UNIVERSITY OF NAIROBI

SCHOOL OF COMPUTING AND INFORMATICS

JOB SCHEDULING IN GRID COMPUTING USING SIMULATED ANNEALING

BY

SOLOMON THUO MAINA

A RESEARCH REPORT SUBMITTED IN PARTIAL FULFILMENT FOR THE REQUIREMENTS FOR THE AWARD OF THE MASTER OF SCIENCE IN COMPUTER SCIENCE

AUGUST 2012

DECLARATION

I certify that the research work presented in this report is my original work and has not been submitted for any other academic award at any other university. All work in this research project is my own with exception of acknowledgement of mentioned research work carried by other researchers.

Signature………………………………… Date………………………………….

Solomon Thuo Maina

P58/8961/2006

This report has been submitted in partial fulfillment of the requirements for the Masters of Science Degree in Computer Science of the University of Nairobi with my approval as the University Supervisor.

Signature………………………………… Date………………………………….

Dr. Elisha Opiyo

Project Supervisor

School of Computing and Informatics

University of Nairobi

ACKNOWLEDGEMENTS

I would like to specially thank Dr. Elisha Opiyo for his dedicated supervision and immense patience with me as I undertook this study.

I would also like to thank my family, in particular my dear wife, and friends for their support and understanding during this period of study.

ABSTRACT

Grid computing has emerged as a platform to aggregate heterogeneous resources into a virtual computing resource that can provide non trivial quality of service to end users. A Key problem in grid computing platforms is the mapping of tasks to resources in order to achieve higher performance and economic use of resources. Grid scheduling is a complex problem and has been proved to be an NP Hard problem. Research has been conducted to study intensely the challenges of scheduling in Grid computing environments. In this research we present a model that continuously improves the Makespan, the time spent from the start of the first task in a job to the end of the last task of the job. We use the Alchemi Grid computing platform to implement and evaluate the model. The underlying scheduling algorithm in the model is the simulated annealing algorithm. We have developed a prototype of the model and embedded it in the Alchemi Grid platform as an alternative scheduling module to the default scheduling module that utilizes the First-In-First-Out scheduling algorithm. We have evaluated the model against the default scheduling model in Alchemi and found it to produce better scheduling performance (lower makespans).

TABLE OF CONTENTS

1 INTRODUCTION

1.1 BACKGROUND

Carl and Ian (2002) defined the Grid as “A computational grid is a hardware and software infrastructure that provides depend-able, consistent, pervasive, and inexpensive access to high-end computational capabilities."

Computation grids have in recent years become practical to implement with increase in bandwidth, communication, lower costs of hardware and increase in the number of problems they can solve.

A computational Grid connects geographically distributed computers allowing their computational power and storage capabilities to be shared. With the rapid explosion of data and computational needs, distributed systems and Grids are gaining more attention to solve the problem of large-scale computing.

Grids have been used in many areas to solve complex problems some of the notable areas are:-

- World Wide Telescope. Advances in digital astronomy enable the systematic survey and the collection of vast amounts of data from telescopes gathered all over the world.
- Biomedical Informatics Research Network. A project that brings together biomedical imaging data to be used for research and, hence, to improve clinical cases.
- Virtual Screening on Desktop Computers. A drug discovery application in which, an intra-Grid composed of desktop PCs is used for virtual screening of drug candidates.
- Infrastructure for Multiplayer Games. Buttery.net is a service provider for the multiplayer video gaming industry. It uses Grid technologies to deliver scalable services to game developers.

Task scheduling is an integrated part of parallel and distributed computing. Intensive research has been done in this area and many results have been widely accepted. With the emergence of the computational grid, new scheduling algorithms are needed to address the challenges that have emerged in Grid Computing.

1.2 THE PROBLEM STATEMENT

In Grid computing, job or task scheduling is the assignment or mapping of jobs or tasks to grid resources, one of the main objectives in job scheduling is the minimize or optimize some cost-function specified by the user. Resource provide useful functions such as computing power, memory storage and network bandwidth. Scheduling in Grid computing is an NP-Hard problem Hesham and Ali (1994). There is no single best way to schedule jobs on the available resources. As such, there are numerous techniques used in grid scheduling.

This research focuses on scheduling using an evolutionary algorithm, the simulated annealing algorithm, on an open source grid environment known as Alchemi Grid, developed and maintained by the University of Melbourne. The Alchemi Grid provides scheduling using a greedy algorithm MCT (Minimum Completion Time) with a First-In-First-out queue. This native scheduling algorithm in the Alchemi Grid does not put into consideration the resource characteristics when factoring choice of resource to schedule.

We use the simulated annealing algorithm to develop a scheduling framework capable of improving job scheduling on applications that are submitted to the grid often with time. When better schedules are found, they are stored and retrieved for use when such applications present similar jobs or tasks for scheduling. Existing schedules are replace by new ones with improved scheduling characteristics.

We provide a better scheduling framework to improve scheduling based on historical scheduling information stored by Alchemi grid.

1.3 THE SIGNIFICANCE OF THE STUDY

This research will intensely involve the studying Grid Scheduling Algorithms, development and the implementation of a scheduling framework to optimize a Grid System. The research will contribute in providing:-

- An improved computation grid that can optimize application performance and reduce total time taken to complete tasks with time in particular for applications with high submission rates.

- A grid that can be used for further research areas in grid computing in the University of Nairobi and other academic institutions.

1.4 RESEARCH OBJECTIVES

The research to be conducted has the following objectives:-

- Study and analyze Grid Scheduling algorithms.
- Provide a scheduling framework with the goal of reducing the Makespan (time taken to execute) of the jobs submitted. The scheduling framework continuously optimizes makespans of applications or jobs that are scheduled often with time.
- Implement the scheduling framework in the Alchemi Grid platform.
- Use simulated annealing algorithm as the core optimization algorithm of the scheduling grid.
- Experiment, study and analyze the performance model developed for the scheduling framework in the context of the Alchemi Grid platform.

1.5 SUMMARY

In this research an in depth study will be conducted on scheduling techniques in Computational Grids.

A prototype will be developed and a model for optimization and performance management in the Alchemi Grid.

2 LITERATURE REVIEW

In this chapter, we provide a literature review of the grid computing field, define the problem of scheduling in computing grids, provide taxonomy of scheduling techniques and contextualize our scheduling policy within that taxonomy.

2.1 THE GRID

The term "GRID", in the computing field, was borrowed from the electricity and water grid systems to denote a type of a distributed computing system that would aggregate and provide computing power on demand just as end users consume power and water from the electricity and water grids on demand respectively (Ferreira et al., 2005).

Foster (2002) provides a three point check list for a Grid Computing System:-

- A Grid System coordinates resources that are not subject to centralized control. These resources are distributed across different administrative domains.
- A Grid System is built using standard, open, general-purpose protocols and interfaces. Since Grids coordinates heterogeneous resources across difference administrative domains. Resource discovery, access, authentication, authorization based on standard interfaces reduces complexity in the Grid.
- A Grid System delivers or strives to deliver nontrivial quality of service. A Grid System coordinates resource consumers and providers to optimally provide the required quality of service.

Distributed systems can be classified as grids if they exhibit characteristics (Buyya & Murshed, 2002):-

- Multiple Administrative Domains and Autonomy. Resources in a Grid tend to be geographically distributed and across multiple administrative domains i.e. organizations. The key challenge introduced is the autonomy of resources and control by different administrative domains.
- Heterogeneity. A Grid is generally composed of different resources composed of different technologies from laptops to personal computers to super computers.

- Scalability. A Grid can grow and shrink in size dynamically as resources enter and leave the Grid System. This further compounds the problem of bandwidth and latency common found in distributed systems.
- Dynamicity and Adaptability. A Grid consists of a high number of resource providers and consumers. Managing this highly dynamic environment is a key problem. Grids must cope with resource failures and unavailability efficiently and effectively.

2.2 TYPES OF GRIDS

Implementation of real world grid systems is along certain functions or specializations and the types of resources aggregated. Consequently, this has given rise to categories of Grid Systems. The common classification of Grid Systems is by type as follows (Ferreira et al., 2005):-

- Computational Grids
- Data Grids
- Network Grids

2.3 COMPUTATIONAL GRIDS

Computational Grids combine the computing power of individual resource to provide a virtual singular computing resource with the objective of processing jobs faster.

2.4 DATA GRIDS

In Data Grids, the key resource being abstracted is storage. The data storage resources of the Grid are aggregated and provided as the main shared resource. A Data Grid can allow large datasets to be accessed transparently and can be view a massive data storage system composed of numerous storage resource.

2.5 NETWORK GRIDS

The Network Grid is also known as the Delivery Grid. This type of Grid provides redundancy and high performance communication services. It serves to provide fault tolerant communication and additional services such as caching between communication points.

2.6 ARCHITECTURE OF A GRID

Grid Systems have a general architecture consisting of the layers below (Foster et al., 2001):-

- Application Layer
- Collective Layer
- Resource Layer
- Connectivity Layer
- Fabric Layer

The figure below illustrates the interconnection and relationships between the layers.

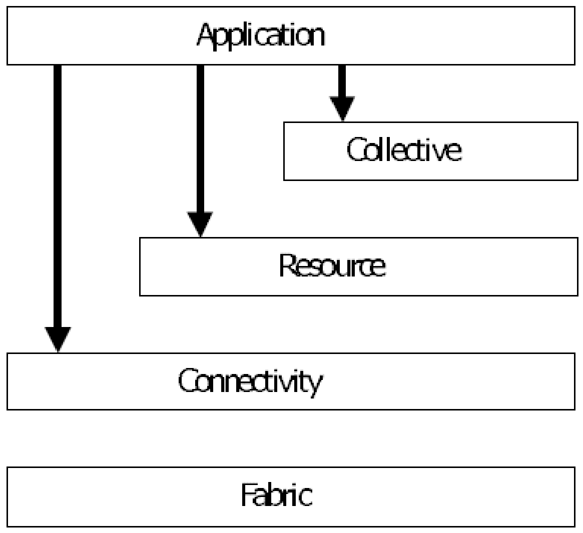

Figure 1: The layered Grid Architecture

2.6.1 Fabric Layer

The Grid Fabric layer provides the physical resources to be shared e.g. computational resources, storage systems, catalogs, network resources and sensors. The fabric provides access control polices to the resources.

2.6.2 Connectivity Layer

The connectivity layer provides the authentication, access control and communication protocols required by the grid. Communications protocols facilitate data transfer between Fabric layer resources. Equally authentication protocols provide secure channels of communication; in particular, they provide tools to identify users and resources.

2.6.3 Resource Layer

The resource layer builds on the connectivity layer protocols and defines protocols for individual resource management. Key functions include secure negotiation, initiation, monitoring, control, accounting, and billing of shared operations on individual resources.

2.6.4 Collective Layer

The Collective layer provides protocols and services across collections of resources. It cuts across collections of resources as opposed to the resources layer that deals with individual resources. The collective layer manages interactions across collections of resources. Typical resources, services and software utilities in this layer include:-

- Directory services
- Co-allocation, scheduling, and brokering services
- Monitoring and diagnostics services
- Data replication services
- Grid-enabled programming systems
- Workload management systems and collaboration frameworks
- Software discovery services
- Community authorization servers
- Community accounting and payment services
- Collaboration services

2.6.5 Application Layer

The uppermost layer the Application Layer consists of end user applications that run on the Grid System environment. Applications have access, through Application Programming Interfaces and Software Development Kits (SDKs), to the layers below to access the Grid System capabilities and functions to provide the desired outcomes.

2.7 GRID SCHEDULING

For a Grid System to fulfill the objective of providing non trivial quality of service, A Grid must manage its resources effectively.

Scheduling in Computational Grids involves the efficient and effective mapping of tasks or jobs to resources, which may be distributed in multiple administrative domains, for execution.

2.7.1 Components of a Grid Scheduler

Grid Scheduling occurs in three stages, that will be elaborated in the sections that follow, namely: - resource discover and filtering, resource selection and scheduling and job submission (Schopf, 2004). From a functionality view point, Figure 2 below provides the logical architecture of a typical Grid Scheduling Sub-System.

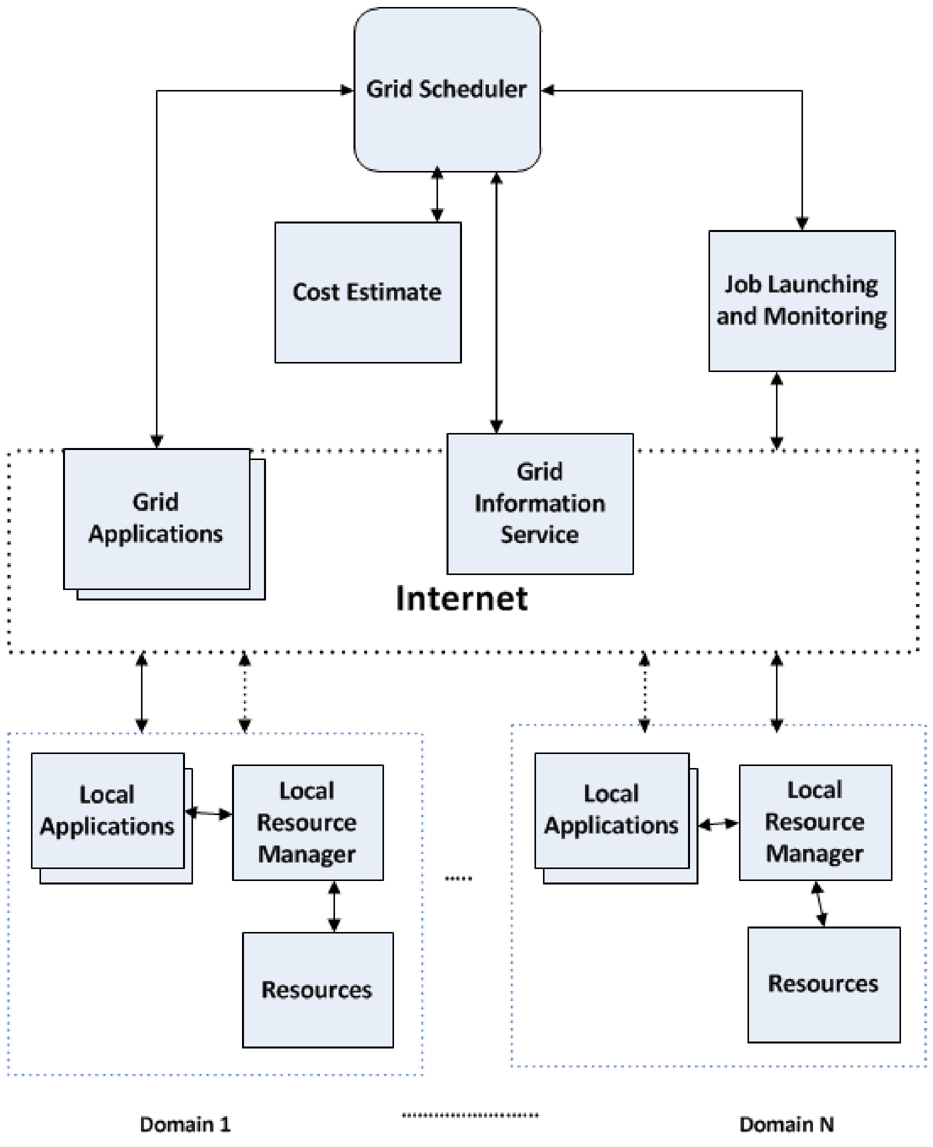

Figure 2: A Logical Grid Scheduling Architecture

A Grid Scheduler is the subsystem in a Grid platform that receives applications from Grid users, selects resources for these applications on the Grid and generates application to resource mappings based on specified objective functions and predicted resource performance. The Scheduler relies on the Grid Information Service to obtain information about the state of Grid resources and applications.

A unique feature of Grid Schedulers is that, they usually do not control Grid resources directly but rather act as intermediaries. This is due to the reason that Grids tend to span multiple administrative domains and resources are under control of a local resource manager and policy.

