all problem instances as well constraint inequalities remained unaltered at a given combination of *n*, k and $\boldsymbol{\rho}$. The three algorithms were tested on the same problem instance, one at a time until the entire ensemble of problem instances were exhausted. Given that different sections of a program can take different times to execute, we only measured the time taken to for each of the algorithms to execute the function *getBestComposite Service (int opt Mode)* in each of the experiments. The Java function *System.CurrentTimeMillis ()* was used to compute the time lapse. Secondly, we ensured that each time the experiment was conducted, the CPU and Memory Utilization of the computer used to conduct experiments remained fairly constant. We realized this by having only the default auto start system services and processes running and only our Java system prototype running during each experiment. Thirdly, the same computer was used throughout the experiments. The computer had the following specifications: LENOVO, Windows

Professional 64 bit (6.1, build 7601), Intel Pentium CPU B960, 2 CPUs @2.20 GHz, 2GB RAM, CPU Utilization, Physical Memory Utilization State at any one time fluctuated between 88% to 93% at any one time, averaging about 90% (or 1.43 GB of 2GB).

In addition to the above measures, to minimize effects of variance by chance in response time observed on different values of *n*, within and between the two treatments, for every problem instance, *10* consecutive measurements of time were taken one at time and the arithmetic average value recorded. This was achieved by executing the workflow 10 times repeatedly. Ten (10) was chosen because it's statistically known that 4-10 repeated measurements are sufficient to significantly reduce the random errors observed on measurements due to uncertainties in the measurement environment. Moreover, for every problem instance, the time measurement on each of the algorithms was immediately successive. For example, at *n=10*, 10 successive measurements of time are taken on L-MIP, then 10 successive measurements on SLUM, then 10 successive measurements on S-MIP. The process is then repeated for *n=15, n=20* etc. This protocol differs from the approach where for n=10, 15,20 .., 10 successive measurements are taken for L-MIP for each *n*, then the procedure repeated for SLUM and then S-MIP. The first approach minimizes the time gap between when measurements on the same subject (problem instance) but on a different treatment (algorithm). This is meant to minimize the impact of intervening system state changes with the passage of time. To put this in context, suppose, at n=10, *20* ,30, 40, 50,60,70 ,L-MIP takes 5 seconds ,10 sec, 20 sec, 40 sec, 80 sec, 160, 320 sec to record ten time measures respectively, if the first method is used the 10 measurements at *n=10* on SLUM would be recorded shortly after 5 seconds, while using the second approach, the 10 measurements at n=10 on SLUM would be recorded after 635 seconds or approximately 10 minutes. If the for some reason, the CPU memory utilization bursts within the 10 minutes delay, the pairwise comparison of the system response time when using L-MIP vs when using SLUM at *n=10* would be somewhat biased.

The same considerations as those stated in section 3.4.1 were taken into account when generating benchmarks for solution quality (SQ) comparison. Unlike in 3.4.1, where the focus is on the hardness, here the focus is how the SQ of SLUM and L-MIP compares on a variety of independent problem instances. Hence for SQ, 40 randomly generated problem instances having *n*=2, 3, 4, ...,

41 webservices per task were used. In section 3.5, the choice of 40 problem instances aided in the selection of appropriate statistical tests of significance as explained in section 3.5.

3.2.2 Solution Quality Efficiency Study

3.2.2.1 Algorithms to be Evaluated

The algorithms that were compared were S-MIP, L-MIP and S-LUM. Recall that all the three algorithms use Mixed Integer Programming (MIP). Note also that both S-MIP and L-MIP are both flat structured except that S-MIP uses global planning while L-MIP uses local planning. SLUM on the other hand, uses global planning like S-MIP unlike the rest is two layered. Solution quality

3.2.2.2 Solution Quality Performance Metrics

A major goal of optimization algorithms is to generate a high quality solution from a large space of alternative solutions within a reasonable amount of time. A common method to gauge the solution quality of the solution value z produced by algorithm A (where A is hypothesized to be suboptimal) on some optimization problem instance p, is to compare the value z against a global optimum value z^* output by an algorithm B that is known to produce a globally optimal solution value z^* for every $p \in P$. We know that the S-MIP algorithm has the property that for every $p \in P$, z^* is output. Let z^B and z^L be the solution values produced by SLUM and L-MIP on the same problem instance. We are interested in measuring the pairwise solution accuracy between z^B and z^* and z^L and z^*. Two commonly used metrics for comparing a suboptimal or approximate algorithm with respect to an optimal one, in terms of solution accuracy are *optimality ratio* (OR) and relation solution *quality (RSQ).* The two metrics have been used previously in studies such as (Coffin & Saltzman, 2000) and Hoos, 2003). The quality metric is one of those discussed in (Eitan , 1981). To make it more intuitive, we convert RSQ to a percentage by scaling it by 100 as per (1). We denote the RSQ of SLUM with respect to SLUM as RSQ_B and RSQ of L-MIP with respect to SLUM as RSQ_L so that (2) and (3) follows. The optimality ratio of L-MIP and SLUM are given by equations 3.4 and 3.5 respectively.

$$\boldsymbol{RSQ = (}[\boldsymbol{z^* - z}\,)/(\boldsymbol{z^*})] * \boldsymbol{100}) \quad \textbf{(3.1)}$$

$$\boldsymbol{RSQ_L = (}[\boldsymbol{z^* - z^L}\,)/(\boldsymbol{z^*})] * \boldsymbol{100}) \quad \textbf{(3.2)}$$

$$RSQ_B = ([z^* - z^B)/(z^*)] * 100) \quad (3.3)$$

$$OR_L = ([z^L)/(z^*)] * 100) \quad (3.4)$$

$$OR_B = ([z^B)/(z^*)] * 100) \quad (3.5)$$

3.2.2.3 Experimental Protocol in Measuring Solution Quality

The solution quality produced by an optimization algorithm is bound to be affected by not only the number of constraints but also the coefficients of the constraint inequalities on the left hand size, the boundary values on the right hand side of the constraint inequalities, and the values within problem instances. We kept these factors invariant and same across the three treatment types during each experimental setup. There was no need for taking several repeated measurements of the solution values for each problem instance, since L-MIP, SLUM and S-MIP are exact algorithms that guarantee to output same solution for a given problem instance no matter how many times the algorithm is invoked on the same problem instance. This differs from probabilistic optimization algorithms that are bound to yield different solutions at different times, other factors kept constant.

As noted earlier, the S-MIP (Zeng et al, 2004) algorithm guarantees global optimality while local planning has no guarantee for global optimality i.e it may yield suboptimal solutions, as experimentally illustrated in (Zeng et al, 2004). The SLUM algorithm in (Mulongo et al, 2015) is hypothesized not to find globally optimal solutions at the "network level" in some cases since it does not consider all QoS attributes at ago, even though it does guarantee optimality within each layer (since global constraints across workflow tasks within a layer are considered). Thus both L-MIP and SLUM are somewhat approximate optimization algorithms relative to S-MIP. Whether or not SLUM or L-MIP finds a global optimum and if not how close the suboptimal solution is from the global optimum may vary from problem instance to problem instance based on the structure of the problem instance. To illustrate this, we will denote G_i as the input webservice QoS graph to a problem instance $P_{i..}$ G_i contains *k* webservice QoS matrices where matrix M_1, M_2 .., M_k corresponds to the set of webservices that can execute worklow task 1, task 2, task *k* correspondingly. The vectors within each matrix are of a fixed length *v* where *v* is the number of QoS attributes. Assume v=7 so the QoS attributes in consideration are the 7 QoS attributes

according to [48]. . For ease of readability, we use "{ }" to represent a graph, "[ ]" to represent a matrix within a graph, "⟨ ⟩" to represent a vector within a matrix, "," is used to separate vectors within a matrix, and matrices within a graph. Let G_1 and G_2 be two webservice QoS graph instances defined according to equation 3.6 and equation 3.7. Examining the two graphs, G_1 seems somewhat systematically and selectively chosen because in each matrix of G_1, one QoS vector (webservice) dominates the other one on all the 7 QoS attributes such that selecting the dominant (best) webservice from each matrix using L-MIP yields a solution global solution. Similarly, selecting the best composite using SLUM will yield network wide global optimum because in layer 1, vector 1 of matrix 1 will be chosen and in layer 2 the same combination will be chosen. Thus if for all problem instances, one vector in each matrix of the input webservice QoS graph dominates the rest on all QoS attributes, it would give a false impression that S-MIP, SLUM and L-MIP all yield globally optimal solutions all the time or all the three at least yield the same cost values (whether optimal or not) for all problem instances.

$$\boldsymbol{G_1}= \left\{ \begin{bmatrix} \langle \mathbf{0.99, 0.95, 1000, 5, 4, 10, 20} \rangle, \\ \langle \mathbf{0.91, 0.85, 800, 3, 3, 20, 50} \rangle \end{bmatrix}, \begin{bmatrix} \langle \mathbf{0.90, 0.88, 100, 3, 4, 40, 20} \rangle, \\ \langle \mathbf{0.95, 0.90, 800, 5, 5\ 15, 10} \rangle, \end{bmatrix} \right\} \tag{3.6}$$

$$\boldsymbol{G_2}= \left\{ \begin{bmatrix} \langle \mathbf{0.91, 0.81, 100, 5, 1, 10, 200} \rangle, \\ \langle \mathbf{0.97, 0.7, 80, 1, 4, 40, 100} \rangle, \end{bmatrix}, \begin{bmatrix} \langle \mathbf{0.90, 0.83, 100, 2, 5, 40, 80} \rangle, \\ \langle \mathbf{0.89, 0.96, 100, 5, 4, 20, 95} \rangle, \end{bmatrix} \right\} \tag{3.7}$$

On the other hand, contrary to G_1, G_2 appears to have a random structure and it's not obvious which webservice within each task is better than the other since some are better than the other on some QoS attributes and worse on other QoS attributes. In this case, it's likely that each of the three algorithms will yield different cost values.

Due to the sensitivity of solution quality to the structure of problem instances, to ensure plausibility of results on solution quality, for each problem instance P_i, the graph G_i was randomly

generated. Both internal and external validity of results as far solution quality is concerned was then ensured because (Mulongo et al, 2016):

i. The QoS matrices within each graph instance are randomized eliminating statistical bias. Thus, any differences observed on objective function cost values on S-MIP, L-MIP and SLUM are not due to mere chance of the structure of the problem ,for example systematic dominance of one QoS vector over the rest for each workflow task.

ii. Each graph instance is independently generated from the other and therefore any differences observed on optimality values across one graph instance is not dependent or related to the other.

iii. The random graph instances generated have monotonically increasing number of QoS vectors (webservices) per task. From a solution quality perspective, this increases the variety of candidate solutions available.

3.3 Algorithm Implementation

The SLUM, L-MIP and S-MIP algorithms were all implemented in Java 7.0 using the Java Optimization Modeler (JOM)[2] tool version 1.15 with a GLPK[3] linear programming optimization solver. Each of these algorithms invoked a program function called *getBestCompositeService (int optMode)* where "optMode" denotes optimization type. The enumeration values for the argument is one of L-MIP=0, SLUM=1 and S-MIP=2.

[2] *www.net2plan.com/jom/*
[3] https://www.gnu.org/software/glpk/

3.4 Data Analysis and Interpretation

The study took a rigorous numerical and statistical approach to the analysis and interpretation of the performance differences of S-MIP, SLUM and L-MIP. Our methodology is mainly informed by the ideas in (Coffin & Saltzman, 2000), (Hoos, 2003; 2009; 2014), (Hoos & Mu, 2015), (Hoos & Mu, 2015) and Goldsmith in (Goldsmith Fredrick Simon, 2009), (Annay Levitin, 2011), (Nudelman, 2005). Using the ideas of (Coffin & Saltzman , 2000), (Nudelman, 2005), Hoos, 2003), (Goldsmith Fredrick Simon , 2009), (Hoos, 2009), (Hoos & Mu (2015), (Levitin, 2011), for example, where appropriate, we derive statistical regression models through model fitting, that describe the empirical scaling behaviour of each of the algorithms with respect to problem instances of monotonically increasing hardness. If the metric is running time, and where meaningful statistical scaling models relating the growth CPU time and the number of webservices per task are obtained, the performance comparison of the three algorithms is first done using the concept of *empirical complexity (EC)* as described in (Coffin & Saltzman , 2000) . EC gives running time performance bounds (empirically) without regard to constant terms, just as is with the case of theoretical algorithm analysis (see section 3.4.2.1). Secondly, where appropriate and based on the regression models obtained, L-Hospital' Rule as described in (Levitin, 2011), is used to determine the average empirical performance of SLUM with respect to either S-MIP or L-MIP (see 3.4.2.2 for details). If the pairwise regression models of SLUM and S-MIP or SLUM and L-MIP, satisfy the conditions in 3.4.2.3, the concept of empirical relative complexity (Coffin & Saltzman, 2000) is then used to quantify the initial as well the empirical asymptotic performance of SLUM with respect to S-MIP or SLUM with respect to L-MIP. In addition to these techniques, we define some descriptive statistical measures of analysis for running time in 3.4.2.5 to augment the analysis. In case no meaningful statistical models for running time were obtained (unlikely to be the case based on the analytic considerations of chapter 1), then the use of sample means or medians coupled with normality or non-parametric tests as described in section 3.4.2.4 were

followed. We summarize the methods used to analyze runtime performance efficiency as follows:

i. *Statistical regression models* (linear, polynomial and exponential) where fitted on data to capture the relationship between the CPU running time (in seconds) and the problem instance size (number of webservices per workflow task, with a fixed number of tasks). Goodness fit tests and significance tests were applied to the regression models. Regression analysis helped us to answer the research question *RQ1.1: How does the running time of SLUM grow with increasing number of webservices per task?.* The complexity class of our proposed method was determined and compared with that of the baseline and the alternative algorithms.

ii. *Empirical Relative Complexity and Empirical Relative Complexity Coefficients* (Coffin & Saltzman, 2000) were used to compare the express the running time of the proposed algorithm as a function of the baseline algorithm, S-MIP i.e , using an equation of the form $t_{eB} = \beta_0 (t_{eA})^{\beta 1}$,where t_{eB} is the running time of SLUM , t_{eA} is the running time of S-MIP. β_0 is parameter showing how many times SLUM performs faster (or slower) than S-MIP initially (small sized problem instances) while $(t_{eA})^{\beta 1}$ is the number of times SLUM is faster (or slower) than S-MIP asymptotically. To obtain the model $t_{eB} = \beta_0 (t_{eA})^{\beta 1}$, first the statistical regression models of both algorithms have to be obtained, and must each be linear after a log transformation. This method was used to address the question *RQ1.2: How much speedup is achievable using SLUM over S-MIP?.* Thus were able to determine how many times SLUM was slower or faster than S-MIP initially and asymptotically using this approach.

iii. *Differential Calculus based on the L-Hospital's Rule:-* We used this method to compute expected average speedup of the algorithm relative to S-MIP. This method also relied on the statistical functions obtained from regression analysis. The expected speedup values obtained empirically were compared to the theoretical speedup results described in chapter 2 that were also computed using L-Hospital's Rule.

iv. Combining ii and iii above, other useful parameters were determined e.g what is the least of number of webservices (virtual enterprises) per workflow task is required for SLUM to be at least X times faster than S-MIP.

v. *Sample Instantaneous Speedup* was used to show the speedup of SLUM relative to S-MIP

at a particular value of *n* (Mulongo et al, 2016)

vi. *Speedup – Phase Transition Graphs* (Mulongo et al, 2016) were used to show the relationship between the speedup of SLUM relative to S-MIP and various Composite Service Phase Transition Rates.

vii. Mean was used to summarize the sample speedup, and the solution quality of the algorithms for comparison purposes. For solution quality, parametric tests were used to test significance in difference of mean values

viii. Graphs were generally used to visualize trends. We also used bar graphs to depict solution quality of the algorithms.

In regard to solution quality, if meaningful regression models of relative solution quality with respect to problem instance size *(n)* are derivable, the RSQ models could be used to describe the scaling behaviour of *RSQ with respect to n* for SLUM and L-MIP. Further, if a scatter plot of RSQ_B vs *n* and RSQ_L vs *n* or *log* RSQ_B vs *n* and log RSQ_L are strongly linear, then the slope test could be used to detect performance solution quality performance differences between L-MIP and SLUM. If no suitable model is derivable or the linearity condition is not satisfied, then either normality tests or nonparametric tests as described in section 3.4.1.2 were used to detect RSQ performance differences between the two algorithms. A similar approach is recommended by the authors in (Coffin & Saltzman, 2000) and the effectiveness of the approach illustrated by the same author on a wide range of optimization problems in (Coffin & Saltzman, 2000).

3.4.1 Analysis and Interpretation of Relative Solution Quality and Optimality Ratio

3.4.1.1 Detecting Solution Quality Difference between L-MIP and SLUM using Slope Test

As said earlier, this test is conducted if linearity exists between both the pairs RSQ_L vs *n* and RSQ_B vs *n* or *ln* RSQ_L vs *n* and *ln* RSQ_B vs *n*. Equation 3.8 was used to generalize the function describing the linear relation between RSQ_L vs or *ln* RSQ_L vs *n*, and equation 3.9 to mean the function describing the linear relation between RSQ_B vs *n* or *ln* RSQ_B vs *n*.

$$\boldsymbol{y_{in}^1 = \beta_{01} + \beta_{11} n + \epsilon_{ni}}, \quad (3.8)$$

$$\boldsymbol{y_{in}^2 = \beta_{02} + \beta_{12} n + \epsilon_{ni}}, \quad (3.9)$$

Where y_{in}^1 = RSQ_L or y_{in}^1 = *ln* RSQ_L, y_{in}^2 = RSQ_B or y_{in}^2 = *ln* RSQ_B , β_{01} and β_{02} are the

intercepts representing heuristic effects (Coffin & Saltzman, 2000). , β_{11} and β_{12} are slopes, and $€_{ni}$ are random variations.

We set the null and alternative hypotheses in equation 3.10. If any of the null hypotheses is rejected, then the two algorithms differ in RSQ performance. Further if H_0 in equation 3.11 is accepted, then it was concluded that the problem size effect on RSQ_L is the same as the problem size effect on RSQ_B. Similarly if RSQ_B in (10) is accepted, then the heuristic effect on RSQ_B is equal on RSQ in both algorithms.

$$\mathbf{H_0 : \beta_{01} = \beta_{02} \text{ vs } H_1 : \beta_{01} \neq \beta_{02}} \quad (3.10)$$

$$\mathbf{H_0 : \beta_{11} = \beta_{12} \text{vs } H_1 : \beta_{11} \neq \beta_{12}} \quad (3.11)$$

3.4.1.2 Detecting Solution Quality Performance Differences

To check for performance differences in solution quality between SLUM and L-MIP, we used the paired Student t-test if the performance differences were normally distributed or non-parametric tests otherwise. To determine normality of the distribution of the performance differences, we used Shapiro Wilk test to test for normality of the performance differences between the pairs. Other tests of normality include Kolmogorov-Smirnov, Lilliefors and Anderson-Darling tests (Razali & Wah, 2011). However, Razal and Wah (2011) established that the Shapiro-Wilk test is more accurate in detecting normality of distribution in data hence the rationale for our choice of the method. With the Shapiro-Wilk test, we computed the statistic *W* and compared the value of W against the critical value W_c at a significance level of 0.05. The more close the value of *W* is close to 1 the more likely the data are normally distributed. If *W* is determined to be more than W_c then it could concluded that there is no reason to believe that the data are not normally distributed, otherwise we could concluded that the differences in RSQ values are not normally distributed. In case the differences in RSQ values do not follow a normal distribution, we could use either the sign test or the Wilcoxon matched pairs signed rank test. In section **3.3.3,** we explained that our optimization problem instances were randomly generated. Holdger H. Hoos (2003) states that if the benchmark optimization problem instances are random in nature, the binomial sign test or Wilcoxon matched pairs signed rank test could be used to detect performance differences (assuming that the test of normality has failed on the data). Since the Wilcoxon tests assumes

symmetry of the data sample, we would choose the method if a histogram plot of the performances differences was fairly symmetrical in shape. If on the contrary, the histogram was asymmetrical, the sign test that has no symmetry assumptions could be used. In the case of using either of the nonparametric, tests, the null hypothesis is that the median relative solution quality for SLUM and L-MIP is the same at a 95% confidence interval. On the other hand, for the paired student t- test, the null hypothesis is that the mean RSQ_B and the mean RSQ_L are equal.

3.4.2 Analysis of CPU Running time Performance Differences

3.4.2.1 Running time Analysis using Growth of Functions and Statistical Regression Models

First, we define *empirical complexity (EC)* of an algorithm according to Coffin & Saltzman (2000) as the function describing the growth of the empirical running time of the algorithm with respect to the problem instance size *n*. Like the theoretical counterpart, the empirical complexity can be regarded as the running time statistical regression model without the constant terms. For example, suppose the statistical model capturing the growth of some algorithm is a polynomial of the form $\beta_1 n^{\beta_2} + \beta_0 n + c$, then as per the definition of Coffin & Saltzman (2000) , the EC of this algorithm is denoted by $O(n^{\beta_2})$.

Here and henceforth, any regression model of the form in equation 3.12 will be said to belong to the linear empirical complexity function $O(n)$, while any regression model of the form in equation 3.13 will be said to belong to the polynomial empirical complexity function $O(n^{\beta_2})$ and any regression model of the form in equation 3.14 will be said to belong to the exponential empirical complexity function $\left(O(e^{o(n)}\right)$. For simplicity, let $C_l = O(n)$, $C_2 = O(n^{\beta_2})$ and $C_3 = \left(O(e^{o(n)}\right)$.

Define $T_{eL}(n)$, $T_{eA}(n)$ and $T_{eB}(n)$ as the parameterized running time empirical functions of the number of candidate webservices per workflow task *n*, of the local planning optimization strategy as in (Zeng et al, 2004), S-MIP and SLUM respectively. We use $T_e(n)$ to imply $T_{eL}(n)$, or $T_{eA}(n)$ or $T_{eB}(n)$. Let g_l , g_p and g_e respectively, be linear, polynomial and exponential functions of *n* of the form in equations 3.12 ,3.13 and 3.14 respectively

$$\boldsymbol{\mu_1} = \boldsymbol{g_l(n) = \beta n + c} \tag{3.12}$$

$$\boldsymbol{\mu_2 = g_p(n) = \beta_1 n^{\beta_2} + \beta_0 n + c} \quad (3.13)$$

$$\boldsymbol{\mu_3 = g_e(n) = \beta_0 e^{\beta_1 n}} \quad (3.14)$$

We would like to establish the running time statistical regression function that significantly describes $T_{eL}(n)$, $T_{eA}(n)$ and $T_{eB}(n)$. To determine whether $T_e(n) = \mu_i$, *i=1, 2 or 3*, we used statistical regression tests where the sample data points were fitted on the model μ_i, i=1, 2, 3, one at a time. The R^2 statistic was used to test goodness of fit of the model μ_i on the data. If the R^2 *<0.8,* we automatically accepted the null hypothesis that μ_i does not fit the data. Otherwise, we performed a further test to check if the percentage of fit is significant. We set $p = 0.05$. If the computed p value is greater than 0.05, we accepted the null hypothesis that the model μ_i , despite having an acceptable goodness of fit, does not significantly fit the data at $p = 0.05$. Otherwise we accepted the alternative hypothesis that the R^2 value is significant and hence the μ_i model fits the data. The null hypothesis and alternative hypothesis are summarized as below.

$$\boldsymbol{H_0 : T_e(n) \neq \mu_i \text{ if, } R^2 < 0.8 \text{ or } (R^2 \geq 0.8 \text{ and } p > 0.05)} \quad (3.15)$$

$$\boldsymbol{H_1 : T_e(n) = \mu_i \text{ if, } R^2 \geq 0.8 \text{ and } p < 0.05} \quad (3.16)$$

We used the Data Analysis ToolPack and the Real Statistics Resource Pack Microsft Excel 2013 plugins to perform the regression tests

It's possible that for instance $T_{eA}(n)$ is significantly described by more than one kind of regression model e.g $g_p(n)$ and $g_e(n)$ so that (15) and (16) follows.

$$\boldsymbol{T_{eA}(n) = g_p(n)} \quad (3.17)$$

$$\boldsymbol{T_{eA}(n) = \mu_3 = g_e(n)} \quad (3.18).$$

According to equation 3.17, $T_{eA}(n) \in O(n^{\beta_2})$, *since* $g_p(n) \in O(n^{\beta_2})$ and $T_{eA}(n) \in (O(e^{o(n)})$ given that $g_e(n) \in (O(e^{o(n)})$ according to equation 3.18.

In the case where the running time function is significantly described by two or three regression models, then we concluded that $T_e(n)$ is tightly empirically lower bounded by the smallest empirical complexity class and upper bounded the largest empirical complexity class, noting that $C_l \ll C_2 \ll C_3$. We will use the notation $T_e(n) = \mu_i$ to denote that the $T_e(n)$ *significantly* fits

on the model μ_i, $T_e(n) \neq \mu_i$, otherwise where I is a value on the closed interval [1,2,3]. Further, we will use $T_e(n) = \{\mu_2, \mu_3\}$ to imply both the models μ_2 and μ_3 significantly fit the function $T_e(n)$Thus, $T_e(n) = \{\mu_2, \mu_3\} \rightarrow, T_e(n) \in \{C_{2,} C_3\}$. If two of the three or all the three algorithms are lower bounded by the same empirical complexity class, then we could conclude that on average, theoretically the algorithms perform the same initially (ignoring the constant terms). If two of the three or all the three algorithms are empirically upper bounded by the same empirical complexity class, then we could conclude that on average, theoretically the algorithms perform the same asymptotically (ignoring the constant terms). If the empirical lower bound of one algorithm *X* is the empirical upper bound of the other algorithm *Y*, then clearly Y is far more efficient than *X.* If two of the three or all the three algorithms share the same lower bound empirical complexity class, as well as the upper bound empirical complexity class, then theoretically, we could conclude that the algorithms have the same runtime performance. As an example, assume $T_{eL}(n) = \{C_1, C_2\}$, $T_{eB}(n) = \{C_2, C_3\}$ and $T_{eA}(n) = \{C_2, C_3\}$. As per our definitions and criteria, we can conclude that L-MIP is far faster than both S-MIP and SLUM, while both SLUM and S-MIP have the same theoretical performance since both of them are have a polynomial lower bound and an exponential upper bound.

3.6.2.1.1 Testing $H_1 : T_e(n) = \mu_1$

This is the most straightforward case. Using the " (linear) regression feature in Data Analysis ToolPack, we provided the $T_e(n)$ values as Y variable values and n values as X variable values. The output was recorded. In addition, a graph of $T_e(n)$ vs n was drawn and a linear regression trend line added using Microsoft excel native capabilities. The coefficient, intercept and R^2 values obtained using excel trend line feature were then counterchecked against the corresponding values obtained using the ToolPak addin.

3.6.2.1.2 Testing $H_1 : T_e(n) = \mu_2$

Recall that this test entails fitting $T_e(n)$ on the polynomial regression model given in equation 3.13 and determining the parameters β_0 β_1, and c for some known β_2. In this work, we set $\beta_2 = k = 2$ thus making the assumption that μ_2 is a quadratic function. This assumption is reasonable in general, since we saw in section 1 that global optimization, algorithms conceptually take time proportional to n^k to generation of candidate composite webservices. Thus by setting $\beta_2 = 2$,

equation 3.19 holds.

$$\boldsymbol{\mu_{2\,=} g_p\,(n) = \beta_1\, n^2 + \beta_0\, n + c} \qquad (3.19)$$

In equation 3.19, we have two independent variables *n (X* variable 1) and n^2 (X variable 2). We computed the range of values of n^2 from known values of *n.* To test that $\boldsymbol{T_e\,(n)}$ is quadratic, we used multiple linear regression analysis. This was accomplished using Excel Data Analysis ToolPak, where we input the range of $\boldsymbol{T_e\,(n)}$ values in the Y input range and all the values in the range *X* variable 1 and X variable 2. The output contains among other items the values of the coefficients $\beta_{1\,and}\ \beta_0$ and the intercept C. In addition, a scatter plot of $\boldsymbol{T_e\,(n)\ vs\ n}$ was done in Excel and polynomial regression fitted on the curve using Excel "Add Trend line" feature. The coefficients and the intercept values and the R^2 values obtained using the scatter plot were compared with those obtained using ToolPak.

3.6.2.1.3 Testing H_1 **:** $\boldsymbol{T_e\,(n) =}\ \mu_3$

As described above, this test involved establishing whether or not $\boldsymbol{T_e\,(n)}$ fits on some exponential function μ_2 (as in equation 3.14. Since μ_3 is of the form $\beta_0\, e^{\beta_1 n}$, if indeed $\boldsymbol{T_e\,(n)}$ grows exponentially with respect to *n* then *log* $\boldsymbol{T_e\,(n)}$ should be linear with respect to *n*. Following this, we transform equation 3.14 to equation 3.20 by taking natural logarithms on both sides.

$$\boldsymbol{\ln\ \ T_e\,(n) =\ ln\,\beta_0\ +\ \beta_1 n\ =\ \beta'_0\ +\ \beta_1 n} \qquad (3.20)$$

From equation 3.20 we applied linear regression using ToolPak where the *Y* Input range takes on the range of ln $T_e\,(n)$ and the X input range takes the range of values of *n.* The scatter plot ln $T_e\,(n)\ vs\ n$ is also drawn and linear trendline obtained in a manner similar to the one explained in the preceding sections. Additionally the graph $T_e\,(n)\ vs\ n$ (expected to be exponential graphically) is drawn and exponential fitting using the Excel trendline feature done. The β_1 value obtained from ToolPak is directly crosschecked against the β_1 values obtained from the linear regression and exponential regression modes of the scatter plot ln $T_e\,(n)\ vs\ n$ and $T_e\,(n)\ vs\ n$ respectively. On the other hand, β'_0 value obtained from the graph ln $T_e\,(n)\ vs\ n$ is directly counterchecked against the intercept value obtained using ToolPak whereas the inverse of β'_0 or the inverse of the intercept value of the ToolPak output is checked against the β_0 obtained from direct exponential regression fitting of the curve $T_e\,(n)\ vs\ n$.

3.4.2.2 *Running time Expected Relative Speedup Analysis using Limits of Growth of Functions*

Once the performance regression models were established for each of the algorithms, the analysis that followed in section 3.6.2.1 aimed to characterize the performance of each of the algorithms into empirical complexity classes ignoring the constant terms. The analysis in 3.6.2.1 therefore is tantamount to the theoretical analysis of algorithms. In this section, the algorithms that *theoretically* share the same upper bound empirical complexity class are further analyzed for average (practical) performance using differential calculus and L-Hospital's Rule and limits theory. The rationale for this analysis, is that even if the running time of two algorithms are characterized by the same "worst case" empirical complexity, it's of practical relevance to analyze which one is better than the other on average .Using the example in 3.6.2, even if $T_{eA}\,(n)$ and $T_{eB}\,(n)$ have an exponential time upper bound, it is likely that $T_{eB}\,(n)$ is better than $T_{eA}\,(n)$**,** at least informed by the analysis in chapter two and also in (Mulongo et al, 2015;2016). In fact, it's easy to quickly tell from the scaling graphs of $T_{eB}\,(n)\ vs\ n$ and $T_{eA}\,(n)$ vs n. Moreover, the coefficients or constant terms obtained through the statistical tests in 3.6.1.1 can hint which of the two algorithms grows faster in running time, even if the two algorithms share the same upper bound empirical complexity class.

We will term the performance efficiency gain (or loss) of SLUM with respect to either S-MIP or L-MIP as *n* tends to infinity as *SLUM Expected Speedup (SES).* SES is a function or constants that tells how many times SLUM is faster or slower than S-MIP (if *SES* is computed with respect to S-MIP) or faster or slower than L-MIP (if *SES* is computed with respect to L-MIP). Let SES^g and SES^l represent SES with respect to S-MIP and SES with respect to L-MIP. Let $f(n)$ be the empirical regression equation (function) representing the runtime function of SLUM and $g(n)$ be the empirical regression equation (function) representing the runtime growth of either L-MIP or S-MIP. Further, in this analysis, we use $T_e\,(n)$ to refer to either $T_{eA}\,(n)$ or $T_{eL}\,(n)$.

To show that as $n\ \to \infty$, $T_e\,(n) \gg T_{eB}\,(n)$, we need to show that $\frac{T_e\,(n)}{T_{eB}\,(n)} \to \infty, n\ \to \infty$ **.** The converse is to show that $\frac{T_{eB}\,(n)}{T_e\,(n)} \to 0, n\ \to \infty$**.** We adopt the former. Applying L-Hospital's Rule, equation 3.21 is true.

$$\lim n \to \infty \frac{T_e\,(n)}{T_{eB}\,(n)} = \lim n \to\ \infty\ \frac{\partial\, T_e\,(n)}{\partial\, T_{eB}\,(n)} \tag{3.21}$$

Mathematically, the SES defined here is given by equation 3.22.

$$\boldsymbol{SES = \lim n \rightarrow \infty \frac{\partial T_e(n)}{\partial T_{eB}(n)}} \quad (3.22)$$

Thus SES is the slope of the function $T_e(n)$ with respect to $T_{eB}(n)$ for large enough *n.* Two outcomes are possible. The first case is that SES is a constant (real number). The second case is that SES is a function of *n.* In the first case, to show that $T_{eB}(n)$ is more efficient than $T_e(n)$, it suffices to show that $SES > 1$, otherwise for the latter case, we have to show that the function $SES \rightarrow \infty, n \rightarrow \infty$. Consequently, we make the null and alternative hypothesis below.

$$\boldsymbol{H_0 : T_e(n) =\ll T_{eB}(n)} \quad (3.23)$$

$$\boldsymbol{H_1 : T_e(n) \gg T_{eB}(n)} \quad (3.24)$$

H_0 will be accepted if $SES \leq 1$ *or* $SES(n) \rightarrow 0, n \rightarrow \infty$ .Otherwise H_1

In the case where SES is a function of *n,* it's essential to determine the value of *n* for which the value of the slope > 1. We will call the value of *n* at which the expected speedup is more than 1 as the *expected critical point* and denote it by n_{CE} . The larger the $\boldsymbol{n_{CE}}$ the more remote the chances are that a small scale virtual enterprise broker will benefit from the efficiency of our method as opposed to an alternative technique. The significance of this is so that a virtual enterprise broker, for instance can determine how many virtual enterprise service providers per workflow task the broker needs in order to benefit from using our approach. Because we do not foresee a situation where SLUM is faster than L-MIP, the analysis and determination of n_{CE} will only be with respect to S-MIP.