Grid Information Service Module

The Grid Information Service (GIS) module fulfills the role of providing information about the status of available resources to the grid scheduler. The GIS is responsible for collecting and predicting resource state information such as CPU processing capacity, memory size, network bandwidth, storage requirements, subtask dependencies in jobs, software available and load of a site or sites at a given time.

Properties about applications to be scheduled are important when making feasible schedules. Properties such as approximate instruction quantity, memory, storage and dependencies are often required. The GIS mainly uses application profiling and analogical benchmarking techniques to build suitable performance models. Application profiling (AP) is used to obtain properties of applications where as analogical benchmarking (AB) provides a measure of how well a given resource can perform a given type of a job.

Cost Estimate

The Cost Estimate module uses the information obtained using the AP and AB techniques and the preferred performance model to produce the cost estimate of candidate schedules. The scheduler chooses a schedule among the proposed schedules to best optimize the defined objective functions.

Launch and Monitoring Module

The Launch and Monitoring module (LRM), also known as the binder, provides executes a selected schedule by submitting applications to mapped resources, staging input data and executable programs where required and monitoring their execution.

Local Resource Manager Module

The Local Resource Manager (LRM) is responsible for local scheduling inside a resource domain and reporting resource information the GIS. Local resource scheduling entails local scheduling of Grid application jobs and also jobs from the local domain.

2.7.2 Stages of Grid Scheduling

.As put forward by Schopf (2004), Grid Scheduling involves three main phases: - Resource discovery, System selection and Job execution. These phases, and the steps that make them up, are shown in Figure 3.

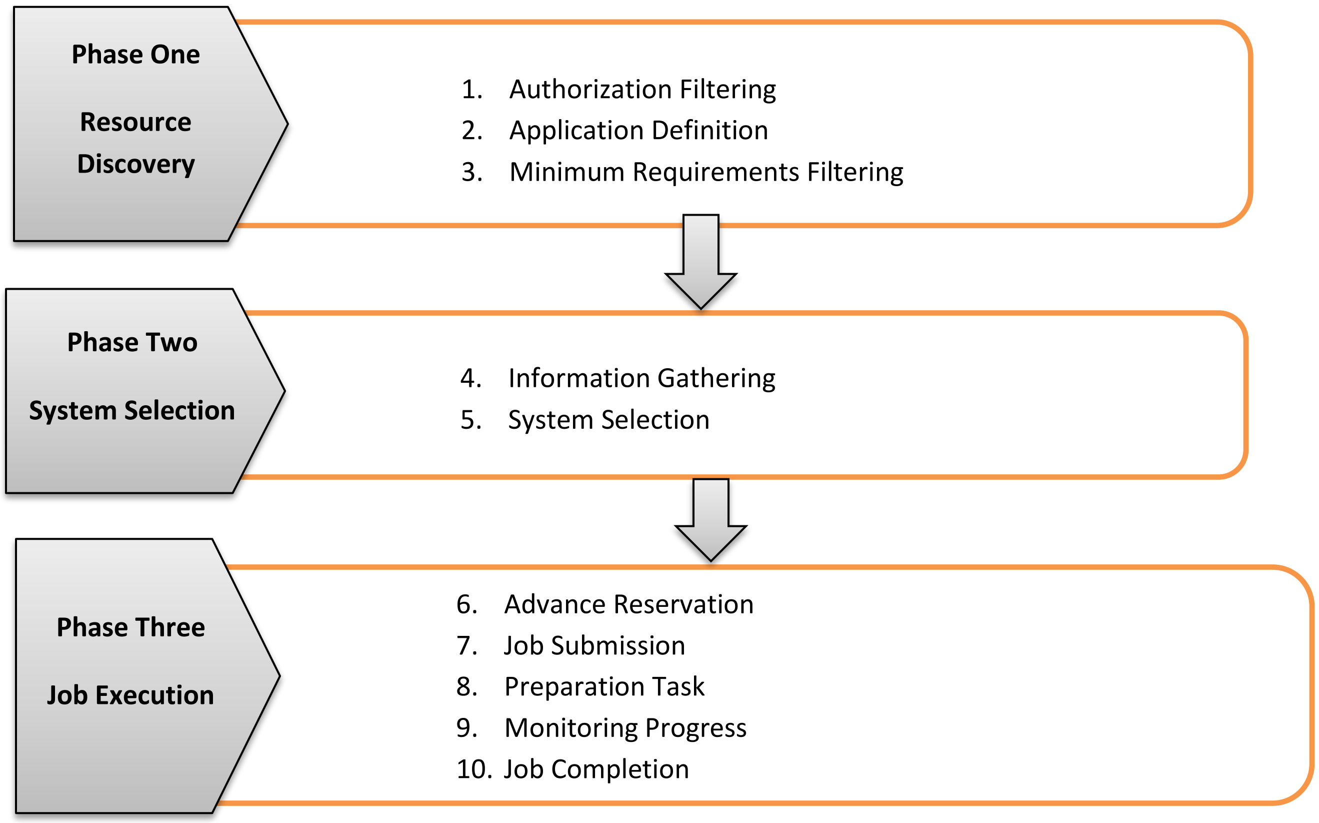

Figure 3: Three Phases of Grid Scheduling

2.7.2.1 ***Resource Discovery***
The resource discovery phase provides a set of potential resources that have passed the minimum requirements. This list serves as an input to the second phase, System Selection. Resource discovery is performed in three main steps:-

- Authorization filtering
- Application requirements definition
- Minimum requirements filtering

Authorization Filtering

This step provides secure access to resources on the Grid. Users have access only to resources they have permission to on the Grid. The available list of resource is filtered based on the user credentials.

Application Requirements Definition

In this step the user specifies the minimum requirements needed for the job(s) to be executed. These details can be categories into:-

- Static requirements such as hardware specifications, architecture and Operating System(s).
- Dynamic requirements such RAM, CPU and disk space that vary with time.

Minimum Requirements Definition

The application requirements information is used to match the resources available and further filter the potential resource list.

In this step resources that do not meet the minimum requirements are filtered out.

2.7.2.2 ***System Selection***
The System Selection phase involves strategies that select a single resource to schedule the job from the list of candidate resources. The selection is generally performed in two steps:-

- Dynamic information gathering
- System selection

Dynamic Information Gathering

This step entails collecting dynamic information about the resources and passing it to the user. This information is necessary to make the best job to resource mapping.

System Selection

Dynamic information gathered about resources is used to select the qualifying resource or set of resources to use. Various approaches are used including heuristics.

2.7.2.3 *Job Execution*

This phase involves the execution of jobs which includes file staging and cleanup. This phase occurs in six steps:-

- Advance reservation
- Job submission
- Preparation tasks
- Monitoring progress
- Job completion

Advance Reservation

Advance reservation entails reserving resources for use at a future time. This is sometimes necessary in-order to get the best or guaranteed performance from the Grid System.

Job Submission

Once resources are ready, the next step is job submission. This step entails submitting the application to the resources. Job submission varies in complexity depending on the type of application or job and the requirement. Job submission may involve the use of scripts, staging and setup.

Preparation Tasks

The preparation step includes acquiring a reservation, setup and staging of the application or job. In complex preparation steps, FTP or file transfers may be necessary to ensure the data is in place for use.

Monitoring Progress

Monitoring of progress is necessary so as to inform users on the status of their jobs or applications and any issues arising during execution.

Job Completion

When jobs terminate normally, end users need to be notified of the job or task completion status.

2.7.3 The Grid Scheduling Problem

Scheduling in Computational Grids presents key challenges when it comes to coordinating resource sharing in dynamic and multi-organizational environments. Resources that constitute a Grid are varied; these include computers, storage, software applications and data, connected through high speed networks and middleware software. The middleware software in overall provides security, monitoring, resource management and other services.

2.7.3.1 *Challenges of Scheduling In Grid Computing*

The problem of scheduling has been thoroughly researched in traditional parallel and distributed systems. A lot of work in the Grid computing scheduling space has been borrowed from traditional parallel and distributed computing. Many of the early algorithms were carried over and have over time been optimized and new ones developed to cope with grid scheduling challenges.

Traditional parallel and distributed systems have the following characteristics (Berman, 1998):-

- All resources exist in a single administrative domain.
- The system scheduler has control and manages all of the resources.
- The resource pool tends to be homogenous in composition.
- Resource contention is managed in a predictable way according to set local policies. Thus, impact of contention on application performance is sufficiently predictable.
- Movement of data for staging and subsequent computations is handled in a highly predictable process such that it can be viewed as a constant overhead.

The problem of scheduling in computational grids differs from the traditional distributed and parallel systems in that (Zhu, 2003):-

- Heterogeneity and Autonomy
- Performance Dynamism
- Resource Selection and Computation-Data Separation

Heterogeneity and Autonomy

A Grid system typically spans across multiple administrative domains that are geographically distributed. As a result, resources that constitute the Grid are varied in architecture, capabilities and capacity. These resources are not under direct control of the Grid. They managed autonomously in their respective domains.

These two factors pose a great challenge to Grid schedulers in ensuring non trivial quality of service is delivered to end users. Heterogeneity creates the challenge in that there are differences job processing and data access capabilities amongst the resources. Autonomy makes it difficult for Grid Schedulers to estimate the cost of executing a task on given resources since they cannot violate local policies. Under such circumstances, the scheduler attempts to make the best job to resource mapping.

Performance Dynamism

A Grid system is a highly dynamic environment. The status of the resources changes over time. A resource can be engaged by the local resource manager to perform a high priority task thus impacting negatively of the grid tasks. Additionally, new resources can become available and existing ones can become unavailable thus changing the overall computing capabilities of the grid as time progresses. Thus Grid schedulers have to contend with performance fluctuations.

Resource Selection and Computation-Data Separation

In a Grid System, resources and data tend not to be in the same site but are rather geographically distributed. The cost of moving application executable code and data to the resources selected to perform the job cannot be ignored. Bandwidth becomes an important factor when choosing resources to perform tasks.

2.8 GRID SCHEDULING ALGORITHMS

Scheduling in Grid computing is an NP-Hard problem; there is no single best way to schedule jobs on the available resources. Pairing of tasks or jobs to resources is an NP-Hard problem (Rewini, Lewis & Ali, 1994). As such, there is no such a single algorithm that gives the best performance depending based on selected objective functions of scheduling.

The common approach is to select the most suitable algorithm, for scheduling purposes, given the characteristics of the jobs, resources and network connectivity in some cases.
Previous research on scheduling algorithms in traditional distributed and parallel systems has yielded poor results because grid systems and traditional distributed and parallel systems do not compare directly.

2.8.1 Objective Functions

In a Grid Computing environment, there are two major parties; resource consumers and resource producers. These two parties are driven by often different and conflicting objectives or goals. Resource consumers are mainly Grid users who submit various applications and a mostly concerned with the performance of their applications e.g. the total time taken to run the application. Resource producers on the other hand provide resources for sharing in the Grid environment. Resource producers are mainly concerned with the performance of their resources e.g. the resource utilization in a given period and sometimes the economic benefits associated with the utilization.
Object functions in Grid Scheduling provide a means to classify the conflicting objective of consumers and producers. They can be classified into two main categories: - application centric and resource centric (Zhu, 2003). Figure 4 shows the objective functions.

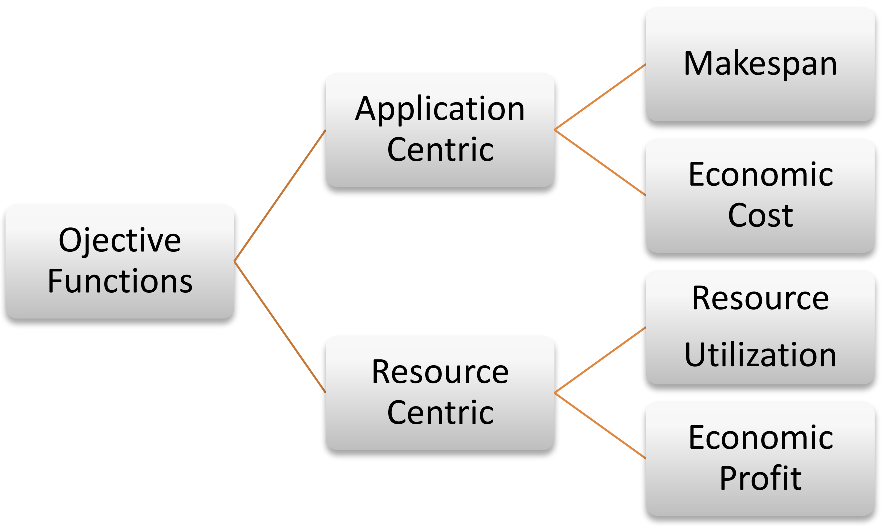

Figure 4: Objective Functions

2.8.1.1 *Application Centric*

Application centric scheduling algorithms aim to optimize the performance of each application. The most common application centric objectives are Makespan and Economic Cost.

Makespan

Makespan is the time spent from the start of the first task in a job to the end of the last task of the job. This is the commonest application centric objective function. The key goal in such scheduling algorithms is to minimize the Makespan.

Economic Cost

In today's distributed computing environments, it is not uncommon to find resources that charge for their use. In such a case the objective is to minimize the consumption of resources and subsequently lower the economic costs.

2.8.1.2 *Resource Centric*

Scheduling algorithms with resource centric scheduling objectives are concerned with the optimal performance of the resources, in particular resource utilization and the economic profit derived from their use.

Resource Utilization

Under resource utilization objectives two key objectives are to maximize:-

- Throughput – the ability of resources to process a certain number of jobs in a given period of time.
- Utilization – the percentage of time a resource is engaged. Low utilization signals idle resources and wastage.

Economic Profit

In Grid models where economic models are used, resource providers are concerned with maximizing the total economic benefits derived from the usage of their resources.

2.8.2 Taxonomy of Grid Scheduling Algorithms

Casavant & Kuhl (1988) provided taxonomy of scheduling algorithms in general purpose parallel and distributed computing systems. The taxonomy applies to Grid Scheduling algorithms by virtue of a Grid system being a special type of a distributed system. Scheduling algorithms fall within a subset of this taxonomy. The figure below shows the graphically illustrates the taxonomy (Casavant & Kuhl 1988).

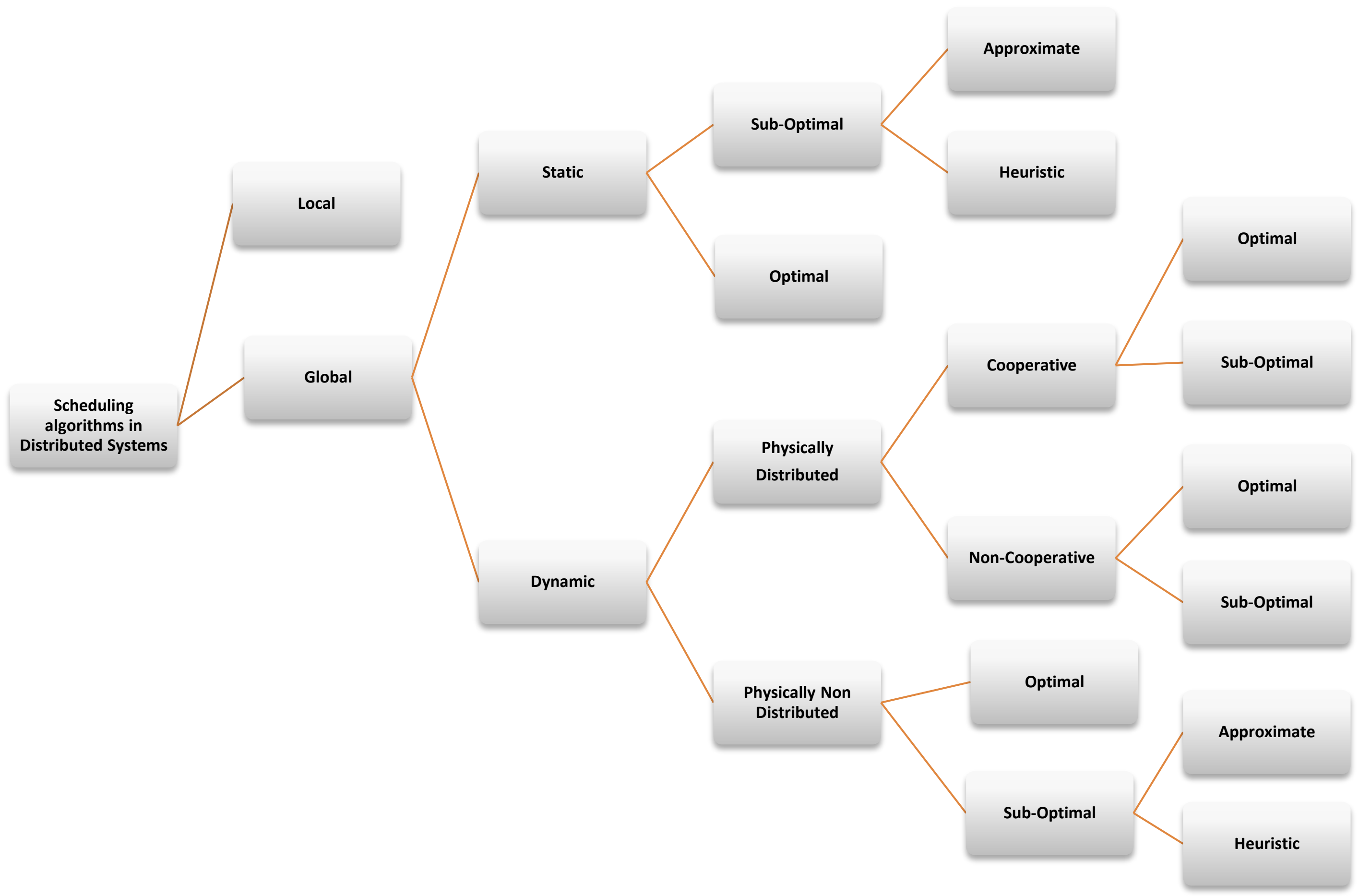

Figure 5: A hierarchical taxonomy of scheduling algorithms

2.8.2.1 *Local Scheduling versus Global Scheduling*

At the highest level, scheduling algorithms can be classified under local and global categories. A Local scheduling policy is concerned with allocating resources to processes on a single CPU. Global scheduling policy on the other hand uses the information about the system to allocate processes to multiple processes to optimize global performance objectives. Grid Scheduling algorithms are classified under the global branch.

2.8.2.2 *Static versus Dynamic*

At this level, the selection takes into account of the time when the scheduling decisions are made. Dynamic scheduling policies make task to resource mappings on the fly as the application executes. Dynamic scheduling applies when it is difficult for the scheduler to make estimates about the running time of applications. In static scheduling, a task comprising a job is assigned once to a resource. The cost estimate of running the application can be determined before the application is executed on the Grid. The static model has the advantage of providing an overall view of tasks and costs of computing those tasks.

The key advantage of static scheduling is its simplicity, being able to determine easily the cost of the tasks. The problem it introduces is that of poor coping with eventualities such as:-

- Performance dynamism of the resources.
- Unavailability of the resources.
- Changes in communication bandwidth.

Dynamic schedules have the advantage of not requiring a run-time profile of an application before execution. Dynamic scheduling is achieved through the techniques below:-

- Unconstrained First-In-First-Out (FIFO)
- Balanced constraints technique.
- Cost constrained techniques.
- Hybrid of static and dynamic techniques.

Unconstrained FIFO

This is an opportunistic load balancing technique with a greedy perspective. The resource with the shortest waiting queue of shortest waiting queue time is selected for task allocation. The key advantage is its simplicity but it is far from optimal.

Balanced Constraints Techniques

This approach rebalances loads on all resources periodically by shifting tasks from one waiting queue to another. This approach has the advantage of distributing tasks to resources evenly. The approach suffers from communication delays in environments with dynamic communication resources.

Cost Constrained Techniques

This approach is an improvement of the balanced constraints techniques. The key difference is that this level includes the cost of communication.

Hybrid of Static and Dynamic Techniques

This category of scheduling algorithms takes advantage of the strengths of static and dynamic policies. Static scheduling is applied on the parts of the application that always execute and dynamic on the other parts that are uncertain.

2.8.2.3 *Optimal versus Sub-Optimal*

In scheduling scenarios where it is possible to obtain the information about the state of resources and jobs, it optimal task to resource mappings could be made based on some objective function such as Makespan and maximum resource utilization. But since the problem of scheduling is NP-Complete and further complicated by performance dynamism of resources, it is a challenge to prove the optimality of scheduling algorithms. Current research, attempts to find sub-optimal solutions.

2.8.2.4 *Approximate versus Heuristic*

Suboptimal solutions are further categorized into approximate and heuristic classes. For computationally infeasible problems, approximate algorithms avoid searching the entire solution space for the most optimal solution, but are rather satisfied when they arrive at a sufficiently good solution.

The factors which determine whether this approach is worthy of pursuit include (Casavant & Kuhl, 1988):

- Availability of a function to evaluate a solution.
- The time required to evaluate a solution.
- The ability to judge the value of an optimal solution according to some metric.
- Availability of a mechanism for intelligently pruning the solution space.

Heuristic category of algorithms under the sub-optimal branch represents the algorithms which make the most realistic assumptions about a priori knowledge concerning process and system loading characteristics. It also represents the solutions to the scheduling problem which cannot give optimal answers but only require the most reasonable amount of cost and other system resources to perform their function. The evaluation of this kind of solution is usually based on experiments in the real world or on simulation.

2.8.2.5 ***Physically Distributed versus Non Physically Distributed***

In Physically distributed scheduling scenarios involve decentralized scheduling, where the job of scheduling in the dynamic grid environment is shared by physically distributed schedulers. Non-physically distributed refers to scheduling scenarios where the scheduling functions are centralized. The centralized strategy is easy to implement but difficult to scale. On the other hand decentralization of scheduling provides scalability but is difficult to implement.

2.8.2.6 ***Cooperative versus Non-Cooperative***

The degree of autonomy of the nodes when scheduling tasks determines whether the algorithms can be further classified under cooperative and non-cooperative categories. Under the non-cooperative category, individual schedulers act autonomously with regard to how their resources should be used and optimization independent of the effects of the decision on the rest of system. Good examples of non-cooperative schedulers are the application level schedulers.

In the cooperative case, each Grid scheduler executes its share of the scheduling task in concert with other Grid schedulers to satisfy a common system-wide goal. The local policy of the scheduler makes scheduling decisions such that they try to achieve a global goal.

2.8.3 Task Dependencies

A task in a grid application is termed as a dependent task if it cannot start until all its parents are done. An independent task has parent task and can be scheduled to execute independently of other tasks.

The common dichotomy used is dependency versus independency. Figure 6 illustrates this dichotomy.

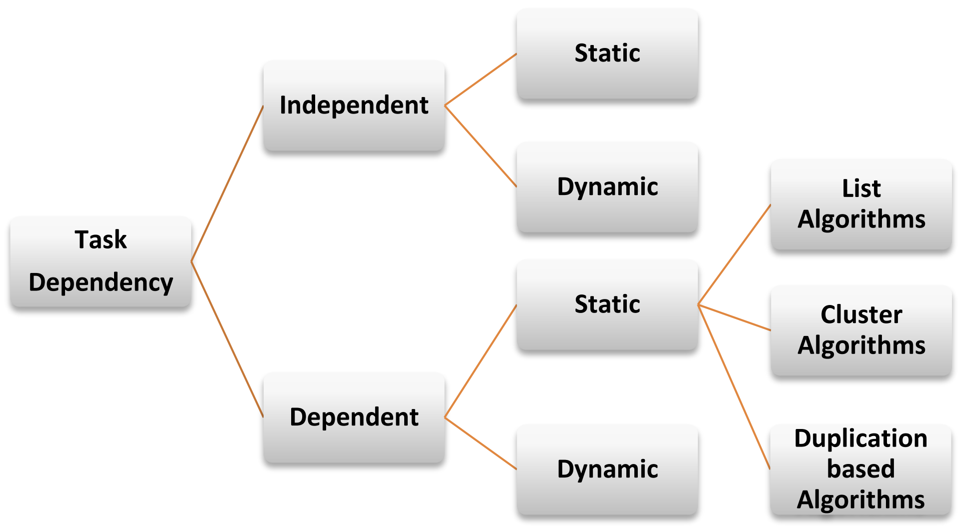

Figure 6: Task dependency taxonomy of Grid scheduling algorithms

2.8.3.1 *Independent Task Scheduling*

From the Grid system perspective, the strategy is to assign independent tasks to resources available to maximize certain objective functions.

Static independent algorithms work with a priori execution cost estimate of tasks. Static algorithms with performance estimates include:-

- MET (Minimum Execution Time): MET performs a task to resource mapping to the resource with the best execution time for that task irrespective of whether the resource is available or not at that time.
- MCT (Minimum Completion Time): MCT performs a task to resource mapping arbitrarily to any resource with the minimum expected completion time for that task.
- Min-Min: The Min-min heuristic begins with the set U of all unmapped tasks. Then, the set of minimum completion time M for each task in U is found. Next, the task with the overall minimum completion time from M is selected and assigned to the corresponding machine (hence the name Min-min). Last, the newly mapped task is removed from U, and the process repeats until all tasks are mapped.
- Max-Min: The Max-min heuristic is very similar to Min-min. It also begins with the set U of all unmapped tasks. Then, the set of minimum completion time M is found. Next, the task with the overall maximum from M is selected and assigned to the corresponding

machine (hence the name Max-min). Last, the newly mapped task is removed from U, and the process repeats until all tasks are mapped.

- XSuffrage: The rationale behind Suffrage is that a task should be assigned to a certain host and if it does not go to that host, it will suffer the most. For each task, its suffrage value is defined as the difference between its best MCT and its second-best MCT.
- Task Grouping: Under Task Grouping, tasks are grouping according to their computational requirements. Tasks in the same group are sent to the same resource which completes the in a given time.

Static algorithms are affected by the heterogeneity of tasks and resources and thus cannot be used in all scheduling scenarios.

Dynamic algorithms for independent task scheduling work without performance estimates of the tasks.

2.8.3.2 *Dependent Task Scheduling*

The most common model for dependent task scheduling is the directed acyclic graph (DAG). In DAG the node s of the graph represent the task and the edges the precedence order. A DAG can carry additional information on its nodes and edges in the form of weights. Weights can carry metrics such as computational and communication costs.
Just as in independent task scheduling, static algorithms work with a priori execution cost estimate of tasks. Dynamic algorithms on the other hand are able to work without such information.

List Heuristics

Under list heuristics, tasks are assigned priorities and placed in a list ordered in decreasing magnitude of priority. When resource contention occurs, the selection of tasks is based on the priority, with the task with the higher priority having a higher precedence.

Clustering Heuristics

Under clustering heuristics, tasks with common characteristics are clustered together. During scheduling tasks in the same cluster are assigned to the same resource.

Duplication Based Algorithms

Duplication based scheduling is utilizing resource idle time to duplicate predecessor tasks. This may avoid the transfer of results from a predecessor to a successor, thus reducing the communication cost. So duplication can solve the max-min problem.

2.8.4 Non Traditional Approaches to Grid Scheduling

Natural and human society can be compared to grid systems. They share common characteristics such as consumers, producers, diversity, geography, resource contention and so on. Natural systems seem to cope with resource management problems and have done so for centuries. The relative success has drawn researchers in Grid computing and resulted in active research. Figure 7 below provides taxonomy of non-traditional scheduling approaches.

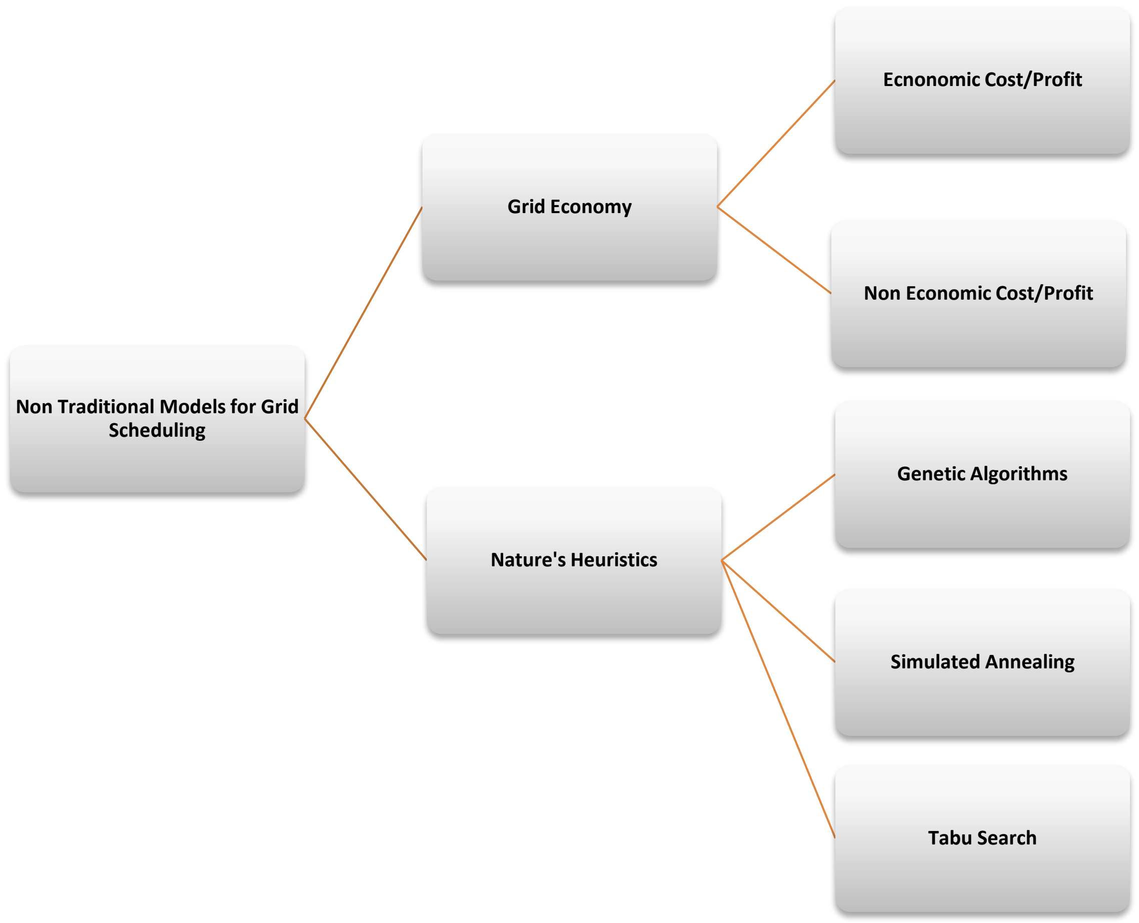

Figure 7: Taxonomy of non-traditional scheduling approaches

2.8.4.1 *Grid Economic Model*

A Grid system has consumers and producers each with their own objectives often conflicting. The Grid environments tend to be very dynamic, decentralized and competitive with applications vouching for resources that are limited. This can be likened to a market with producers and consumers. Based on the notion of a market, economic models for resource management and scheduling have been developed to optimize certain objective functions (Buyya et al., 2002).

A typical market is composed of producers, consumers and commodities. These can be mapped directly to resource owners, resource users including applications and computing resources respectively. Economic theories are established based on the study of market behaviors. Price and quality of commodities are parameters to the decision making procedures of consumers and providers e.g. consumers normally want to get better services (e.g., a smaller Makespan) with a budget as small as possible (cost), while a providers normally want to get a higher resource utilization to raise its profits.