3.4.2.3 Runtime Performance Correlation based on Empirical Relative Complexity

The analysis in section 3.6.2.1 concerned characterizing the empirical complexity of L-MIP, SLUM and S-MIP with an aim to determining and comparing their theoretical limits. In 3.6.2.2, the analysis targeted (practical –all terms in the regression equation considered) average performance of SLUM with respect to S-MIP. In this section, the analysis aims to compare the initial practical and asymptotic practical performance of $T_{eB}(n)$ against $T_e(n)$, where $T_e(n)$ could be $T_{eA}(n)$ or $T_{eL}(n)$. The analysis is carried out using the method by Coffin & Saltzman (2000). If the regression model obtained by either plotting $T_{eB}(n)$ vs $T_e(n)$ is linear or by plotting $ln\, T_{eB}(n)$ vs $ln\, T_e(n)$ is linear, then the resultant regression model of the form in

equation 3.25 can be used to tell whether $T_{eB}(n)$ is better or worse than $T_e(n)$ initially and by how much, and also show whether $T_{eB}(n)$ is better or worse than $T_e(n)$ asymptotically and by how much. This can be can easily be checked graphically. However, graphs wouldn't quantify the magnitude of relative differences initially and asymptotically. Also, the methods in preceding two subsections cannot reveal these levels of detail.

Picking from 3.6.1, we saw that $T_{eA}(n)\ \left(O(e^{o(n)}\right))$ and $T_{eB}\left(n\left(O(e^{o(n)}\right)\right)$**.** By plotting a graph of ln $T_{eB}\ vs\ ln\ T_{eA}(n)$**,** a linear regression of the form in equation 3.25 holds. Note, t_{eA} = $\boldsymbol{T_{eA}(n)}$**.**

From equation 3.25 we obtain equation 3.26.

$$\boldsymbol{ln\ T_{eB}(n) = ln\ \beta_0 + \beta_1\ ln\ t_{eC} + €} \quad \textbf{(}3.25)$$

$$\boldsymbol{T'_{eB}(n) = \beta_0\ t_{ec}^{\beta_1}} \quad (3.26)$$

Thus given the running time algorithm *A* (S-MIP), running time of algorithm *B* (SLUM) in terms of t_e can be estimated using (26). Coffin & Saltzman (2000) refers to the function $O(t_e{}^{\beta_1})$ as the empirical relative complexity of algorithm B relative to algorithm *C* and the parameter β_1 as the empirical relative complexity coefficient of algorithm *B* with respect to algorithm *C*, where algorithm C is either L-MIP(algorithm L) or S-MIP (algorithm A). When $\beta_1 < 1$**,** then empirically, *B* is asymptotically much faster than *C* (Coffin & Saltzman , 2000). . Otherwise when $\beta_1 > 1$, then , *B* is asymptotically much slower than C (Coffin & Saltzman , 2000) . As $n\rightarrow\infty, \boldsymbol{T'_{eB}(n)} \approx t_{eA}{}^{\beta_1}$ **(**Coffin & Saltzman , 2000**).** On the other hand, when $\beta_0 > 1$, it means that algorithm C is faster than B for small enough *n*, while when $\beta_0 < 1$, it means that algorithm C is slower than algorithm B for small enough *n* **(**Coffin & Saltzman , 2000). **.** We made two sets of hypotheses, one on the parameter β_0 and the other on β_1 . The hypotheses are captured in equations 3.27, 3.28, 3.29 and 3.30. The null hypothesis in equation 3.27 below claims that the initial performance of both algorithms is equal while the corresponding alternative hypothesis in equation 3.28 claims that the initial performance is not the same. H_0 in equation 3.29 states that the asymptotic performance of the two algorithms is the same while the alternative hypothesis H_1

$$\boldsymbol{H_0 : \beta_0 = 1} \quad (3.27)$$

$$\boldsymbol{H_1 : \beta_0 \neq 1} \quad (3.28)$$

$$\boldsymbol{H_0 : \beta_1 = 1} \quad (3.29)$$

$$\boldsymbol{H_1 : \beta_1 \neq 1} \quad (3.30)$$

3.4.2.4 Performance Difference Detection using Parametric/Non Parametric Tests

As explained earlier on, these tests shall be applied if the performance comparison between SLUM vs S-MIP or SLUM vs L-MIP using the method in 3.6.2.3 is not feasible. Coffin & Saltzman (2000) notes that CPU running times exhibit increasing non constant variance so that any attempt to use tests with normality assumptions may not yield plausible results. To confirm this, we used Shapiro Wilk test to test for normality of the performance differences between the pairs. If the data were normally distributed, we use the paired student t-test, otherwise we use either the signed test or the Wilcoxon matched paired test.

3.4.2.5 Runtime Analysis using Sample Instantaneous Speedup and Sample Mean Speedup

All the preceding methods of analysis are based on inferential statistics. Inferential statistics provide more rigorous tools (than descriptive statistics) of estimating population parameters based on sample data (Howel C. David, 2013). Nevertheless, descriptive statistics can be useful tools in summarizing sample data (Howel C. David, 2013) .We define two descriptive statistics: SLUM *Sample Instantaneous Speedup (SSIS) and SLUM Sample Mean Speedup* (SSMS). SSIS and SSMS with respect to S-MIP and with respect to L-MIP are denoted as $SSIS_g$, $SSMS_g$, $SSIS_l$ and $SSMS_l$, and defined according to (31) , (32), (33) and (34) respectively.

$$\boldsymbol{SSIS_g = (T_{eL}(n))/(T_{eB}(n))} \quad \textbf{(3.31)}$$

$$\boldsymbol{SSMS_g = \left(\sum_1^N SSIS_g\right)/N} \quad \textbf{(3.32)}$$

$$\boldsymbol{SSIS_l = (T_{eL}(n))/(T_{eL}(n))} \quad (3.33)$$

$$\boldsymbol{SSMS_l = \left(\sum_1^N SSIS_l\right)/N} \quad (3.34)$$

Where *N* is the sample size (number of problem instances).

Thus, SSIS is the speedup of SLUM for a specified problem instance of size *n*. $SSIS < 1$ means that SLUM is slower than the alternative algorithm for some specific value of *n*. $SSIS = 1$, means that SLUM has equal efficiency with the alternative algorithm for some *n* while SLUM is faster than the alternative strategy for some *n* when $SSIS > 1$. $SSMS$ has a similar interpretation, although over the set of all *N* sample problem instances. Similar to n_{CE} (see section 3.6.2.3) will define n_{CS} as the value of *n* beyond which $SSIS > 1, for\ all\ n > n_{CS}$. Thus $for\ all\ n < n_{CS}$,

$SSIS \leq 1$. Therefore, even without performing the detailed regression analysis, $SSIS$, . $SSMS$ and n_{CS} can pre-empt some interesting performance behaviour of SLUM. Because we do not foresee a situation where SLUM is faster than L-MIP, we only determine n_{CS} will only be with respect to S-MIP.

3.5 Chapter Summary

In this chapter we discussed the research methodology followed in order to study the runtime performance efficiency and the solution quality of our proposed model SLUM against the standard related models- S-MIP and L-MIP. We discussed the overall research process, followed by a detailed discussion on research design. Research design addressed the issues concerning the metrics of measurements of runtime efficiency and solution quality, in which respectively, CPU time and relative solution quality (RSQ) were adopted. The design also addressed the issues of sampling where for CPU time, 16 problem instances, having 5, 10, .., 80 service providers per task and two tasks for each problem instance were chosen. The choice of 16 samples for CPU runtime was found to be sufficient in relation to: i) previous related studies, ii) the fact that for CPU runtime, the difficulty of the problem instance as opposed to merely the size of the sample, is more important and iii. the methods of analysis and the nature of statistical tests applied were robust enough i.e regression analysis, empirical relative complexity analysis and differential calculus. Further, the composite phase transition rate values were seven and were distributed over the interval [0.1], including the 0 and 1. The distribution was selected carefully so that we could observe the behaviour of SLUM efficiency as the composite phase transition rate progressively tends to zero and progressively tends to 1. For solution quality, 40 samples were determined as the right sample size to evaluate differences in solution quality among the three algorithms. The number was chosen because differences in mean performance was the only viable method for assessing solution quality differences by the very nature of this metric and also in regard to previous studies. Normality tests were identified as the method of analysis given that the sample size was more than the minimum number 30 required for test of normality. Other issues discussed were, the number of runs in which we determined for CPU runtime, 10 runs would be performed for each experiment. This owes to the fact that measurements of CPU runtime taken even within the same interval might have some variances. For solution quality, it was determined that only one run was sufficient because the algorithms under text are deterministic in terms of the solution

returned. For this reason, provided the optimization variables remained invariant, running the same experiment many times would still yield the same utility value of the objective function. We also addressed the issue of how problem instances were generated and their structure. The answer here was that simulated webservices were programmed to generate random vectors of seven QoS values. The vectors of related webservices were packed into one matrix, leading to two matrices leading to a n by *7* matrix, where n as said varied from 5, 10 , to 80.

4 CHAPTER 4: RESULTS AND DISCUSSIONS

The two specific research objectives of this study were:-

1. Design a layered hierarchical mixed integer programming model for the composite webservice selection problem following the concepts from the theory of Layering as Optimization Decomposition.
2. Evaluate the performance of the SLUM model against the single layered global planning technique (S-MIP) and the local planning method (L-MIP) in terms of two metrics:
    i. Running time (performance efficiency) and;
    ii. Solution quality.

Objective number one was achieved by way of presenting the proposed Service Layered Utility Maximization (SLUM) model as detailed in section 2.10, after conducting literature review.

Objective number two, roman number one was partially achieved through the theoretical mathematical analysis presented in section 2.14. As explained in chapter 3, it was not possible to mathematically model the relative solution quality of the algorithms prompting for an experimental approach. Further the experimental approach as detailed in chapter 3, besides serving as a verification tool of the theoretical results, was designed to answer the research questions outlined in section 2.13, some of which could not be answered through theoretical mathematical analysis. The research questions were:-

RQ1: For a composite webservice selection problem having a workflow with k tasks and v alternative webservices per task, how does the runtime efficiency of *SLUM compare with that of S-MIP and L-MIP* when each is used to solve the problem? The specific research questions arising from this question are:

RQ1.1: How does the running time of SLUM grow as the number of service providers per task increase?

RQ1.2: How does the running time growth of SLUM compare with that of S-MIP and L-MIP?

RQ1.3: How much speedup is achievable when using SLUM over S-MIP to autogenerate composite webservices given a business workflow having n webservices per task?

RQ1.4: What is the minimum number of service providers per workflow task that a virtual enterprise broker needs to have in order to benefit from the relative efficiency of SLUM when compared to S-MIP?

RQ2 that was outlined in section 2.13 of chapter two. The question is, *RQ2*: How does the average solution quality of SLUM compare with that of L-MIP and S-MIP? This leads us to the following specific research questions:

RQ2.1: What is the average relative solution quality (percentage accuracy of quality) of the composite webservices generated by SLUM relative to S-MIP?

RQ2.2: What is the percentage difference in the relative solution quality by SLUM relative to L-MIP?

This chapter presents the experimental results of the study. The results are later analyzed in response to the to the above research questions. Recall that the specific research questions were designed towards addressing objective number two.

A significant amount of the results, analysis and discussions reported here can also be found in (Abiud W. M. et al, 2016a). In sections 4.1 to 4.8 results on the running time behaviour of the three algorithms are presented. Recall that from chapter two, we theoretically showed that the running time growth $T_{eB}(n)$ and hence the speedup Ω of SLUM relative to the baseline algorithm (S-MIP) varies directly as the initial number of webservices (virtual enterprise service providers) per task n, and inversely as the *composite service phase transition rate* ρ. Hence, in this study, in order to answer research questions *RQ1*, we designed the experiments such that the variation of $T_{eB}(n)$ with n and the speedup Ω were reported at specific average values of ρ. Recall also from chapter 3 that ρ cannot be determined directly a priori since at the start of the experiment, it's not possible to know how many webservices will proceed to phase two of the optimization process. However, by varying one or some of the boundary values of the webservice QoS constraints, the number of webservices that get promoted to phase two can be indirectly controlled. Holding the set of input data set constant, we used a trial and error process, where we tuned some of the constraints in order to achieve a desired average value of ρ. At the end of each experiment, the mean value ρ_{avg} was computed. Since ρ_{avg} varies on the continuous closed interval [0,1], we

designed a number of experimental setups having ρ_{avg} values on this range. The experiment setup were as follows.

The first setup consisted of the case where ρ_{avg}= 0.0296 approaches zero, representing the special case when $\rho=0$ as described in chapter two , section 2.14.2. This is the case where the average speedup Ω relative to S-MIP is expected to be approximately 2^k for large enough n. The second experiment setup had ρ_{avg}=1, which is the special case described in chapter two section 2.14.1. As per section 2.14.1, when ρ_{avg}=1, the average expected to be approximately 2^{k-1} for a large enough *n.* The other experiments had ρ_{avg} values in between, specifically $\rho_{avg}=0.61$, $\rho_{avg}=0.45$, $\rho_{av}=0.36$, $\rho_{avg}=0.13$ and $\rho_{avg}=0.064$.

In each of the above setups, k was fixed at 2 and the same ensemble of input problem instance set was used. The problem instance set consisted of 16 problem instances having *n=5, 10, 15, ... 80.*

The results produced in each of the experiments were analyzed following the methodology described in section 3.6.2:

i) Descriptive Statistics (Sample Instantaneous Speedup), Scatter plots of running time, growth

ii) Statistical Regression Analysis to quantitative characterize the running time empirical function. Makes use of the scatter plots in (i) above

iii) Expected Speedup using L-Hospitals Law to determine average speed up for large *n.* Makes use of the regression functions in (ii) above

iv) Initial and asymptotic speedup using Empirical Relative Complexity analysis. Makes use of the analysis in (ii) above.

At the end, we summarize the key results on running time in section 4.9.

In section 4.10, we present findings on solution quality of SLUM in relation to the two other algorithms. The analysis follows the methodology established in section 3.4.1 of chapter three.

In section 4.11, we present a detailed discussions of the results. In our discussion, we link the findings to our research questions. Further we explain our results linking them to the analytic and theoretical considerations of chapter two. In this section, we also report any "an unexpected results" that have no immediate scientific/theoretical basis.

4.1 Running Time Analysis when mean Composite Service Phase Transition, ρ=0.0296

4.1.1 Running time Scaling Scatter Plots and Simple Descriptive Statistics (Sample Speedup)

It can be observed that for each of the three algorithms, the time taken to find a solution is consistently increasing with respect to *n*; figure 10 confirms this observation. Moreover, the running time of SLUM, S-MIP and L-MIP nearly the same at *n=5.* Beyond *n=5*, the running time of L-MIP is persistently lower than that of SLUM and S-MIP. In fact, both SLUM and S-MIP running time grows more than double compared to L-MIP for every increment in *n.* For example, when *n*=10, t_{eA} = 1.3, t_{eB} = 1.3 and t_{El}=0.68. Thus, at *n*=10, both SLUM and S-MIP are more than two times slower than L-MIP. When *n* is doubled from 10 to 20, it's possible to see that both SLUM and S-MIP are about 4 times slower than L-MIP. Table 12 also shows that SLUM is generally slower than S-MIP for all *n<40* since we can see that for *n<40*, the SIS<1. On the other hand, SLUM is consistently faster than S-MIP for *n>40* since SIS>1 for all *n>40.*

Table 12: CPU Running time Data when Phase Transition Rate ρ=0.0296

N	***N-Squared***	$t_{eB}(s)$	$t_{eA}(s)$	$t_{eL}(s)$	***ln*** (t_{eA})	***ln*** (t_{eB})	***ln (teL)***	SIS_n
5	25	0.65	0.59	0.55	-0.53	-0.43	0.6	0.91
10	100	1.3	1.3	0.68	0.26	0.26	0.39	1
15	225	1.96	1.96	0.8	0.67	0.67	0.22	1
20	400	4.2	4	0.66	1.39	1.44	0.42	0.95
25	625	4.6	4.2	0.8	1.44	1.53	0.22	0.91
30	900	6.9	6.35	0.88	1.85	1.93	0.13	0.92
35	1225	9.9	9.6	0.88	2.26	2.29	0.13	0.97
40	1600	13.3	14.5	1	2.67	2.58	0	1.09
45	2025	15.8	22	1	3.09	2.76	0	1.39
50	2500	19.3	32.5	1.3	3.48	2.96	0.26	1.68
55	3025	22.5	40.3	1.4	3.7	3.11	0.34	1.79
60	3600	30	62	1.5	4.12	3.4	0.41	2.07
65	4225	37	87	1.66	4.46	3.61	0.51	2.35
70	4900	47	118	1.72	4.77	3.85	0.54	2.51
75	5625	60	155	1.9	5.04	4.09	0.64	2.58

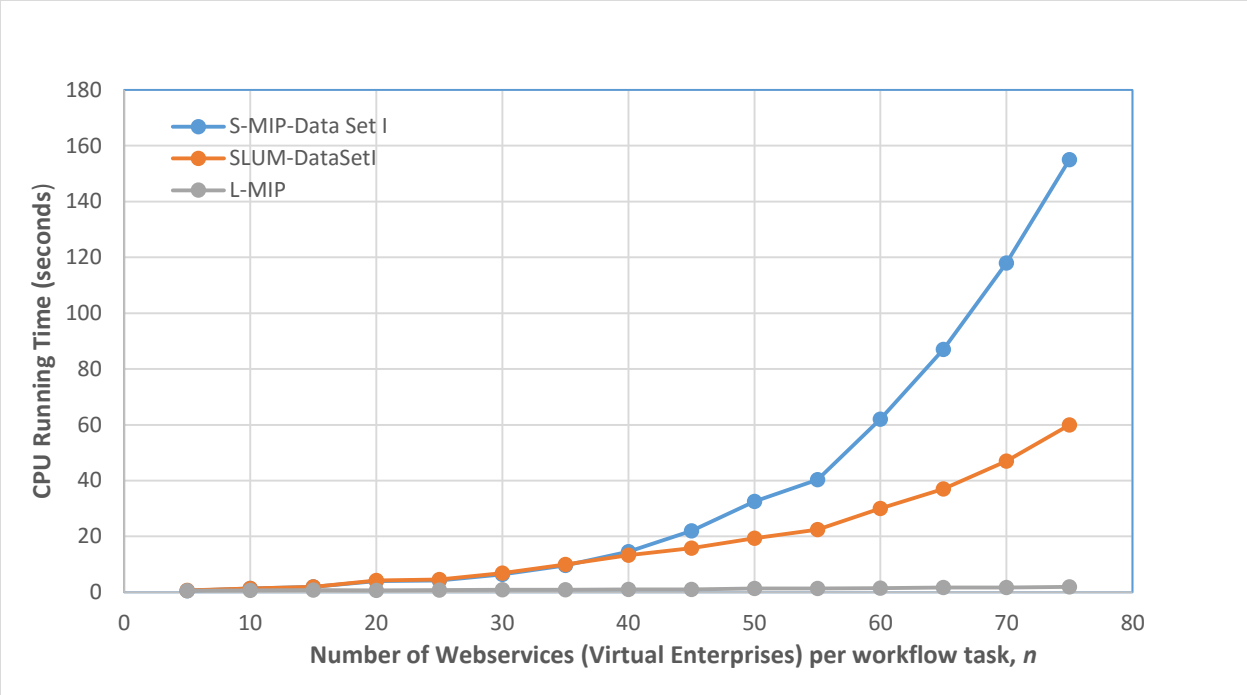


Figure 10 Empirical Running Time Growth Scatter Plot of L-MIP, SLUM and SMIP ρavg = 0.0296.

Therefore from table 3 and figure 10, we have that $n_{cs} = 40$.

Figure 11 below shows how the linear regression models of the running time growth of SLUM, S-MIP & L-MIP at a fixed composite service phase transition rate p=0.0296.

The table 4 below shows a summary of the linear regression, regression and exponential regression statistics for each of the three algorithms : SLUM, S-MIP and L-MIP.

4.1.2 CPU Running Time Growth Analysis via Linear, Polynomial and Exponential Regression

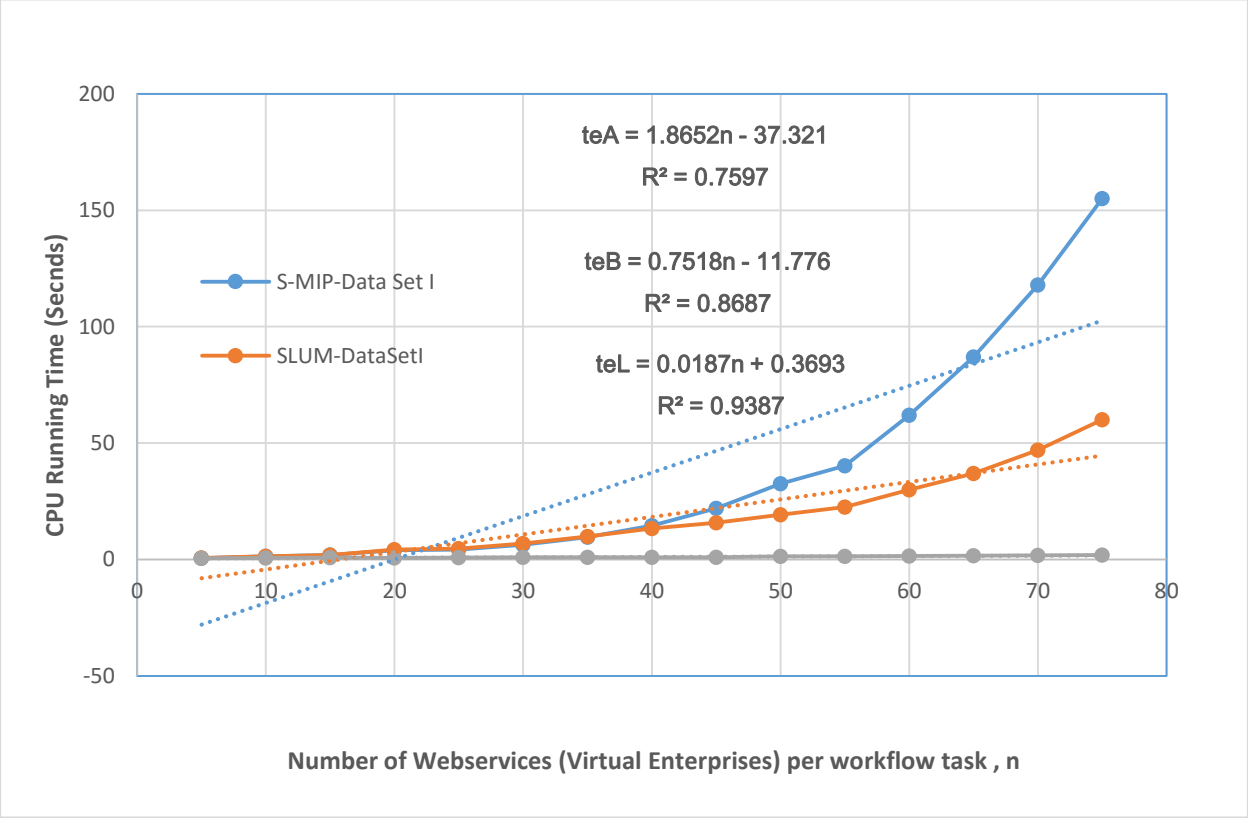


Figure 11 Empirical Running Time Growth Linear Regression Analysis at ρavg = 0.0296

Table 13: CPU Running time Goodness of fit and Significance Results when Phase Transition Rate ρ=0.0296

	Regression Model Type									
	Linear Regression			Polynomial Regression				Log-linear regression)		
Algorithm	R^2	p_1	p_i	R^2	p_1	p_2	p_i	R^2	p_n	p_i
L-MIP	0.93	$1.95*10^{-5}$	0.0004	0.978	0.86	0.000127	$4.5*10^{-6}$	0.23	0.08	0.35
S-LUM	0.868	$4.3*10^{-7}$	0.007	0.987	0.004	0.000001	0.024	0.97	$1.5*10^{-10}$	0.38
S-MIP	0.78	$2.5*10^{-5}$	0.007	0.98	$6.6*10^{-5}$	$3.0*10^{-7}$	0.0025	0.995	$4.0*10^{-15}$	0.00004

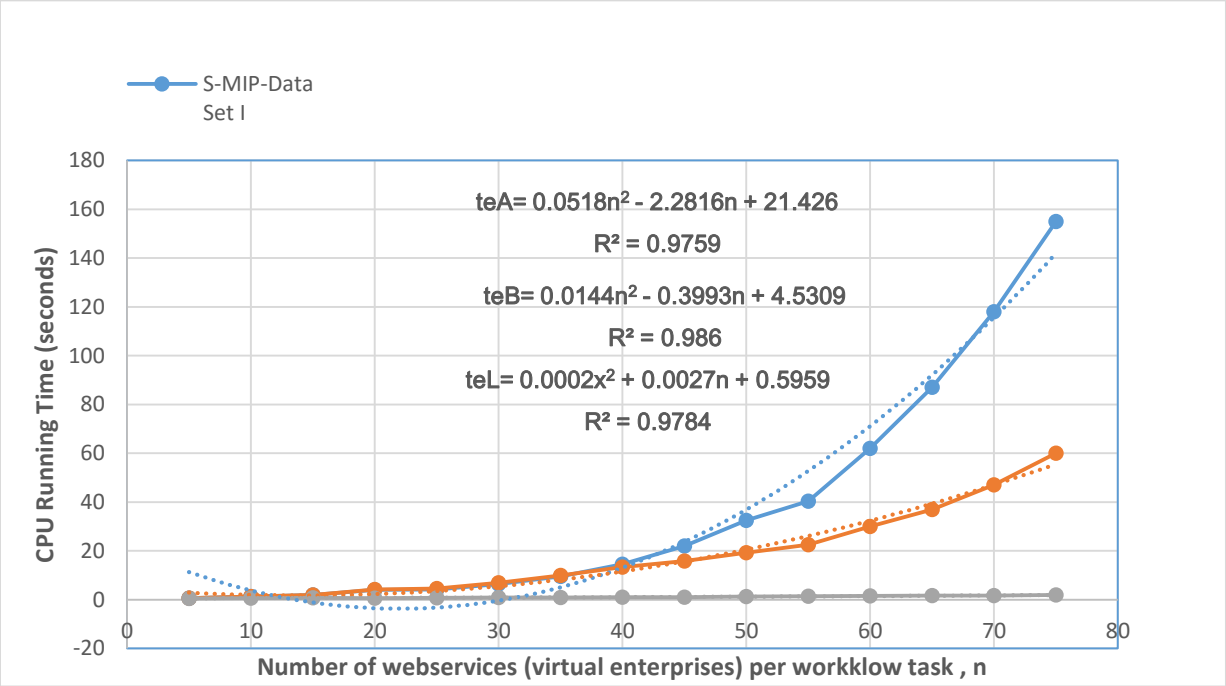


Figure 12 Empirical Running Time Growth – Polynomial Regression Curves at ρ_{avg} = 0.0296

Figure 12 above shows how the polynomial regression models of the running time growth of SLUM, S-MIP & L-MIP at a fixed composite service phase transition rate p=0.0296. On the other hand, figure 13 below shows the log-linear regression models of the running time growth of SLUM, S-MIP & L-MIP at a fixed composite service phase transition rate p=0.0296. Note that as described in chapter 3, the log-linear regression was used to text the exponential runtime growth of each of the three.

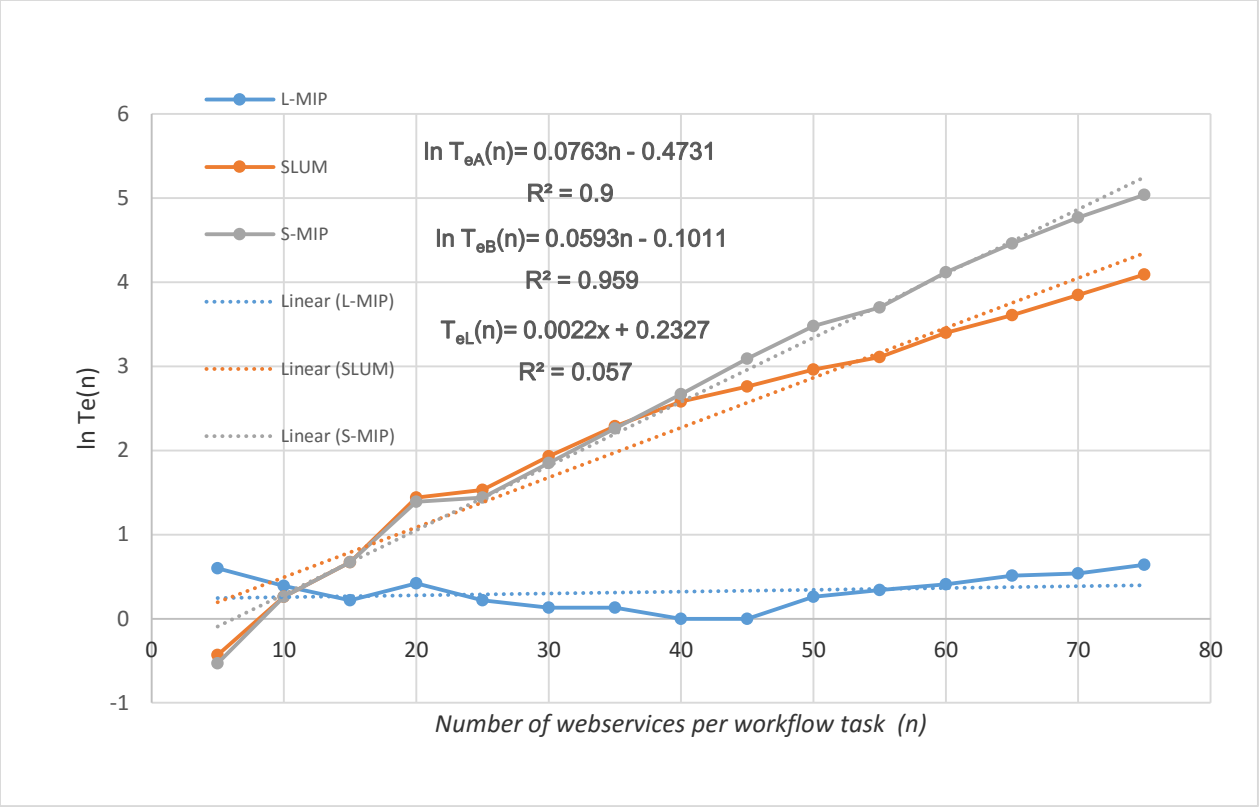


Figure 13 Empirical Running Time Growth – Log Linear Regression at ρ_{avg} = 0.0296

4.1.2 SLUM Expected Speedup via L. Hospital's Law

Applying the L-Hospital's Law, we compute the SLUM expected speedup (SES) as per the methodology in chapter three. Using the statistics from figure 12 and figure 13, the SES value under polynomial growth, SES_P is determined as per equation 4.1 and. the SES function under exponential growth is given equation 4.2.

$$SES_p = \lim_{n\to\infty} d(t_{eA})/d((t_{eB}) = \frac{0.0518}{0.0144} = 3.6 \quad (4.1)$$

$$SES_E = \lim_{n\to\infty} d(t_{eA})/d((t_{eB}) = \frac{0.6231e^{0.0763n}}{0.904e^{0.0593n}} 0.6893e^{0.017n} \quad (4.2)$$

The function $0.6893e^{0.017n}$ can be used to compute the expected speedup for a given number of service providers per workflow task at ρ=1. To compute, n_{CE} ,the critical value beyond which

SLUM is faster than S-MIP, we could set the expected SES at 1.1 and solve the inequality in equation 4.1.

$$0.6893\mathrm{e}^{0.017\mathrm{n}} \geq 1.1 = 0.0117nlne = \ln 1.1 \rightarrow n \geq 8 \quad (4.3)$$

Equation 4.3 means that if a virtual enterprise broker had at least 8 virtual enterprises per task, and the current transition rate is 0.029, they would enjoy a 10% increase in speedup when using SLUM instead of S-MIP.

Similarly, the least of number of service providers per task needed to achieve an expected speedup of 3.6 is given as per equation 4.4 below.

$$0.6893\mathrm{e}^{0.017\mathrm{n}} \geq 3.6 = 0.0117nlne = \ln 3.6 \rightarrow n \geq 109 \quad (4.4)$$

4.1.3 Initial and Asymptotic Speedup via Empirical Relative Complexity under Exponential Growth

We plot log t_{eB} vs log t_{eA} as shown in figure 14. The relationship log t_{eB} vs log t_{eA} seems to be strongly linear. We checked the significance of the relationship using the Microsoft Excel ToolPak plugin, and obtained a p value of 0.01 for the intercept and a p value of 3.85 * 10^{-13} for the X variable. These results indicate that the linear relationship is not only strong but statistically significant at a significance level of 0.05. The linear relationship allowed to us to compute the values of β_0 and β_1 in the equations 3.25 and 3.26. We obtained $\beta_0 = 0.2506 \; and \; \beta_1 = 0.7833 \rightarrow t_{eB} = 1.28{t_{eA}}^{0.783} \rightarrow \beta_0 = 1.28$ and $\beta_1 = 0.7833$. Since $\beta_0 > 1$, we reject the null hypothesis in equation 3.27 and accept the alternative hypothesis of equation 3.28. Additionally, $\beta_1 < 1$ and therefore we reject the null hypothesis in equation 3.29 and accept the alternative hypothesis in equation 3.30. We thus conclude that S-MIP is 1.28 times faster than SLUM initially, but SLUM is more efficient than S-MIP asymptotically since ${t_{eB}} = {t_{eA}}^{0.783}$ for large enough $t_{eA.}$

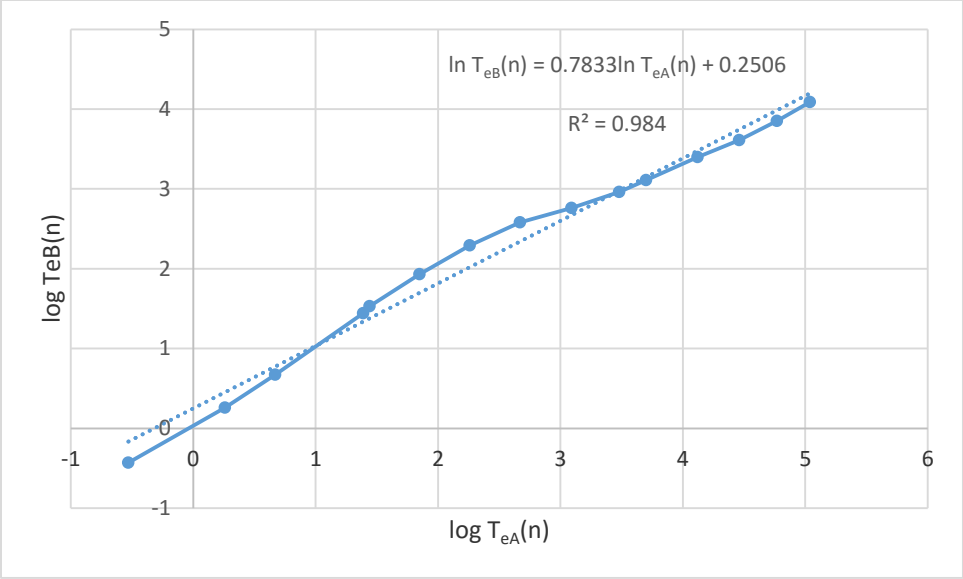


Figure 14 Empirical Relative Complexity –log-log Scatter plot at ρ_{avg} = 0.0296

4.2 Running Time Analysis when Mean Composite Service Phase Transition, ρ=1

4.2.1 Running time Scaling Scatter Plots and Simple Descriptive Statistics

Table 5 presents the CPU runtime performance of SLUM and S-MIP with respect to problem instances of increasing empirical hardness. As explained earlier, optimization inequality constraints were tuned once such that for all problem instances, all candidate webservices evaluated during stage one were all promoted for evaluation in phase 2. The goal was to ensure that any variation in performance between SLUM and S-MIP is not attributed to the service elimination effect. The data shows that until *n=45,* the performance of SLUM is marginally worse than that of S-MIP. Beyond *n=45,* the performance of SLUM is steadily better than that of S-MIP. Moreover the relative speedup of S-MIP ***Ssi*** increases steadily, starting at 1.017 when *n=45* and grows to 1.2 at *n=120.* The scatter plot in figure 15 and the SLUM Instantaneous speedup curve (SISC) in figure 4.8 reinforce the observations.

Table 14: CPU Running time Data when Phase Transition Rate ρ=1

N	*TB (s)*	*TA (s)*	*Ssi*	*N*	*TB (s)*	*TA (s)*	*Ssi*
5	0.62	0.56	0.9032	60	42.2	43.4	1.0284
10	1.37	1.3	0.9489	65	59.1	61	1.0321
15	1.86	1.7	0.9140	70	79.89	83.1	1.0402
20	2.64	2.4	0.9091	75	100.76	104.3	1.0446
25	3.87	3.49	0.9018	80	130.09	138.2	1.062354
30	5.3	5	0.9434	85	165.54	177	1.069211
35	7.45	6.95	0.9329	90	218.17	235	1.07712
40	10.7	10.5	0.9813	95	254.81	275.6	1.081592
45	15.74	16	1.0165	100	312.01	339.3	1.087455
50	22.2	22.4	1.0090	110	481.76	530	1.100139
55	31	31.6	1.0194	120	673.85	748	1.110038

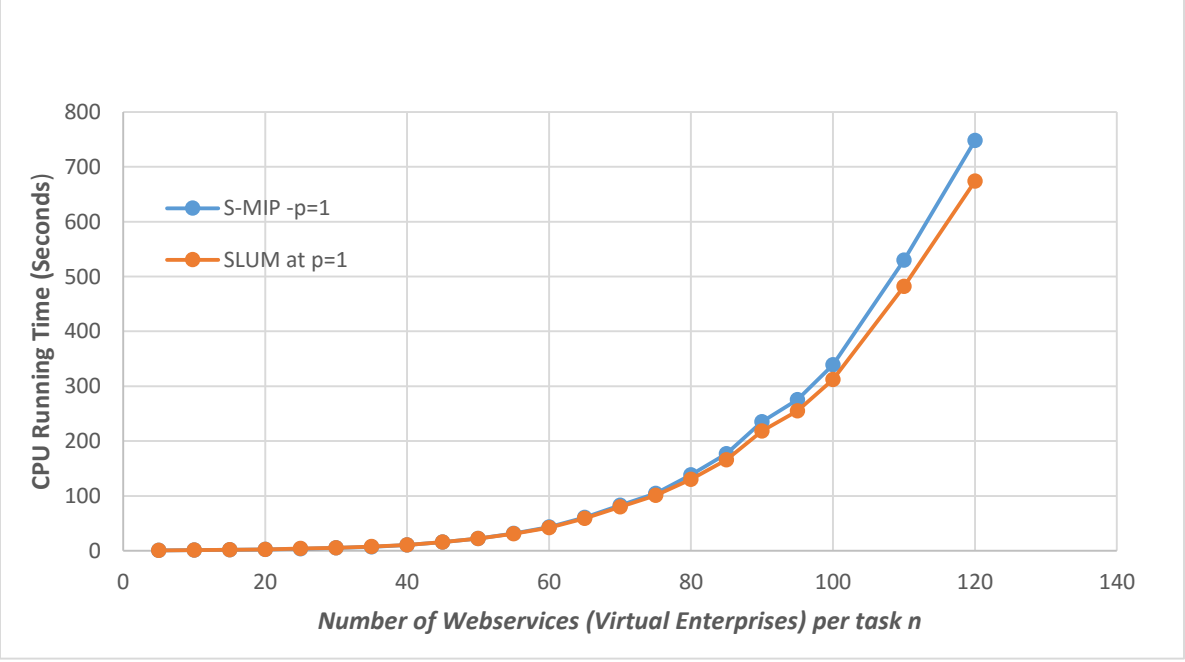


Figure 15 Empirical Running Time Growth Scatter Plot at ρ_{avg} = 0.0296

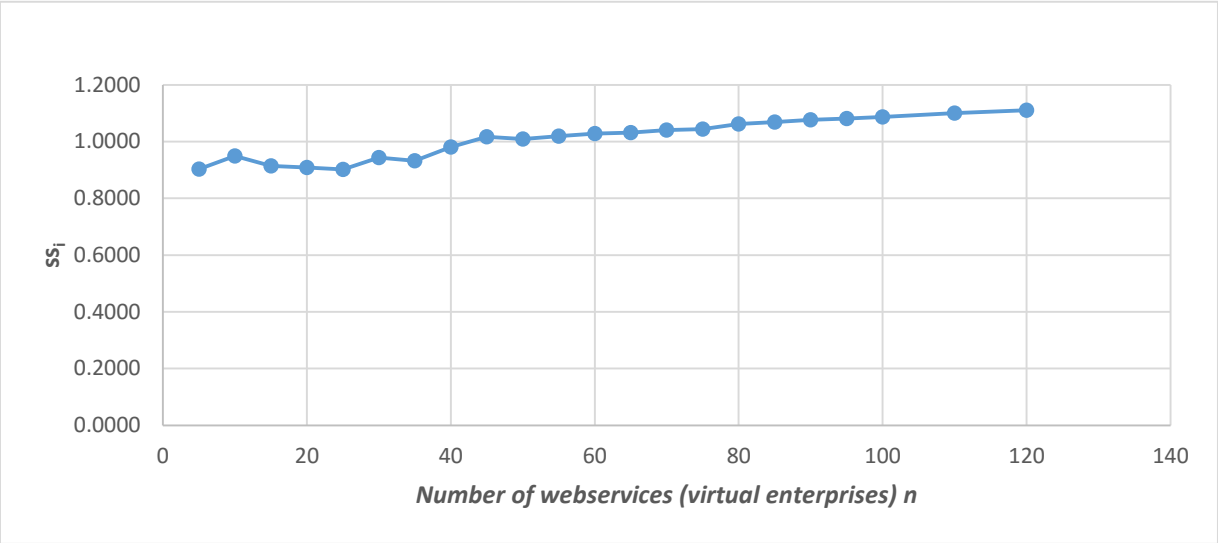


Figure 16: Speedup Growth Curve vs Number of Service Providers per workflow task at $\rho_{avg} = 1$

4.2.2 Statistical Regression Models: Linear, Polynomial & Exponential

Exponential regression of the curves in figure 15 yielded the equations $t_{eA} = 0.7793e^{0.0624n}$ at $R^2 = 0.9835$ and $t_{eB} = 0.8676e^{0.0605n}$ at $R^2 = 0.9836$. Thus we conclude that both SLUM and S-MIP exhibit exponential growth in running time. By substituting equation (10) with the two exponential equations, and setting determined n_{SSC}=60 as the minimum number of webservices that would be required for SLUM to be at least faster than S-MIP

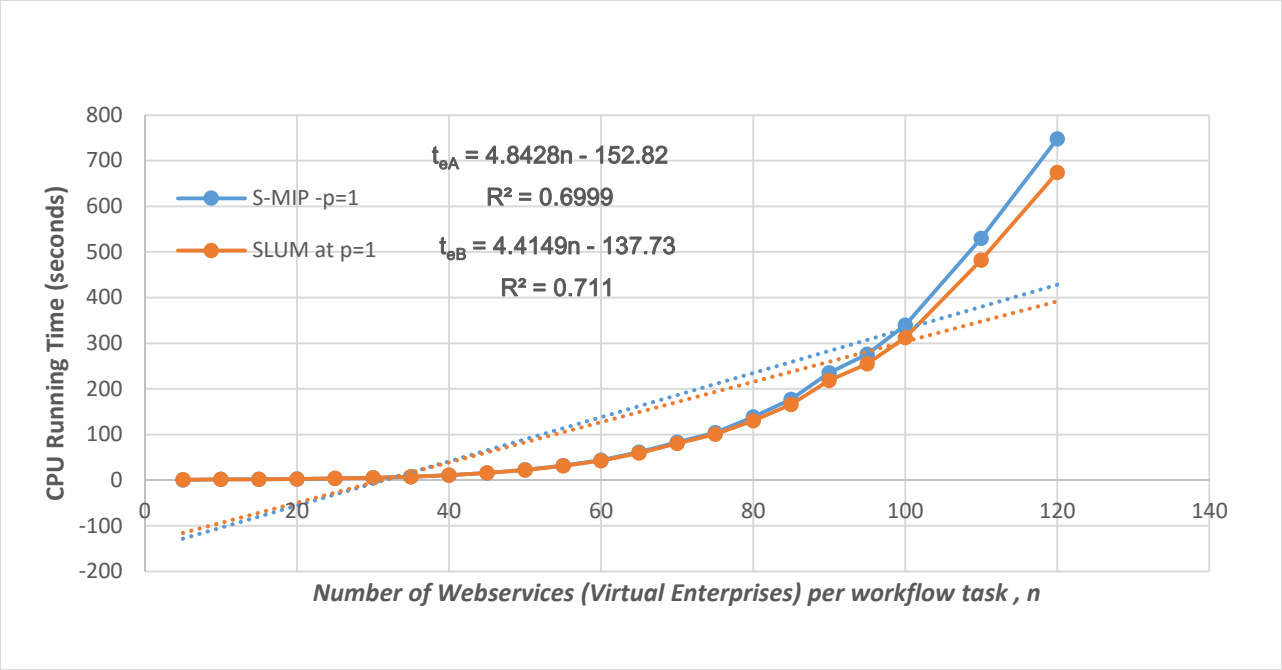


Figure 17 Empirical Running Time Growth:- Linear Regression Curves at $\rho_{avg} = 1$

Figure 17 above shows linear regression models/equations on each of the three algorithms – S-MIP, SLUM and L-MIP when the composite service phase transition rate is unity i.e when all webservices are promoted from the SCUM layer to the SPUM layer. On the other hand, figure 18 below shows the polynomial regression models for S-MIP, SLUM and L-MIP when the composite service phase transition rate is unity.

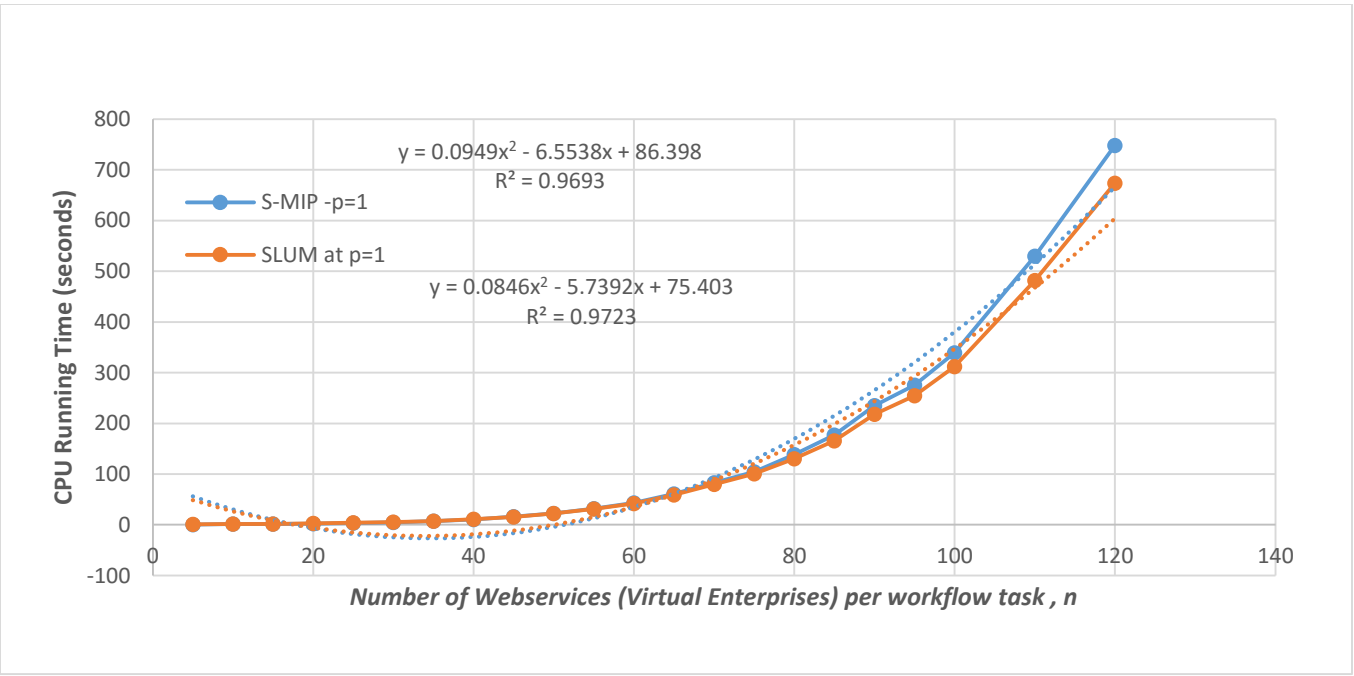


Figure 18 Empirical Running Time Growth:- Polynomial Regression Curves at $\rho_{avg} = 1$

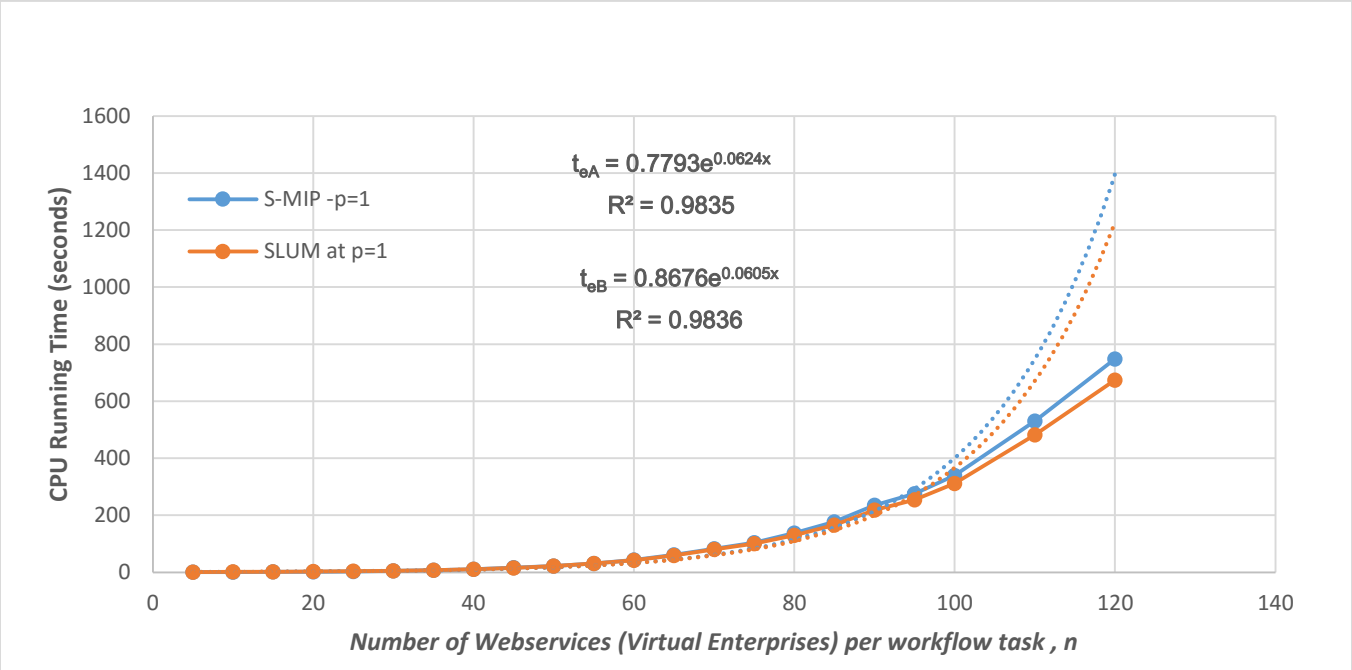


Figure 19 Empirical Running Time Growth :- Exponential Regression Curves ρ_{avg} =1

Figure 19 above shows exponential regression models equations on each of the three algorithms – S-MIP, SLUM and L-MIP when the composite service phase transition rate is unity i.e when all webservices are promoted from the SCUM layer to the SPUM layer. On the other hand, figure 18 below shows the polynomial regression models for S-MIP, SLUM and L-MIP when the composite service phase transition rate is unity.

4.2.3 *SLUM* Expected Speedup via L-Hospital's Law

Applying the L-Hospital's Law, we compute the SLUM expected speedup (SES) as per the methodology in chapter three. Using the regression equations from the figure 18 and figure 19, the SES value under polynomial growth, SES_P is determined as per equation 4.5 below and the SES function under exponential growth is given equation 4.6

$$SES_p = \lim_{n\to\infty} d(t_{eA})/d((t_{eB}) = \frac{0.0949}{0.0846} = 1.12 \qquad (4.5)$$

$$SES_E = \lim_{n\to\infty} d(t_{eA})/d((t_{eB}) = \frac{0.7793e^{0.0624n}}{0.8676e^{0.0606n}} = 0.9e^{0.0018n} \qquad (4.6)$$

The function $0.9e^{0.0018n}$ can be used to compute the expected speedup for a given number of service providers per workflow task at ρ=1. To compute, n_{CE} ,the critical value beyond which SLUM is faster than S-MIP, we could set the expected SES at 1.1 and solve the inequality in equation 4.7

$$0.9e^{0.0018n} \geq 1.1 = 0.0103nlne = \ln 1.1 \rightarrow n \geq 59 \quad (4.7)$$

Equation 4.7 means that if a virtual enterprise broker had 59 virtual enterprises per task, and the current transition rate is 1, they would enjoy a 10% increase in speedup when using SLUM instead of S-MIP.

4.2.4 *SLUM* Initial and Asymptotic Speedup via Empirical Relative Complexity Analysis

The growth behaviour of the two curves in figure 15 above hint non constant variance and nonnormality of CPU running time. Thus according to Coffin & Saltzman (2000), a variance stabilizing transformation (log transformation in this case) is required. The *log-log* scatter plot in Figure 3 being a straight line confirms the heteroskedasticity of the CPU running time. From Figure 20, we infer that the empirical relative complexity coefficient of SLUM with respect to S-MIP β_1 = 0.9684 while the constant term β_0 =1.1 and thus conclude that initially, S-MIP is 1.1 times faster than SLUM but asymptotically, SLUM is more efficient than S-MIP such that $t_{eB} = t_{eA}^{0.9684}$. This means that if for instance the running time of S-MIP is 1000 seconds, the running time of SLUM would be $1000^{0.9684} = 804$ *seconds*.

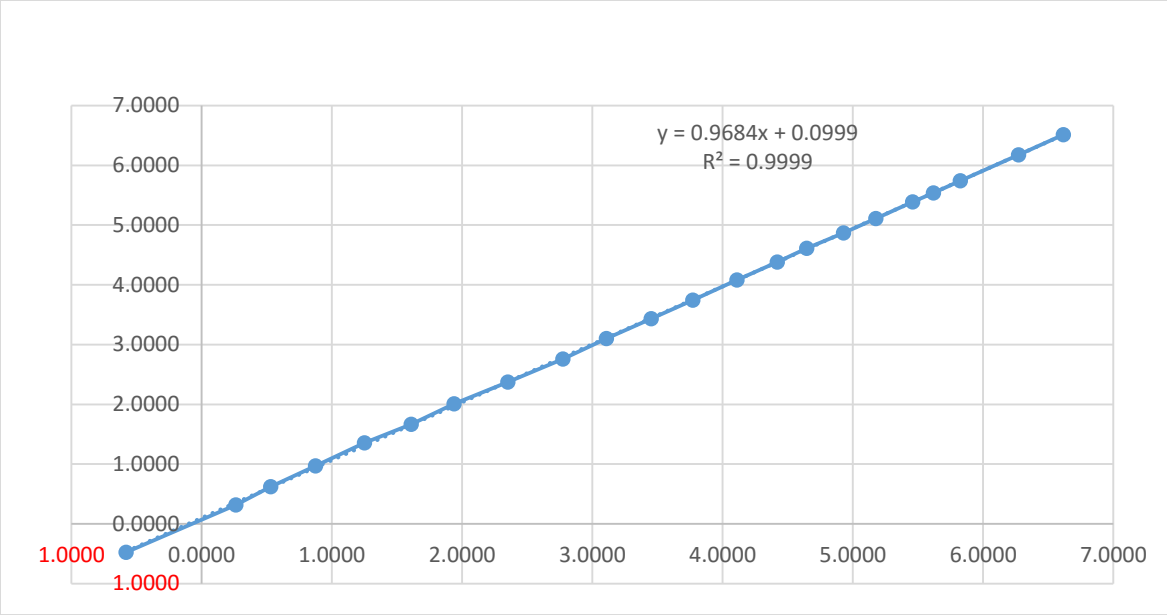


Figure 20 SLUM **Empirical Relative Complexity log-log Curve at $\rho_{avg} = 1$**

Using figure 20, we can also compute the SLUM Expected Speedup function under exponential growth. That's, by considering that over the long run, the function for computing SLUM running time as function of S-MIP running time is $t_{eB} = t_{eA}^{0.9684}$. For any $t_{eA,}$ the rate of change (speedup) of SLUM with respect to S-MIP is SES = $t_{eA} / t_{eA}^{0.9684} = t_{eA}^{0.0316}$. By plotting the function $t_{eA}^{0.0316}$ vs t_{eA} we obtain a graph showing how SES varies with t_{eA} for very large values. The graph is depicted in figure 21. The goal was to empirically estimate the limit of the SLUM expected speedup. Figure 21 reveals that the SES values are increasing with respect to t_{eA}. However, the same figure reveals that the growth in SES is not infinite, but instead seems to approach a limiting value of 2.

Table 15: Expected Relative Speedup of SLUM with respect to S-MIP for large teA values at ρ=1

teA(seconds)	**teB(seconds)**	**SES**
1000	803.8962	1.243942
5000	3820.169	1.308843
10000	7474.807	1.337827

50000	35520.78	1.407627
100000	69502.43	1.438799
500000	330280.2	1.513866
1000000	646249.2	1.547391
10000000	6008970	1.664179
50000000	28555025	1.751005
100000000		1.789781

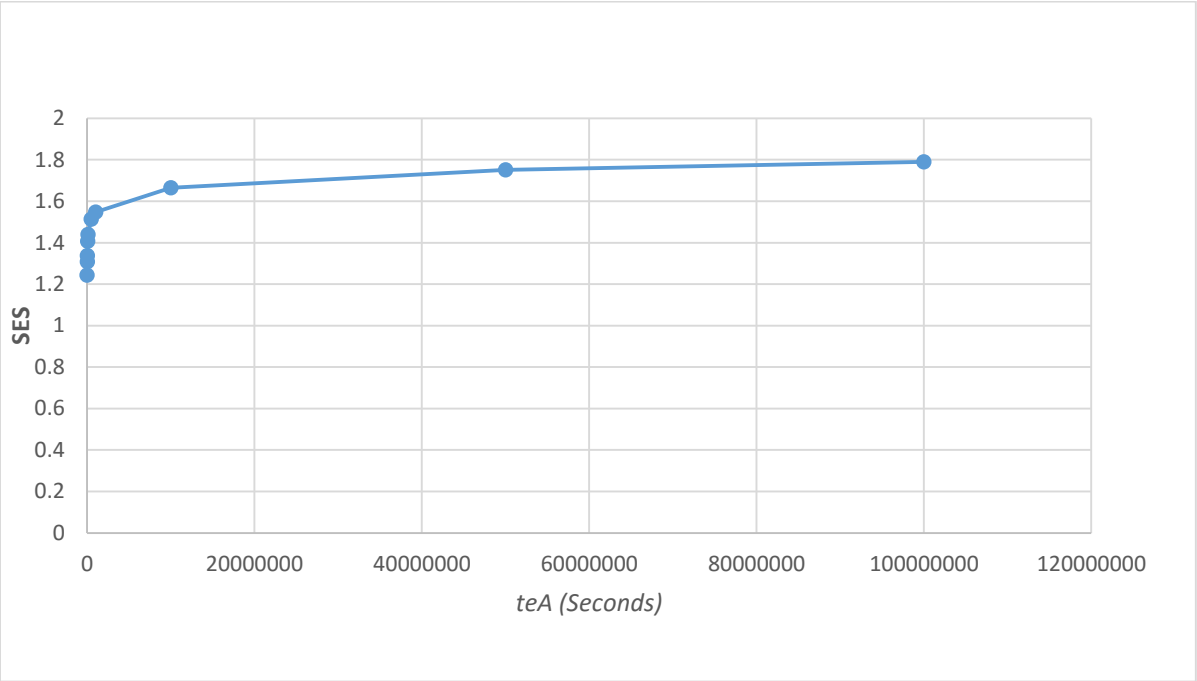


Figure 21 SLUM Expected Speedup Curve under Exponential Growth at ρ_{avg} =1

4.3 Running Time Analysis when Composite Service Phase Transition ρ=0.6

4.3.1 Running time Scaling Scatter Plots and Simple Descriptive Statistics

Table 7 below, gives a summary of the runtime results on the three algorithms- SLUM,S-MIP and L-MIP when the composite service transition rate is 0.6.

Table 16: CPU Running time Data when Phase Transition Rate ρ=0.6

n	n1	n2	g2	g1	p	teA	teB	SSI	ln teA	ln teB
5	5	5	25	25	1	0.577	0.65	0.887692	-0.54991	-0.43078
10	10	10	100	100	1	1.4	1.5	0.933333	0.336472	0.405465
15	15	15	225	225	1	1.8	1.95	0.923077	0.587787	0.667829
20	20	20	400	400	1	2.4	2.6	0.923077	0.875469	0.955511
25	25	25	625	625	1	3.4	3.9	0.871795	1.223775	1.360977
30	30	30	900	900	1	5.1	5.2	0.980769	1.629241	1.648659
35	20	38	760	1225	0.620408	7	6.9	1.014493	1.94591	1.931521
40	32	32	1024	1600	0.64	10.7	8.5	1.258824	2.370244	2.140066
45	31	32	992	2025	0.489877	15.6	11.3	1.380531	2.747271	2.424803
50	30	37	1110	2500	0.444	22.6	15.1	1.496689	3.11795	2.714695
55	31	37	1147	3025	0.379174	31.6	19.7	1.604061	3.453157	2.980619
60	33	32	1056	3600	0.293333	44	26	1.692308	3.78419	3.258097
65	30	37	1110	4225	0.262722	60.1	34	1.767647	4.09601	3.526361
70	32	27	864	490	0.17632	81	42	1.9285	4.39444	3.73767

				0	7			71	9	
75	28	35	980	5625	0.174222	106	55.6	1.906475	4.663439	4.018183
80	25	35	875	6400	0.136719	135	71	1.901408	4.905275	4.26268

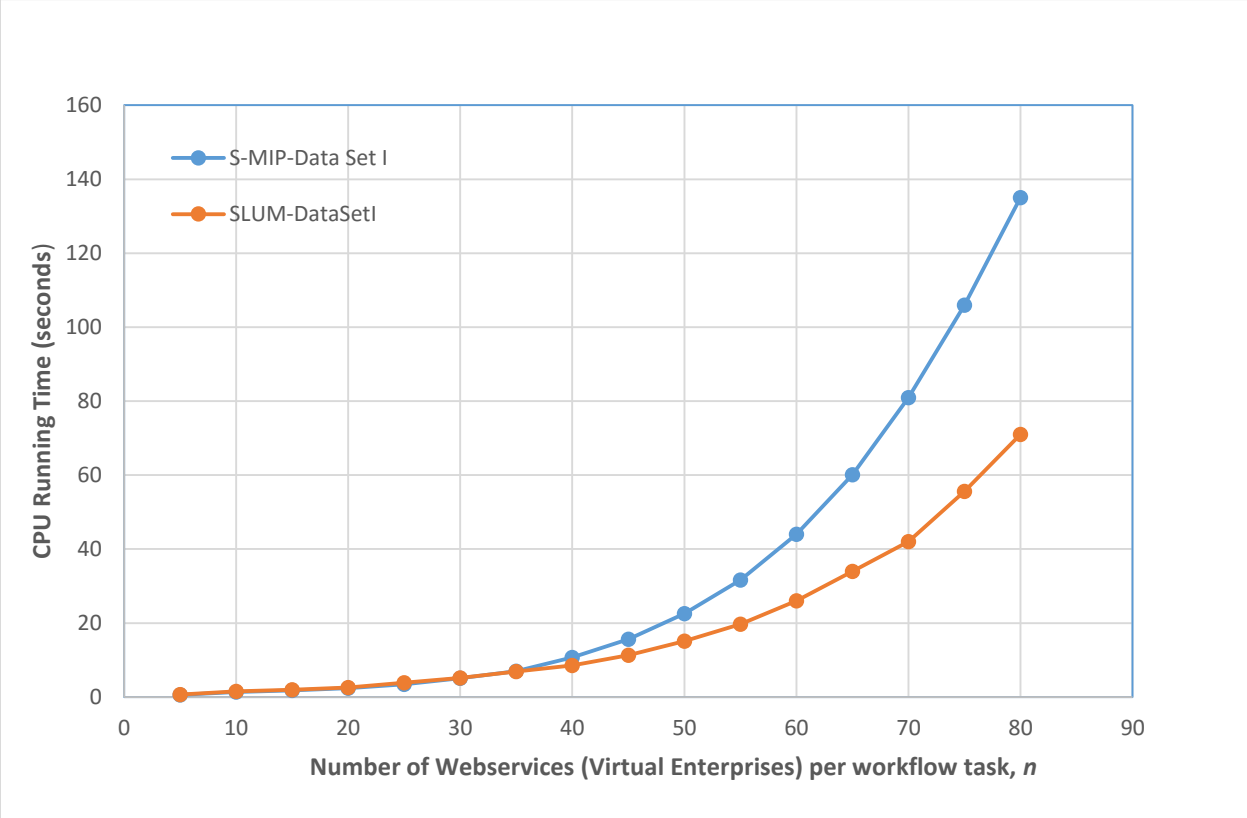


Figure 22 Empirical Running Time Growth Curves at ρ_{avg} = 0.6

Figure 22 shows the runtime growth of SLUM vs S-MIP when the composite service phase transition rate is 0.6. The red line shows the runtime curve for SLUM and the blue is the curve for S-MIP.

4.3.2 Statistical Regression Models: Linear, Polynomial & Exponential at p=0.6

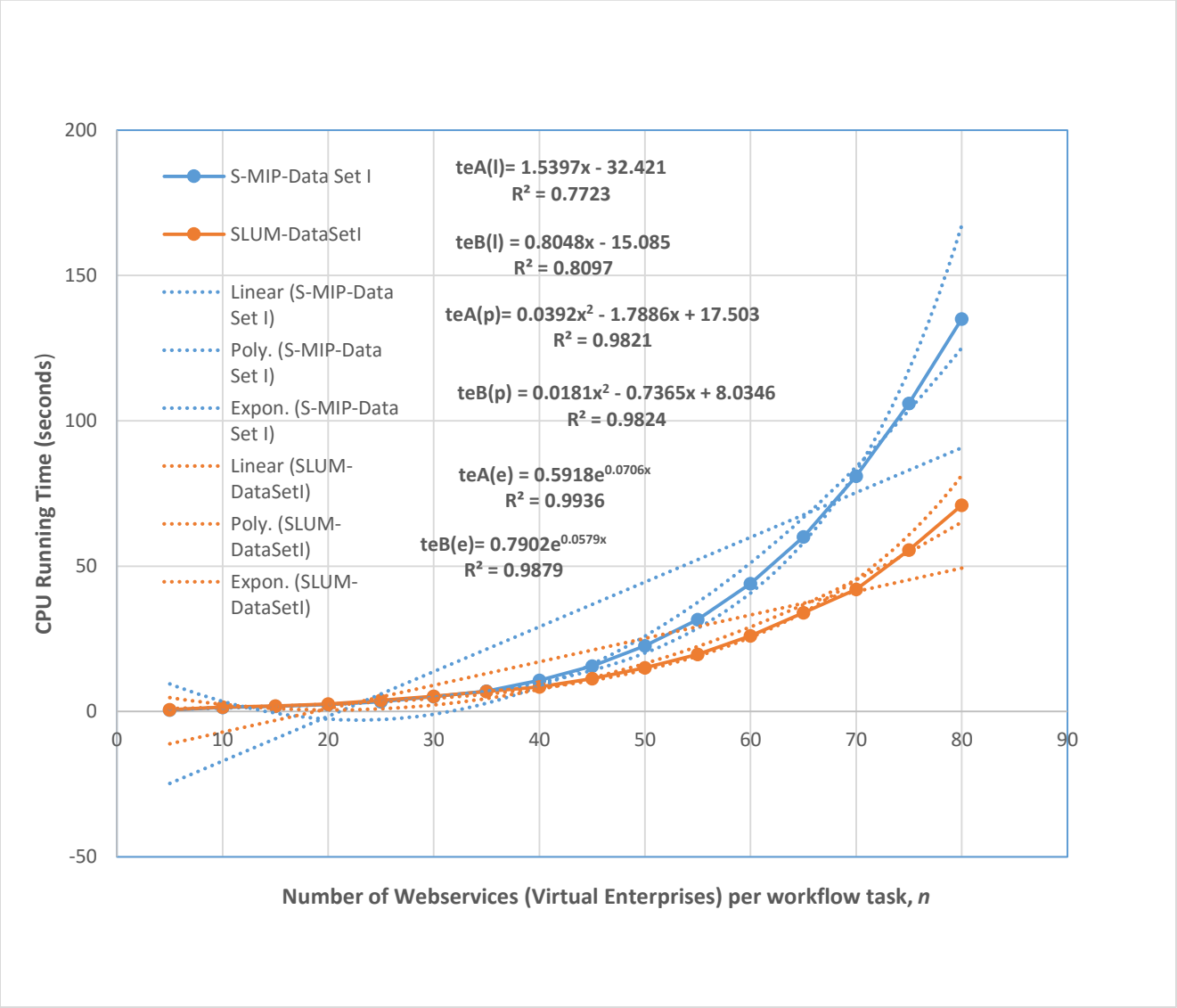


Figure 23 Empirical Running Time Growth :-Linear, Polynomial and Exponential Regression Curves at ρ_{avg} = 0.6

4.3.3 Expected Speedup via L-Hospital's Law

Applying the L-Hospital's Law, we compute the SLUM expected speedup (SES) as per the methodology in chapter three. Using the regression equations from figure 23, the SES value under polynomial growth, SES_P is determined as per equation 4.8. The SES function under exponential growth is given equation 4.9

$$SES_p = \lim_{n\to\infty} d(t_{eA})/d((t_{eB}) = \frac{0.0392}{0.0181} = 2.16 \quad (4.8)$$

$$SES_E = \lim_{n\to\infty} d(t_{eA})/d((t_{eB}) = \frac{0.5918e^{0.0706n}}{0.7902e^{0.0579n}} = 0.74e^{0.0127n} \quad (4.9)$$

The function $0.74e^{0.0127n}$ can be used to compute the expected speedup for a given number of service providers per workflow task at ρ=0.6. To compute, n_{CE}, the critical value beyond which SLUM is faster than S-MIP, we could set the expected SES at 1.1 and solve the inequality in equation 4.1

$$0.74e^{0.0127n} \geq 1.1 = 0.0094nlne = \ln 1.1 \rightarrow n \geq 10 \quad (4.10)$$

Equation 4.10 means that if a virtual enterprise broker had 10 virtual enterprises per task, and the current transition rate is 0.6, they would enjoy a 10% increase in speedup when using SLUM instead of S-MIP..