Interaction amongst consumers and producers in a Grid system using economic models use methods such as bargain, bid and auction. Buyya et al., (2002) apply the Commodity Market, Tender / Contract-Net and Auction models to the Grid computing scheduling space.

2.8.4.2 *Natures Heuristics*

Nature provides numerous examples of large complex systems that resemble parallel and distributed systems. In many ways, these natural systems share common characteristics with Grid computing systems. Properties such as non-determinism, parallelism and adaptability are the common to both natural systems and grid computing environments (Abraham, Buyya & Nath, 2000). The three main algorithms inspired by nature are:-

- Genetic Algorithms (GA)
- Simulated Annealing (SA)
- Tabu Search (TS)

Genetic Algorithms

GA is a large space search evolutionary technique that closely mimics the natural processes of genetics. The goal of a GA search is to find the best chromosome with the optimal fitness value. Crossover is the process of swapping certain sub-sequences in the selected chromosomes. Mutation is the process of replacing certain sub-sequences with some task-mapping choices new to the current population. Both crossover and mutation are done randomly.

A GA algorithm contains the following main steps:-

- Population generation
- Chromosome evaluation
- Crossover and mutation

Population Generation

A population is a set of chromosomes each of which is a potential schedule. The initial seed population is generated using other heuristic algorithms such as Min-min.

Chromosome Evaluation

Each chromosome is evaluated by fitness function against some fitness value such as the Makespan.

Crossover and Mutation

Crossover operation selects a random pair of chromosomes and selects a random point in the first chromosome. The operation selects portions of the chromosomes from that point to the end and exchanges sections within. This translates to machine and task reassignments. The mutation operation randomly selects a task within a chromosome and randomly reassigns it to a new machine.

Chromosomes from this mutated population are reevaluated severally until a stop criterion is met.

The stopping criteria during the reevaluation iterations occur when be:-

- No improvement in recent evaluations
- All chromosomes converge to the same mapping
- A cost bound is met.

Simulated Annealing

SA like the GA is large space search technique based on the physical process of annealing. Annealing is a metallurgical process of obtaining low-energy crystalline state of a solid. At the beginning, the temperature is increased to very high levels until the solid melts. The temperature is subsequently and iteratively lowered. The gradual lowering of the temperature allows the atoms of the melted solid to arrange locally, in a thermal equilibrium that is an optimal state.

By analogy, the thermal equilibrium is an optimal task to resource mapping, the temperature is the total completion time of the mapping and the change of temperature is the process of change in a task to resource mapping.

At high temperature less optimal mappings exist and as temperature lowers more optimal mappings are chosen and the likelihood of accepting worse mappings is probabilistically minimized. An implementation of the simulated annealing algorithm for Grid scheduling is described as follows (Braun et al., 2000):-

- Generation of the initial solution
- Generation of neighborhood solution and the acceptance strategy of generated solution
- Temperature update
- Termination of the solution

Generation of the Initial Solution

The initial solution (schedule) is generated randomly using a uniform distribution. Alternatively one can use greedy heuristics to generate it. The initial system temperature is the Makespan of the initial solution.

Generation of Neighborhood Solution and the Acceptance Strategy of the Generated Solution

The next step is to perturb the initial solution just as it is done in GA. The new Makespan is evaluated with reference to the old one. If the new Makespan is better the solution is accepted and replaces the old one. If the new Makespan is worse than the old one, a uniform random number $z \in [0,1]$ is generated. This number is compared with another number y generated as follows:-

$$\frac{1}{1+e^{\left(\frac{oldmakespan-newmakespan}{current\ temparature}\right)}}$$

If z>y the new worse solution is accepted otherwise it is rejected and the old solution kept. As the temperature is iteratively lowered it becomes increasingly more difficult to accept a worse of solution.

Temperature Update

After each perturbation the system temperature is lowered to a fraction of its current value such as 90%. This completes one iteration of the SA.

Termination of the Solution

The algorithm terminates when either there is no change in the solution found so far or the temperature is approaching zero.

A perturbation scheme under SA entails:-

- Randomly swapping a task from one resource to another.
- Transferring a task from one resource to another.
- Adding a new resource and transferring a task to it.

Tabu Search

Taboo Search is another heuristic that avoid local optima by repeatedly making moves from one solution to another located in its neighborhood.

TS grid scheduling algorithm begins with a random mapping of tasks to resources. This initial solution is generated from a uniform distribution. To perturb the current solution, at this point the initial solution, to explore the solution space a short hop is performed. A short hop finds the nearest local minimum within the solution space. A short hop is performed by considering each possible pair of tasks, each possible resource assignments and keeping other assignments unchanged. If the new Makespan is an improvement the new solution is kept. A short hop ends when:-

- Every pairwise remapping considerations have been explored without improvements.
- Limit of total number of successful hops is reached.

On completion of the short hop the final solution is added to the tabu list.
The tabu list keeps track of explored regions of the solution space. A new random mapping is generated and must vary from the tabu list entry by at least half of the machine assignments. This process is known as the long hop. A long hop moves the algorithm to an unexplored region in the solution space that is yet to be searched. A short hop procedure is repeated after the long hop.

The stopping criterion for the iterative search is when the total number of successful short hops and long hops equals the limit set.

2.8.5 Scheduling Under Quality of Service Constraints

In heterogeneous and dynamic environments characterized by autonomous and semi-autonomous administrative domains, quality of service (QOS) is a major concern for applications. Two techniques of dealing with resource dynamics and heterogeneity have been proposed (Diot & Seneviratne, 2003):-

- Guaranteed approach
- Best effort approach

Guaranteed Approach

The Guaranteed approach is based on the notion of negotiating and re-negotiating quality of service contracts with the resource providers.

Best Effort Approach

In the absence of QOS guarantees above the best effort approach is used under which applications adapt themselves to the operating environment.

2.8.6 Strategies for Managing Dynamic Resource Performance

Resource in a Grid environment may at times have fluctuations in performance and availability. This is because resources belong to administrative domains that have control over them and thus behave autonomously or semi autonomously. This presents a major challenge to Grid schedulers. Current Grid schedulers handle this problem in three main ways:-

- Scheduling based on just-in-time information from Grid Information Services
- Performance prediction
- Dynamic rescheduling at run time

Scheduling Based on Just-In-Time Information from Grid Information Services

The GIS is a central component of a Grid System. It provides a catalog of users, applications, resources and services participating in the Grid. It serves a vital function to the Grid Scheduler by providing resource discover and monitoring. This way the scheduler is able to know when resource join or leave the Grid or become overloaded.

Performance Prediction

In dynamic environments prediction serve an important role of guiding future decisions. Prediction can be based on:-

- Prediction accuracy. Accurate prediction based on runtime parameters.
- Historical information collected over time.
- Prediction based on Workload modeling.

Dynamic Rescheduling

When resource prediction is unavailable or not feasible, rescheduling provides an alternative way to deal with resource dynamism. Strategies such as job migration are used in rescheduling. Previous scheduling decisions are alter to suite the current Grid state.

2.9 MIDDLEWARE FOR GRID COMPUTING

A Grid Middleware is a software stack that provides the functions of a Grid to end users enabling them to utilize Grid resources. The list below consists of some of the widely used Grid middleware:-

- ARC

- gLite
- Globus toolkit
- GridWay
- The OMII-UK distribution
- Oracle Grid Engine
- Alchemi
- Aneka

2.9.1 Advanced Resource Connector

Advanced Resource Connector (ARC) is a grid computing middleware introduced by NorduGrid. It provides a common interface for submission of computational tasks to different distributed computing systems and thus can enable grid infrastructures of varying size and complexity. ARC includes data staging and caching functionality, developed in order to support data-intensive grid computing. ARC is open source software distributed under the Apache License.

2.9.2 gLite

gLite is a grid computing middleware used by the CERN LHC for experiments and other scientific domains. The gLite Grid provides a framework for building applications tapping into distributed computing and storage resources across the Internet. The gLite services were adopted by more than 250 computing centers and used by more than 15000 researchers in Europe and around the world.

2.9.3 Globus Toolkit

The Globus Toolkit is an open source toolkit for building computing grids developed and provided by the Global Alliance organization.

2.9.4 GridWay

GridWay is an open source meta-scheduling technology that enables large-scale, secure, reliable and efficient sharing of computing resources managed by different Distributed Resource Management (DRM) systems, such as Sun Grid Engine, Condor and so on within a single organization or scattered across several administrative domains.

2.9.5 The OMII-UK Distribution
The OMII-UK distribution is GridSAM; a job submission interface for submitting computational jobs to commonly used distributed resource management systems such as Condo, PBS, SGE, etc.

2.9.6 Oracle Grid Engine
Oracle Grid Engine previously known as Sun Grid Engine (SGE) is open source grid middleware developed and maintained by the Oracle Corporation

2.9.7 Alchemi
Alchemi is an open-source .NET grid computing framework that allows you to aggregate the computing power of intranet and Internet-connected machines into a virtual supercomputer and to develop applications to run on the grid. The Grid is developed and maintained by the University of Melbourne.

2.9.8 Aneka
Aneka is a grid and cloud computing platform developed and supported by the University of Melbourne. It is the commercial successor of the Alchemi Grid. Aneka supports various programming models such as Task Programming, Thread Programming and MapReduce Programming. The platform provides tools for rapid creation of distributed applications and their seamless deployment on private or public Clouds.

2.10 THE CONCEPTUAL MODEL
The above literature has provided a very useful background in conduction this research. Going by the research objectives above, we introduce a learning function and performance model that is implanted into the Alchemi Grid. The conceptual diagram below provides an overview of the areas of interest for this research work.

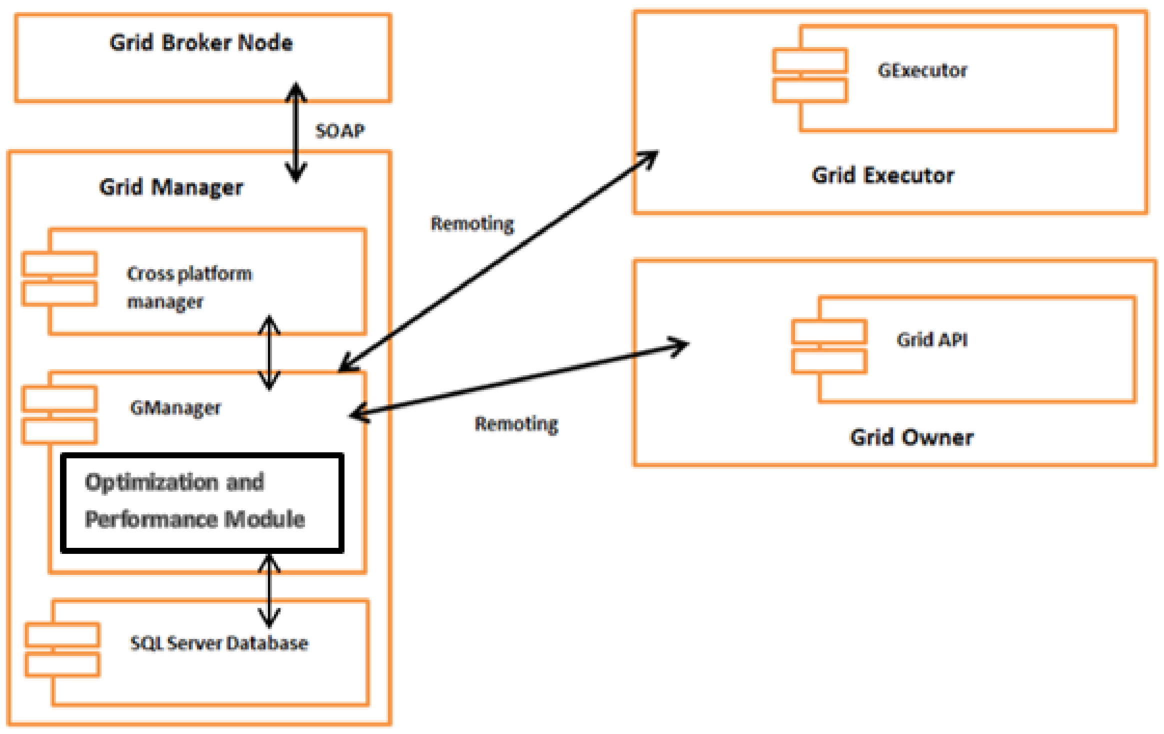

Figure 8: Conceptual Architecture for the Optimization and Performance Module

Grid Manager

The Grid Manager is the head node for the Alchemi Grid. It is responsible for the management of all Alchemi Grid Resources. It also acts as the scheduler by receiving applications and mapping them to executors. Grid Manager aggregates the computing power of the executors connected to it.

Grid Executor

The Grid Executor is a grid resource available to provide processing power for execution of Grid Applications.

Grid Owner

Grid Owner is a Grid user who submits Grid Applications to the Grid Manager for execution.

Grid Broker Node

Grid Broker Node provides external communication to resources and other Grids outside the current topology of the Grid. It provide cross platform integration with other diverse technologies.

SQL Server Database

SQL Server is the data repository for the entire Grid. All data regard to the state of resources and execution of grid applications is stored in this database.

Learning and Performance Management Module

This is the subsystem we introduce in this research. The subsystem will use the repository data to model application profiles, resource performance and characteristics. The module will use a learning function to improve the Grid scheduling functions.

3 METHODOLOGY

This chapter outlines how the research project was carried out. The research explores the various scheduling techniques in Grid computing and implements a performance model with a learning function on a typical computational Grid, the Alchemi Grid. The research work involves development of a prototype and experimentation. The following steps were carried out in the research work:-

- Survey of relevant literature
- Software methodology
- Analysis and design of the system
- Development of a performance model
- Implementation of the system
- Experimentation
- Data collection and analysis

3.1 SURVEY OF RELEVANT LITERATURE

Literature that touched on the aspects of Grid computing, scheduling algorithms, machine learning and artificial intelligence was studied. In particular greater emphasis was laid on the problem of scheduling in Computational Grids and research work done grid scheduling algorithms. Algorithms inspired by nature were delved into simulated annealing algorithm was chosen to implement components of the performance model and learning function.

3.2 SOFTWARE METHODOLOGY AND PARADIGM

The software methodology adopted in this research work is the ICONIX agile software development method. ICONIX is a simple and easy to use agile software development method base on UML. It encourages iterative software development. An iterative approach was used; sections of the prototype were developed in incremental stages.

The software programming paradigm used was object oriented programming. The Alchemi Grid used in this research is developed using an object oriented programming language C#. It is highly modularized in its design.

3.3 ANALYSIS AND DESIGN OF THE SYSTEM

A proper analysis was done of the existing Grid System and in particular the scheduling subsystem. A new design was created to incorporate the new scheduling algorithm that utilized more optimal schedules discovered by the optimization component of the Performance Manager. An optimization and performance management module was designed and embedded into the Grid Resource Manager.

3.3.1 Design and Setup of the Virtual Grid Environment

A virtual Grid Environment was designed setup up using VMWARE virtualization software. Virtual machines were created and given differential speeds to imitate heterogeneity in actual Grid Systems.

3.3.2 Design of Performance Manager

A Performance Manager component was designed to analyze historical resource information and produce better schedules for certain applications.

3.4 REVERSE ENGINEERING OF THE GRID BINARIES

Obtaining the Grid source code was a very difficult process. The existing code base was in an unknown state and full of bugs. The have a guarantee of a running grid system. The existing Grid binaries were reverse engineered to obtain the original source code. This code was used to modify the functionality of the Grid to include custom code and algorithms.

3.5 IMPLEMENTATION OF THE SYSTEM

Virtual Grid Environment

A Virtual Grid Environment was setup to mimic an actual Grid System. Virtual machines were created and given differential speeds to imitate heterogeneity in actual Grid Systems.

Development of the Performance Manager

The Grid Performance Manager, that includes a learning component, was developed using the .NET platform using the C# language. The module was embedded in the Grid Resource Manager so that it can effectively interact with the Grid and enhance scheduling.