Similarly, the virtual enterprise broker could be interested to determine at what value of n they could achieve the expected speedup of 2.576 if the transition rate were 0.064. We solve the equation in 4.11

$$0.0094nlne \geq 2.16 = 0.0094nlne = \ln 2.16 \rightarrow n \geq 81 \quad (4.11)$$

4.3.4 Initial and Asymptotic Speedup via Empirical Relative Complexity Analysis

The growth behaviour of the two curves in figure 22 hint non constant variance and nonnormality of CPU running time. Thus according to Coffin & Saltzman (2000), a variance stabilizing transformation (log transformation in this case) is required. The *log-log* scatter plot in Figure above being a straight line confirms the heteroskedasticity of the CPU running time. From Figure 24, we infer that the empirical relative complexity coefficient of SLUM with respect to S-MIP at ρ=0.6, **β_1 = 0.82** while the constant term β_0 =1.2 and thus conclude that at ρ==0.6 , SLUM is initially

slower than S-MIP but asymptotically faster than S-MIP. Initially S-MIP is 1.2 times faster than S-MIP and asymptotically, SLUM's running time is given by $t_{eB} = t_{eA}^{0.82}$. This means that if for instance the running time of S-MIP is 1000 seconds, the running time of SLUM would be $1000^{0.82} = 2888$ *seconds*. This is equivalent to a speed of 3.4 times.

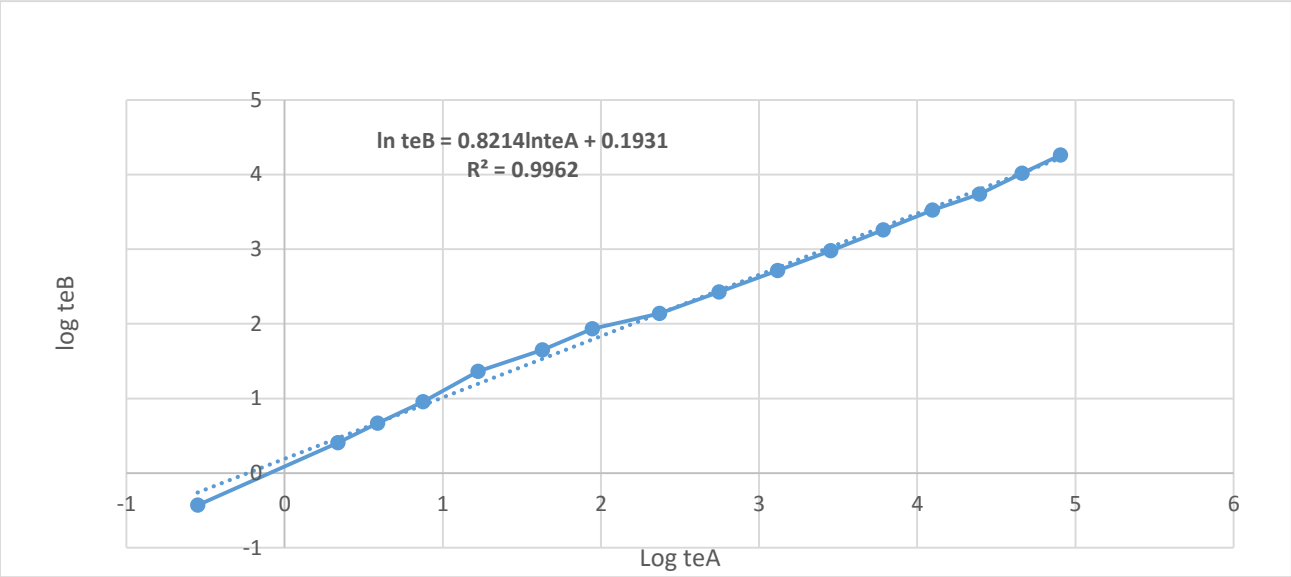


Figure 24 SLUM Empirical Relative Complexity –log-log Curve at $\rho_{avg} = 0.6$

4.4 Running Time Analysis when Composite Service Phase Transition, ρ=0.45

4.4.1 Running time Scaling Scatter Plots and Simple Descriptive Statistics

Table 8 below shows the runtime values of SLUM vs S-MIP at variable values of *n* when the composite service phase transition rate is 0.45.

Table 17: CPU Running time Data when Composite Service Phase Transition Rate ρ=0.45

n	ln(n)	TB (s)	TA (s)	ln(TA)	ln(TB)	ρ1	ρ2	g1	g2	g2/g1	Ssi
5	1.609438	0.56	0.56	-0.57982	-0.57982	3	5	25	15	0.6	1
10	2.302585	0.87	1.3	0.262364	-0.13926	10	10	100	100	1	1.494253

15	2.70805	1.35	1.72	0.542324	0.300105	2	2	225	4	0.017778	1.274074
20	2.995732	1.9	2.4	0.875469	0.641854	2	1	400	2	0.005	1.263158
25	3.218876	3.9	3.8	1.335001	1.360977	25	25	625	625	1	0.974359
30	3.401197	5.4	5.2	1.648659	1.686399	30	30	900	900	1	0.962963
35	3.555348	6.5	7.6	2.028148	1.871802	30	28	1225	840	0.685714	1.169231
40	3.688879	8.6	10.7	2.370244	2.151762	32	32	1600	1024	0.64	1.244186
45	3.806662	10.91	16.3	2.791165	2.38968	31	32	2025	992	0.489877	1.494042
50	3.912023	15.6	22.4	3.109061	2.747271	30	37	2500	1110	0.444	1.435897
55	22.04	19.7	32	3.465736	2.980619	31	37	3025	1147	0.379174	1.624365
60	4.094345	25.9	43.7	3.777348	3.254243	33	32	3600	1056	0.293333	1.687259
65	4.174387	34	61	4.110874	3.526361	30	37	4225	1110	0.262722	1.794118
70	4.248495	44.1	82.5	4.412798	3.78646	32	27	4900	864	0.176327	1.870748
75	0	55.29	107	4.672829	4.012592	28	35	5625	980	0.174222	1.93525
80		71.7	134.18	4.899182	4.272491	35	28	6400	980	0.153125	1.871409

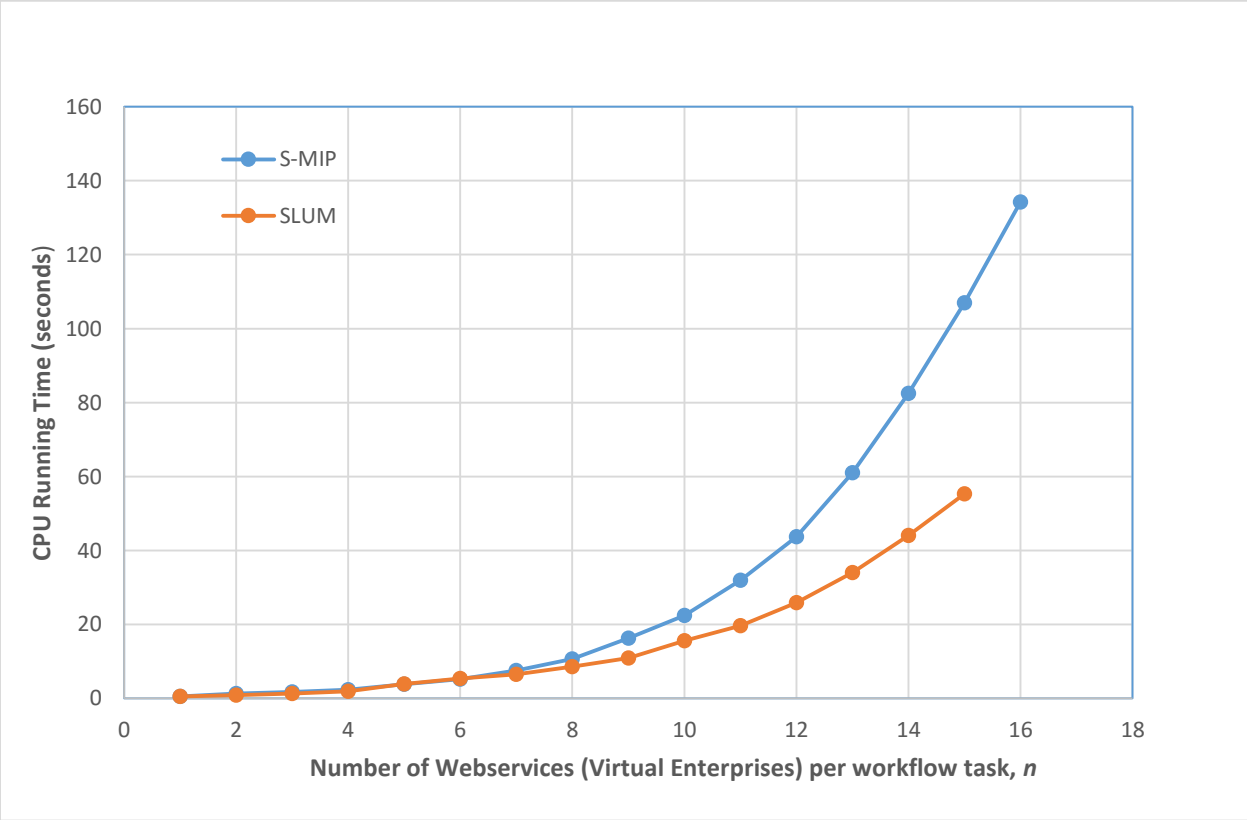


Figure 25 Empirical Running Time Growth Curves at ρ_{avg} = 0.45

Figure 25 above shows the runtime growth of SLUM vs S-MIP when the composite service phase transition rate is 0.45. The red line shows the runtime curve for SLUM and the blue is the curve for S-MIP.

4.4.2 4.4.2 Statistical Regression Models: Linear, Polynomial & Exponential at *p=0.45*

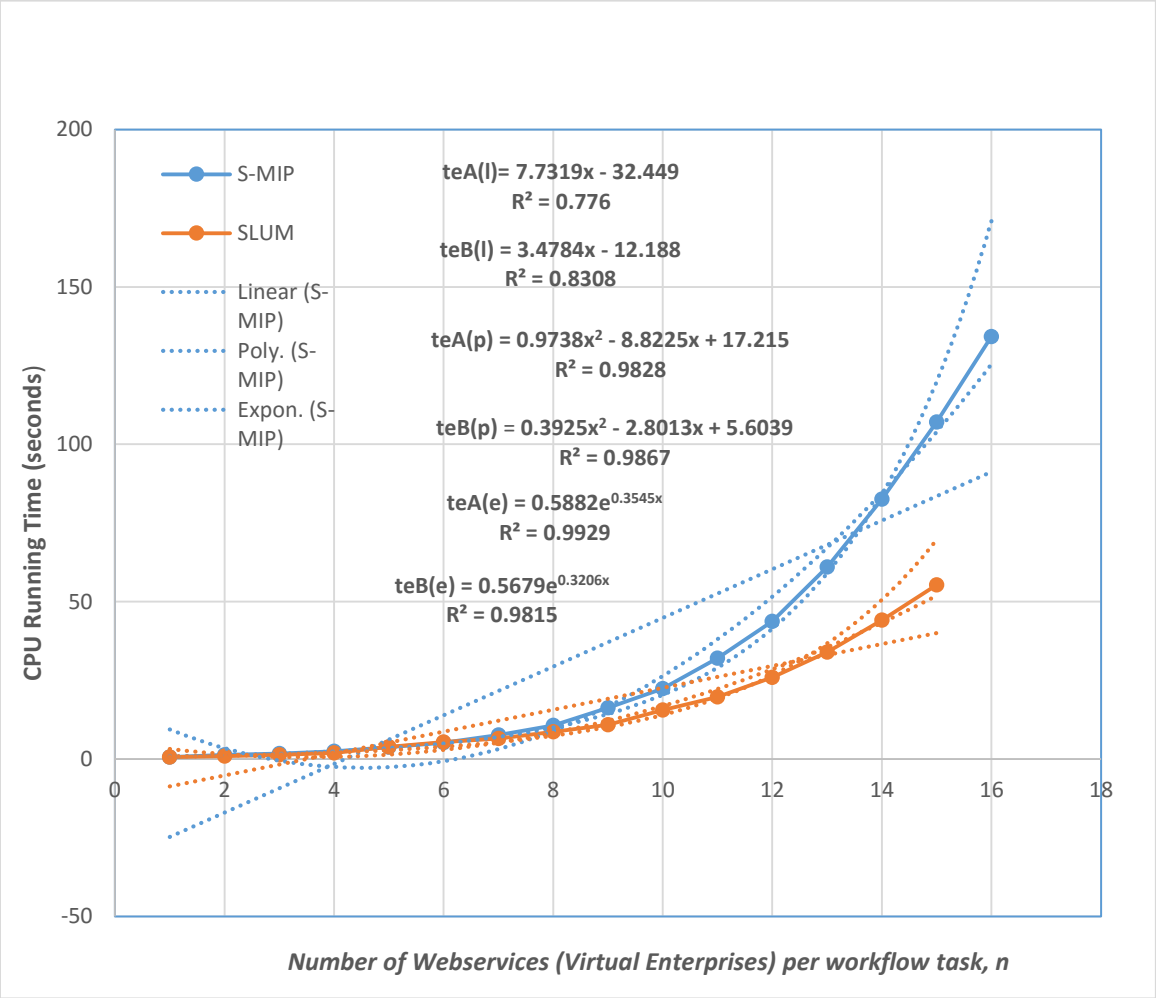


Figure 26 Empirical Running Time Growth – Linear, Polynomial and Exponential Curves at ρ_{avg} = 0.45

4.4.3 SLUM Expected Speedup via L-Hospital's Law

Applying the L-Hospital's Law, we compute the SLUM expected speedup (SES) as per the methodology in chapter three. Using the regression statistics from figure 26, the SES value under polynomial growth, SES_P is determined as per equation 4.11. The SES function under exponential growth is given equation 4.12.

$$SES_p = \lim_{n\to\infty} d(t_{eA})/d((t_{eB}) = \frac{0.09738}{0.03925} = 2.48 \quad (4.11)$$

$$SES_E = \lim_{n\to\infty} d(t_{eA})/d((t_{eB}) = \frac{0.5882e^{0.3545n}}{0.5678e^{0.3206n}} = 1.036e^{0.0339n} \quad (4.12)$$

The function $0.81e^{0.0115n}$ can be used to compute the expected speedup for a given number of service providers per workflow task at ρ=0.36. To compute, n_{CE}, the critical value beyond which SLUM is faster than S-MIP, we could set the expected SES at 1.1 and solve the inequality in equation 4.1

$$1.036e^{0.0339n} \geq 1.1 = 0.0351nlne = \ln 1.1 \rightarrow n \geq 3 \quad (4.13)$$

Equation 4.13 means that if a virtual enterprise broker had 3 virtual enterprises per task, and the current transition rate is 0.45, they would enjoy a 10% increase in speedup when using SLUM instead of S-MIP.

Similarly, the virtual enterprise broker could be interested to determine at what value of n they could achieve the expected speedup of 2.48 if the transition rate were 0.45. We solve the equation in 4.14

$$1.036e^{0.0339n} \geq 2.48 = 0.0351nlne = \ln 2.48 \rightarrow n \geq 25 \quad (4.14)$$

4.4.4 Initial and Asymptotic Speedup via Empirical Relative Complexity Analysis

The growth behaviour of the two curves in figure 25 hint non constant variance and nonnormality of CPU running time. Thus according to Coffin & Saltzman (2000), a variance stabilizing transformation (log transformation in this case) is required. The *log-log* scatter plot in figure 27 below being a straight line confirms the heteroskedasticity of the CPU running time. From Figure 27, we infer that the empirical relative complexity coefficient of SLUM with respect to S-MIP at ρ=0.45, β_1 =0.887 while the constant term β_0 =0.94 and thus conclude that at ρ=0.45 , SLUM is both initially and asymptotically faster than S-MIP. Initially SLUM is 1.06 times faster than S-MIP and asymptotically, SLUM's running time is given by $t_{eB} = t_{eA}^{0.887}$. This means that if for instance the running time of S-MIP is 1000 seconds, the running time of SLUM would be $1000^{0.887}$

= *458 seconds*. Which is equivalent to a speed of 2.1 times.

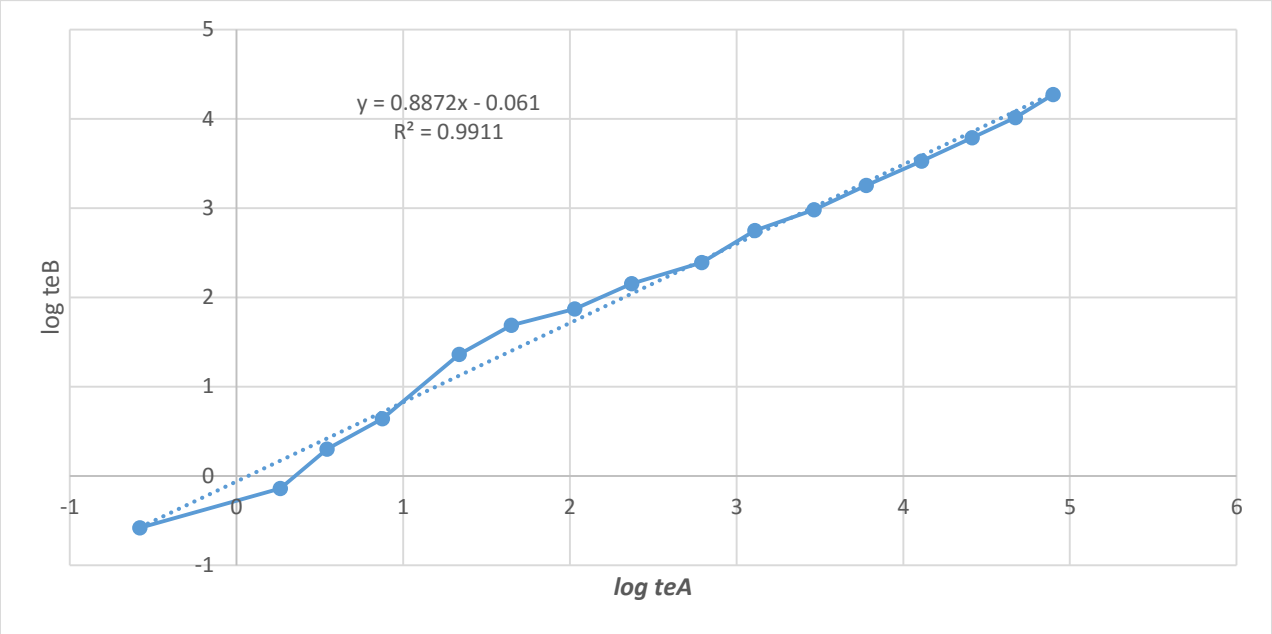


Figure 27 SLUM Empirical Relative Complexity –log-log Curve at ρ_{avg} = 0.45

4.5 Running Time Analysis when Composite Service Phase Transition, ρ=0.36

4.5.1 Running time Scaling Scatter Plots and Simple Descriptive Statistics

The data in table 9 and the figure 28 below show that the running time of both S-MIP and SLUM increases none linearly as n grows larger. We also observe that initially, SLUM lags behind S-MIP until *n*=35. Beyond *n*=35, the growth of SLUM is persistently slower than S-MIP. Beyond *n=40*, the performance differences between the two algorithms become conspicuous. We also note that the SLUM Sample Instantaneous speedup steadily increases as n grows larger. For example the SIS value at *n=35*, *n=40* , *n=50*, *n=70* is 1.17,1.24, 1.53 , 1.84 respectively.

Table 9 below shows the runtime response time values of SLUM vs S-MIP when the composite service phase transition rate at p value of 0.36. ***ρ1*** is the number of services that were eliminated

for task 1 and ***ρ2*** were the number of tasks that were eliminated for task 2. The product of ***g2 = ρ2. ρ2*** gives the number of composite services that transitioned to the SPUM layer. ***g1*** is the number of composite services available before the optimization process begins. The ration ***g2/g1*** is the composite service transition rate at a given *n*.

Table 18: CPU Running time Data when Composite Service Phase Transition Rate ρ=0.36

N	***TB (s)***	***TA (s)***	***ρ1***	***ρ2***	***g1***	***g2***	***g2/g1***	***Ssi***	***ln(TA)***	***ln(TB)***
5	0.625	0.57	3	1	25	3	0.12	0.912	-0.56212	-0.47
10	1.35	1.28	1	2	100	2	0.02	0.9481	0.24686	0.300105
15	1.89	1.88	2	2	225	4	0.0178	0.9947	0.631272	0.636577
30	5.3	5.1	30	30	900	900	1	0.9623	1.629241	1.667707
35	6.45	7.6	30	28	1225	840	0.6857	1.1783	2.028148	1.86408
40	8.6	10.7	32	32	1600	1024	0.64	1.2442	2.370244	2.151762
45	12.8	17.1	31	32	2025	992	0.4899	1.3359	2.839078	2.549445
50	15.7	24	30	37	2500	1110	0.444	1.5287	3.178054	2.753661
55	21.5	34.56	31	37	3025	1147	0.3792	1.6074	3.542697	3.068053
60	28	47.9	33	32	3600	1056	0.2933	1.7107	3.869116	3.332205
65	36.18	64	30	37	4225	1110	0.2627	1.7689	4.158883	3.588506
70	47.01	86.43	32	27	4900	864	0.1763	1.8385	4.459335	3.85036
75	59	110	28	35	5625	980	0.1742	1.8644	4.70048	4.077537

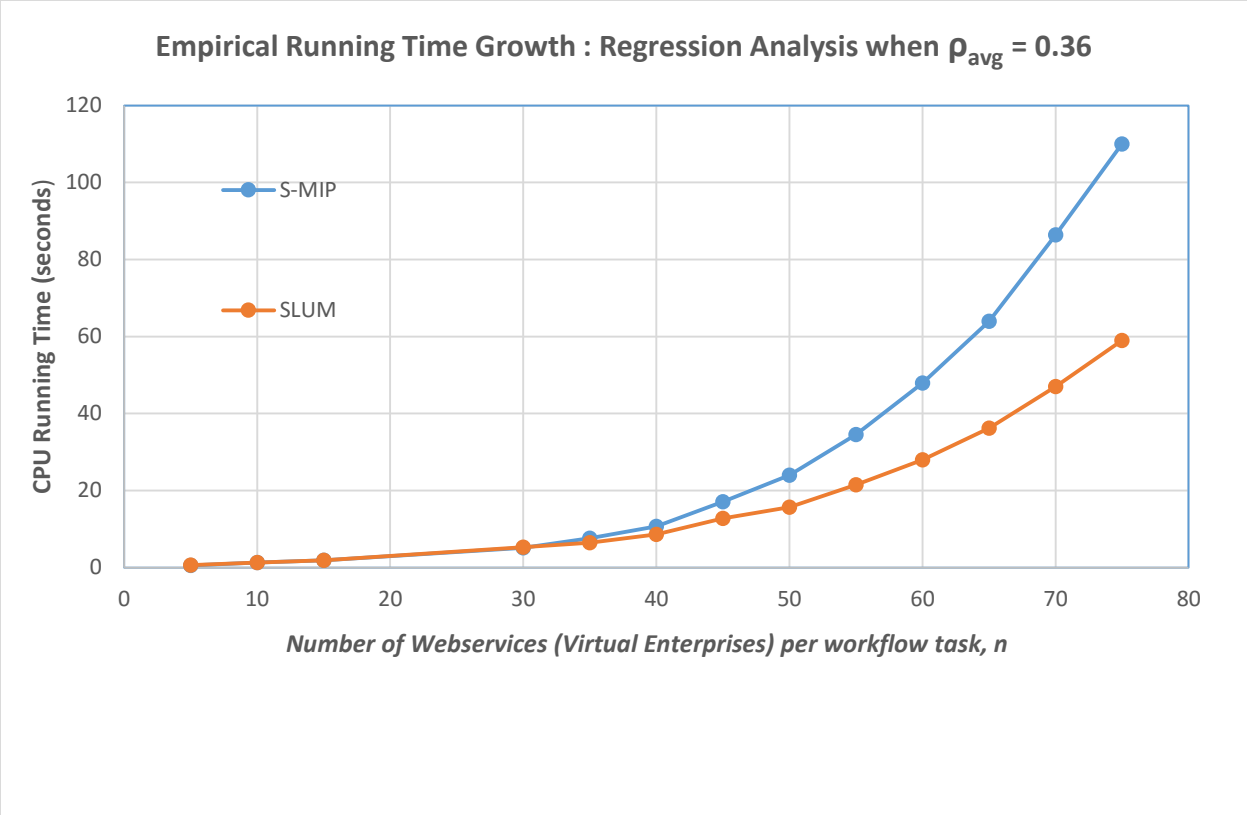


Figure 28 Empirical Running Time Growth Curves at $\rho_{avg} = 0.36$

4.5.2 Statistical Regression Models: Linear, Polynomial & Exponential

Table 19. CPU Running time Regression Statistics when Phase Transition Rate ρ=0.36

	Linear Model	*Polynomial Model*	*Exponential Model*	*R^2 (Linear)*	*R^2 (POL)*	*R^2 (EXP)*
S-MIP	1.36n - 26	$0.0369n^2 - 1.5558n + 13.018$	$0.5602e^{0.0732n}$	0.76	0.9868	0.9931
SLUM	$0.737n - 7.12.64$	$0.0173n^2 - 0.6338n + 5.8815$	$0.6882e^{0.0617n}$	0.81	0.9893	0.9881

Table 10 above shows a summary of the linear, polynomial and exponential regression models at

R² and their corresponding goodness of fit as shown by the ***R²* values**. Figure 29 below captures the same statistics graphically.

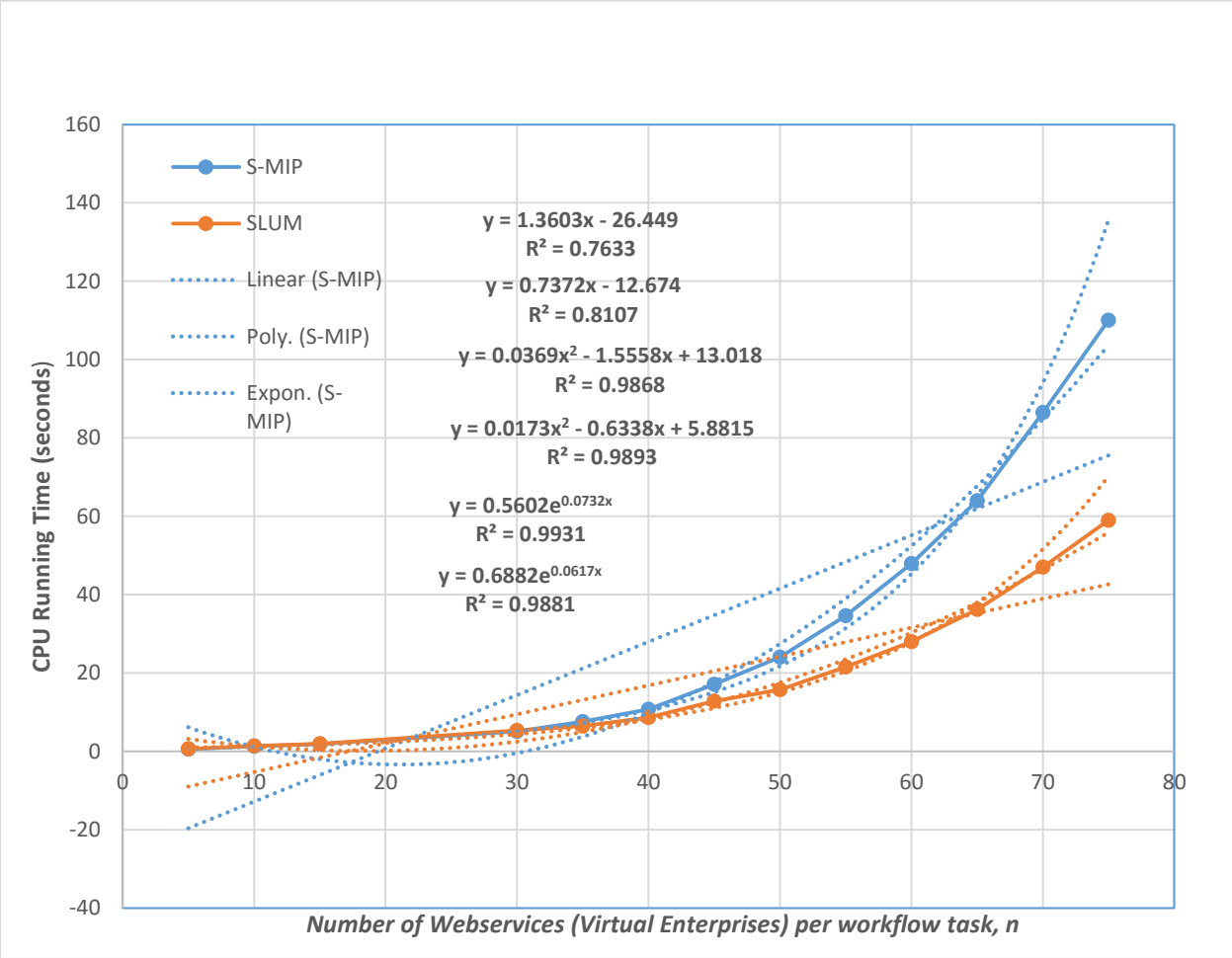


Figure 29 Empirical Running Time Growth –Linear, Polynomial and Exponential Regression Curves at ρ_{avg} = 0.36

4.5.3 Expected Speedup via L-Hospital's Law

Applying the L-Hospital's Law, we compute the SLUM expected speedup (SES) as per the methodology in chapter three. Using the regression equations from the table in 10 and figure section 29 above, the SES value under polynomial growth, SES_P is determined as per equation 4.15. The SES function under exponential growth is given equation 4.16.

$$SES_p = \lim_{n\to\infty} d(t_{eA})/d((t_{eB}) = \frac{0.0369}{0.0173} = 2.13 \quad (4.15)$$

$$SES_E = \lim_{n\to\infty} d(t_{eA})/d((t_{eB}) = \frac{0.5602e^{0.0732n}}{0.6882e^{0.0617n}} = 0.81e^{0.0115n} \quad (4.16)$$

The function $0.81e^{0.0115n}$ can be used to compute the expected speedup for a given number of service providers per workflow task at ρ=0.36. To compute, n_{CE}, the critical value beyond which SLUM is faster than S-MIP, we could set the expected SES at 1.1 and solve the inequality in equation 4.1

$$0.81e^{0.0115n} \geq 1.1 = 0.00932lne = \ln 1.1 \rightarrow n \geq 10 \quad (4.17)$$

Equation 4.17 means that if a virtual enterprise broker had 9 virtual enterprises per task, and the current transition rate is 0.36, they would enjoy a 10% increase in speedup when using SLUM instead of S-MIP.

Similarly, the virtual enterprise broker could be interested to determine at what value of n they could achieve the expected speeup of 2.13 if the transition rate were 0.36. We solve the equation in 4.18

$$0.81e^{0.0115n} \geq 2.13 = 0.00932nlne = \ln 2.13 \rightarrow n \geq 81 \quad (4.18)$$

4.5.4 Initial and Asymptotic Speedup via Empirical Relative Complexity Analysis

The growth behaviour of the two curves in figure 28 above hint non constant variance and nonnormality of CPU running time. Thus according to Coffin & Saltzman (2000), a variance stabilizing transformation (log transformation in this case) is required. The *log-log* scatter plot in Figure above being a straight line confirms the heteroskedasticity of the CPU running time. From Figure 30, we infer that the empirical relative complexity coefficient of SLUM with respect to S-MIP at ρ=0.36, β_1 = 0.8437 while the constant term β_0 =1.12 and thus conclude that at ρ=0.36 , SLUM is initially slower and asymptotically faster than S-MIP. Initially S-MIP is 1.12 times faster than S-MIP and asymptotically, SLUM's running time is given by $t_{eB} = t_{eA}^{0.8374}$. This means

that if for instance the running time of S-MIP is 1000 seconds, the running time of SLUM would be $1000^{0.8374} = 331$ *seconds*. Which is equivalent to a speed of 3.01 times.

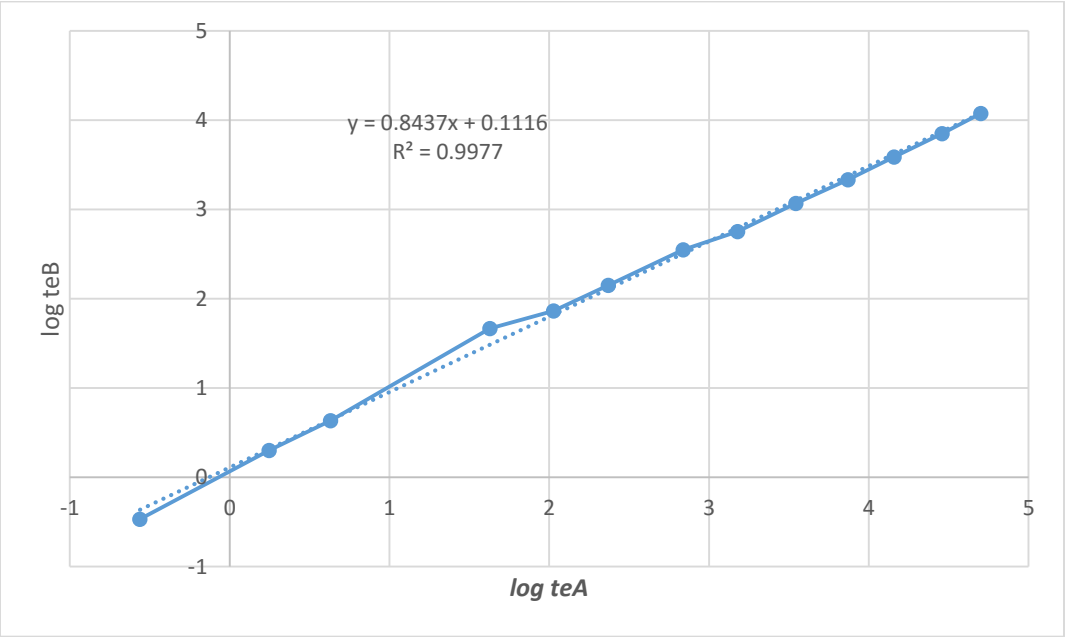


Figure 30 SLUM Empirical Relative Complexity –log-log Curve at $\rho_{avg} = 0.36$

4.6 Running Time Analysis when mean Composite Service Phase Transition, ρ=0.13

4.6.1 Running time Scaling Scatter Plots and Simple Descriptive Statistics

Table 20.: CPU Running time Data when Composite Service Phase Transition Rate ρ=0.13

n	TB (s)	TA (s)	ρ1	ρ2	g1	g2	g2/g1	Ssi	ln(TA)	ln(TB)
5	0.67	0.58	5	5	25	25	1	0.865672	-0.54473	-0.40048
10	1.25	1.39	5	7	100	35	0.35	1.112	0.329304	0.223144

15	1.76	2.1	7	8	225	56	0.248889	1.193182	0.741937	0.565314
20	2.25	2.75	8	4	400	32	0.08	1.222222	1.011601	0.81093
25	2.7	3.6	11	7	625	77	0.1232	1.333333	1.280934	0.993252
30	3.73	5.2	7	6	900	42	0.046667	1.394102	1.648659	1.316408
35	5.1	7.2	9	7	1225	63	0.051429	1.411765	1.974081	1.629241
40	6.3	10.99	4	6	1600	24	0.015	1.744444	2.396986	1.84055
45	8.88	16.13	10	6	2025	60	0.02963	1.816441	2.780681	2.183802
50	12.7	23.1	8	5	2500	40	0.016	1.818898	3.139833	2.541602
55	17.28	31.7	9	5	3025	45	0.014876	1.834491	3.456317	2.84955
60	23.1	44.4	7	6	3600	42	0.011667	1.922078	3.793239	3.139833
65	31	60.3	4	9	4225	36	0.008521	1.945161	4.099332	3.433987
70	42.9	85	4	7	4900	28	0.005714	1.98044	4.394449	3.71113
75	53.8	107	6	4	5625	24	0.004267	1.981352	4.672829	3.985273
80	68.7	140	5	6	6400	30	0.004688	2.037846	4.919981	4.229749

The data in table 11 above shows that the running time of both S-MIP and SLUM increases none linearly as n grows larger. We also observe that initially, SLUM lags behind S-MIP until n=10. Beyond n=10, the growth of SLUM is persistently slower than S-MIP. Beyond *n=20,* the performance differences between the two algorithms become conspicuous. We also note that the SLUM Sample Instantaneous speedup steadily increases as n grows larger. For example the SIS value at *n=5, n=10* , *n=50, n=70* is 0.86,1.1, 1.8 , 1.98 respectively.

4.6.2 Statistical Regression Models: Linear, Polynomial & Exponential

Table 12 below shows the S-MIP, L-MIP and SLUM linear, polynomial and exponential regression equations and the goodness of fit statistics captured through the R^2 values.

Table 21: CPU Running time Regression Statistics when Composite Service Phase Transition Rate ρ=0.13

	Linear Model	***Polynomial Model***	***Exponential Model***	R^2 ***(Linear)***	R^2 ***(POL)***	R^2 ***(EXP)***
S-MIP	1.59n – 33	$0.0405\ n^2 - 1.1849n + 10.864$	$0.634e^{0.0699n}$	0.76	0.98	0.99
SLUM	0.779n - 15	$0.092n^2 - 0.3843n + 3.7664$	$0.628e^{0.0597n}$	0.78	0.98	0.99

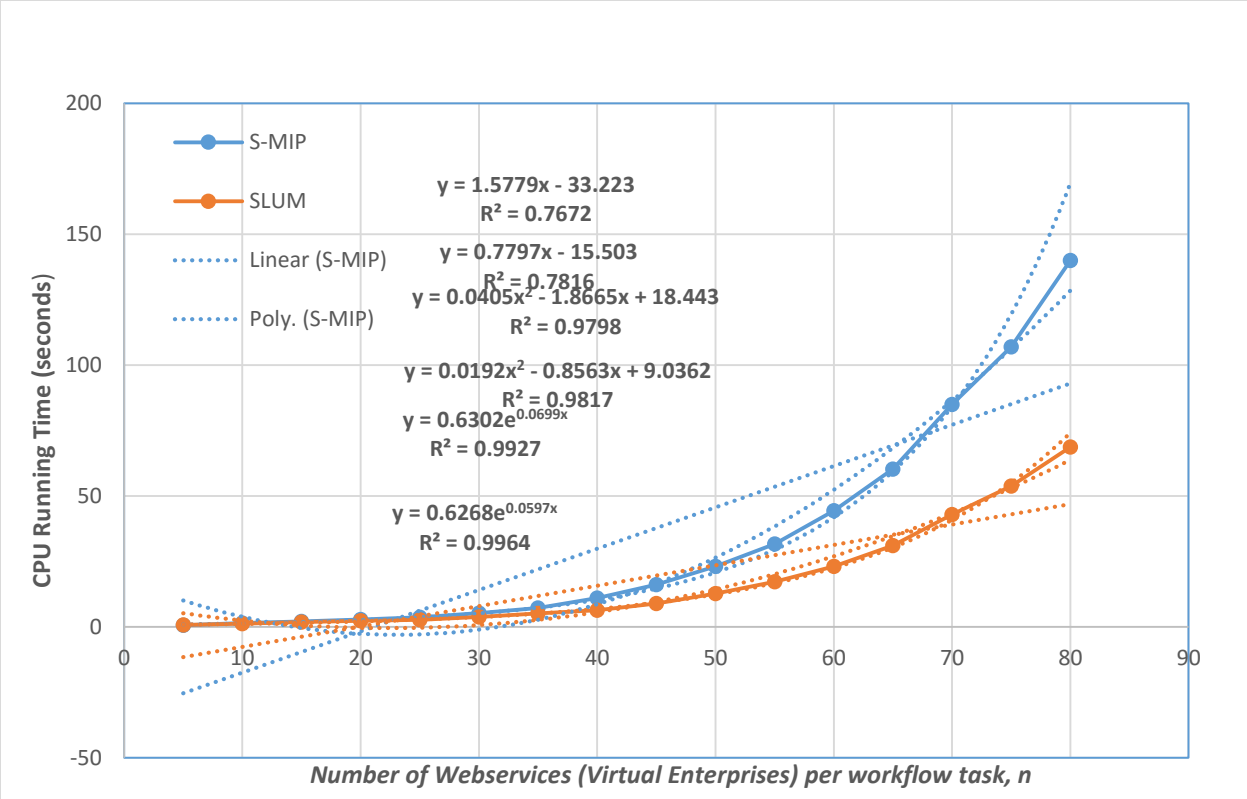


Figure 31 Empirical Running Time Growth – Linear, Polynomial and Exponential Regression Curves at ρ_{avg} = 0.13

Figure 31 shows the linear, polynomial and exponential statistical regression models for S-MIP and L-MIP at ρ=0.13.

4.6.3 Expected Speedup via L-Hospital's Law under

Applying the L-Hospital's Law, we compute the SLUM expected speedup (SES) as per the methodology in chapter three. Using the regression equations from table 12 and figure 31., the SES value under polynomial growth, SES_P is determined as per equation 4.19. The SES function under exponential growth is given equation 4.20.

$$SES_p = \lim_{n\to\infty} d(t_{eA})/d((t_{eB}) = \frac{0.0405}{0.0192} = 2.1 \quad (4.19)$$

$$SES_E = \lim_{n\to\infty} d(t_{eA})/d((t_{eB}) = \frac{0.634e^{0.0699n}}{0.628e^{0.0597n}} = 1.01e^{0.0102n} \quad (4.20)$$

The function $1.01e^{0.0102n}$ can be used to compute the expected speedup for a given number of service providers per workflow task at ρ=0.13. To compute, n_{CE} ,the critical value beyond which

SLUM is faster than S-MIP, we could set the expected SES at 1.1 and solve the inequality in equation 4.21

$$1.01e^{0.0102n} \geq 1.1 = 0.0103nlne = \ln 1.1 \rightarrow n \geq 9 \quad (4.21)$$

Equation 4.21 means that if a virtual enterprise broker had 9 virtual enterprises per task, and the current transition rate is 0.13, they would enjoy a 10% increase in speedup when using SLUM instead of S-MIP. Examining the data in table 1 above, we that at n=5, the speedup is 0.8 and at n=10, the speedup is 1.32. Therefore a speedup of 1.1 should lie in between n=5 and n=10. Thus the solution to equation 4.21 holds.

Similarly, the virtual enterprise broker could be interested to determine at what value of n they could achieve the expected speedup of 2.1 if the transition rate were 0.13. We solve the equation in 4.22

$$1.01e^{0.01n} \geq 2.1 = 0.0103nlne = 2.1 \rightarrow n \geq 72 \quad (4.22)$$

4.6.4 Initial and Asymptotic Speedup via Empirical Relative Complexity

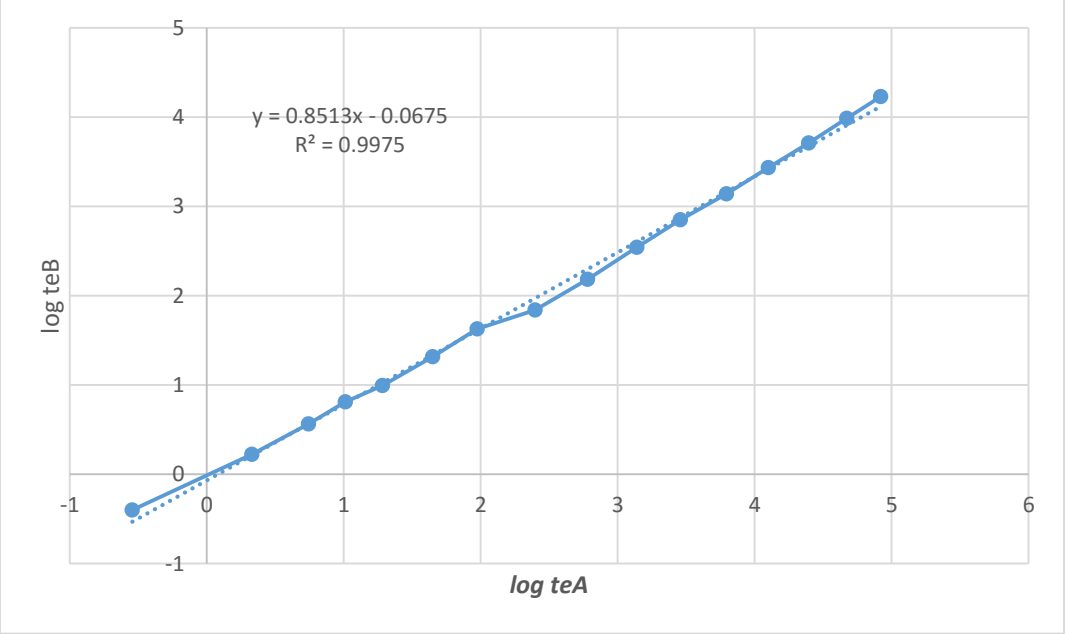


Figure 32 SLUM Empirical Relative Complexity –log-log Curve at ρ_{avg} = 0.13

The growth behaviour of the two curves in figure 31 hint non constant variance and nonnormality of CPU running time. Thus according to Coffin & Saltzman (2000), a variance stabilizing transformation (log transformation in this case) is required. The *log-log* graph in figure 32 above being a straight line confirms the heteroskedasticity of the CPU running time. From Figure 32, we infer that the empirical relative complexity coefficient of SLUM with respect to S-MIP at ρ=0.13, β_1 = 0.851 while the constant term β_0 =0.93 and thus conclude that at ρ=0.13 , SLUM is both initially and asymptotically faster than S-MIP. Initially SLUM is 1.07 times faster than S-MIP and asymptotically, SLUM's running time is given by $t_{eB} = t_{eA}^{0.851}$. This means that if for instance the running time of S-MIP is 1000 seconds, the running time of SLUM would be $1000^{0.851} = 357$ *seconds*. Which is equivalent to a speed of 2.8 times.

4.7 Running Time Analysis when mean Composite Service Phase Transition, ρ=0.064

4.7.1 Running time Scaling Scatter Plots and Simple Descriptive Statistics

Table 22: CPU Running time Data when Composite Service Phase Transition Rate ρ=0.064

n	n1	n2	g2	g1	p	teA	teB	SSI	ln teA	ln teB
5	3	4	12	25	0.48	0.577	0.65	0.887692	-0.54991	-0.43078
10	2	4	8	100	0.08	1.3	0.98	1.326531	0.262364	-0.0202
15	2	2	4	225	0.017778	1.8	1.39	1.294964	0.587787	0.329304
20	6	1	6	400	0.015	2.7	2.044	1.320939	0.993252	0.714909
25	2	3	6	625	0.0096	3.7	2.3	1.608696	1.308333	0.832909
30	3	3	9	900	0.01	5.2	3.4	1.529412	1.648659	1.223775
35	2	3	6	1225	0.004898	7.1	4.6	1.543478	1.960095	1.526056
40	2	4	8	1600	0.005	10.7	6.3	1.698413	2.370244	1.84055
45	2	5	10	2025	0.004938	16.3	9	1.811111	2.791165	2.197225
50	6	4	24	2500	0.0096	22.7	12.2	1.860656	3.122365	2.501436
55	2	4	8	3025	0.002645	31.6	17.4	1.816092	3.453157	2.85647
60	2	3	6	3600	0.001667	44	22.8	1.929825	3.78419	3.126761
65	30	37	1110	4225	0.262722	60.1	29	2.072414	4.09601	3.367296
70	7	4	28	4900	0.005714	81	34	2.382353	4.394449	3.526361

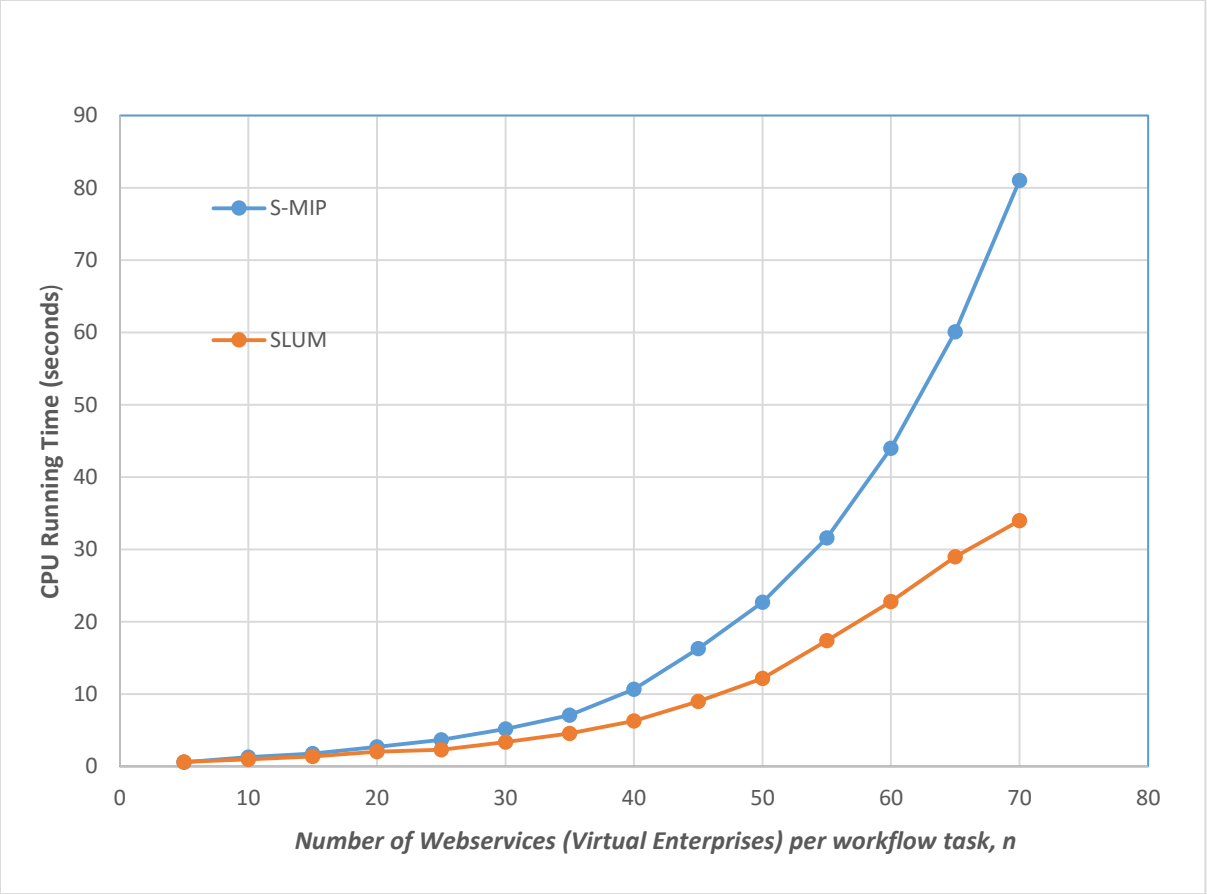


Figure 33 Empirical Running Time Growth Curves at $\rho_{avg} = 0.064$

The data in the table and the figure shows that the running time of both S-MIP and SLUM increases none linearly as n grows larger. We also observe that initially, SLUM lags behind S-MIP until n=10. Beyond n=10, the growth of SLUM is persistently slower than S-MIP. Beyond *n=20*, the performance differences between the two algorithms become conspicuous. We also note that the SLUM Sample Instantaneous speedup steadily increases as n grows larger. For example the SIS value at *n=5*, *n=10* , *n=50*, *n=70* is 0.887,1.32, 1.5 , 2.38 respectively.

4.7.2 Statistical Regression Models: Linear, Polynomial & Exponential

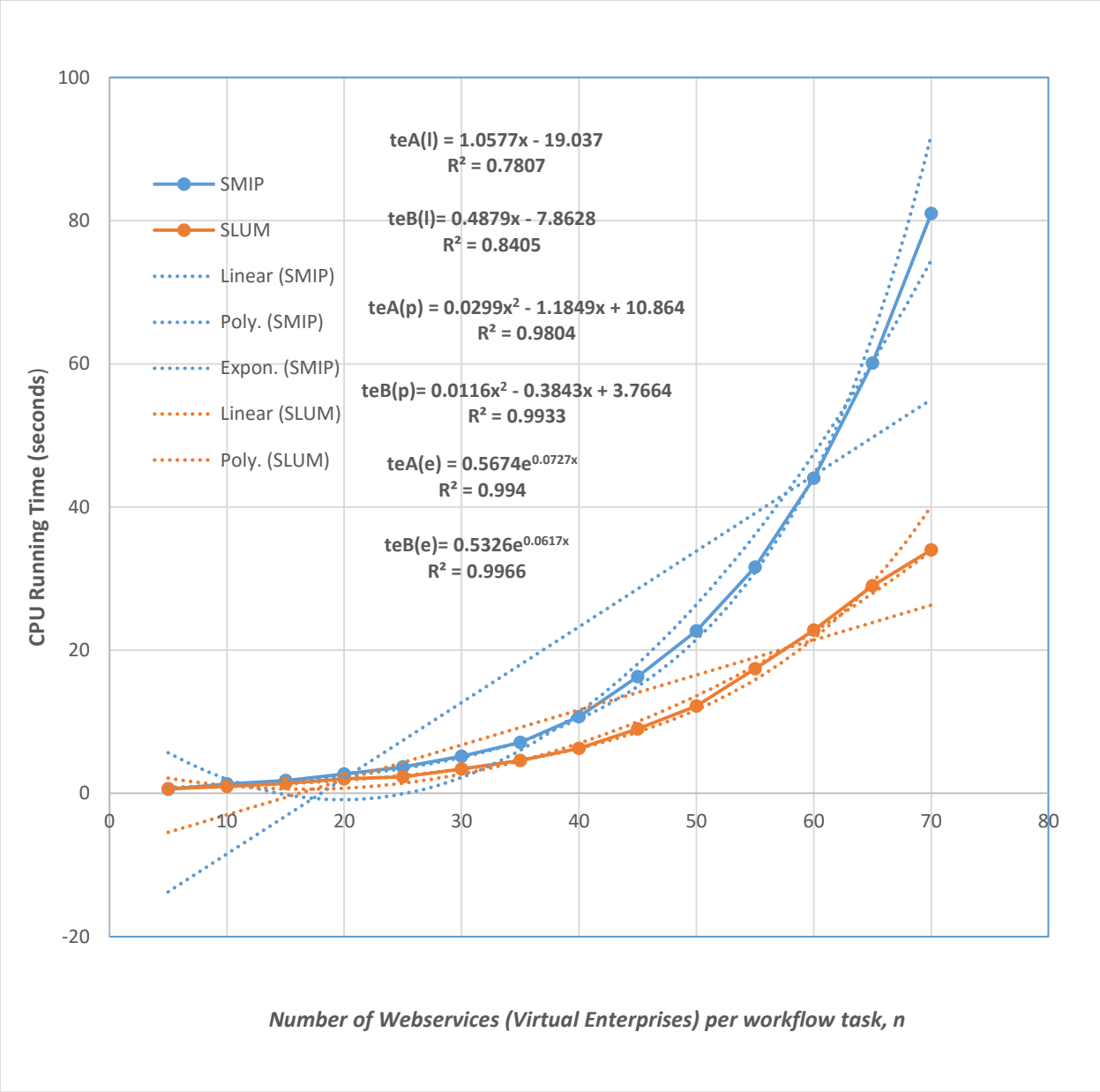


Figure 34 Empirical Running Time Growth- Linear, Polynomial and Exponential Regression Curves at ρ_{avg} = 0.064

The figure 34 above and the regression equations in table 14 below show that the growth curves of both SLUM and S-MIP at ρ=0.064 has near perfect polynomial and a near perfect exponential growth. However, while S-MIP has no linear growth since the R^2 value for linear regression is below 0.8 for S-MIP, SLUM exhibits some linear growth characteristics since R^2=*0.84.*

Table 23: CPU Running time Regression Statistics when Composite Service Phase Transition Rate ρ=0.064

	Linear Model	*Polynomial Model*	*Exponential Model*	*R^2 (LN)*	*R^2 (POL)*	*R^2 (EXP)*
S-MIP	1.0577n - 19.037	0.0299 n^2 - 1.1849n + 10.864	$0.5674e^{0.0727n}$	0.7807	0.9804	0.994
SLUM	0.4879n - 7.8628	$0.0116n^2$ - 0.3843n + 3.7664	$0.5326e^{0.0617n}$	0.8405	0.9933	0.9966

4.7. 3 Expected Speedup via L-Hospital's Law

Applying the L-Hospital's Law, we compute the SLUM expected speedup (SES) as per the methodology in chapter three. Using the regression equations from the table 14 , the SES value under polynomial growth, SES_P is determined as per equation 4.23. The SES function under exponential growth is given equation 4.24.

$$SES_p = \lim_{n\to\infty} d(t_{eA})/d((t_{eB}) = \frac{0.0299}{0.0116} = 2.5776 \quad (4.23)$$

$$SES_E = \lim_{n\to\infty} d(t_{eA})/d((t_{eB}) = \frac{0.5674e^{0.0727n}}{0.5326e^{0.0617n}} = 1.065e^{0.01n} \quad (4.24)$$

The function $1.065e^{0.01n}$ can be used to compute the expected speedup for a given number of service providers per workflow task at ρ=0.064. To compute, n_{CE} ,the critical value beyond which SLUM is faster than S-MIP, we could set the expected SES at 1.1 and solve the inequality in equation 4.25

$$1.065e^{0.01n} \geq 1.1 = 0.01065nlne = \ln 1.1 \rightarrow n \geq 9 \quad (4.25)$$

Equation 4.25 means that if a virtual enterprise broker had 9 virtual enterprises per task, and the current transition rate is 0.064, they would enjoy a 10% increase in speedup when using SLUM instead of S-MIP. Examining the data in table 1 above, we that at n=5, the speedup is 0.8 and at n=10, the speedup is 1.32. Therefore a speedup of 1.1 should lie in between n=5 and n=10. Thus the solution to equation 4.25 holds.

Similarly, the virtual enterprise broker could be interested to determine at what value of n they could achieve the expected speedup of 2.576 if the transition rate were 0.064. We solve the equation in 4.26.

$$1.065e^{0.01n} \geq 2.576 = 0.01065nlne = \ln 2.576 \rightarrow n \geq 88 \quad (4.26)$$

4.7.4 Initial and Asymptotic Speedup via Empirical Relative Complexity Analysis

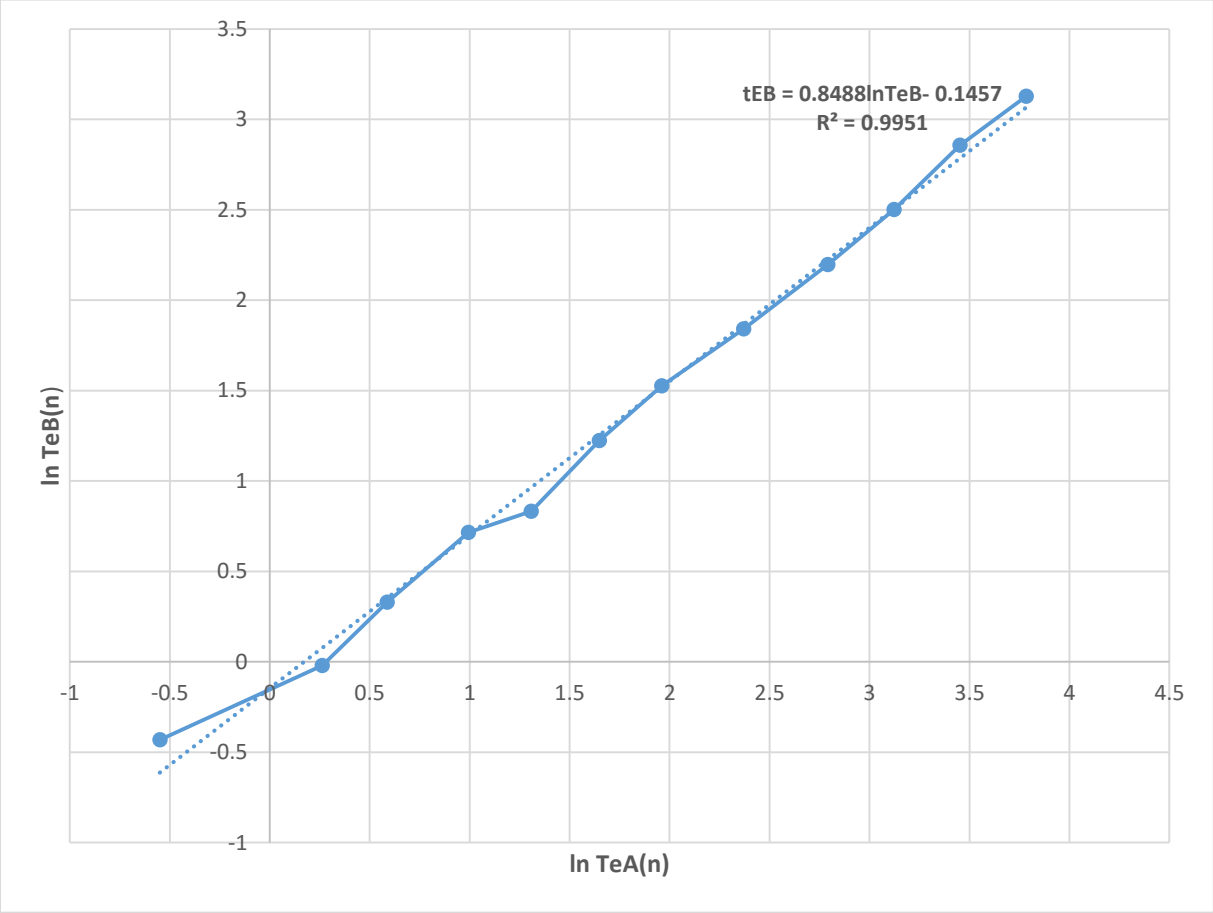


Figure 35 SLUM Empirical Relative Complexity –log-log Curve at ρ_{avg} = 0.064

The growth behaviour of the two curves in figure 34 above hint non constant variance and nonnormality of CPU running time. Thus according to Coffin & Saltzman (2000), a variance stabilizing transformation (log transformation in this case) is required. The *log-log* scatter plot in Figure above being a straight line confirms the heteroskedasticity of the CPU running time. From

Figure 35, we infer that the empirical relative complexity coefficient of SLUM with respect to S-MIP at ρ=0.064, β_1 = 0.84 while the constant term β_0 =0.88 and thus conclude that at ρ=0.064 , SLUM is both initially and asymptotically faster than S-MIP. Initially SLUM is 1.15 times faster than S-MIP and asymptotically, SLUM's running time is given by $t_{eB} = t_{eA}^{0.84}$. This means that if for instance the running time of S-MIP is 1000 seconds, the running time of SLUM would be $1000^{0.84} = 331$ *seconds*. Which is equivalent to a speed of 3.01 times.

4.8 Summary of Key CPU Running Time Results

4.8.1 Variation of Running Time vs Number of Service Providers under the various ρ values

Figure 36 below shows that generally the smaller the transition rate the slower the growth in running time and the better the performance.

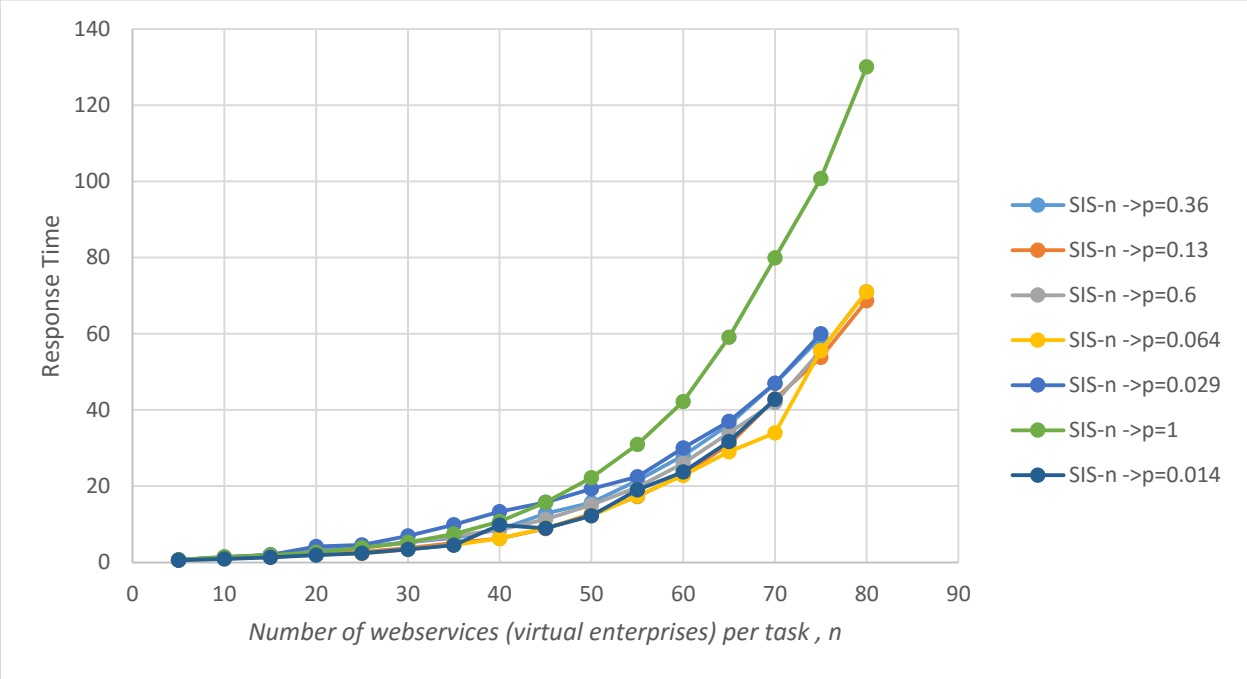


Figure 36 Summary -SLUM Running Time Growth at different composite service phase transition rates

4.8.2 Variation of Speedup vs Number of Service Providers per task under the various ρ values

In the legend on of the graph in figure 37 below, *SIS* stands for 'SLUM Instantaneous Speedup' as defined earlier in chapter 3. *SIS-n* means the SIS value at given value of *n*, whereas the symbol ρ carries the usual meaning as earlier defined in section 2.14.