The tools and technologies used in development were:-

- .NET Platform
- Visual Studio 2010
- C# language
- SQL Server 2008R2

3.6 EXPERIMENTATION

The experimentation phase entailed running experimental Grid Applications on the virtual Grid Environment. A compute intensive experiment was carried out.

Compute Intensive Experiment

The Grid application was designed to perform very large calculations. On execution of the application the computation was distributed amongst the Grid nodes according to the scheduling policy.

Parameters

The Virtual machines were configured with different speed ratios to provide the semblance of heterogeneity of resources and differences in computing capabilities of the Grid Nodes.

The Grid Application parameters were:-

- Numbers of tasks were varied.

3.7 DATA COLLECTION AND ANALYSIS

The Grid System records all its data in the SQL Server database. The performance model and learning function also recorded their data in the same database in separate tables.

Results of the Grid scheduler under the native algorithm were extracted, tabulated, analyzed and graphically presented. Additionally, the results of the scheduler with the advice of the performance manager derived by the learning component extracted, tabulated, analyzed and graphically presented. Comparison of the results under both schemes subsequently done and conclusions were made from the findings.

4 ANALYSIS AND DESIGN

In this chapter, we provide the design and analysis of the propose performance manager and scheduling algorithm that incorporates more optimal schedules observed by the optimizer. We also implement the performance management module in the Grid Resource Manager and the scheduling algorithm in the Grid Scheduler.

Alchemi Grid was chosen as the target computational Grid to implement the modules. The key reason of choosing Alchemi is in its simplicity of design and implementation without sacrificing flexibility, scalability, reliability and extensibility.

4.1 ALCHEMI GRID ARCHITECTURE

The Alchemi Grid follows the master-worker parallel programming paradigm in which a central component dispatches independent units of parallel execution to workers and manages them. This smallest unit of parallel execution is a grid thread, which is conceptually and programmatically similar to a thread object. The Figure 8 below illustrates the architecture of the Alchemi Grid.

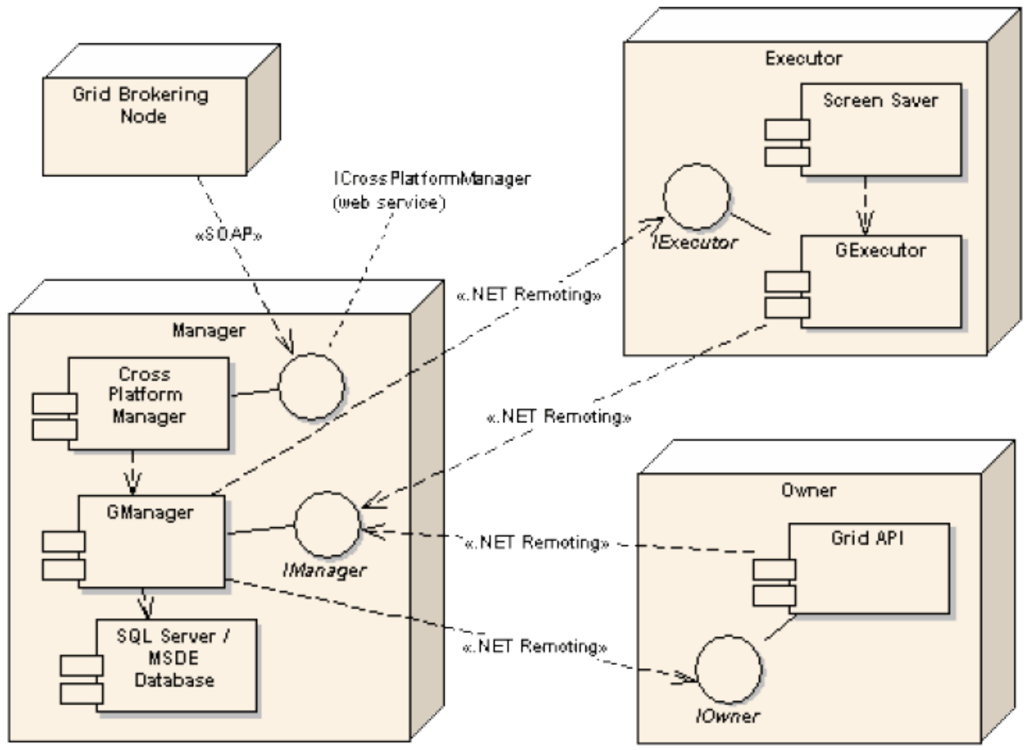

Figure 9: Architecture of the Alchemi Grid

Manager Component

The Manager manages the execution of grid applications and provides services associated with managing thread execution. The Executors register themselves with the Manager which in turn keeps track of their availability. Threads received from the Owner are placed in a pool and scheduled to be executed on the various available Executors. A priority for each thread can be explicitly specified when it is created within the Owner, but is assigned the highest priority by default if none is specified. Threads are scheduled on a Priority and First Come First Served (FCFS) basis, in that order. The Executors return completed threads to the Manager which are subsequently passed on or collected by the respective Owner.

Executor Component

The Executor accepts threads from the Manager and executes them. An Executor can be configured to be dedicated, meaning the resource is centrally managed by the Manager, or non-dedicated, meaning that the resource is managed on a volunteer basis via a screen saver or by the user.

For non-dedicated execution, there is one-way communication between the Executor and the Manager. In this case, the resource that the Executor resides on is managed on a volunteer basis since it requests threads to execute from the Manager. Where two-way communication is possible and dedicated execution is desired the Executor exposes an interface (**IExecutor**) so that the Manager may communicate with it directly. In this case, the Manager explicitly instructs the Executor to execute threads, resulting in centralized management of the resource where the Executor resides.

Thus, Alchemi's execution model provides the dual benefit of:

- Flexible resource management i.e. centralized management with dedicated execution vs. decentralized management with non-dedicated execution.

- Flexible deployment under network constraints i.e. the component can be deployment as non-dedicated where two-way communication is not desired or not.

Owner

Grid applications created using the Alchemi API are executed on the Owner component. The Owner provides an interface with respect to grid applications between the application developer and the grid.

The Owner submits threads to the Manager and collects completed threads on behalf of the application developer via the Alchemi API.

Cross-Platform Manager

The Cross-Platform Manager is a generic web services interface that exposes a portion of the functionality of the Manager in order to enable Alchemi to manage the execution of platform independent grid jobs. Jobs submitted to the Cross-Platform Manager are translated into a grid threads which are then scheduled and executed. Thus, in addition to supporting the grid-enabling of existing applications, the Cross-Platform Manager enables other grid middleware to interoperate with and leverage Alchemi on any platform that supports web services.

4.1.1 Alchemi Grid Class Diagram

.NET Remoting allows a .NET object to expose its functionality across application domains. Remoting is used for communication between the four Alchemi distributed grid components as it allows low-level interaction transparently between .NET objects with low overhead. The objects remoted using .NET Remoting within the four distributed components of Alchemi, the Manager, Executor, Owner and Cross-Platform Manager are instances of **GManager**, **GExecutor**,**GApplication** and **CrossPlatformManager** respectively.

GManager, **GExecutor**, **GApplication** derive from the **GNode** class which implements generic functionality for remoting the object itself and connecting to a remote Manager through the **IManager** interface.

The Manager executable initializes an instance of the **GManager** class, which is always remoted and exposes a public interface **IManager**. A key point to note is the fact the **IManager** interface derives from **IExecutor**. This allows a Manager to connect to another Manager and appear to be an Executor. This is the means by which the architecture supports the building of hierarchical grids.

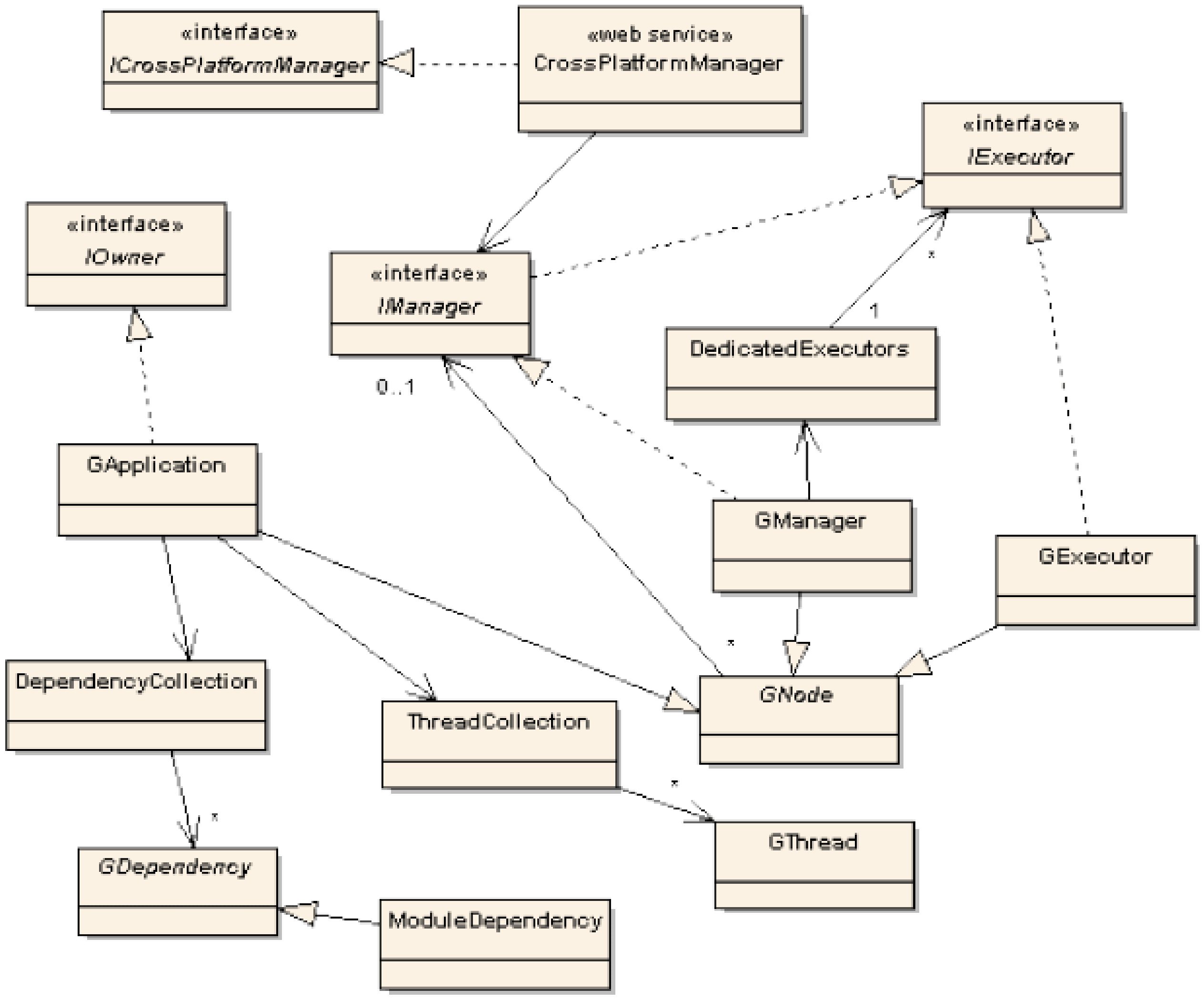

Figure 10: Class Diagram of Alchemi Grid

4.2 OPTIMIZATION AND PERFORMANCE MANAGER MODULE DESIGN

The learning and performance manager module is designed to be embedded in the Grid Manager component.

4.2.1 Conceptual Model

The diagram below depicts the conceptual model of the research work to be undertaken.

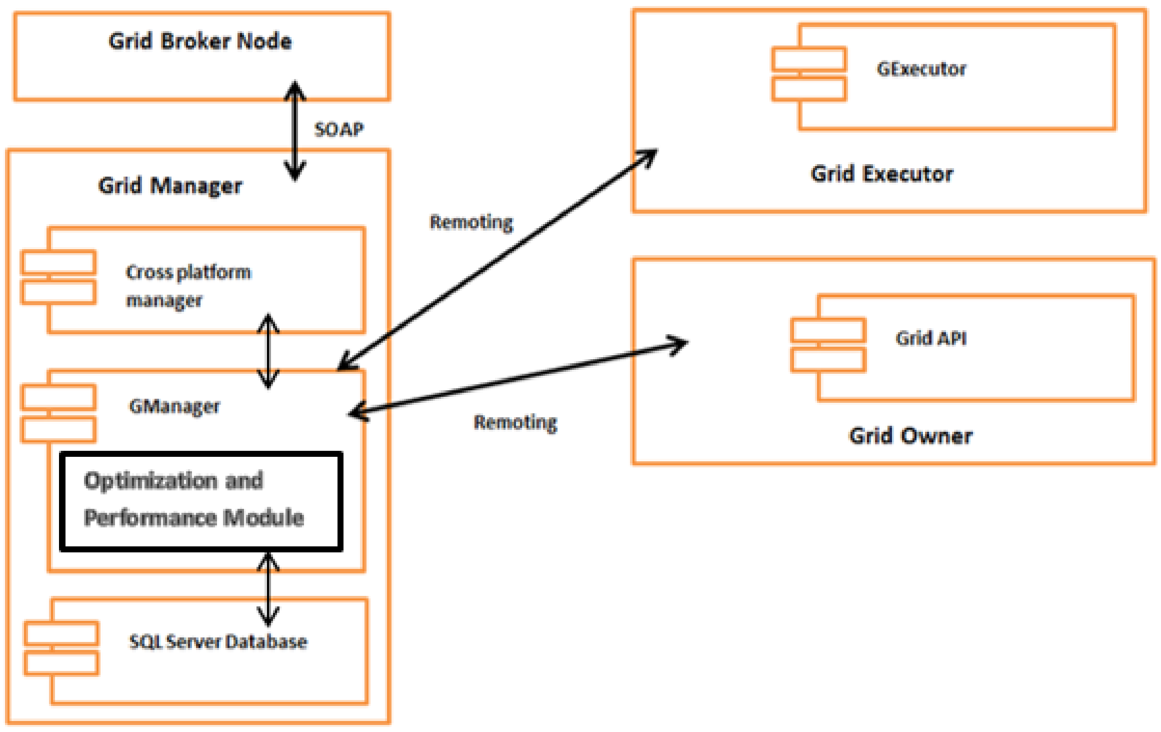

Figure 11: Conceptual Model of the Modified Alchemi Grid with the Optimizer

The main function of the optimization and performance module is to provide better schedules with time for application that execute on the Grid.

4.2.2 Design of the Optimization and Performance Manager Module

To implement to optimization and performance module, the **ManagerContainer** class of the Alchemi Grid is modified. The **Optimizer** class is added which starts a single thread that runs in the background. The class invokes the **PerformanceManager** class that is responsible for finding better schedules for each type of application found without an optimization. The class retrieves historical performance records of each type of application and feeds the **SimulatedAnnealingAlgorithm** class. The **SimulatedAnnealingAlgorithm** class utilizes application level objective function of minimizing the Makespan of the application. The class uses the simulated annealing algorithm to find more optimal schedules using schedules derived from historical records. The current scheduling algorithm is MCT (Minimum Completion Time) type algorithm with a FIFO queue. Figure 11 below shows the class diagram of the system module.