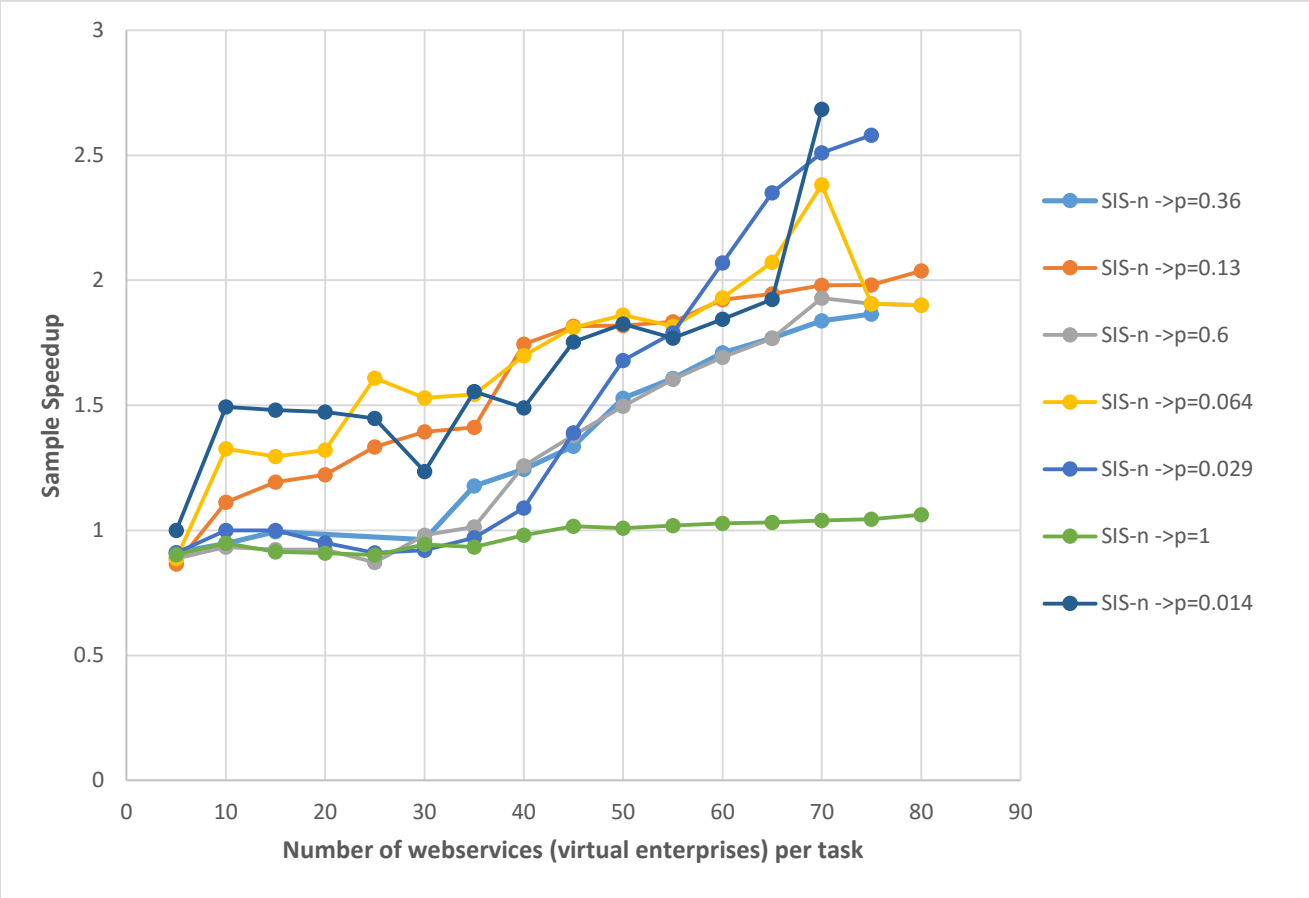


Figure 37 Summary -SLUM Speedup vs Number of Service Providers per Task at different phase transition rates

In figure 37, we plot the speedup that was observed against increasing number of service providers per task at a given transition rate. The following can be observed. That at a constant transition rate, the speedup of SLUM with respect to grows larger as the number of service providers grows larger. However, we also see that the speedup hits a limit (does not grow infinitely). The third observation is that reducing the transition rate accelerates increases the maximum speed achievable at any number of service providers. For example the speedup at n=10, when p=0.029 is 1.5 times, against

1.2 at p=0.13 and below 1 when p=1. These observations are expected as per the theoretical performance models developed in chapter two where we found that the speedup and transition rates are inversely related.

4.8.3 Expected Speedup vs Composite Service Phase Transition Rates under Polynomial Growth

In table 24, the following conclusions can be drawn. First, the empirical speedup generally increases with a decrease in transition rate. However, we note that the speedup at p=0.13 = 2.133 > speedup at p=0.064 =2.1. We treat this as an outlier. The second conclusion is that the empirical speedups are all smaller in value than their expected theoretical counterparts. This is expected because of several reasons. The theoretical expected speeds have an assumption of *n* being very large. Secondly, the theoretical model ignores constant terms which in practice could have contributed to some performance inefficiencies in our model.

Table 24: Summary Data: Expected Speedup vs Phase Transition Rates under Polynomial Growth

P	SLUM Polynomial Function	S-MIP Polynomial Function	Empirical Speedup(Ω_s)	Theoretical Speedup(Ω_s)
0.0296	$0.0144n^2$-0.3993n+ 4.530	$0.0518\ n^2$-2.2816n + 21.426	3.5972	4
0.064	$0.0116n^2$-0.3843n + 3.766	$0.0299n^2$- 1.1849n + 10.864	2.5779	3.7594
0.13	$0.0192n^2$-0.8567n+ 9.0362	$0.0405n^2$- 1.8665n + 18.443	2.109375	3.5398
0.36	$0.0173n^2$ - 0.634n + 5.886	$0.0369n^2$-1.5558n +13.018	2.133	2.9411
0.45	$0.3925n^2$- 2.8013n + 5.6039	$0.9738n^2$- 8.8225n + 17.215	2.481	2.7586
0.61	$0.0181n^2$-0.7365n+ 8.0346	$0.0392n^2$- .789n + 17.503	2.17	2.484
1	$0.0846n^2$-5.7392n+ 75.403	$0.0949\ n^2$ -6.554n + 86.398	1.12	2

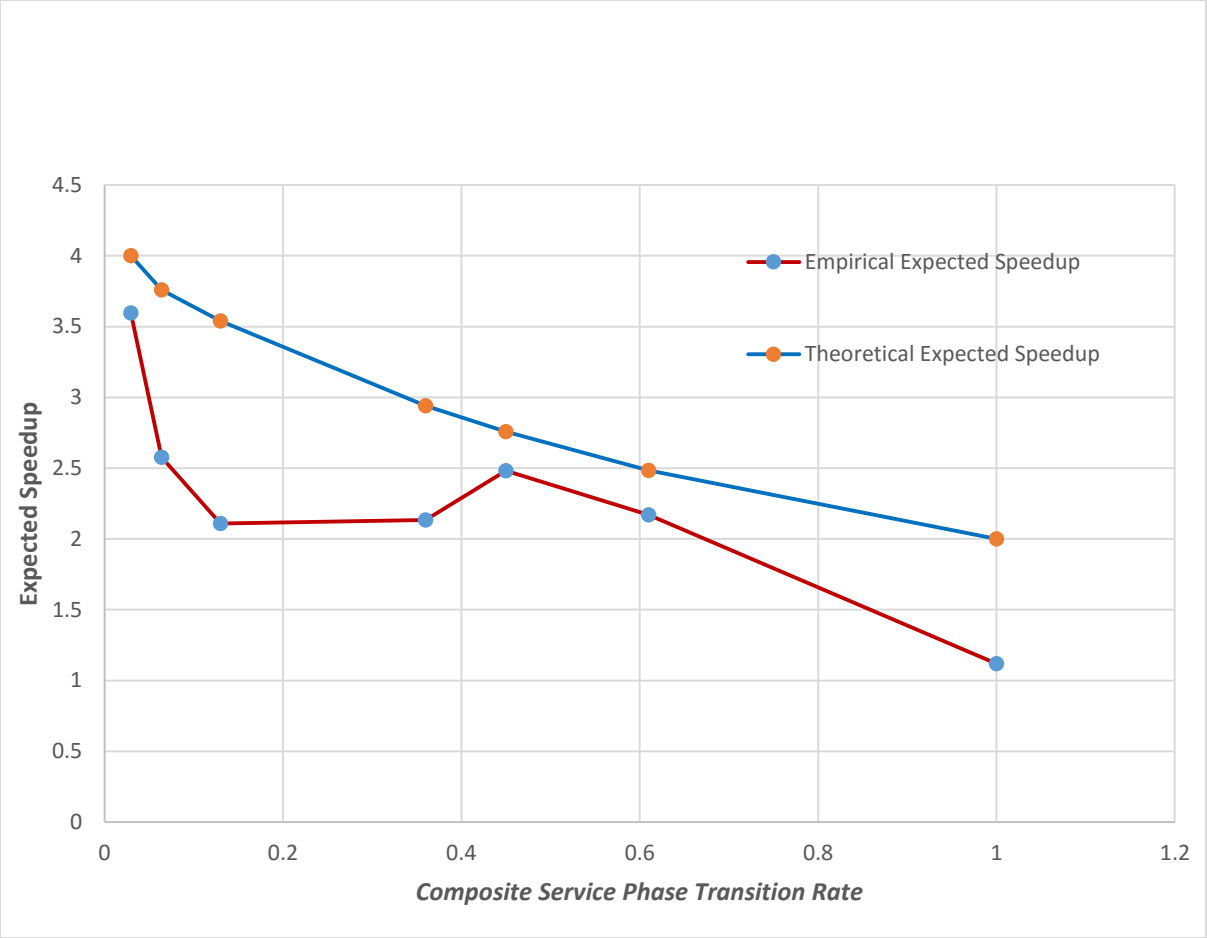


Figure 38 Summary – SLUM Expected Empirical and Expected Theoretical Speedup with respect to phase transition rates

The figure 38 alongside depicts a plot of speedup vs composite service transition rate. Both the empirical and theoretical curves are drawn. We deduce that both curves are decreasing functions. We also observe that the empirical curve is under the theoretical curve.

4.8.4 Expected Speedup vs Composite Service Phase Transition Rates under Exponential Growth

Table 25: Exponential Expected Speedup functions under various Phase Transition Rates

ρ	SLUM Exponential Function (t_{eB})	S-MIP Exponential Function (t_{eA})	Empirical Speedup function $\lim_{n\to\infty} d(t_{eA})/d((t_{eB})$	n_{CE} at Ω = 1.1(Solve $\lim_{n\to\infty} d(t_{eA})/d((t_{eB}) \geq 1.1$
0.0296	$0.904e^{0.0593n}$	$0.6231e^{0.0763n}$	$0.6893e^{0.017n}$	8
0.064	$0.5326e^{0.0617n}$	$0.5674e^{0.0727n}$	$1.065e^{0.01n}$	9
0.13	$0.6268e^{0.0597n}$	$0.6302e^{0.0699n}$	$1.01e^{0.0102n}$	9
0.36	$0.6882e^{0.0617n}$	$0.5602e^{0.0732n}$	$0.81e^{0.0115n}$	10
0.45	$0.5679e^{0.3206n}$	$0.5882e^{0.3545n}$	$1.036e^{0.0339n}$	3
0.61	$0.7902e^{0.0579n}$	$0.5918e^{0.0706n}$	$0.74e^{0.0127n}$	10
1	$0.8676e^{0.0605n}$	$0.7793e^{0.0624n}$	$0.9e^{0.0018n}$	59

Table 25 shows the speedup exponential functions at different transition rates. The functions can be used to answer the question: How much speedup given *n* and given that the transition rate was *p?* The solution is obtained by equating the function with the desired speedup and solving for *n.* The results show that for all transition rates except where p=1, at n>=10, a speedup of 1.1 times is guaranteed. The result mean that a virtual enterprise broker operating at least 10 service providers per task, regardless of the transition rate (except in rare cases where p=1), they are guaranteed to enjoy at least a 10% increase in efficiency by using SLUM over S-MIP. The functions could be used to compute speedups at any other value of *n*.

4.8.5 Variation of Initial and Asymptotic Coefficients vs Composite Service Phase Transition Rates

Table 26 below shows the variation of initial performance of SLUM relative to S-MIP and the asymptotic performance of SLUM with respect to S-MIP. A general trend is that SLUM is slower than S-MIP initially since the β_0**>1** generally. Exceptions to this were noted e.g when **ρ=0.064 , ρ=0.13 and p= 0.45**, where SLUM is marginally faster. The graph of β_0 vs **ρ** in figure 39 shows that the initial performance of SLUM is generally poorer than S-MIP. The reason for this is that SLUM experiences the sequential overheads of having to select the best composite service in two sequential phases, where S-MIP does only once (Mulongo et al, 2015).

Table 26: Initial and Asymptotic Performance Coefficients vs Phase Transition Rates

(ρ)	**Log t_{eB} vs log t_{eB} Equation**	**Initial Performance Parameter (β_0)**	**Asymptotic Performance :- Empirical Relative Complexity Coefficient**	**$1/(\beta_1)$**
0.0296	log TeB(n) = 0.7833 log TeA(n) + 0.2506	1.28	0.7883	1.2685526
0.064	log TeB(n)= 0.8488 log TeA(n) - 0.146	0.8644	0.8488	1.1781338
0.13	log TeB(n)= 0.8513 log TeA(n) - 0.0675	0.94082	0.8513	1.174674
0.36	log TeB(n)= 0.8437log TeA(n) + 0.1116	1.1181	0.8437	1.1852554
0.45	log TeB(n)= 0.8872 log TeA(n) - 0.061	0.941	0.8872	1.1271416
0.61	log TeB(n) = 0.8214 log TeA(n) + 0.1931	1.213	0.8214	1.2174336
1	log TeB(n)= 0.9684 log TeA(n) + 0.0999	1.1	0.9684	1.0326311

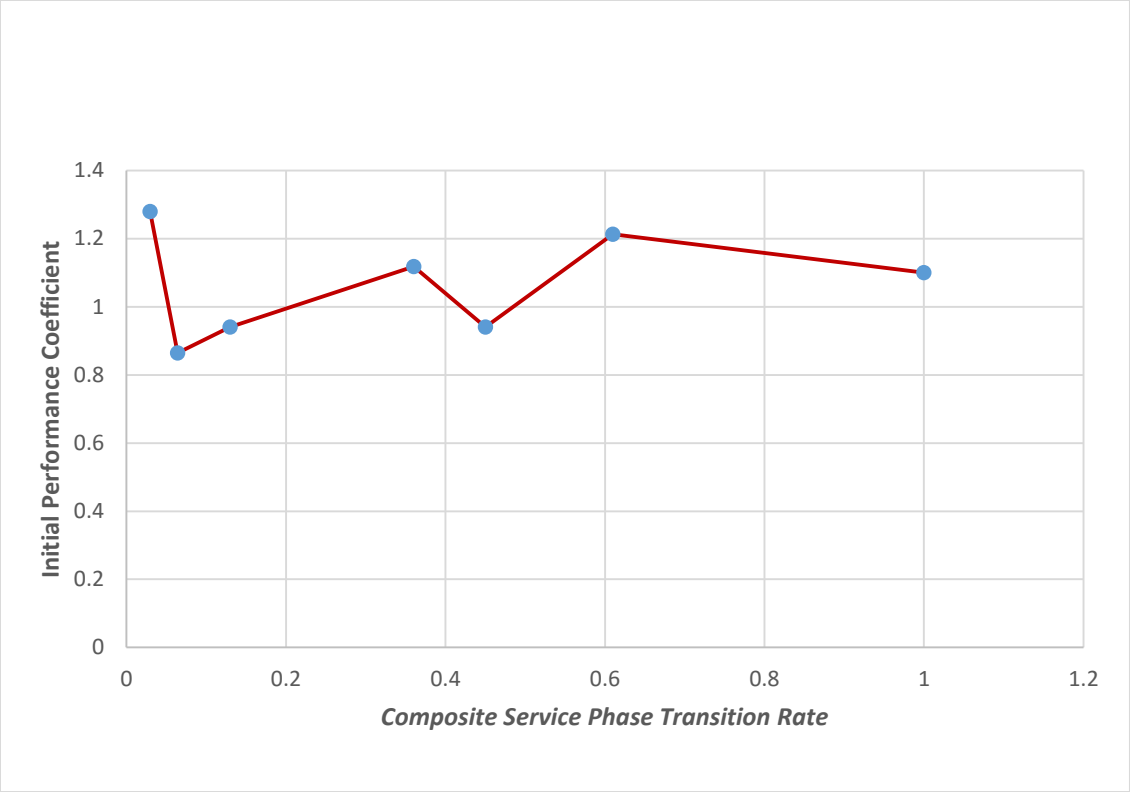


Figure 39 Summary –Variation of Initial Performance Parameter β_0 with respect to Phase Transition Rate ρ

The graph in figure 39 shows how the initial performance of SLUM relative to that of S-MIP varies with different values of the composite service phase transition rate ρ.

On the other hand, the converse is true for the empirical relative complexity β_1, which generally increases with an increase in ρ *and reduces with a reduction in* ρ. The graphs in figure 40 and 41 prove this. Since empirical relative complexity (Coffin & Saltzman, 2000) is a well-known method of comparing the relative speeds of two algorithms, the direct correlation between our method of transition rate , confirm the theoretical analysis that the speedup of SLUM was likely to decrease with an increase in transition rate. This was equivalent to proving that increasing transition rate increases the empirical relative complexity coefficient and vice versa, which this study has achieved.

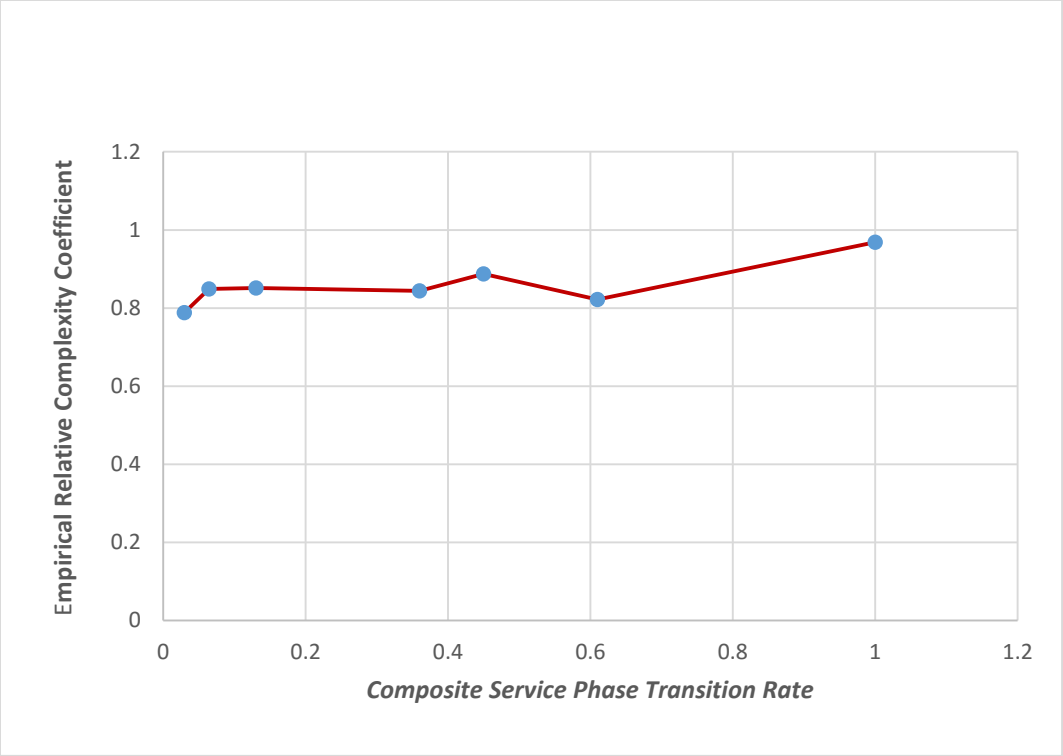


Figure 40 Summary –Variation of Empirical Relative Complexity Coefficient β_1 with respect to Phase Transition Rate ρ.

Figure 40 shows how the empirical relative complexity and hence how the asymptotic performance of SLUM with respect to S-MIP varies with the composite service phase transition rate.

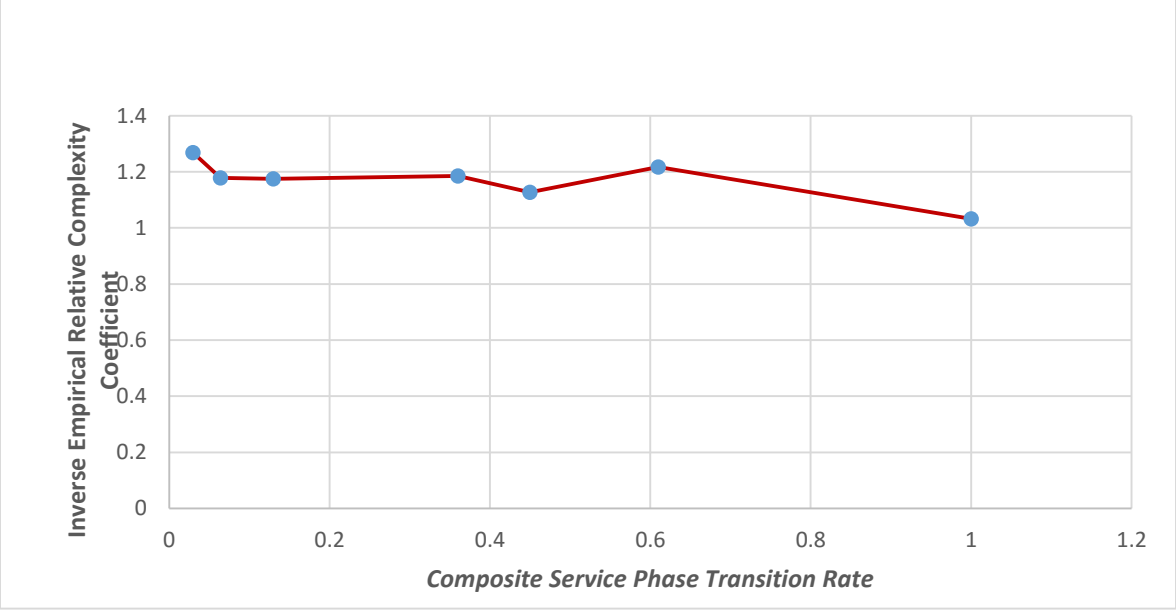


Figure 41 Summary –Variation of Inverse of Empirical Relative Complexity Coefficient ($1/\beta_1$) with respect to Phase Transition Rate ρ

4.9 Solution Quality and Optimality Results

Table 27: Solution Quality Performance Results

N	Z^B	Z^L	Z^*	RSQ_B	RSQ_L	OR_B	OR_L
2	0.69	0.58	0.69	0.00	16.67	100.00	83.33
3	0.61	0.73	0.73	16.44	0.15	83.56	100
4	0.68	0.50	0.68	0.00	26.50	100.00	73.50
5	0.59	0.56	0.75	21.33	25.11	78.67	74.89
6	0.70	0.65	0.73	4.11	10.62	95.89	89.38
7	0.77	0.60	0.77	0.00	22.22	100.00	77.78
8	0.82	0.80	0.83	1.20	3.26	98.80	96.74
9	0.72	0.61	0.72	0.00	14.74	100.00	85.26
10	0.75	0.65	0.79	5.06	17.92	94.94	82.08
11	0.70	0.58	0.72	2.78	19.22	97.22	80.78
12	0.70	0.61	0.7	0.00	13.28	100.00	86.72
13	0.67	0.00	0.7	4.29	100.00	95.71	0.00
14	0.63	0.00	0.69	8.70	100.00	91.30	0.00
15	0.60	0.68	0.68	11.76	0.60	88.24	99.40
16	0.59	0.64	0.75	21.33	15.04	78.67	84.96
17	0.72	0.65	0.75	4.00	13.33	96.00	86.67
18	0.67	0.69	0.75	10.67	7.35	89.33	92.65
19	0.65	0.64	0.72	9.72	10.63	90.28	89.38
20	0.71	0.56	0.77	7.79	27.65	92.21	72.35
21	0.66	0.55	0.74	10.81	25.51	89.19	74.49
26	0.76	0.78	0.69	2.56	11.45	97.44	88.55
27	0.74	0.79	0.79	6.33	0.31	93.67	99.69
28	0.65	0.77	0.60	15.58	21.79	84.42	78.21

29	0.73	0.77	0.74	5.19	4.23	94.81	95.77
30	0.74	0.78	0.68	5.13	13.43	94.87	86.57
31	0.75	0.79	0.77	5.06	2.53	94.94	97.47
32	0.66	0.8	0.74	17.50	6.99	82.50	93.01
33	0.70	0.78	0.67	10.26	13.93	89.74	86.07
34	0.69	0.83	0.71	16.87	14.41	83.13	85.59
35	0.79	0.79	0.60	0.00	24.29	100.00	75.71
36	0.68	0.74	0.62	8.11	15.72	91.89	84.28
37	0.72	0.79	0.72	8.86	8.23	91.14	91.77
38	0.80	0.82	0.74	2.44	9.51	97.56	90.49
39	0.62	0.79	0.66	21.52	17.04	78.48	82.96
40	0.66	0.73	0.70	9.59	4.30	90.41	95.70
41	0.77	0.8	0.63	3.75	20.77	96.25	79.23

In table 27 above, the optimization solution values for SLUM, L-MIP and S-MIP respectively are given in the columns labelled $Z^{B,}$ Z^{L} and Z^{*} for problem instances of varying size.. The rows in the grey background denote infeasibility i.e no solution was found for the problem instances with the given problem size. Such instances were excluded from analysis.

From the results, we observe that for every problem instance, S-MIP solution value $\boldsymbol{Z^{*}}$ is the highest value of the three algorithms, implying that S-MIP yields more quality solutions than both SLUM and L-MIP. This result confirms our considerations in section 3.2.2. Thus the result allowed us to compute RSQ_B, $RSQ_{L,}$ OR_B and OR_L as defined in equations (2), (3), (4) and (5) respectively. In table 1, a RSQ value of 100% denotes that no solution was found (or 100% error rate). For example, we see that for the problem instances with *n=13* and those with *n=14, L-MIP* failed to find a solution where SLUM and S-MIP did. For fair analysis and comparison, we excluded results that contained RSQ=100%. By computing the mean optimality ratio from the data provided, we determine that the mean optimality ratio of SLUM ≈93b3T CVC %, implying an average error rate (RSQ) of ≈7%. On the other hand, the mean OR_L value of L-MIP is ≈88% or RSQ_L ≈12%. These simple descriptive statistics suggest that SLUM generally generates more optimal solutions on

average than L-MIP by approximately 5%. We shortly validate this claim using one of the tests described in section 3.5.1. Despite the fact that SLUM seems to have a larger mean *OR or smaller RSQ*, there are some cases where L-MIP yields more quality solutions, for example, in table 1 L-MIP is outperforms SLUM when *n*=16, 27, 39 etc. However, as visualized in figure 42 and figure 43, SLUM generally has more quality solutions than L-MIP. With reference to the problem instances with n=*3, n=9, n=12* , it can also be seen that both SLUM and S-MIP are able to obtain a globally optimal solutions in some cases.

Our inferential statistical analysis followed the procedure established in section 3.5.1 to compare the RSQ_B, *vs* RSQ_L, and therefore by implication OR_B *vs* OR_L. Figure 42 is a graph of *RSQ vs n.* An alternative representation of the results is captured by the bar graph in figure 43. The curves RSQ_B *vs n* and RSQ_L *vs n* both hint the following: the variation of *RSQ vs n* is nonlinear, and the performance differences between RSQ_B and RSQ_B do not seem to be constantly increasing with *n.* Thus, the slope test based on linear regression as described in 3.5.1.1 was not found to be an appropriate comparison technique. The alternative approach detailed in section 3.5.1.2 was used instead. Since our sample size *N=38* and solution quality differences do not exhibit heteroskedasticity, under the assumptions of the central limit theorem, we assume that the *RSQ or OR* are normally distributed. We carried out a Shapiro Wilk test on the 38 sample performance differences, to verify our normality assumption. We obtained *W=0.93* against the critical value W_c =0.938, at a significance of level of 0.05 and 38 degrees of freedom. The result confirms our assumption of normality. Therefore, the paired Student t- test was used to verify the significance of the 5% mean difference in solution quality between SLUM and L-MIP. The null hypothesis is rejected if either t-stat< -t critical two tail or t-stat> t-critical two tail. The results of the significance test are presented in table 2. In table 3 below, the *t-stat = -3.182* and *-t-critical two tail = -2.03*. Since *-3.182 <-2.03*, we reject the null hypothesis and conclude that there are more than 95% chances that SLUM yields solutions with better quality by 5%.

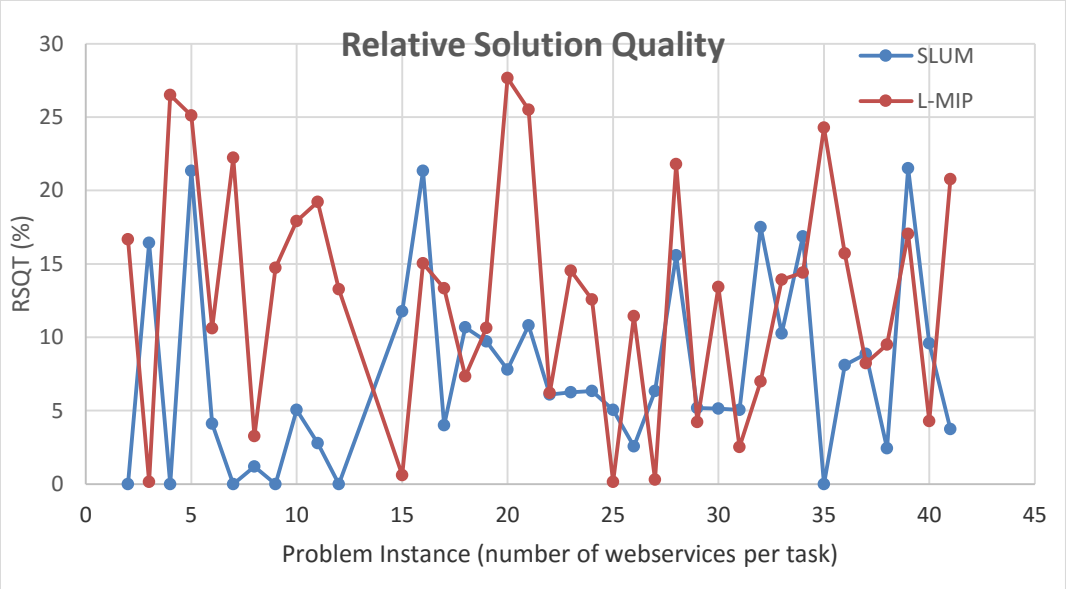


Figure 42 Line Graph Showing Relative Solution Quality of SLUM & L-MIP

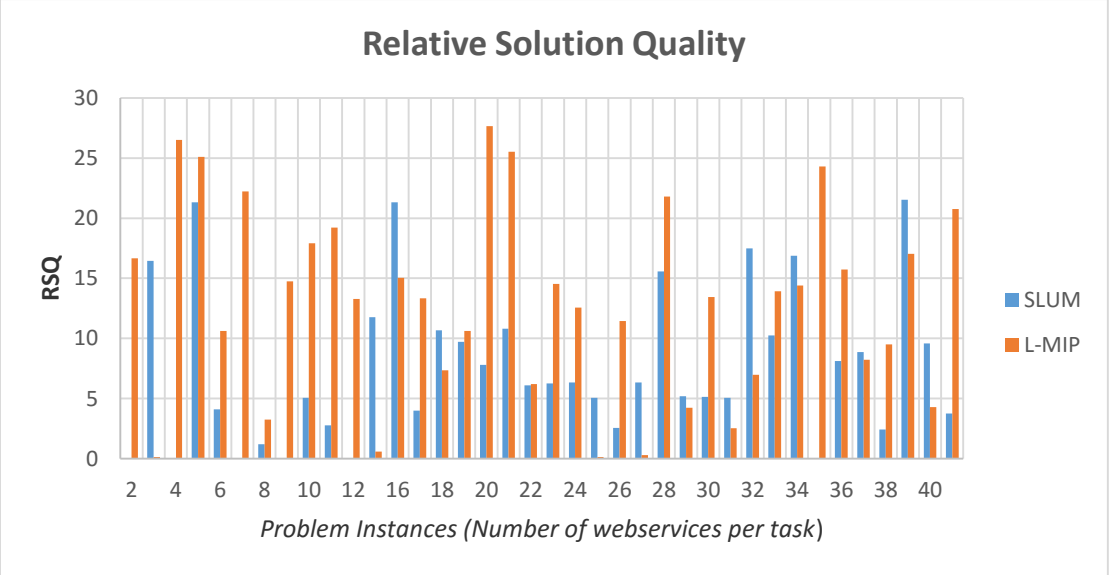


Figure 43 Bar Graph Showing Relative Solution Quality of SLUM and L-MP with respect to S-MIP

Table 28 : Paired Student -t Test Results on SLUM & L-MIP Relative Solution Quality

	SLUM	*L-MIP*
Mean	7.618157895	12.95210526
Variance	40.33246949	62.92169815
Observations	38	38
Pearson Correlation	-0.034624739	
Hypothesized Mean Difference	0	
df	37	
t Stat	-3.182521967	
P(T<=t) one-tail	0.00147773	
t Critical one-tail	1.68709362	
P(T<=t) two-tail	0.00295546	
t Critical two-tail	2.026192463	

4.10 Discussion of Results

As per section 1.5, the thesis sought to answer the research questions below:

RQ1: For a composite webservice selection problem having a workflow with *k* tasks and *v* alternative webservices per task, how does the runtime efficiency of SLUM compare with that of S-MIP and L-MIP when each is used to solve the problem*?* The specific research questions arising from this question are:

RQ1.1: How does the running time of SLUM grow as the number of service providers per task increase?

RQ1.2: How does the running time growth of SLUM compare with that of S-MIP and L-MIP?

RQ1.3: How much speedup is achievable when using SLUM over S-MIP to autogenerate composite webservices given a business workflow having n webservices per task?

RQ1.4: What is the minimum number of service providers per workflow task that a virtual enterprise broker needs to have in order to benefit from the relative efficiency of SLUM when compared to S-MIP?

RQ2 that was outlined in section 1.5 of chapter two is: How does the average solution quality of SLUM compare with that of L-MIP and S-MIP? This leads us to the following specific research questions:

RQ2.1: What is the average relative solution quality (percentage accuracy of quality) of the composite webservices generated by SLUM relative to S-MIP?

RQ2.2: What is the percentage difference in the relative solution quality by SLUM relative to L-MIP?

4.10.1 Running Time

RQ1.1: How does the running time of SLUM grow as the number of service providers per task increase?

First, as per the results in section 4.8.1, we conclude that the running time growth of SLUM increases as the initial number of webservices per task get larger. At the same time, based on the empirical results, we conclude that the running time growth also grows larger the larger the composite service phase transition rate **ρ** and grows slower for smaller values of **ρ.** Hence the empirical results are in agreement with our theoretical model $T_B\,(n) = (\boldsymbol{n^k}(\boldsymbol{q_1}/\boldsymbol{q_t})^{\boldsymbol{k}} + \prod_1^k(\mathrm{n}-\epsilon_i)\,(\boldsymbol{q_2}/\boldsymbol{q_t})^k)$

We have determined that regardless of the value of the phase transition rate **ρ,** SLUM exhibits very strong and very statistically significant polynomial running time growth in the number of service providers per workflow task (see the results in section 4.2 to 4.8). This finding is consistent with the theoretical model $T_B\,(n) = (\boldsymbol{n^k}(\boldsymbol{q_1}/\boldsymbol{q_t})^{\boldsymbol{k}} + \prod_1^k(\mathrm{n}-\epsilon_i)\,(\boldsymbol{q_2}/\boldsymbol{q_t})^k)$ (see equation 2.48 in chapter two). Given that *k* , $\boldsymbol{q_1}, \boldsymbol{q_t}$ and $\boldsymbol{q_2}$ are generally much smaller than *n* and further k, $\boldsymbol{q_1}, \boldsymbol{q_t}$ and $\boldsymbol{q_2}$ are fixed, the theoretical model in equation 2.48 is approximately a polynomial of degree *k.* The statistical polynomial regression functions obtained under all the seven different values of **ρ** had a degree of two (2) (see section 4.8 for a summary of the functions), which happens to the value of k at 2 that was fixed for all the experiments. At the same time, in all the setups, we also observed that SLUM exhibits a strong and statistically significant exponential growth in the

number of service providers, even though, generally, the exponential growth is not as strong as polynomial growth at a fixed transition rate. This implies that for very large number of virtual enterprise service providers per workflow task, our prosed model SLUM, is highly likely to suffer exponential state space explosion, and thus not likely to guarantee polynomial time solution to very large problems. In conclusion we could say that SLUM is *superpolynomial* and hence non deterministic polynomial. This result is not surprising since as discussed in chapter one and two, we saw that all Mixed Integer Programming algorithms suffer exponential state explosion for very large problem sizes. The finding that SLUM has a superpolynomial running time growth despite being faster than the state of the art (see previous and subsequent sections for evidence) is an empirical reinforcement that the dynamic webservice composition problem remains NP hard.