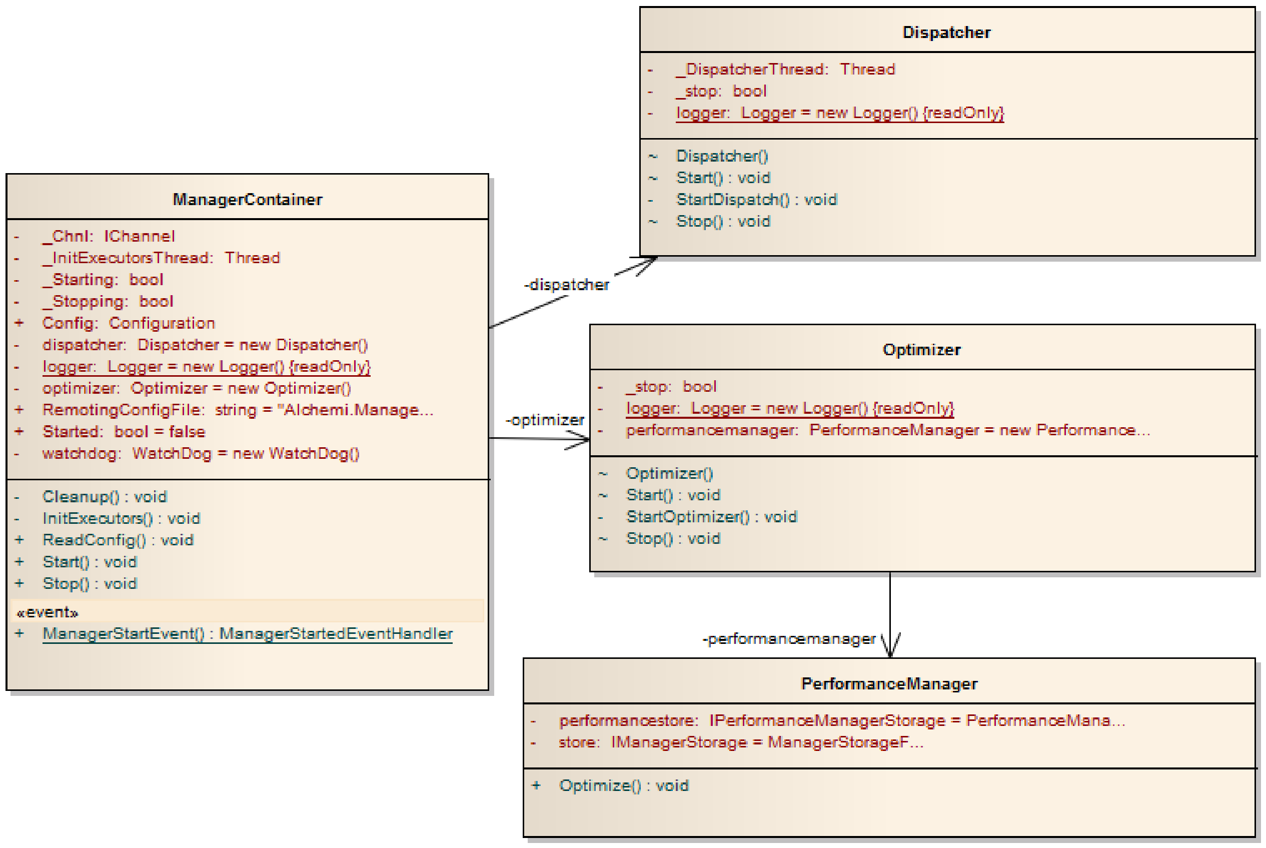
Dispatcher
- _DispatcherThread: Thread
- _stop: bool
- logger: Logger = new Logger() {readOnly}
~ Dispatcher()
~ Start() : void
- StartDispatch() : void
~ Stop() : void
ManagerContainer
- _Chnl: IChannel
- _InitExecutorsThread: Thread
- _Starting: bool
- _Stopping: bool
+ Config: Configuration
- dispatcher: Dispatcher = new Dispatcher()
- logger: Logger = new Logger() {readOnly}
- optimizer: Optimizer = new Optimizer()
+ RemotingConfigFile: string = "Alchemi.Manage...
+ Started: bool = false
- watchdog: WatchDog = new WatchDog()
- Cleanup() : void
- InitExecutors() : void
+ ReadConfig() : void
+ Start() : void
+ Stop() : void
«event»
+ ManagerStartEvent() : ManagerStartedEventHandler
-dispatcher
-optimizer
Optimizer
- _stop: bool
- logger: Logger = new Logger() {readOnly}
- performancemanager: PerformanceManager = new Performance...
~ Optimizer()
~ Start() : void
- StartOptimizer() : void
~ Stop() : void
-performancemanager
PerformanceManager
- performancestore: IPerformanceManagerStorage = PerformanceMana...
- store: IManagerStorage = ManagerStorageF...
+ Optimize() : void

Figure 12: Class diagram of the primary design of the Optimizer and Performance Manager Module

The scheduling module requires that any concrete implementation of a Scheduling Algorithm implements the interface **IScheduler**. There are two native implementations of **IScheduler** that come with the Alchemi Grid: - **DefaultScheduler** and **MappingScheduler**. **DefaultScheduler** implements the FIFO scheduling algorithm. **MappingScheduler** implements a form of a prioritized mapping of tasks to executors. Our Simulated Annealing Scheduling Algorithm is implemented by the **SimulatedAnnealingScheduler** that looks up matching schedules generated by the **PerformanceManager** using the **SimulatedAnnealingAlgorithm** for executing applications.

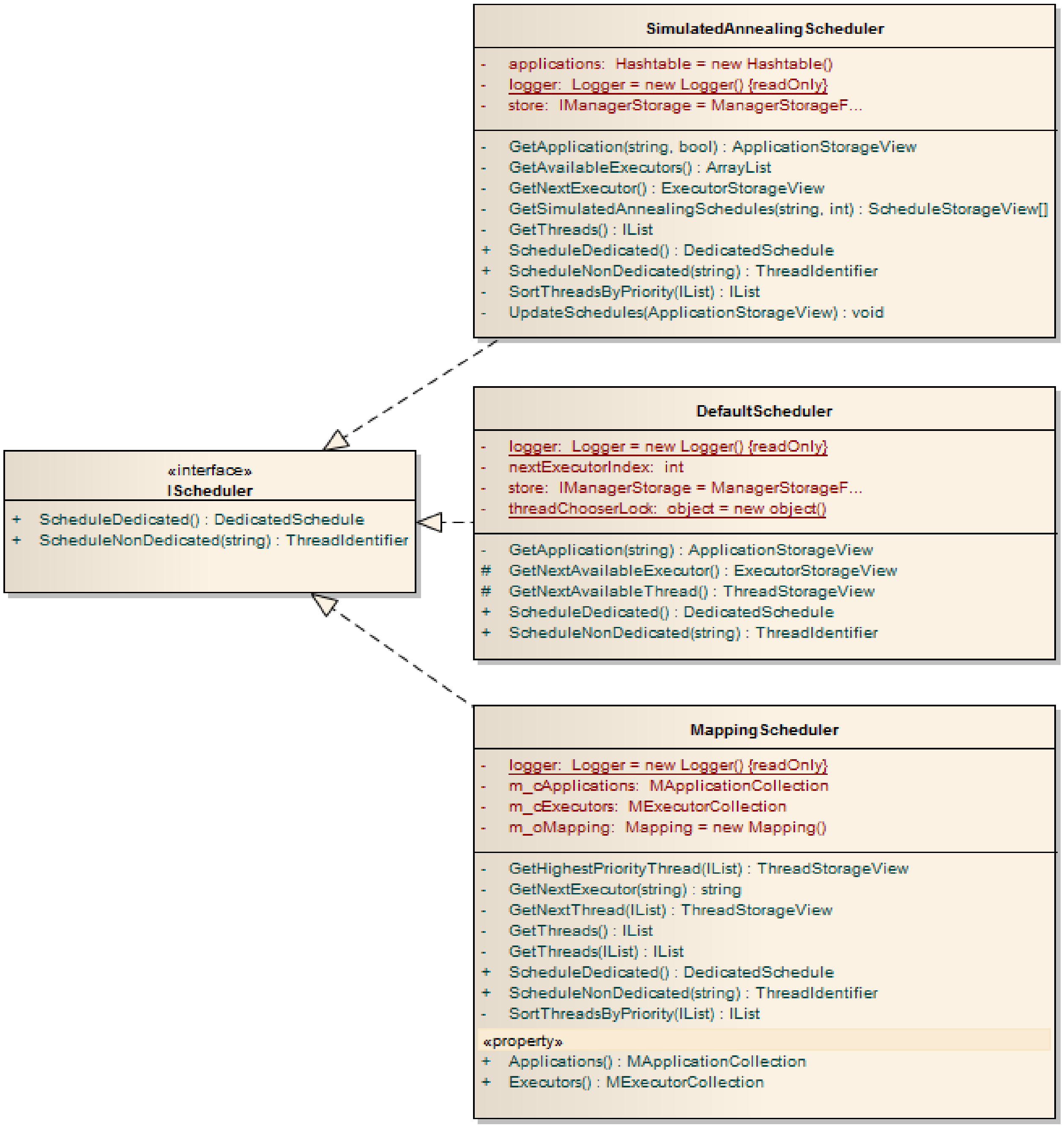
SimulatedAnnealingScheduler
- applications: Hashtable = new Hashtable()
- logger: Logger = new Logger() {readOnly}
- store: IManagerStorage = ManagerStorageF...
- GetApplication(string, bool) : ApplicationStorageView
- GetAvailableExecutors() : ArrayList
- GetNextExecutor() : ExecutorStorageView
- GetSimulatedAnnealingSchedules(string, int) : ScheduleStorageView[]
- GetThreads() : IList
+ ScheduleDedicated() : DedicatedSchedule
+ ScheduleNonDedicated(string) : ThreadIdentifier
- SortThreadsByPriority(IList) : IList
- UpdateSchedules(ApplicationStorageView) : void
«interface»
IScheduler
+ ScheduleDedicated() : DedicatedSchedule
+ ScheduleNonDedicated(string) : ThreadIdentifier
DefaultScheduler
- logger: Logger = new Logger() {readOnly}
- nextExecutorIndex: int
- store: IManagerStorage = ManagerStorageF...
- threadChooserLock: object = new object()
- GetApplication(string) : ApplicationStorageView
# GetNextAvailableExecutor() : ExecutorStorageView
# GetNextAvailableThread() : ThreadStorageView
+ ScheduleDedicated() : DedicatedSchedule
+ ScheduleNonDedicated(string) : ThreadIdentifier
MappingScheduler
- logger: Logger = new Logger() {readOnly}
- m_cApplications: MApplicationCollection
- m_cExecutors: MExecutorCollection
- m_oMapping: Mapping = new Mapping()
- GetHighestPriorityThread(IList) : ThreadStorageView
- GetNextExecutor(string) : string
- GetNextThread(IList) : ThreadStorageView
- GetThreads() : IList
- GetThreads(IList) : IList
+ ScheduleDedicated() : DedicatedSchedule
+ ScheduleNonDedicated(string) : ThreadIdentifier
- SortThreadsByPriority(IList) : IList
«property»
+ Applications() : MApplicationCollection
+ Executors() : MExecutorCollection

Figure 13: Class diagram of the primary design of the optimization and performance manager prototype

The Grid, at run-time, creates an instance of the **ManagerContainer** class invoke a static method in the **InternalShared** class. The invocation creates a single instance of the configured Scheduler from one of the following classes: - **DefaultScheduler**, **SimulatedAnnealingScheduler** or **MappingScheduler** as per the Grid configuration. The Grid **ManagerContainer** instance instantiates an object of type Dispatcher that begins the task of scheduling jobs and distributing to grid executors. Subsequently, an Instance of type Optimizer is started that commences the task of reviewing historical job completions and optimizing schedules for future use. The Optimizer creates an Instance of the **PerformanceManager** that runs the **SimulatedAnnealingAlgorithm** on application tasks to obtain optimized schedules for those applications. The diagram below is a sequence diagram of the interactions amongst the classes in the prototype to fulfill optimization.

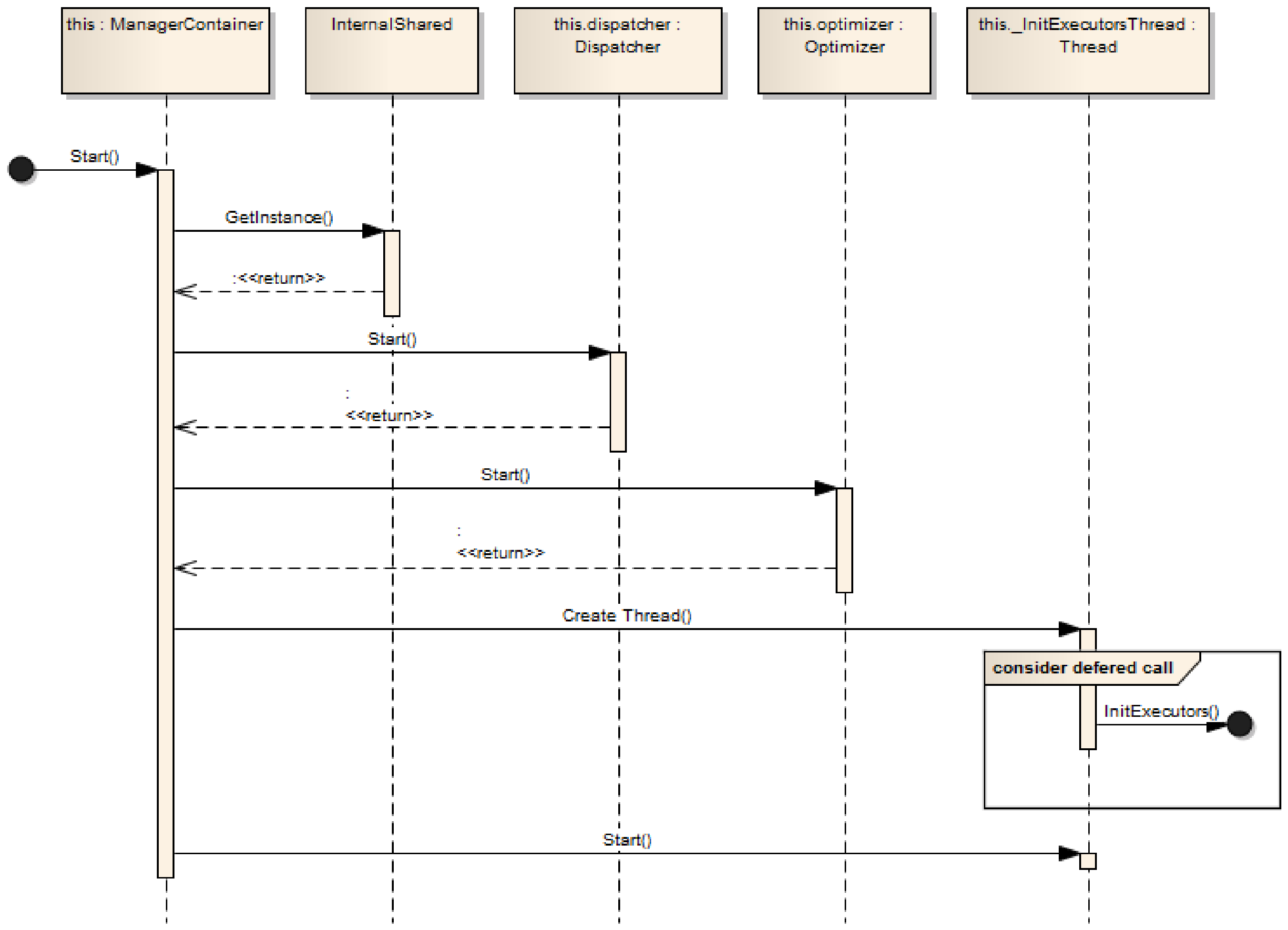

Figure 14: Sequence diagram for Optimization by the Performance Manager

To highlight how the Scheduler is invoked, the diagram below shows the sequence of interactions between the **InternalShared** class that is invoked by the **ManagerContainer** class. A Singleton instance of **InternalShared** is created and subsequently a scheduler creation factory instance is created that sets the actual scheduling algorithm to be used from the Grid configurations.