RQ1.2: How does the running time growth of SLUM compare with that of S-MIP and L-MIP?

In regard to *RQ1.2,* we determine that the running time of S-MIP is empirically bounded between polynomial and exponential growth –this implies that the running time of S-MIP is nondeterministic polynomial or superpolynomial. In a theoretical sense, we could conclude that for an infinitely large number of service providers per workflow task, both SLUM and S-MIP are equally bad in performance. This is expected because as per the analytic considerations of chapter two, ignoring constant terms, the initial global search space for both algorithm is $\boldsymbol{n^k}$**.** For very huge *n*, this search space can require exponential effort (Benatallah, et al, 2004). However, as will shortly be explained, in practice SLUM is much faster than S-MIP. On the other hand, we saw that the running time growth of L-MIP has a polynomial upper bound. In a L-MIP is multiple factors faster than both S-MIP an L-MIP. The result confirms the analysis in chapter two and also reinforces the fining of previous studies that L-MIP is much faster than S-MIP.

As per the criteria in 3.5.2.1, we can conclude that the running time growth of L-MIP, $T_{eL}(n)$ is empirically bounded between $O(n)$ and $O(n^k)$ i.e $O(n) \leq T_{eL}(n) \leq O(n^k)$, while $T_{eB}(n)$ *and* $T_{eA}(n)$ are both empirically bounded between polynomial and exponential empirical complexity classes, so that e $O(n^k) \leq T_{eA}(n) \leq O(e^{o(n)})$ and $O(n^k) \leq T_{eB}(n) \leq O(e^{o(n)})$. On the basis of empirical complexity, we therefore conclude that L-MIP guarantees solutions within polynomial time, while both SLUM and S-MIP do not guarantee solutions in polynomial time since they have a

polynomial empirical lower bound but exponential empirical upper bound. Thus, L-MIP is far more efficient than both SLUM and S-MIP.

Despite having found that both SLUM and S-MIP are superpolynomial theoretically, and therefore are equally bad in theoretically, there are plausible practical performance differences between the two methods. Looking at the graphs of running time for the seven different experimental setups, and the summary statistics in section 4.8, it's evident that irrespective of the transition rates, SLUM is much faster than S-MIP (see the response to RQ1.3 for a more quantitative treatment of the performance comparison between SLUM and S-MIP).

RQ1.3:- How much speedup is achievable when using SLUM over S-MIP to autogenerate composite webservices given a business workflow having n webservices per task?

The answer to this question comes from figure 37, the plot of speedup vs number of service providers for different transition rates and figure 38 having a plot of expected speeup vs transition rates. We saw that at a constant transition rate, the speedup of SLUM with respect to grows larger as the number of service providers grows larger. However, we also see that the speedup hits a limit (does not grow infinitely). The third observation was that reducing the transition rate accelerates increases the maximum speed achievable at any number of service providers. For example the speedup at n=10, when p=0.029 is 1.5 times, against 1.2 at p=0.13 and below 1 when p=1. Figure 38 on the hand, shows that on average, it's possible for virtual enterprise brokers to hit an average speed up of 3.6 times when using SLUM. We see that:-

1. Based on the descriptive and inferential statistics in section 4.8.2 :- the speeup vs number of service provi ers graph shows that :-
    i. Virtual enterprise brokers with as low 10 to 20 service providers per workflow task could enjoy speeups of up to 1.5 times when using SLUM as opposed to S-MIP.
    ii. Virtual enterprise brokers with service providers per task in the range 30 to 50 could enjoy nearly two fold speed up when using SLUM instead of S-MIP.
    iii. Virtual enterprise brokers with more than 60 service providers could enjoy more than two times speedup.
2. Using the results in section 4.8.3 that are based on differential calculus using L-Hospital's Rule. We established that for any number of service providers that is large enough, virtual enterprise

brokers could expect a worst mean speedup of 1.1 times (10% efficiency gain) when the transition rate approaches 100% and up to 3.6 times (260% gain efficiency gain) when the transition rate approaches zero. Note that the transition rate depends on the current values of webservices QoS attributes and the current QoS constraint models) of the problem instance. Since we see that at 60 to 70 service providers per task, it's possible to hit an average speed up of 3.5 times, then the 260% improvement in speed is very practicable. In chapter two, we established a theoretical model that captures the expected speedup of SLUM w.r.t. The model is model $\Omega = \frac{(2)^k}{1+\ \rho}$. Thus a plot of Ω should exhibit inverse rational function. The empirical expected speedups determined via L-Hospital's rule (see section 4.8.3) confirm the inverse trend between speedup and transition rate. Secondly, a comparison between the empirical expected speedup values at given transition rates with the expected theoretical results are in synchrony albeit with some deviations. However, note that the empirical values are all less than their theoretical counterparts. This is expected because, as said earlier, theoretical models do not take into account the practical limitations of the execution platform. The error deviations could be attributed to: 1) machine dependent factors and 2) random errors. The conclusion is that on average the virtual enterprise broker can practically expected speedups of up to $\frac{(2)^k}{1+\ \rho}$.

3. Using the method of empirical relative complexity analysis (see chapter 3) or Coffin & Saltzman (2000), we have empirically demonstrated that even though both SLUM and S-MIP could suffer exponential state space explosion, in practice, the running time of SLUM grows much slower compared to S-MIP. The summarized results in section 4.8.4 show that for all composite service phase transition rates **ρ** on the interval [0,1], the empirical relative complexity coefficient $\boldsymbol{\beta_1}$ of SLUM with respect to S-MIP is less than 1. In fact, based on the statistics, the mean value of $\boldsymbol{\beta_1}$ is about 0.85 and the modal value between 0.8 and 0.9 . A $\boldsymbol{\beta_1}$ value of 0.85 roughly corresponds to a transition rate value of 0.36. At ρ=0.36, the speedup is about 2.1 times. Note that at *k=2*, transition rate of 0.36 means that more than half of the initial service providers per task get promote for phase two optimization.

Although SLUM is generally much faster than S-MIP, using the empirical relative complexity measure by Coffin & Saltzman (2000), we found that for a small number of webservices per task, S-MIP is about 1.2 times faster than SLUM even though there were a few cases where SLUM is

initially an asymptotically better than S-MIP . We also used L-Hospital's rule and limits theory to arrive at $n_{CE} = 22$ i.e beyond 22 webservices per worklow task, SLUM has a relative speedup larger than 1 and therefore faster than S-MIP. These results could be attributed to the fact that initially, SLUM suffers the sequential overhead of having to formulate and instantiate the optimization problem twice on two sequentially partitioned problem instances, first one at the SCUM layer then later at the SPUM (Abiud et al, 2016). At a fixed transition rate, the overhead is steadily overcome by the relative advantage of the layering as decomposition optimization approach as *n* grows larger and beyond 22, the sequential overhead is completely overcome and the superior performance of SLUM becomes apparent. This empirical result reinforces the theoretical claim that even when decomposition is formulated on sequential algorithms, relative performance speedups arise from the Superlinear growth of the problems being solved (Byod et al, 2003).

RQ1.4: What is the minimum number of service providers per workflow task that a virtual enterprise broker needs to have in order to benefit from the relative efficiency of SLUM when compared to S-MIP?

The answer to this research question comes from the analysis of the results as in subsections 4.2.3, 4.3.3, 4.4.3, 4.4.3, 4.5.3, 4.6.3 and 4.7.3, titled *" Expected Speedup via L-Hospital's Law".* Under exponential growth, in each of these sections, the speedup of SLUM with respect to S-MIP (SES) is an exponential function in *n.* We have consolidated the results of these subsections in section 4.8.4. Figure 37 can also be quickly used to answer the question without rigorous mathematical analysis.

From the results, we conclude that the minimum number of service providers beyond which SLUM is faster than S-MIP depends on the transition rate. The lower the rate the lower the value n_{CE}, and the better the case for the virtual enterprise broker. For instance, using the results in section 4.8.4 we see that to achieve a minimum speedup of 1.1 times, at any transition rate value $n_{CE} \leq 10$. This means that a virtual enterprise broker operating at least 10 service providers per task could be guaranteed at least a 10% increase in efficiency when using SLUM as opposed to S-MIP. An exception exists, where $n_{CE=59}$ when the transition rate is 1. The implication is that is at any one time whenever the transition rate is 1, at that point in time, any virtual enterprise broker having

less than 59 service providers per workflow task would not benefit from SLUM. However, in a dynamic service environment, the transition rate will be seldom stationary and like said earlier, will rarely be 1. This is to mean that the frequency of a virtual enterprise broker with as few enterprises as ten per task are high.

From figure 37, we see that at a rate of 0.029, a virtual enterprise broker can enjoy a speedup of 1.5 at n_{CE}=10, thus 10 becomes the minimum at this rate. At $\rho=0.13$, the minimum value of n_{CE} is still *10* but a lower speedup of 1.3 times speedup. At $\rho=0.6$, the minimum value of *n* is 40 providers with 1.3 times speedup. At $\rho=1$, the minimum value of *n* is about 120 service providers with 1.2 times speedup. However, given that $\rho=1$ represents an extreme (special) cases, it's unlikely that on the average case, a virtual enterprise broker with less than 120 service provider would fail to enjoy the speedup of SLUM. From the graph in section 4.8.2, we can see that for all transition rates except at $\rho=1$, SLUM performs better than S-MIP just above 30 service providers per task. One method to validate this is by $\rho=0.6$ (a realistic but the most pessimistic value). At k=2, $\rho=0.6$ means that more than 75% of service providers per task satisfied the QoS constraints of layer optimization. For example, *k*=60, yields 3600 composite services initially. Sixty percent transition rate, means a total of 2160 composite services proceeding to layer two for final selection, which is roughly 47 out 60 services per task transiting. Having set $\rho=0.6$, and assuming exponential growth (worst case), the exponential regression functions at $\rho=0.6$ could be used to estimate the minimum number of service providers required for SLUM to have a speedup of at least *S* times. From the results, at $\rho=0.6$, we have $\mathbf{t_{eA} = 0.5918e^{0.0706n}}$ **and** $\mathbf{t_{eB} = 0.7902e^{0.0579n}}$. Applying L-Hospital's rule as per the methodology established in chapter three, we have SLUM Expected speedup at some value of *n* given as SES = $\mathbf{0.5918e^{0.0706n} / 0.7902e^{0.0579n} = 0.75e^{0.0127n}}$

By solving the inequality $\mathbf{0.75e^{0.0127n} \geq S}$, one can estimate the minimum value of *n* required to achieve at a speedup of at least *S* times. Let's we have that S = 1.3 (20% increase in speed) . We have $\mathbf{0.75e^{0.0127n} \geq 1.3}$. **Which yields $n \geq 27$. Lowering S=1.2, yields $n \geq 20$ and $n \geq 10$ at S=1.1.** Thus, in general a virtual enterprise broker having at least 20 service providers per task is guaranteed at least a 10% improvement in speed when using SLUM instead of S-MIP provi the transition rate is less than 100%.

4.10.2 Relative Solution Quality of SLUM vs L-MIP with respect to S-MIP

RQ2.1: What is the average relative solution quality (percentage accuracy of quality) of the composite webservices generated by SLUM relative to S-MIP?

In response to research question **RQ2.1,** the results in section 4.9 indicate that the optimality ratio of SLUM is about 93% on average. We conclude that SLUM suboptimal relative to S-MIP. This finding confirms our analytic considerations of section 2.11 of chapter two, our early hypothesis of our recent work in (Abiud et al, 2015) and the results in (Abiud W. M et al, 2016). The reason for this result is because, although SLUM considers all global constraints eventually, it does so in two layered steps, so that at one layer only global constraints related to the QoS attributes at that layer are considered. Thus, while SLUM guarantees global optimality within each layer, it does not guarantee the same at the *network* level because optimization at each layer does not take into account the constraints at the other layer (Mulongo et al, 2015; 2016).

RQ2.2: What is the percentage difference in the relative solution quality by SLUM relative to S-MIP?

We found that SLUM on average is significantly more optimal than L-MIP by approximately 5%. We draw the conclusion that SLUM on average yields more quality solutions than L-MIP. The reason why SLUM has a better solution quality performance could be qualitatively attributed to the fact that while SLUM considers global constraints albeit in two partial steps, L-MIP does not at all take into account global constraints, and thus, the chance of L-MIP ignoring potentially better webservices across the workflow are higher than SLUM (Mulongo et al, 2016). A possible quantitative explanation for the same observation is as follows (Mulongo et al, 2016): for each *k* by *n* workflow, L-MIP only considers the QoS matrices of *n* webservices at a time. Given that globally, the maximum number of possible solutions is n^k, for every optimization decision L-MIP takes, L-MIP ignores the QoS *of* n^k-*n* other possible solutions. SLUM on the other hand, at each layer, considers approximately only half of the QoS matrices per workflow task i.e recall that each worklow task is mathematically represented as matrix. Each matrix contains a set of *n* QoS vector, where each vector is a webservice. By considering only half of the QoS attributes at a time, SLUM

only considers *n/2(*half of the QoS information) options per task. Thus at each layer, SLUM considers only $(n/2)^k$ possible options, while it ignores the remaining $(n)^k - (n/2)^k$ options. At a constant *k, i*t is possible to show that $(n)^k - (n/2)^k < n^k - n$, $n>4$. For instance, let $n=6$ and $k=2$, $(n)^k - (n/2)^k = 27$, while $n^k - n = 30$, and when $n=60$, $k=2$, $(n)^k - (n/2)^k = 2700$ and $n^k - n = 3540$. Since solution quality depends on the variety of candidate solutions (Holdger Hoos, 2003) (see also chapter three), SLUM has more chance of yielding more optimal solutions than L-MIP. Nevertheless, we also observed that in some instances, L-MIP, despite ignoring a larger number of candidate solutions, is able to produce solutions with better quality than SLUM. This isn't surprising because, in addition to the variety or set of candidate solutions considered, as illustrated in section 3.2.3.3, the statistical structure of the problem instances (QoS matrices) can affect the quality of the solution produced by one algorithm compared to the other. However, provided the problem instances are random in nature, for a very large number of problem instances, with monotonically increasing number of webservices per workflow task, there are more than 95% chances that SLUM will produce more quality solutions than L-MIP as indicated by the statistical tests.

Even if L-MIP performs worse than SLUM on average, we observed that its mean optimality ratio with respect to S-MIP is 87%. This is still an impressive performance. The study conducted by Ardagna et al (2007) established that global planning MIP algorithms for webservice composition outperform local planning MIP algorithms designed for the same task by 20% to 30% on average in terms of solution quality. This implies that, at the very best, local planning MIP strategy has a mean optimality of 80% according to Ardagna et al (2005). Although, our results report a figure slightly larger than 80%, our finding that L-MIP has 87% mean solution quality, is a reinforcement of the results by Ardagna et al (2007).

5 CHAPTER 5: CONCLUSIONS & CONTRIBUTIONS

In general, webservice composition is a critical business capability of modern virtual organizations (Rabelo et al, 2007; 2008). Composite Web services can save a substantial amount of time and cost for developing new software applications and enhancing the interoperability and collaboration among the various virtual enterprises within collaborative virtual organizations (Dongsong et al, 2005). In particular, *dynamic webservice composition* (DWSC) is an important technological as well as a business capability for enabling delivery of highly adaptable, customized and personalized services to web service consumers (Mulongo et al , 2015;2016a; 2016b), (Farhan et al, ,2010). DWSC could benefit a Virtual enterprise broker in the following ways (Abiud et al, 2016a):

i. Improved likelihood of the service consumer obtaining high quality solutions because the best composite service is selected from a pool of many potential solutions. Even in the event that no suitable solution is found that satisfies the consumer, the user can be provided with the list of feasible solutions and choose whether or not one of them nearly satisfies them.
ii. Through re-planning strategies, workflows that are dynamically bound to webservices at runtime are more likely to survive failures through selection of different execution paths hence boosting system reliability and customer experience.

However to autogenerate and adapt composite services that maximize the utility of various service consumers whose preferences differ from time to time, remains a multiple criteria decision making problem whose solution cannot be guaranteed in reasonable time. As explained in previous chapters, Mixed Integer Programming (MIP) is the most appropriate method for efficiently modelling decision problems that involve linear integer, real and binary variables. Current MIP formulations for the webservice composition problem exploit two alternative strategies – *local planning* which is demonstrably polynomial time but lacks the ability to capture global constraints and therefore generally suboptimal. The alternative strategy is *global planning*, which can capture both local and global constraints. The global planning MIP guarantees global optimality but cannot guarantee a solution in polynomial time. All the existing global planning MIP methods for the webservice composition problem follow a flat structured model in which one monolithic mixed integer optimization program is formulated and solved in one shot (Mulongo et al, 2015; 2016a;

2016b). The limitation is that such MIP models grow faster in search space as the number of optimization decision variables grow larger (Mulongo et al, 2015; 2016a). Other than performance limitations, a second gap in all existing service composition strategies is that end users are required to specify weights and QoS constraints on too many QoS attributes, some of which are too technical to discern (Abiud W.M et al, 2015) – it can be too tedious for the end user (Zeng et al, 2004). Even though Benatallah (2004) hypothesizes that a hierarchical optimization approach to the dynamic webservice composition problem could yield a more efficient solution albeit with solution quality tradeoff, until this study there no such an approach.

In line with our research goal, this study aimed to design more efficient Mixed Integer Programming dynamic composite webservice selection strategy that does not deny service consumers an opportunity to specify all their critical local and global webservice QoS constraints. This research goal was pursued through two specific research objectives. The research objectives were:

i. Design a layered hierarchical mixed integer programming model for the composite webservice selection problem following the concepts from the theory of *Layering as Optimization Decomposition*.
ii. Evaluate the performance of the SLUM model against the single layered global planning technique (S-MIP) and the local planning method (L-MIP) in terms of two metrics:
    i. Running time (performance efficiency) and;
    ii. Solution quality.

In relation to the above research objectives, the study also sought to answer the following two main research questions.

RQ1: For a composite webservice selection problem having a workflow with *k* tasks and *n* alternative webservices per task, how does the runtime efficiency of *SLUM compare with that of S-MIP and L-MIP* when each is used to solve the problem*?* The specific research questions arising from this question are:

RQ2: How does the average solution quality of SLUM compare with that of L-MIP and S-MIP? This leads us to the following specific research questions?

Through pursuing the above research objectives and seeking answers to the above research questions, this study makes the contributions and conclusions described from section 5.1.

5.1 Contributions

5.1.1 A Two Layer Architecture and Model MIP Model for the Webservice Composition.

Our main contribution is that we have pioneered the application of the theory of *Layering as Optimization Decomposition* (from a conceptual perspective) to solving the dynamic webservice composition problem more efficiently using the global planning strategy described in chapter two. Layering as Optimization Decomposition (Mung, 2006;2007) & (Steve Low , 2013) is an architectural as well as mathematical framework that has been used to reformulate the classical Network Utility Maximization (NUM) problem, so that it's solved in a layered fashion, the results being more optimal TCP/IP networks in efficiency, throughput and network resource allocation. Although the theory has its roots in the communication networks field, this study has argued that the *dynamic webservice composition problem* resembles the network utility maximization problem, and consequently recast, the well-known (single layered) global planning mixed integer programming model (S-MIP) for service composition pioneered by Zeng et al (2004), into a two layered hierarchical MIP model called SLUM: *Service Layered Utility Maximization model,* inspired by the conceptual aspects of the Layering as Optimization Decomposition theory.

The research output was a conceptual architecture together with the underlying mathematical models. The two layered MIP model is documented in (Mulongo et al, 2015) and its performance analysis presented in ((Mulongo et al, 2016a; 2016b). The key ideas in SLUM are as follows.

i. Like in network utility maximization problem, NUM (Kelly et al, 1998) based on Layering as Optimization Decomposition, in SLUM, the original webservice composition problem is partitioned in two MIP optimization problems – each subproblem is tackled at its layer. One layer is concerned with the maximization of end user utilities. The objective function here is formulated in terms of decision variables related to financial burden of the user and efficiency i.e *response time, reputation, etc.* The other layer, is concerned with the maximization of the utility of the virtual enterprise broker (and their service providers). The objective function at the service provider layer is modelled in terms of low level technical webservice QoS

parameters such as *throughput*, *availability*, *reliability* etc. The mathematical optimization model at each of the two layers follows the global planning MIP model originally defined in Zeng et al (2004). The two MIP subproblems are then solved sequentially.

ii. Optimization at the Service Provider Utility Maximization layer is invisible to the end user yet the efficiency benefits of the optimization at the service provider are propagated to the end user without their knowledge. This form of abstraction is also employed in the network utility maximization problem, where improvements in the physical layer performance due to novel optimization algorithms at the physical layer are propagated to the application layer without the awareness of the end user.

The study has analytically theoretically and empirically shown that the two layered MIP approach is more efficient than the flat structured owing two main factors:-

i. *Space reduction due to sequential decomposition*. When a problem is decomposed and the resultant subproblems solved sequentially (as opposed to a parallel solution), efficiency benefits from the decomposition will still be achieved due to the theory that the complexity of computational problems grows more than linearly as a function of the input size (Byod et al,, 2003). The significance of this is that virtual enterprise brokers operating a workflow that has *k* sequential tasks could benefit from faster average speed of up to 1.5 times using a two layer global planning MIP strategy than using a single layer global planning to MIP strategy, even in the absence of webservice elimination at the first phase of optimization. More evidence for this contribution is given in sections 5.2.2 to 5.2.8, and can also be found in (Mulongo et al, 2016b).

ii. *Space reduction due to early service provider elimination.* We have shown that other than the relative speed gain due to superlinearity of the complexity of computation problems, a two layered MIP architecture inherently benefits from further efficiency gains due to early elimination of some service providers who do not satisfy the end user QoS constraints. In particular, according to section 4.8.3 and (Mulongo et al, 2016a), we showed that in the presence of elimination, a two layer global planning MIP model could be on average up to 4 times faster than the single layered global planning MIP model.

The performance efficiency results obtained provide an empirical proof that decomposition, even when applied to problems sequentially, eventually yields significantly more efficient solutions due to the super linearity of the complexity of computational problems as the problem size rises (Byod et al,, 2003). Concurrently, the results support the thesis of layering as decomposition (Mung ,2006) & (Low, 2012)), as a more efficient mathematical as well as architectural method for problems that inherently can be reformulated in multiple layers of abstraction- we provide the first proof where the composite webservice selection problem is concerned. Moreover, the results show that the relative expected efficiency gain of layering as decomposition with respect to non-layered is limited by the sequential overheads, hence achieving theoretical maximum expected speedup with respect to S-MIP may not be feasible.

5.1.2 Runtime Performance Evaluation of the Two Layer MIP Model.

Based on research question **RQ1** above, the study makes several contributions to the body of knowledge as regards the general runtime performance efficiency of SLUM, the initial as well asymptotic performance of SLUM with respect to S-MIP and L-MIP and expected speedup of SLUM with respect to S-MIP. The contributions & conclusions are described in subsections 5.1.2.1, 5.1.2.2, and 5.1.2.3.

5.1.2.1 A Theoretic Runtime Performance Model for the Two Layer MIP Model

Starting with the specific research question:

RQ1.1: How does the running time of SLUM grow as the number of service providers per task increase?

Through the analysis in section 2.12, the theoretical conclusion is that the runtime efficiency of SLUM is given by $\boldsymbol{n^k(1/2)^k + \prod_1^k(n-\epsilon_i)\,(1/2)^k}$.This theoretical performance model constitutes our second major contribution to the body of knowledge.

More conclusions can be drawn from this model. Firstly, that unlike in flat structured workflow based dynamic webservice composition approaches (L-MIP and S-MIP) in this case, the performance of layered approaches such as the proposed model (SLUM) is not only affected by the number of service providers ***n,*** and the number of workflow tasks **,** but also by the number of service providers that are eliminated per workflow task at the SCUM layer, ϵ_i. The larger the ϵ_i value the larger the efficiency and vice versa. Hence, the significance is that the magnitude of the average speedup gains expected by virtual enterprise brokers from SLUM depends also on the

number of early eliminated service providers.

Secondly, that runtime growth of SLUM as the $\boldsymbol{n}$ grows larger, at a fixed k is likely to be polynomial but also that at a fixed n, the runtime is likely to be exponential in k. The conclusion therefore is that theoretically, SLUM's runtime performance is superpolynomial and consequently non deterministic polynomial. However, examining the coefficients of the model leads to the conclusion that SLUM is theoretically more efficient than S-MIP, since the latter's performance model is given by $\boldsymbol{n^k}$. Further, the SLUM runtime performance model shows that as anticipated, SLUM has a poorer performance than L-MIP given that the latter's runtime performance model of the L-MIP is nk, which is linear time.

5.1.2.2 Empirical Runtime Performance Characterization of the Two Layer MIP Model

Through a series of experiments and using statistical regression analysis when k is fixed at 2, the study found that SLUM has both very strong and statistically significant quadratic and exponential runtime growth. Note that the quadratic growth is a polynomial growth when $k=2$. This consistutes our third major contribution – the first empirical proof that a two layer MIP model for the dynamic webservice composition problem is superpolynomial. Thus, our contribution here is in showing the a key limitation of layered MIP approach to webservice composition which is "despite the two layer MIP model having been proven to be on average and asymptotically more efficient than the single layered one" (see subsequent subsections) it still suffers from exponential state explosion. Through this finding, to the webservices research community, we provide further empirical evidence that the dynamic composite webservice selection considering global constraints, remains nondeterministic polynomial hard problem, and thus remains a significant problem deserving further research (Abiud et al 2016;2016b).

Given that SLUM is superpolynomial at $k=2$ and since $k=2$ is the smallest workflow in the number of tasks and is already superpolynomial, and the absolute performance of cannot be better at larger values of k, we therefore conclude that SLUM is generally superpolynomial both in theory in practice for all k.

5.1.2.3 Theoretic and Empirical Relative Performance Evaluation of the Two Layer MIP Model.

The contributions made under this section arise from the three research questions below:

RQ1.2: How does the running time growth of SLUM compare with that of S-MIP and L-MIP?

RQ1.3: How much speedup is achievable when using SLUM over S-MIP to autogenerate composite webservices given a business workflow having *n* webservices per task?

RQ1.4: What is the minimum number of service providers per workflow task that a virtual enterprise broker needs to have in order to benefit from the relative efficiency of SLUM when compared to S-MIP?

The study answered the research questions using both theoretical algorithm analysis and empirical analysis following a series of experiments. The resultant contributions and conclusions are given in section 5.1.2.3.1 to 5.1.2.3.5.

5.1.2.3.1 Complexity Analysis: SLUM & S-IMP are both theoretically & Empirically Superpolynomial & L-MIP is Polynomial time.

Following the analysis from section 2.12, from a theoretical perspective, the SLUM runtime performance model is given by $T_B = \boldsymbol{n^k(1/2)^k} + \prod_1^k(\mathrm{n}-\in_i)\,\boldsymbol{(1/2)^k}$ .S-MIP performance model $T_A = \boldsymbol{n^k}$ and the L-MIP runtime performance model is $T_C = nk$. Intuitively, both S-MIP and SLUM are superpolynomial theoretically since for both algorithms, for a fixed *n*, the runtime would grow exponentially in *k,* and grow in polynomial time in *n* with a degree of *k,* for a fixed *k*. Thus, ignoring constant terms, we can conclude that both the two layer MIP (SLUM) and the single layered MIP (S-MIP) are superpolynomial and equally bad in performance. As explained in the preceding subsection, empirical results also confirmed that both SLUM and S-MIP exhibit polynomial and exponential growth in runtime. On the other hand, L-MIP worst case runtime growth not only polynomial time but multiple orders faster than both S-MIP and SLUM i.e $nk \ll \boldsymbol{n^k}$ and $nk \ll \boldsymbol{n^k(1/2)^k} + \prod_1^k(\mathrm{n}-\in_i)\,\boldsymbol{(1/2)^k}$.

5.1.2.3.2 SLUM is much faster than S-MIP on average theoretically for large values *n for all k.*

Although from a complexity analysis point of view SLUM and S-MIP are superpolynomial, from the theoretical performance models, quantitatively, SLUM is significantly faster than S-MIP i.e

$\boldsymbol{n^k}(\frac{\mathbf{1}}{\mathbf{2})^k} + \prod_1^k(\mathrm{n}-\epsilon_i)\left(\frac{\mathbf{1}}{\mathbf{2}}\right)^k \ll \boldsymbol{n^k}$. This brings us to our fourth main contribution- we derive a theoretic SLUM speedup model relative to S-MIP. Based on the analysis of section 2.12, Following the model $(\boldsymbol{n^k}(\boldsymbol{q_1}/\boldsymbol{q_t})^{\boldsymbol{k}} + \prod_1^k(\mathrm{n}-\epsilon_i)\,(\boldsymbol{q_2}/\boldsymbol{q_t})^k)$, the study introduced the concept of *Composite Service Phase Transition Rate* ρ . ρ is the ratio of the number of alternative composite webservices available after phase one optimization to the number of alternative composite webservices available before the start of phase one optimization. Thus $\rho = (\prod_1^k(\mathrm{n}-\epsilon_i))/(\boldsymbol{n^k})$ and lies on the interval [0,1]. Using L-Hospital's rule, we went ahead to show that when the number of service providers per task is large enough, the expected speedup of SLUM with respect to S-MIP is generally given by the, function $\frac{(\mathbf{2})^{\boldsymbol{k}}}{\mathbf{1}+\ \rho}$. At $\rho = 1$, the speedup is $(2)^{k-1}$ and at $\rho = 0$, speedup is $(2)^k$. Hence, theoretically, we learn that the average speedup $\frac{(\mathbf{2})^{\boldsymbol{k}}}{\mathbf{1}+\ \rho}$ is on the interval $[(\mathbf{2})^{\boldsymbol{k-1}}, (\mathbf{2})^{\boldsymbol{k}}]$ for a large enough *n.* Another lesson learned from the theoretical model is that the average speedup is inversely proportional to ρ. The average speedup model suggests that virtual enterprise brokers operating a large number of service providers per task could expect an average speedup on the interval $[(\mathbf{2})^{\boldsymbol{k-1}}, (\mathbf{2})^{\boldsymbol{k}}]$ in practice. For instance, for two task workflow, a speedup between 2 and 4 could be expected. We also conclude that the expected relative SLUM speedup is inversely proportional to the composite service phase transition rate ρ. This analysis also shows that for large enough *n*, SLUM is on average many orders faster than S-MIP. Another important conclusion from the speedup model $\frac{(\mathbf{2})^{\boldsymbol{k}}}{\mathbf{1}+\ \rho}$ is that at $\rho = 1$, we have a speedup $(\mathbf{2})^{\boldsymbol{k-1}}$. This is the average speedup arising from pure sequential decomposition when all service providers are promoted from the SCUM layer to the SPUM layer any speedup is not due to service provider elimination.

5.1.2.3.3 SLUM is much faster than S-MIP on average practically for large values *n* on a two task workflow.

To verify the theoretical speedup model, the study conducted a series of experiments for a range of problem instances with increasing empirical hardness in the number of service providers per task at various values of the composite phase transition rate. The number of workflow tasks was fixed at 2. The summarized empirical results in table 15 of section 4.8.3, table 16 of section 4.8.4, and the Speedup vs ρ graph in figure 38 of section 4.8.3 leads to various conclusions and contributions regarding the average empirical performance of our proposed two layer MIP model versus the state of the art single layer MIP model:

i. The empirical speedup values obtained at all composite service phase transition rates including at ρ =1, are greater than 1. We conclude that, SLUM is not only theoretically faster than S-MIP on average, but also practically faster than S-MIP on average.

ii. Secondly, the empirical speedup values are larger at smaller ρ values and smaller at larger ρ values. This empirical finding verifies the inverse relation between the expected SLUM speedup and the composite phase transition rate as captured in the theoretical model $\frac{(\mathbf{2})^{\boldsymbol{k}}}{\mathbf{1}+\ \rho}$. Hence for a fixed number of service providers per workflow task, virtual enterprise brokers and hence service consumers will experience faster relative speeds at time instants when the composite service phase transition rate is higher than when it's smaller in value.

iii. Thirdly, the empirical speedup ρ value, obtained via L-Hospital's law tends towards the corresponding theoretical value. For example, from table 4.8.3, at $\rho = 0.0296$, expected theoretical speedup is 3.885 while the computed is 3.6. However, all empirical expected speedups are below their theoretical counterparts by some error margin. The conclusion is that although the empirical expected speedup approaches the estimated theoretical speedup and hence fairly approximates the theoretical model, in practice, it's difficult to hit the envisaged theoretical maximum speedup gains (Mulongo et al, 2016a). This is due to sequential computational overheads incurred by the two layer

model that are not theoretically captured into the theoretical model (Mulongo et al, 2016a).

Fourth, the empirical speedup values at smaller ρ values are closer to their corresponding theoretical values than at larger ρ values. This could be attributed to the fact that the sequential overheads experienced by SLUM are larger at larger ρ and vice versa.

5.1.2.3.3 SLUM is about slower than S-MIP for small values of *n* below certain threshold on a two task workflow.

If the two layered model is not faster than the single layered model until a certain minimum number of service providers, then another question arising from RQ1.2 is: *how faster is the single layered model initially better than the layered model*? Using empirical relative complexity (Coffin & Saltzman, 2000), we established that although the initial relative performance of SLUM is also dependent on the transition rate, SLUM is generally slower than the single layer MIP approach initially when the number of service providers is below a certain threshold. Specifically, using the empirical relative complexity analysis method, we established that on a two task workflow, the S-MIP is about 1.3 faster than SLUM initially. Thus the conclusion is that single layered MIP models could be more efficient for small scale webservice composition problems than the two layered approach.

5.1.2.3.5 SLUM is practically asymptotically faster than S-MIP for large values of *n* above a certain threshold on a two task workflow.

In addition the to initial and average performance comparison and in relation to RQ1.2, the study sought to understand the empirical asymptotic performance of the two layer MIP model compared with the single layered model. To do this, we used the empirical relative complexity coefficients (Coffin & Saltzman ,2000). For $\rho = 0.0296$ (much closer to zero), we obtained an empirical relative complexity coefficient of 0.783 compared with the 0.96 obtained when $\rho = 1$). For the remaining five ρ values set at 0.064, 0.6, 0.45, 0.36, and 0.13 yielded empirical relative complexity coefficients between 0.783 and 0.96 that were generally directly proportional to the transition rate. The finding that the empirical relative complexity coefficients are directly proportional to transition rate is a cross validation of the theoretical performance model obtained in chapter two. The finding that at $\rho = 1$, the empirical coefficient is less than 1 means that even without service

elimination, SLUM is not only faster than S-MIP on average but also asymptotically. These results provide an empirical evidence of our earlier theoretical conclusions that decomposition even when done sequentially can lead to improved efficiency.

5.1.2.3.6 On a two task workflow, in practice at a fixed transition rate, the relative speedup of SLUM grows larger as n grows larger and at a fixed number of n.

In regard **to RQ1.3: How much speedup is achievable when using SLUM over S-MIP to autogenerate composite webservices given a business workflow having *n* webservices per task?** The study also investigated the effect of varying the number of service providers per task at a fixed ρ value on the running time of the two layered model compared to the single layered approach. By plotting graphs of speedup vs number of service providers per task, for various ρ values, we observed that the expected speedup of SLUM generally grows larger as the number of service provider's increase. For example at $\rho = 0.36$, $n = 10, 40, 50$ and 70 , the speedups were 1, 1.3, 1.5, and nearly 2 respectively. At $\rho = 0.0296$, the speedups at n =10, and n=70 were 1.5 and 2.6 respectively. In general, we establish that the relative speedup of SLUM at a given value can be determined from the exponential speedup functions derived at a particular value of k and ρ. For example, at $k=2$ and $\rho = 0.0296$, a virtual enterprise broker would expect $0.6893e^{0.017n}$.

The exponential speedup functions can also be used to answer the research question:

RQ1.4: What is the minimum number of service providers per workflow task that a virtual enterprise broker needs to have in order to benefit from the relative efficiency of SLUM when compared to S-MIP?

To determine the minimum number of service provider n_{min} , the exponential speedup function at a particular value of k and ρ can be used to compute n_{min} by setting the desired speedup value. For example, at $k=2$ and $\rho = 0.0296$, to determine n_{min} needed to obtain a 50% gain, one would solve the equation $0.6893e^{0.017n} \geq 1.5$.

5.1.2.3.7 Relative Speedup of SLUM has an elastic limit with respect to the number of virtual enterprises per task.

We observed that the speedup does not grow infinitely with the number of service providers but rather hits a limit. From these results, we make an important contribution towards the scalability/elasticity characteristics of layered MIP algorithms for the webservice composition problem. The contribution is that beyond a certain value of *n*, virtual enterprise brokers would no longer expect any more relative efficiency gains from the two layer MIP model.

5.1.3 Empirical Evaluation of SLUM's Solution Quality and Optimality.

The main research question regarding solution quality was:

RQ2: How does the average solution quality of SLUM compare with that of L-MIP and S-MIP?

The results obtained show that layered MIP has an average solution quality of 93%, which is 7% less optimal than the single layered. However, the same results showed that the local planning MIP approach has an average solution quality of 87%, which is 5% less optimal less than the layered MIP. The finding that L-MIP has an optimality of 87% reinforces the study in (Ardagna, 2007) which established that L-MIP could be an average 20% to 30% worse than the global MIP strategy. The conclusion is that a two layer MIP model is generally suboptimal but could on average produce more quality solutions than the local planning MIP algorithms. Analytically, any scheme exploiting layered optimization is bound to yield a suboptimal solution (Mung, 2006). Therefore the proposed two layer MIP model was hypothesized to be suboptimal. However, the error deviation of the two layered MIP model from the global optimum was unknown.

5.1.4 The Algorithm Selection Problem for the Virtual Enterprise Broker: S-MIP vs L-MIP vs SLUM

Considering the foregoing, the overall and practical contribution is that for virtual enterprise brokers to gain maximum benefit from dynamic webservice composition, there is a need to combine the three techniques, given that none of the methods is adequate in all situations. This transforms to what (John, 1976) terms as *The Algorithm Selection Problem.* In this case, the question becomes, which of the three algorithms should the Virtual enterprise broker use under what circumstances? Our contribution to this is that:-

i. In scenarios where there is no need for global webservice QoS constraints, the local planning mixed integer programming is the most ideal technique to use especially in ultra-low latency webservice enabled collaborative online stock trading platforms. In such web applications, the tolerable waiting limit for end users is 2 seconds (Neilson, 1993;2009) , (Akamai, 2009) & (Nah, 2004)

ii. Below 10 service providers per work task, the difference between SLUM and S-MIP is below 1 second. Where there are requirements for global constraints and strict requirements for 100% optimality, the single layered global planning MIP is better than both SLUM and L-MIP since SLUM does not guarantee global optimality whereas L-MIP lacks support for global constraints and at the same time is suboptimal

iii. Where there is need to address global QoS constraints and the Virtual Enterprise Broker has more than 10 service providers per workflow task, then SLUM is the best tradeoff.

iv. Based on (Mulongo et al, 2015), if there are no strict requirements on timelines and some marginal error in solution quality is tolerable , then SLUM dominates over both S-MIP and L-MIP because S-MIP is less efficient than SLUM while L-MIP is less optimal than SLUM.

v. If the target service consumers average users, and usability is a great concern, SLUM dominates over S-MIP and L-MIP because unlike the rest, SLUM does not require users to directly specify constraints on low technical parameters.

5.1.5 Methodological Contributions

When analyzing and comparing the performance of two or more composite webservice selection algorithms, their running time, and consequently their relative average speedup can be analyzed empirically as functions of the problem input size (in this case, the number of webservices per task is the problem size). Further, the relative initial and asymptotic performance efficiency of two algorithms could be analyzed graphically by plotting running time vs problem size, and even more formally by exploiting the concept of empirical relative complexity as in (Coffin & Saltzman, 2000). All these approaches have been followed in this thesis (see section 4.1 to 4.7). However, when comparing the relative performance of a layered composite service selection algorithm vs a non-layered counterpart or even vs another layered algorithm, a second dimension arises- the *Composite Service Phase Transition Rate, ρ* as defined by the author in this thesis, a method for visualizing the variation of the relative performance of one algorithm with respect to the other as a function of the rate ρ was needed. Previously, no study has explored such a method. In any case as stated earlier, to the best of our knowledge, there exists no any other study that exploited the concept of "a hierarchically layered mixed integer programming "model for the composite webservice selection problem. The study fills the gap through the below contributions:-

5.1.5.1 **Ω-ρ** *Graph*

This type of graph shows how the speedup of an algorithm *B* relative to algorithm *A* varies with increasing values of ρ. For example, in this work, figure 38 in section 4.8.3 shows the **Ω-ρ** graph of our proposed SLUM algorithm with respect to the baseline algorithm S-MIP. The graph quickly tells one that the relative speedup of SLUM with respect to S-MIP declines with increasing ρ value. Where a mathematical model exists that correlates the relative speedup **Ω** with the transition rate **ρ,** two **Ω-ρ** graphs can be plotted, one obtained through empirical analysis and another obtained by substituting certain **ρ** values in the mathematical model. The empirical and theoretical **Ω-ρ** graphs can then be compared where the empirical graph can be used to verify the theoretical model and vice versa. This is exactly what this study did. Using figure 38 in section 4.8.3 as a reference, we have one theoretical **Ω-ρ** graph and one empirical **Ω-ρ** graph. Both of the graphs confirm the same general trend that the speedup of SLUM with respect to S-MIP declines with increasing transition rate, thus validating the theoretical results of section 2.13.

Besides SLUM, future layered approaches to the composite webservice selection problem, may or may not exploit mixed integer programming model. And even if they did, variations in the formulation of the layered approach could be possible. Nevertheless, the invariant concept here is that in any of the layered approaches, candidate composite services are bound to be eliminated as they pass through the layers and thus the notion of "Composite Service Phase Transition Rate" remains. Thus, the **Ω-ρ** graph is envisaged to be a useful tool of algorithm performance efficiency comparison in the context of any layered approaches to composite service selection. In this case, **Ω-ρ** graph becomes a vital visualization tool depicting effect of service elimination through the phases on the relative performance of the new layered algorithm being investigated.

5.1.5.2 β_0-ρ Graph

As explained throughout this study, other than average performance, the initial and asymptotic performance of two algorithms can be compared more formally using the concept of empirical relative complexity analysis as defined and explicated in (Coffin & Saltzman, 2000). The only condition for this kind of analysis is that a *log-log* graph of the running times of both algorithms must yield a graph that is significantly linear. This study takes the work of (Coffin & Saltzman, 2000) further to introduce the notion of β_0-ρ graph. The graph shows how the relative initial performance $\beta_{0,}$ of a layered algorithm *B* varies with the transition rate ρ. For example, in this work, in section 4.8.5, figure 39 shows how the initial performance of our proposed model with respect to S-MIP varies with **ρ.** In our case, the variation although somewhat noisy, shows that the initial performance of SLUM is generally poorer than S-MIP and that the effect of **ρ** is almost negligible.

5.1.5.3 β_1-ρ Graph

This graph is very similar to the β_0-ρ graph except that it shows the relative asymptotic performance of a layered algorithm with respect to another (non-layered) as a function of the transition rate. Figure 40 in section 4.8.5 captures this concept. The conclusion drawn from the **β_1-ρ** graph in figure 40 is that the asymptotic performance parameter β_1 (computed as per Coffin & Saltzman (2000)) and the phase transition parameter ρ are directly proportional to each other. Since according to Coffin & Saltzman (2000) a larger value of β_1 , where β_1 is on the interval [0,1], shows a poorer relative asymptotic performance than a smaller value, and given that from the definition of the parameter ρ, and the derived speedup model $\frac{(2)^k}{1+\ \rho}$, for large enough n, the speedup

is poorer at larger ρ values, then the direct correlation between β_1 and ρ is not a surprising result.

5.2 Limitations of the Study

The generalized theoretic SLUM speedup model $\frac{(2)^k}{1+\ \rho}$ has been be shown to approximately hold in practice using a set of 112 experiments involving sixteen problem instances whose difficulty ranged from 5 service providers per task to 80 problem instances per task (in steps of 5, two tasks per work and seven ρ whose fairly spread between 0 and 1. Although, the empirical ρ values and their theoretical counter parts were converging, the study showed that the empirical values were all below their corresponding theoretical values. A limiting factor explaining this behaviour is the runtime sequential computational overheads that SLUM has to overcome first before getting faster than S-MIP (Abiud et al, 2016a).

Whereas the model $\frac{(2)^k}{1+\ \rho}$ generally suggests that SLUM's speedup would be much larger at larger *k* values, it would have been interesting to perform more experiments involving $\boldsymbol{k > 2}$ e.g $\boldsymbol{k = 2, 3}$ in order to establish the scalability behaviour of SLUM relative to S-MIP at larger $\boldsymbol{k}$ at larger *k* values at different composite service phase transition rates and at different number of service providers per workflow task. This would have helped illuminate the circumstances and conditions under which SLUM is more beneficial than S-MIP for workflows containing more than two sequential tasks.

5.3 Future Work

It would be desirable to explore further work on the scalability behaviour of SLUM with respect to S-MIP on workflows larger than two sequential tasks.

Analytically due to decomposition and layering, the abstraction afforded by our approach inherently shields end users from the burden of specifying weights and constraints on low level

performance attributes such as reliability, throughput etc. However, it would be desirable in future to carryout qualitative usability studies to compare the user experience/ease of use of our model against the baseline model in order to determine the effect of reduced end user QoS parameters on the usability.

Layering as Optimization Decomposition is a science as well an art requiring human engineering effort (Mung, 2006). There are more than one scheme of layering and each layering scheme could lead to varying degree of runtime efficiency and optimality (Mung, 2006) as proven in the communications network research. This being the pioneering work, we have only advanced one of the possibly many MIP layering schemes that could follow from this one. Future work shall explore alternative MIP layering schemes and their relative performance in terms of runtime execution efficiency and optimality benchmarked. A starting point would be to investigate whether reversing the optimization process in our two layer SLUM model, such that optimization begins at the Service Provider Utility Maximization (SPUM) layer followed by the Service Consumer Utility Maximization (SCUM) layer could yield any improvement in the relative speedup as well as optimality of the SLUM model compared to when the reverse order is done. Recall that as per the practical and philosophical considerations of this study (see section 2.10), the study adopted a layered optimization process beginning with SCUM followed by SPUM.

This study pioneered a layered mixed integer approach to the dynamic composite webservice selection problem. Specifically a two layer approach was formulated, again based on realistic applications at the forefront as elaborated in section 2.10. Whether it's for mere theoretical or practical motivations, future work could explore N layered mixed integer programming approaches to the problem, where $N>2$. Obviously, the complexity in analyses is expected to grow larger as the number of layers increases. However, if approached well, new performance insights never conceivable before could emerge.

6 REFERENCES

Afsarmanesh, H., Sargolzaei, M. & Shadi, M., 2012. A framework for automated service composition in collaborative networks. *Collaborative Networks in the Internet of Services*, pp.63–73.

AgileLoad, 2012. Web Applications Performance Symptoms and Bottlenecks Identification, www.agileload.com, last accessed 30th December 2015.

Ahuja, R.K, Orlin, J.B, 1996. Use of Representative Operation Counts in Computational Testing of Algorithms. Informs Journal on Computing, Vol. 8, summer.

Alifarai, M., Skoutas, D., Risse, T., 2010. Selecting Skyline Services for QoS based Web service Composition, April 26-30, Raleigh NC, USA.

Amit G., Heinz, S. ,David, G., 2010. Formal Models of Virtual Enterprise Architecture: Motivations and Approaches, PACIS 2010 Proceedings.

Ardagna, D., Pernici, B. , 2007. *Adaptive Service Composition in Flexible Processes.* IEEE Trans. on Software Engineering. 2007.

Bachmann, F., 2000. Software Architecture Documentation in Practice: Documenting Architectural Layers, Software Engineering Institute, Carnegie Mellon University.

Bakhshi, M., Hashemi, S.M, 2012. User Centric Optimization for Constraint Webservice Composition Using a Fuzzy Guided Genetic Algorithm System. International Journal on Webservice Computing Vol. 3, No 3., September 2012.

Barr, R.S ,2001. Guidelines for Designing and Reporting on Computational Experiments with Heuristic Methods, March 16, 2001.

Bartalos, P., Bielikova, M., 2011. Automatic Dynamic Web Service Composition: A Survey and Problem Formalization, Computing and Informatics Journal, Vol. 30, 2011, 793–827.

Bartz-Beielstein, T., Preub, M.,2014. Experimental analysis of optimization algorithms: Tuning and beyond. In Theory and Principled Methods for the Design of Metaheuristics, pages 205–245,

Springer, 2014.Also available at http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.301.2339&rep=rep1&type=pdf.[Last accessed March, 2016.

Berthold, T. et al, 2012. Solving mixed integer linear and nonlinear problems using the SCIP Optimization Suite. Proceedings of the 24th RAMP Symposium held at Tohoku University, Miyagi, Japan, 27th September 2012. Also available at http://orsj.or.jp/ramp/2012. Available as ZIB-Report 24. [Last Accessed in May 2016].

Blum, A. & Merrick, F., 1997. Fast Planning Through Planning Graph Analysis. Artificial Intelligence. *Artificial Intelligence*. Available at: http://www.cs.cmu.edu/afs/cs.cmu.edu/user/avrim/www/Papers/planning.ps.gz.

Broadwell, P.M., 2004. Response time as a Performability Metric for Online Services, Report No. UCB//CSD-04-1324, Computer Science Division, University of California, Berkeley.

Byod, S., Xiao, L.,Mutapcic, A., 2003. Notes on Decomposition Methods, Notes for EE3920, Stanford University, Autumn, available at http://web.stanford.edu/class/ee392o/decomposition.pdf [Last accessed 4th April, 2015].

Cook, S., 1971. The Complexity of Theorem Proving Procedures, STOC 71, Proceeding of the third ACM Symposium on the Theory of Computing, pp 151-158,1971.

Camarinha-Matos, L.M. & Afsarmanesh, H., 2007. A comprehensive modeling framework for collaborative networked organizations. *Journal of Intelligent Manufacturing*, 18(5), pp.529–542. Available at: http://link.springer.com/10.1007/s10845-007-0063-3 [Accessed November 18, 2016].

Coffin, M.., Saltzman, M.J., 2000. Statistical Analysis of Computational Tests of Algorithms and Heuristics, INFORMS Journal on Computing, Vol. 12, No. 1, Winter 2000.

David, H.C., 2013. Statistical Methods for Psychology, 8th Edition, ISBN-13: 978-1-111-83548-4.

Dongsong, Z., Minder, C., Lina, Z., 2005. Dynamic and Personalized Web Services Composition in E-Business**,** Information Systems Management 22(3):50-65 · June 2005.

Eitan, Z., 1981. Measuring the Quality of Approximate Solutions to Zero-One Programming Problems. Mathematics of Operations Research, Vol. 6, No. 3, August 1981, USA.

Farhan, K.H., Younus, J.M., Saba, B. , 2010. QoS Based Dynamic Web Services Composition & Execution, International Journal of Computer Science and Network Security, Vol 7 (2), February, 2010.

Gabrel, V., Manuovrier, M., Murat, C., 2013. A linear Program for QoS web service composition based on complex workflow. 2013.

H-CL, Yoon, K., 1981. Multiple Criteria Decision Making, Lecture Notes in Economics and Mathematical Systems, Springer Verlag. J Op. Res Soc. Vol 49(3), pp 237-252, March 1998.

Hoos, H.H., 2003. Introduction to Empirical Algorithmics.

Hoos, H.H., 2009. A bootstrap approach to analysing the scaling of empirical run-time data with problem size. Technical Report TR 2009-16, Dept. of Computer Science, University of British Columbia, 2009.

Hoos, H.H., 2014. On the empirical scaling of run-time for optimal solutions to the travelling salesman problem. European Journal of Operational Research, 238(1), 2014.

Hoos, H.H., Mu, Z., 2015. Empirical Scaling Analyser: An Automated System for Empirical Analysis of Performance Scaling, GECCO '15 July 11-15, 2015, Madrid, Spain, ACM ISBN 978-1-4503-3488-4/15/07, DOI: http://dx.doi.org/10.1145/2739482.2764898, last accessed on 22nd March 2016.

IBM, 2004.Patterns: Service-Oriented Architecture and Webservices. Available at http://www.redbooks.ibm.com/redbooks/pdfs/sg246303.pdf [last accessed June 2016].

John, R.R, 1976. The Algorithm Selection Problem. Computer Science Technical Reports, Report No. 75-76, Computer Science Department, Purdue, University.

Kautz H. and Selman B.(1992). Planning as Satisfiability, available online at http://www.cs.cornell.edu/selman/papers/pdf/92.ecai.satplan.pdf. Last accessed on 5th April, 2015

Kautz H., Selman B., 2004. WALKSAT in the 2004 SAT Competition, available online at http://www.cs.rochester.edu/~kautz/papers/walksat.pdf, . Last Accessed in May 2016.

Kelly F.P, Maulloh A,Tan T. ,1998. Rate Control for Communication Networks. Shadow Prices, Proportional Fairness and Stability.

Kitching, M. ,2010. Decomposition and Symmetry in Constraint Optimization Problems, PhD Thesis, Graduate Department of Computer Science, University of Toronto.

Klotz, E., Newman, A.M., 2012. Practical Guidelines for Solving Difficult Mixed Integer Linear Programs.

Kounev S, Gordon I. and Sachs K. (Eds). SIPEW 2008, LNCS, 5119283-302, 2008, Springer Verlag.

Kusiak, A., Larson, N., 1995. Decomposition and representation methods in mechanical design. ASME J Mechanical Design 117(special 50th anniversary design issue):17–24.

Levitin A., 2011. Introduction to the Design and Analysis of Algorithms, 3rd Edition.

Lifschitz V., 2002. Answer set programming and plan generation. *Artificial Intelligence*, Vol. 138, 2002.

Liu, S.C, 2012. Applying Genetic Algorithm to Select Web services Based on Workflow Quality of Service, Journal of Electronic Commerce, Vol 13(2), 2012.

Low, S., 2013. Scalable Distributed Control of Networks of DER, Computing & Math Sciences and Electrical Engineering, Caltech University.

Mahboobeh M. and Joseph G.D, 2011. Service Selection in Webservice Composition. A comparative Review of Existing Approaches, Springer- Verlag Berlin, Heidelberg, 2011.

Mancini, T., Flener, P., Monshi, A.H., Pearson, J., 2009. Constraint Optimization over Massive Databases in Proceedings of the 16th International Conference RCRA workshop (RCRA 2009).

Molina A. , Flores M. ,1999. A Virtual Enterprise in Mexico: From Concepts to Practice", Journal of Intelligent and Robotics Systems, 26: 289-302.

Monsincat A., Binder, W., 2010. Automated Performance Maintenance for Service Compositions, available at http://www.inf.usi.ch/phd/mosincat/adula.htm [last accessed 10th Nov. 2014]

Mulongo, A. W, Opiyo E. T.O, Odongo W O., 2015. A Hierarchical Multilayer Service Composition Model for Global Virtual Organizations, Computer Science and Information Technology 3(4):91-104, 2015. Also available at http://www.hrpub.org/download/20150510/CSIT1-13503138.pdf [Last Accessed July 2016].

Mulongo, A. W, et al, 2016a: SLUM: Service Layered Utility Maximization Model to Guide Dynamic Composite Webservice Selection in Virtual Organizations, Computer Science and Information Technology Vol. 4, No. 2, 2016. Also available at http://www.hrpub.org/download/20160430/CSIT2-13505724.pdf [Last Accessed July 2016].

Mulongo, A.W, et al (2016b): Superlinear Relative Speedup of the Service Layered Utility Maximization Model for Dynamic Composite Webservice Selection in Virtual Organizations, International Journal of Computer and Information Technology (IJCIT), Volume 5, Issue No.4, July 2016. Also available at http://www.ijcit.com/archives/volume5/issue4/Paper050408.pdf [Last Accessed July 2016].

Mung, C., 2006. Layering as Optimization Decomposition, Electrical Engineering Department, Princeton University. Available online at http://www.ece.rice.edu/ctw2006/talks/ctw06-Mung.pdf. [Last accessed 29th April 2016].

Mung C., Low, S.H, A., Calderbank, R., Doyle, J.C. , 2007. Layering as Optimization Decomposition. Current Status and Open Issues, Electrical Engineering Department, Princeton University.

Mung, C.,et al, 2007. Layering as Optimization Decomposition. Ten Questions and Answers, available at http://web.stanford.edu/class/ee360/previous/suppRead/read1/layer_1.pdf [last accessed June, 2016].

Mu, Z.X., Hoos, H.H., 2015. On the empirical time complexity of random 3-SAT at the phase transition, in the Proceedings of IJCAI, 2015.

Nah H.H.,2004. A Study on tolerable waiting time. How long are web users willing to wait? Behavior and Information Technology, Forthcoming.

Ngoko Y. , Goldman A, and Milojicic D. (2013). Service Selection in Webservice Compositions Optimizing Energy Consumption and Service Response.

Nielsen, J. ,1993. Usability Engineering, Morgan Kaufmann, 1st Edition, September, 1993.

Nngroup (2014). http://www.nngroup.com/articles/response-times-3-important-limits/, updated 2014, Last accessed on 4th April, 2015.

Nudelman, E., 2005. Empirical Approach to the Complexity of Hard Problems: PHD Thesis 2005, Stanford University.

Pan, S., Mao, Q., 2013. Case Study on Webservices Composition Based on Multi-Agent System. Journal of Software, Vol. 8, No 4. April 2013.

Picard, W. et al., 2010. Breeding virtual organizations in a service-oriented architecture environment. *SOA Infrastructure Tools: Concepts and Methods*, pp.375–396.

Rabelo J. Ricardo et al (2007). An Evolving Plug and Play Business Infrastructure for Networked Organizations. International Journal of on Information Technology and Management, 2007.

Rabelo, R.J. et al., 2009. An evolving plug-and-play business infrastructure for networked organisations. *International Journal of Information Technology and Management*, 8(3), p.260. Available at: http://www.inderscience.com/link.php?id=24605 [Accessed November 18, 2016].

Rabelo R., Gusmeroli S. (2008). The ECOLEAD collaborative business infrastructure for networked organizations. Pervasive collaborative networks PRO-VE 2008. Springer, New York, 2008.

Rajendran T. and Balasubramanie P., 2009. Analysis on the Study of QoS Aware Webservices Discovery, Journal of Computing Vol. 1(2), December, 2009.

Rainer, A., 2005. Web Service Composition using Answer Set Programming

Rainer, A., Dorn, J. U. , 2009. MOVE: a generic service composition framework for Service Oriented Architectures. IEE Webservices Challenge 2009.

Rao, J. , 2004. Semantic Web Service Composition via Logic Based Program Synthesis, PhD Thesis, Department of Computer and Information Science, Norwegian University of Science and

Rao, J. & Su, X., 2005. A survey of automated web service composition methods. *Semantic Web Services and Web Process Composition*, pp.43–54. Available at: http://link.springer.com/chapter/10.1007/978-3-540-30581-1_5.

Schahram D., Wolfgang, S., 2005. A Survey on Web Services Composition.

Seogewick, R., Flajolet, P. 2009. An Introduction to the Analysis of Algorithms, Second Edition, Princeton University. Available online at http://aofa.cs.princeton.edu/lectures/lectures13/AA01-AofA.pdf . [Last accessed 18th May, 2016].

Seog, C.O., Dongwon, L. & Soundar, R.T.K., 2006. A comparative illustration of Artificial Intelligence.

September 14, 2009 - Akamai Reveals 2 Seconds as the New Threshold of Acceptability for e Commerce Web Page Response Timeshttp://www.akamai.com/html/about/press/releases/2009/press_091409.html, [Last Accessed 4th April, 2015].

Shade, K.O., Akinde Ronke O, A.O. & Samuel, O.O., 2012. Quality of Service (Qos) Issues in Web Services. *IJCSNS International Journal of Computer Science and Network Security*, 12(1).

Singh K.A (2012). Global Optimization and Integer Programming Networks. International Journal of Information and Communication Technology Research.

Simon, G.F., 2009. Measuring Empirical Computational Complexity, PhD Thesis, Department of Electrical Engineering and Computer Science, University of California at Berkeley, USA.

Songqing, S., Gary, W.G, 2009. Survey of Modeling and Optimization Strategies for High Dimensional Design Problems with Computationally Expensive Black Box Functions, Springer Verlag, Published Online August, 2009.

Terlouw, L.I, Albani, A, 2013. An Enterprise Ontology-Based Approach to Service Specification, IEEE Transactions on Services Computing, Vol. 6, NO. 1, January-March 2013.

The OpenGroup (2007). Service Oriented Architecture. Available at https://www.vanharen.net/Player/eKnowledge/service_orientated_architecture_soa_.pdf [Last Accessed June 2016].

Tramontani, A., 2008. Enhanced Mixed Integer Programming, PhD Thesis. Available at https://core.ac.uk/download/files/330/11012104.pdf. [Last accessed on 30th April 2016].

Xu, B. et al , 2011. Towards Efficiency of QoS driven semantic webservice composition for large scale service oriented systems, Sringer, 211, DOI 10.1007/s11761-011-0085-8.

Yan, F., 2012. Global Optimization Method for Web services composition based on QoS, International Conference on Engineering and Business Management, 2012.

Urban, S.D., Gao, L., 2011. A Survey of Transactional Issues for Web Service, Composition and Recovery, Int. J. Web and Grid Services, Vol. 10 (10), 2011.

Yu, L., 2005a. Cmodels for Tight Disjunctive Logic Programs, In Proc. of WCLP, 2005.

Yu, L., 2005b. Disjunctive Answer Set Programming via Satisfiability, In Proc. of Workshop on ASP, 2005.

Zeng, L. et al., 2004. QoS-aware middleware for Web services composition. *IEEE Transactions on Software Engineering*, 30(5), pp.311–327.

Zeng et al (2009). Configurable Composition and Adaptive Provisioning of Web Services3, IEEE Transactions on Service Computing Vol. 2(1), January, 2009.

Zhu, X., 2006. Discrete Two-Stage Stochastic Mixed-Integer Programs with Applications to Airline Fleet Assignment and Workforce Planning Problems, PhD Thesis, Department of Industrial Systems Engineering, Virginia Polytechnic Institute and State University.

W3C (2001). Web Services Description Language (WSDL) 1.1 W3C Note 15 March 2001. Available at http://www.w3.org/TR/wsdl.html.W3C (2013) http://on.cs.unibas.ch/owls-api/index.html [Last accessed 8th July 2013.

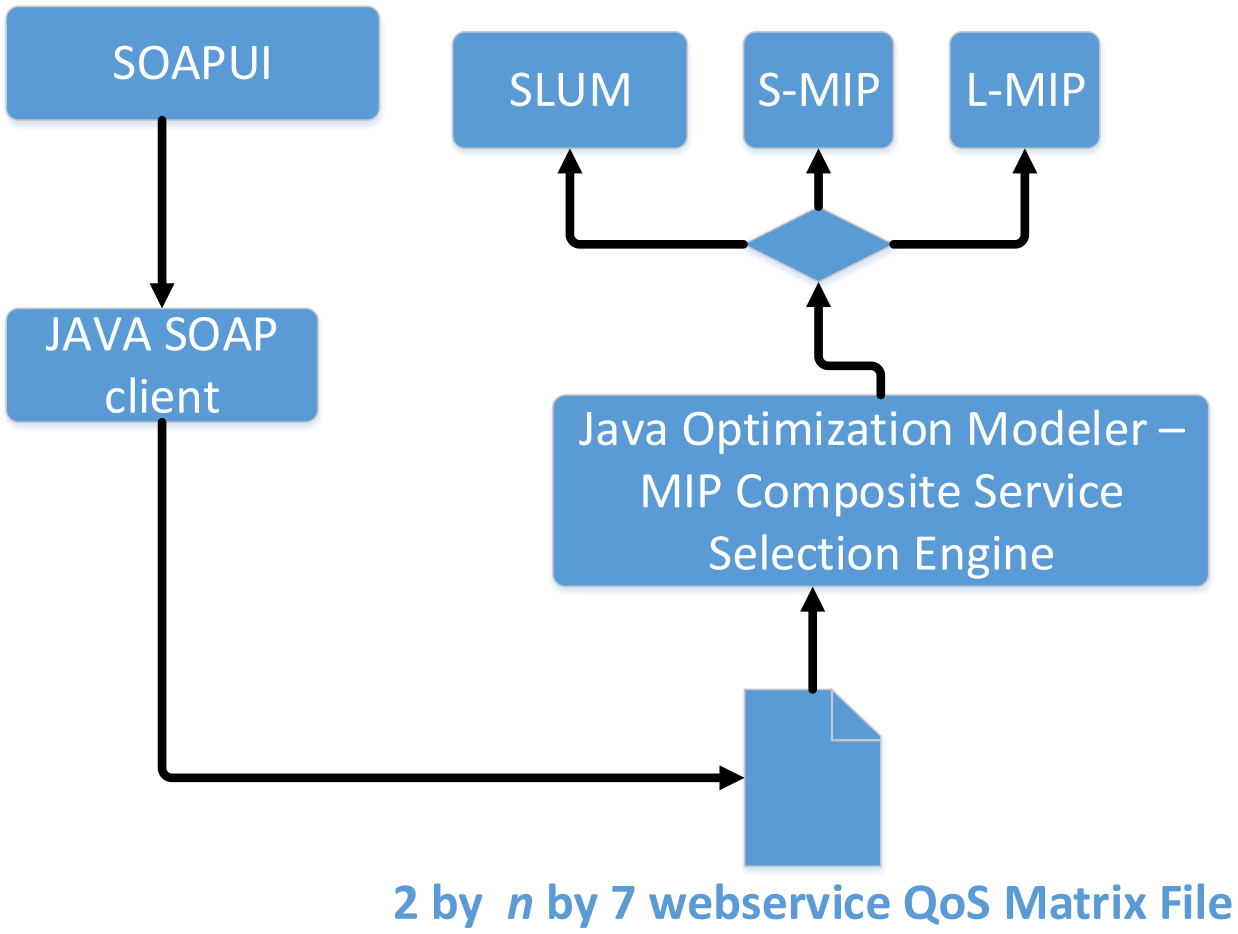
Appendix 1: Composite Webservice Subgraph Program Logic

```
private static double[][][] getWebServiceSubgraph(double[][][] originalGraph,
        int startQoSIndex, int endQoSIndex) {
    double[][][] subgraph = new double[originalGraph.length][][];
    for (int i = 0; i < originalGraph.length; i++) {
        double[][] matrix = originalGraph[i];

        double[][] subMatrix = new double[matrix.length][endQoSIndex - startQoSIndex];

        for (int k = 0; k < matrix.length; k++) {

            for (int j = startQoSIndex; j < endQoSIndex; j++) {

                subMatrix[k][j - startQoSIndex] = matrix[k][j];
            }
        }
        subgraph[i] = subMatrix;
    }
    return subgraph;

}
```

Appendix 2: Composite Webservice Selection Model in Java Optimization Modeler

```
    normalizedSet1 = normalizedAndWeightedGraph[0];
    normalizedSet2 = normalizedAndWeightedGraph[1];
    objectiveFunction = "sum((x1 * c1)+ (x2*c2))";
    op.addDecisionVariable("x1", true, new int[]{1, serviceSet1.length}, 0, 1
    /* decision variable for set 1 services */
    op.addDecisionVariable("x2", true, new int[]{1, serviceSet2.length}, 0, 1
    /* decision variable for set 1 services */

    op.setInputParameter("c1", normalizedSet1);
    op.setInputParameter("c2", normalizedSet2);
    op.setObjectiveFunction("maximize", objectiveFunction);

    if (optimizationMode == 0) {

    op.addConstraint("sum(x1)==1");
    op.addConstraint("sum(x2)==1");
} else {

    op.addConstraint("sum(x1)>=1");
    op.addConstraint("sum(x2)>=1");
    }
```

Appendix 3: Experiment Setup



In the setup above, SOAPUI was used to generate *n* mock webservices per task. As explained in the methodology section, each mock service generated a random vector of 7 QoS values when invoked by a Java SOAP client. The output vector is then store in a file. The data structure stored in the file is three dimension. The first dimension represents the number of workflow tasks, which as explained was fixed at 2 The second dimension is the number of QoS vectors, which map to the number of webservices per task and the last one is the number of QoS attributes, which was fixed at 7. The Java Optimization Modeler (JOM) is a Mixed Integer Programming Library. The composition engine was built on top of JOM. The input to the engine is a 2 by n by 7 graph. The engine based on the configuration, gets the file as input, selects one of the three algorithms to process the input file. The output is one or a 2 by 7 QoS matrix, where the first vector is the best webservice for task 1 and the second vector is the best webservice for task 2.