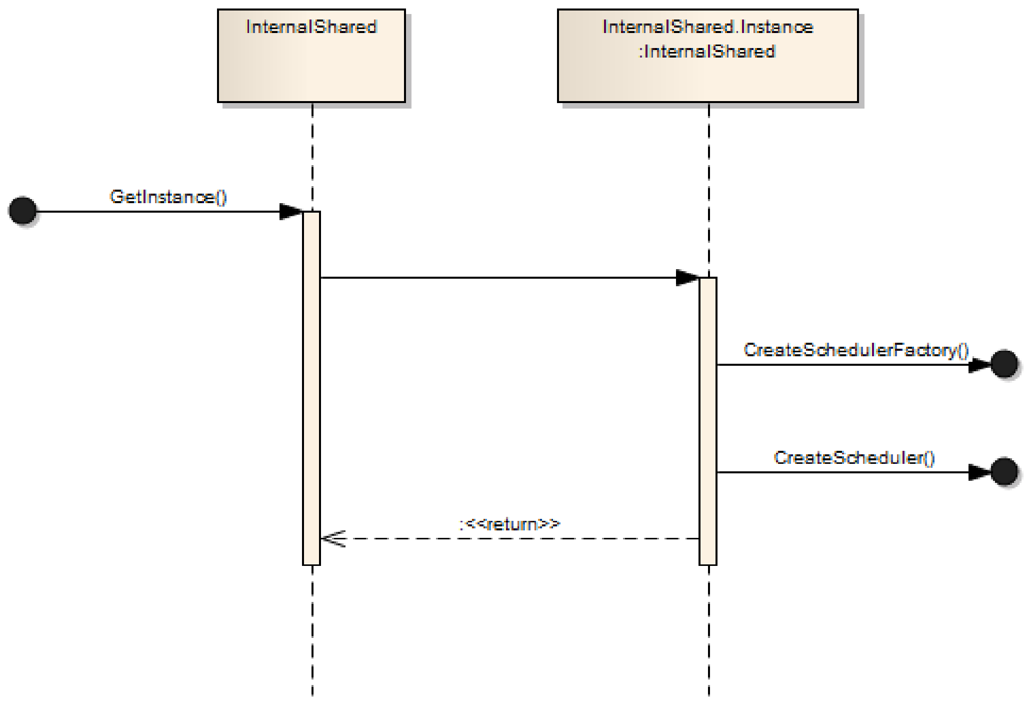

Figure 15: Sequence diagram for Instantiation of the Configured Scheduler

The **SchedulerFactory** instance that is created once the **InternalShared** instance comes to being, creates a scheduler instance abiding to the **IScheduler** interface and as per the grid configurations. The diagram below depicts this interactions in detail.

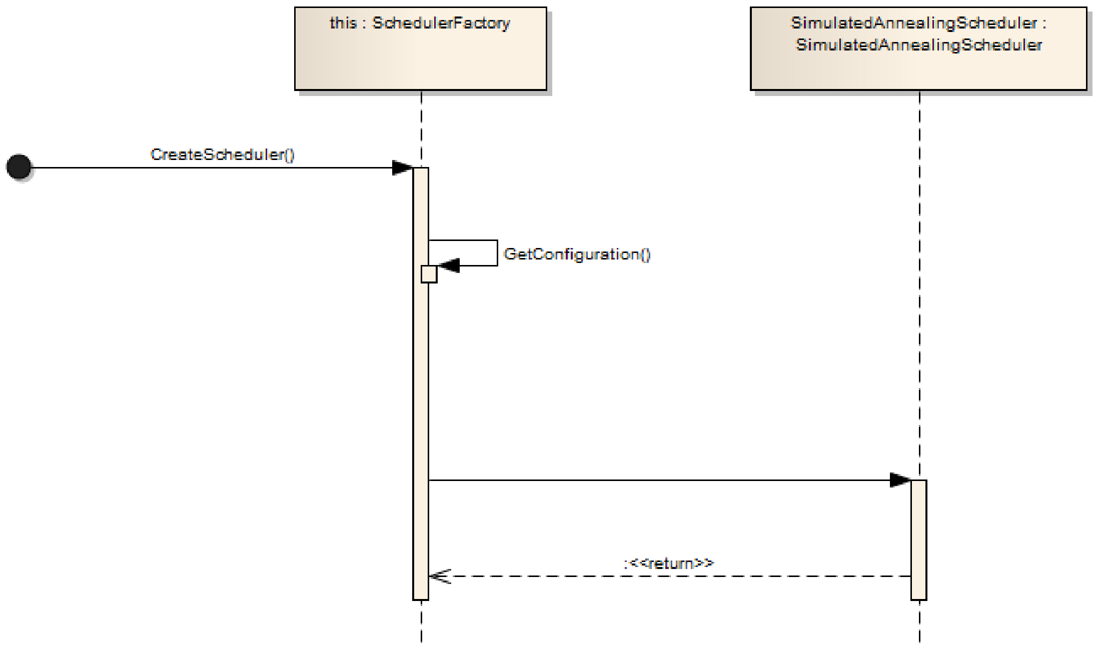

Figure 16: Sequence diagram for Instantiation of the Simulated Annealing Scheduler

4.2.2.1 *Algorithm for the Optimizer and Performance Manager*

The model is governed by a process that continuously comes up with new lower Makespan schedules for applications based on historical task execution data. The simulating annealing algorithm is used to obtain the lower Makespan schedules. The system follows the steps below to complete the learning and performance optimization process:-

1. While the grid manager is not is stop mode
    a. Select all applications types that have historical performance records.
    b. Check if an optimized schedule exists for each application
        1. If no optimized schedule exists
            i. Retrieve previous schedules for the applications.
            ii. Run the simulated annealing algorithm on these schedules.
            iii. If a better schedule (with lower Makespan) is found store it in the database else make no changes
        2. If an optimized schedule exists
            i. Retrieve previous schedules for the application.
            ii. Run the simulated annealing algorithm on these schedules.

        iii. If a better schedule (with lower Makespan) is found update the existing stored schedule

   c. Check for the stopping condition go to step 1 if no stopping condition.

4.2.2.2 *Algorithm for the Simulated Annealing*

The simulated annealing algorithm used is generic simulated annealing scheduling algorithm Stefka (2006). The algorithm pseudo code is listed below:-

1. The Problem Specific Decisions

Step 1. Formulation the problem parameters;

Step 2. Determination of the initial schedule, generate a feasible solution V;

2. The Problem Generic Decisions

Step 3. Initialization the cooling parameters:

i. Set the initial value of the temperature parameter, T;
ii. Set the temperature length L;
iii. Set the cooling rate F;
iv. Set the number of iterations K;

3. The Generation Mechanism, Selecting and Acceptance Strategy of Generated Neighbors

Step 4.

i. Select a neighbor V’ of V where V’ ε I(V)
ii. Let C(V’)=the cost of the schedule V’
iii. Compute the move value Δ =C(V’)-C(V)

Step 5.

i. IF $\Delta \leq 0$ accept V’ as a new solution and set V=V’
ii. ELSE
iii. IF e- Δ /T>θ set V=V’
iv. Where θ is a uniform random number 0<θ<1
v. OTHERWISE retain the current solution V

4. Updating the Temperature

Step 6. Updating the annealing scheduling parameters using the cooling schedule Tk+1=F*Tk k=1, 2,…

5. Termination of the Solution

Step 7. IF the stopping criteria is met THEN

Step 8.

i. Show the output
ii. Declare the best solution
iii. Otherwise go to step 4.

In our scheduling scenario, the temperature is the Makespan of the schedule; initial solution is the schedule from historical records and solution is a schedule with mapping of tasks and machines. The generation of a neighborhood solution is through pairwise exchange and transfer of tasks from one machine to another randomly.

4.2.2.3 *Algorithm for the Simulated Annealing Scheduler*

The simulated annealing scheduler follows the steps below to complete the scheduling process:-

1. Get the next free machine (resource)
2. Get the next task scheduled for execution.
    a. Retrieve application information for the task
    b. Lookup the stored schedule for this application
    c. Retrieve the task in the schedule that matches to the current task
        i. Compare the machine identities for this mapping
        ii. If the machine identities match return the mapping else move to the next available task in the queue and repeat the machine identity check for the task as above.
    d. If no match is found return an empty mapping.
    e. End.

4.3 EXPERIMENT DESIGN

The experiment setup consists of a Windows Server 2008 R2 Virtual Machine environment installed on a host computer as the diagram below illustrates. The Virtual Machines consist of the same operating system Windows.

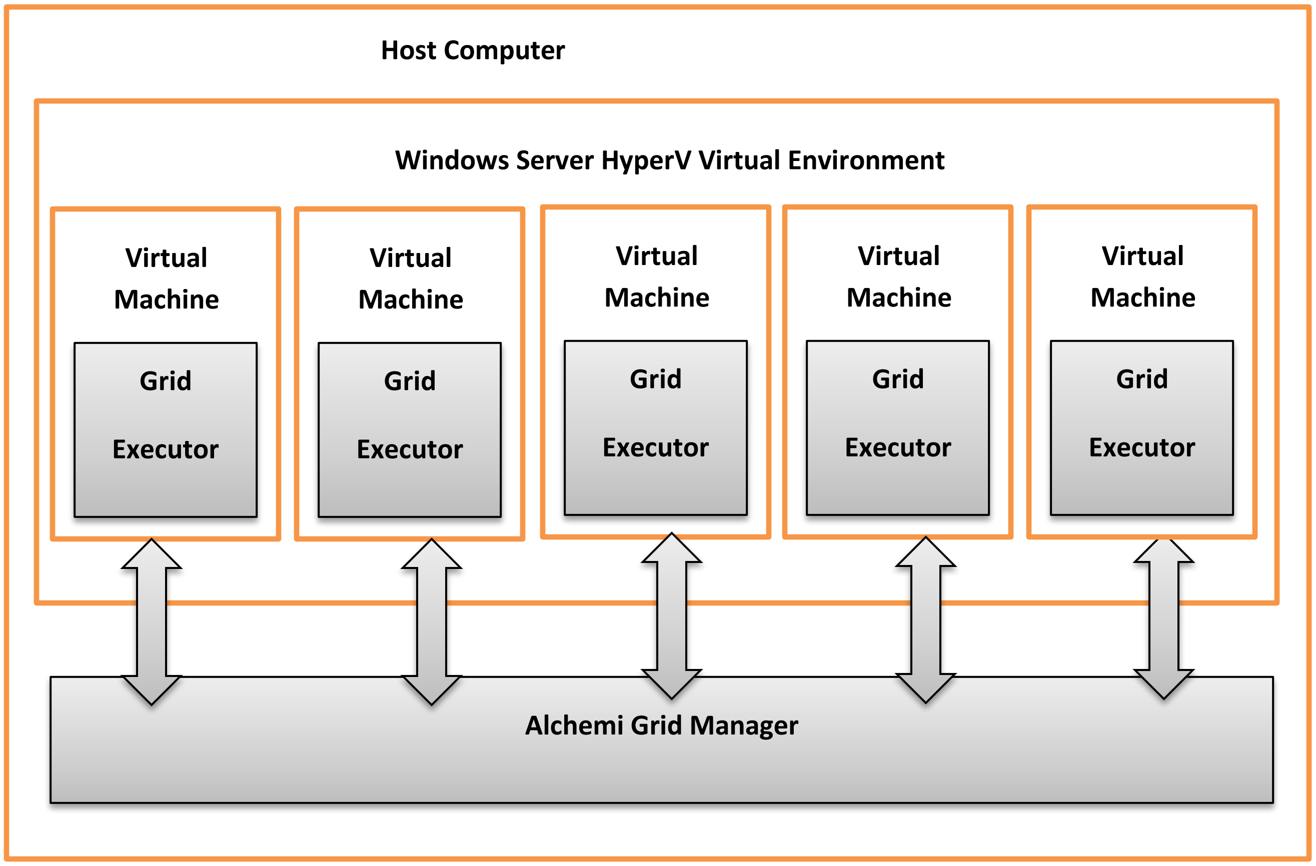

Figure 17: Virtualized Environment

A Compute intensive experiment was carried out with varying task sizes.

Compute Intensive Experiment

A Grid application is designed to perform very large calculations. On execution of the application the computation will be distributed amongst the Grid nodes according to the scheduling policy. The Grid application is named “PI Calculator - Alchemi sample”. This application computes the digits of PI that are keyed in.

Experiment Parameters

The Virtual machines are to be configured with different speed ratios to provide the semblance of heterogeneity of resources and differences in computing capabilities of the Grid Nodes.

The Grid Application parameters to be varied are:-

- The Size of tasks.
- Numbers of tasks.

4.4 ALCHEMI GRID CONFIGURATION

The Grid architecture is design uses the most basic hierarchy as shown in the figure 13 below; a single Alchemi Grid Manager and several Grid Executor nodes.

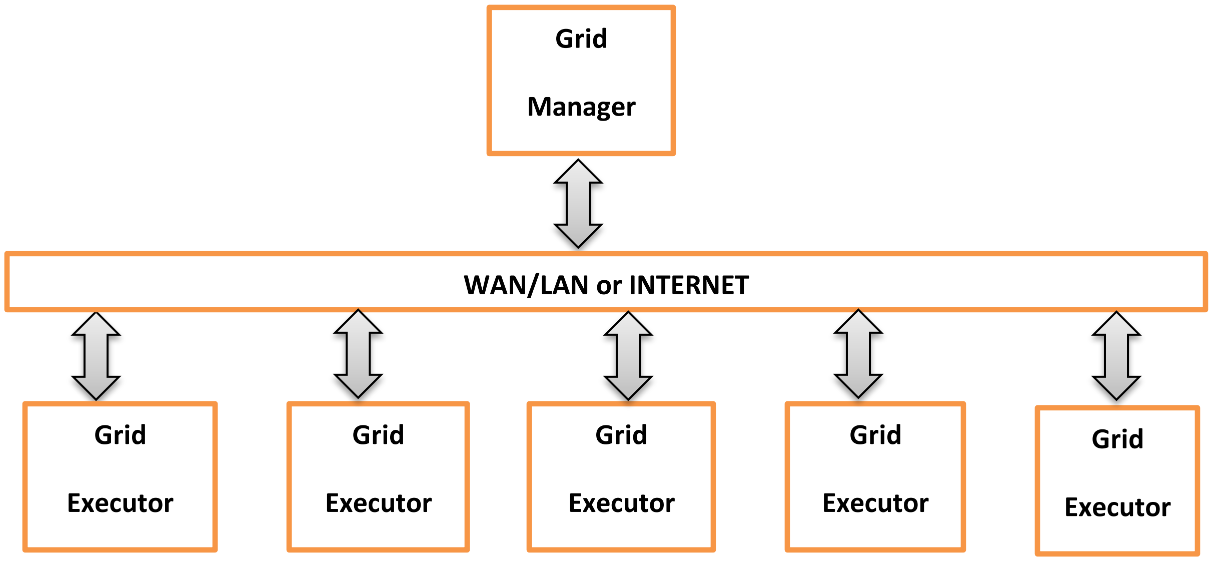

Figure 18: Alchemi Grid Hierarchy

5 IMPLEMENTATION

5.1 SETUP

In this chapter, we provide the implementation specific details of the virtual machine environment, experiment and the developed prototype. The table below shows the software components used.

Item	Specifications
Virtual Machines	Windows Server 2008 R2 Hyper-V Virtual Machine Environment
Database server	SQL Server 2008R2 Enterprise Edition

The table below shows the hardware components needed and used.

Item	Specifications
1 Laptop	8GB RAM 1TB hard disk Pentium Core i3 (Quad Core)

5.1.1 Virtual Environment Setup

Installation of Windows Server 2008 R2 Virtual Machine

The Hyper-V environment comes as a component of Windows Server 2008R2. The component was activated as by installing the Hyper-V software components. The location configured for the Virtual Machines was **C:\VirtualMachines.**

Setup of the Virtual Machines

Five virtual machines were setup for the experiments and virtual machine environment with the following configurations:-

Item	**VM Machine 1**	**VM Machine 2**	**VM Machine 3**	**VM Machine 4**	**VM Machine 5**
Memory	1024MB	512MB	512MB	512MB	512MB
Disk size	20GB	20GB	20GB	20GB	20GB
Number of Virtual Processors Allocated (Maximum 4)	1	1	1	2	4
Percentage of Total System Resources (Percentage) Allocated	100	30	80	100	100
Virtual Machine Resource Usage Limit (Percentage for CPU ulitlization)	25	7	20	50	100
Relative Weight	100	30	60	80	100
Machine Name	Machine1	Machine2	Machine3	Machine4	Machine5
IP	192.168.146.2	192.168.146.3	192.168.146.4	192.168.146.5	192.168.146.6
Operating System	Windows 7	Windows 7	Windows 7	Windows 7	Windows 7

On each virtual machine the .NET framework version 4 was installed and a single Grid executor.

5.1.2 Host Machine Setup

The host machine no major installations were needed apart from SQL Server 2008R2 with an instance identity of "HP\SQLSERVER2008R2". The Host machine was assigned the IP 192.168.146.1.

5.1.3 Experiment Setup

The experiment setup involved installing the grid middleware and configuring its topology.

The Grid Setup

The Alchemi Manager

The Alchemi Grid Manager is installed in the host machine. It is configured with the following parameters in an xml file located at "C:\Users\Administrator\AppData\Roaming\Alchemi\Manager".

```
?xml version="1.0" encoding="utf-8"?>

<Configuration xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

  <ConnectVerified>false</ConnectVerified>

  <DbConnectTimeout>50</DbConnectTimeout>

  <DbMaxPoolSize>5</DbMaxPoolSize>

  <DbMinPoolSize>5</DbMinPoolSize>

  <DbName>Alchemi</DbName>

  <DbPassword>Password01</DbPassword>

  <DbServer>HP\SQLSERVER2008R2</DbServer>

  <DbType>SqlServer</DbType>

  <DbUsername>sa</DbUsername>
  <Dedicated>true</Dedicated>

  <Id />

  <Intermediate>false</Intermediate>

  <ManagerHost>192.168.146.1</ManagerHost>

  <ManagerPort>9001</ManagerPort>

  <OwnPort>9000</OwnPort>

  <SchedulerAssemblyName>Alchemi.Manager</SchedulerAssemblyName>


<SchedulerTypeName>Alchemi.Manager.SimulatedAnnealingScheduler</SchedulerTypeName>

</Configuration>
```

The table below describes the parameters.

Parameter	**Meaning**
ConnectVerified	Verify whether the connection are up
DbConnectTimeout	Number of seconds before a the database connection
DbMaxPoolSize	Maximum number of connections to put the connection pools
DbMinPoolSize	Minimum number of connections to put the connection pools
DbName	Database username
DbPassword	Database password
DbServer	Database Server to connect
DbType	Type of database server e.g. Oracle, Sql Server, etc
DbUsername	Database username
Dedicated	Connect manage in complete dedication mode
Id	Unique Identity of the manager
Intermediate	

ManagerHost	IP of the host
ManagerPort	Port to reach the manager when acting like an Executor
OwnPort	Port to reach the host
SchedulerAssemblyName	Namespace of the class the implements IScheduler
SchedulerTypeName	Name of the scheduling class implementing a scheduling algorithm

The Alchemi Executor

The Alchemi Grid Executor is installed in each the virtual machine. It is configured with the following parameters in an xml file located at C:\Users\Administrator\AppData\Roaming\Alchemi\Executor. The configuration differs from machine to machine in some fields.

```
<?xml version="1.0" encoding="utf-8"?>

<Configuration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <AutoRevertToNDE>false</AutoRevertToNDE>

  <ConnectVerified>true</ConnectVerified>

  <Dedicated>true</Dedicated>

  <HeartBeatInterval>5</HeartBeatInterval>

  <DebugFlag>false</DebugFlag>

  <DebugDelay>-1</DebugDelay>

  <Id>fb1b0481-387f-4e0a-9ec5-0c9c5bd616dc</Id>

  <ManagerHost>192.168.146.1</ManagerHost>

  <ManagerPort>9000</ManagerPort>
  <OwnPort>9002</OwnPort>

  <Password>executor</Password>

  <RetryConnect>true</RetryConnect>

  <RetryInterval>30</RetryInterval>

  <RetryMax>-1</RetryMax>

  <SecureSandboxedExecution>false</SecureSandboxedExecution>

  <Username>executor</Username>

</Configuration>
```

Parameter	**Meaning**
Dedicated	Connect to the Manager in dedicated mode.
HeartBeatInterval	Number of seconds to wait to send a heartbeat to the Manager.
DebugFlag	The delay flag used to vary the speed on computing of the executor.
DebugDelay	The amount in seconds to delay
Id	Id of the executor varies per executor
Password	Database password
Username	Database Server to connect
RetryConnect	flag to retry reconnection to Manager when connection fails
RetryMax	Time in seconds to retry
ManagerPort	Port to reach the manager
SecureSandboxedExecution	Namespace of the class the implements IScheduler

5.1.4 Grid Application Settings

The Grid application parameters for the Application “PI Calculator - Alchemi sample” were input as follows:-

Parameter Name	Parameter Value
Number of threads (Task)	Increase in steps of 100 starting from 100

5.1.5 Scheduling Algorithm Parameters

The Scheduling Algorithm will be executed with the following parameters:-

Parameter Name	Parameter Value
Maximum Iterations	1,000,000
Initial temperature	1,000,000
Temperature cooling rate	0.995

5.1.6 Collecting Results

To collect the results under the Simulated Annealing Scheduler, configure the < SchedulerTypeName > parameter of the Grid Manager and set it to “Alchemi.Manager.SimulatedAnnealingScheduler” and Start it up. To configure the Grid Manager to use the native Scheduler, change the same value to “Alchemi.Manager.DefaultScheduler”. An example of such a configuration is

<SchedulerTypeName>Alchemi.Manager.SimulatedAnnealingScheduler</SchedulerTypeName>
.

5.2 DEVELOPMENT

The learning and performance management module was successfully developed and deployed as part of the Alchemi Grid Manager using C#. A new table was created to store the near optimal schedules discovered during the solution exploration phase.

6 RESULTS

6.1 INTRODUCTION

This chapter provides the results of carrying out the design experiments and assessed the overall performance of the Grid under the learning function and the Performance Management module.

6.2 EXPERIMENT OBJECTIVES

The objectives of the experiment were to assess the performance of the Alchemi Grid under the Native Scheduler that uses a Minimum Completion Time (MCT) strategy with a FIFO queue and compare that to the Simulated Annealing Scheduler. The objective function used for comparison is application centric namely the Makespan.

6.3 EXPERIMENT LAYOUT

The Grid environment was set up in a virtualization platform with five machines with differential processing capabilities. The grid hierarchy was one level as shown in the Diagram below:-

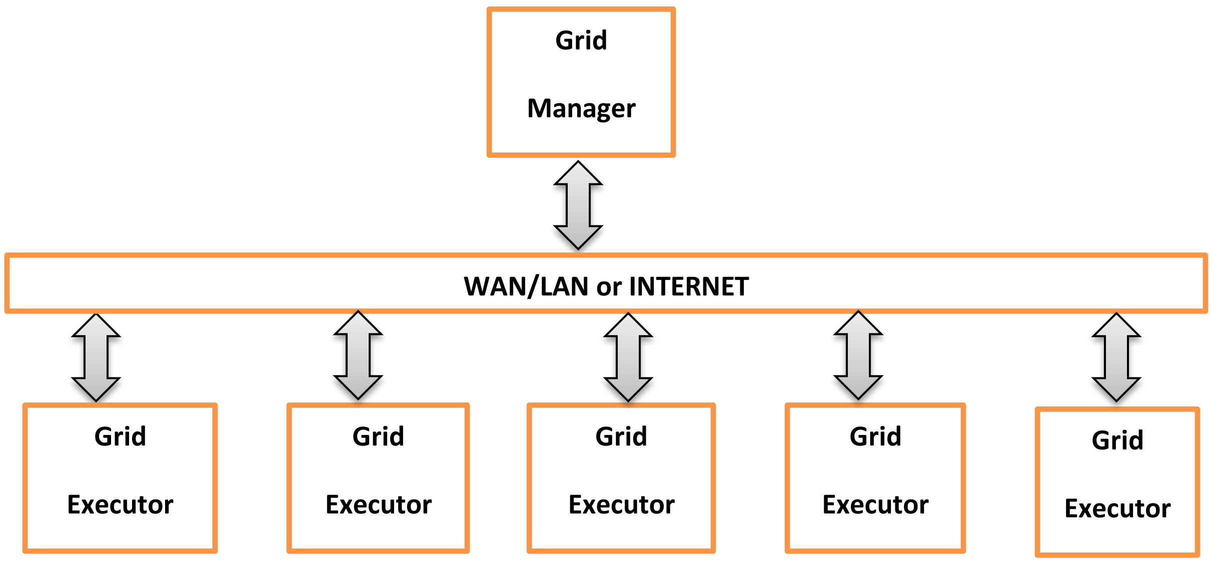

Figure 20: Hierarchical Layout of Grid

A compute intensive application named “PI Calculator - Alchemi Sample” was executed on the Grid with the Native Scheduler. The application runtime statistics were collected from the database. The scheduler was then changed to the Simulated Annealing Scheduler and the experiment repeated. The number of tasks was increased after each experiment in steps of 100.

6.4 PERFORMANCE RESULTS

The results of the experiments are tabulated below.

No	**Number of Tasks**	**MakeSpan Duration (hh:mm:ss) Native Scheduler**	**MakeSpan Duration (hh:mm:ss) Simulated Annealing Scheduler**	**MakeSpan Duration In Seconds Native Scheduler**	**MakeSpan Duration (hh:mm:ss) Simulated Annealing Scheduler**
1	100	01:24:00	00:50:00	84	50
2	200	00:01:26	00:01:15	86	75
3	300	00:01:42	00:01:37	102	97
4	400	00:03:01	00:02:09	181	121
5	500	00:02:48	00:02:39	168	159
6	600	00:04:11	00:03:28	251	208
7	700	00:04:15	00:03:44	255	224
8	800	00:04:23	00:04:17	263	257

When the information is present graphically the comparison is more visible as below:

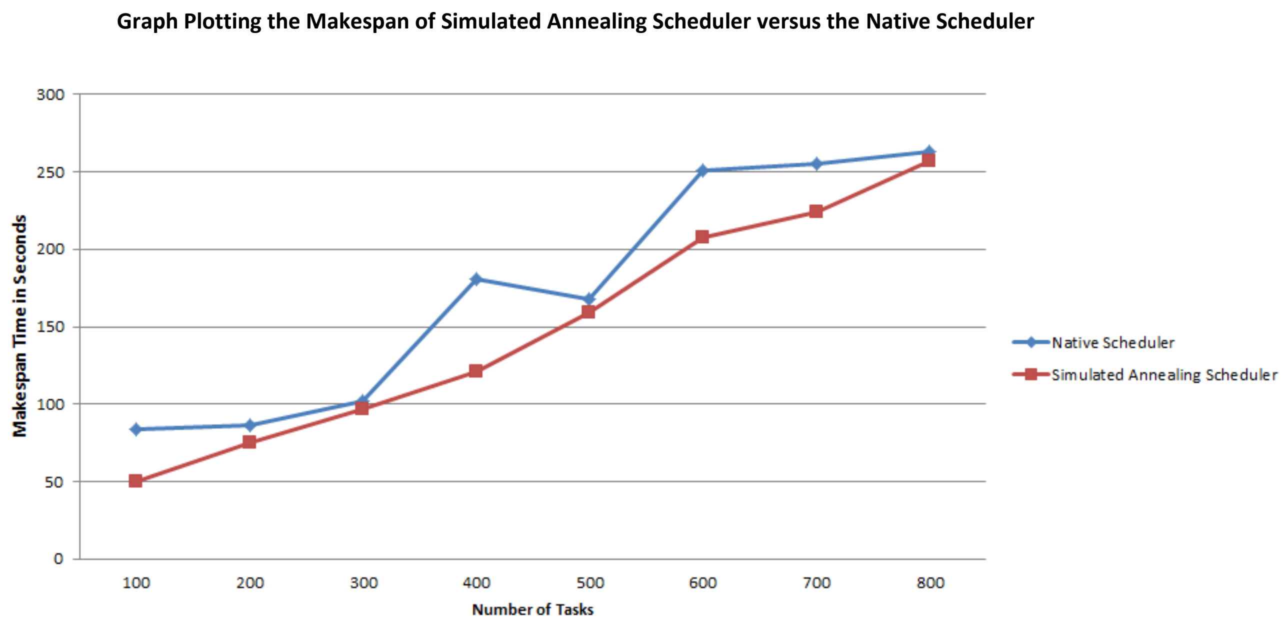

Figure 21: Graph Plotting the Makespan of Simulated Annealing Scheduler versus the Native Scheduler

7 DISCUSSION

7.1 DISCUSSION

We were able to demonstrate that application and resource profiling based on historical records may benefit a Grid Scheduler. The simulated annealing scheduler was able to consistently produce lower makespans than the native scheduler. The simulated annealing scheduler was able to consider resource capabilities prior to scheduling. On the other hand the native scheduler based on the FIFO principle, allots tasks to resources without any such bias. The native scheduler would assign tasks to resources even if the assignment may negatively impact the objective function in this case the Makespan.

Practical implication of the results is in the reservation of resources to perform specific tasks. The model we developed can be used where a large number of compute intensive tasks are executed routinely**.** The developed framework can optimize such applications as they run for a considerable length of time.

From the results, we obtained an average of 14% reduction in Makespan. The optimization and performance module has been successful in obtaining better schedules and keeping a memory of them in the database. A better schedule is kept when found and subsequently optimized.

Given that a computational Grid is a dynamic and heterogeneous computing environment, the profiles based on historical information may not hold. Also the study took only into consideration a fixed number of resources that are dedicated to the Grid. Resources in a Grid leave and join sometimes unpredictably.

The optimization and performance management subsystem may not be suitable in situations where task number and sizes vary dynamically for the same application.

8 CONCLUSION AND FUTURE WORK

8.1 THE CONCLUSION

We presented a model of a scheduling framework that improves the scheduling performance i.e. shorter makespans of common applications scheduled in the grid. The performance of such applications improves with time as better schedules are obtained.

Our implemented model consisted of performance optimizer that utilized the simulated annealing algorithm. The algorithm generated better optimized schedules with shorter makespans for applications that were previously executed in comparison to the native and default scheduling algorithm in the Alchemi Grid. Historical data on task execution times on given resources was used to compute the time taken by individual grid computing resources to complete individual tasks on the grid. This Scheduling algorithm performed a schedule lookup if the application exhibited similar scheduling characteristics to existing optimized schedules.

The virtual computing environment was able to mimic a real gird computing environment by differential allocation of computing resources i.e. CPU, memory and disk-space. CPU was the key determinant in the variation. The virtual computers were connected to the grid in dedicated mode. The juggling of resources amongst virtual machines gave the computing impression of heterogeneity.

We were able to observe that virtual machines with unfavourable allocation of computing resources exhibited lower task completion rates than machines with higher CPU and memory allocations.

8.2 RECOMMENDATIONS AND FUTURE WORK

The research has provided some insight into Grid Scheduling algorithms and their practical implications. The research was limited to a single Grid environment, the Alchemi Grid. Alchemi Grid has since been succeeded by the Aneka Grid that comes with many scheduling algorithms and schemes. The Alchemi Grid currently supports the thread model to task decomposition. Our models such as Map reduce and task model can be considered in future research endeavors.

Our Virtual computing environment gave the computing impression of a grid computing environment though virtual. It would be interesting to evaluate the model on an actual grid composed of heterogeneous computing resources. Our virtual grid only considered CPU as the main computing resource. Research could be conducted to evaluate resource allocation of memory in addition to CPU for data and compute intensive applications. The applications considered in this research were compute intensive only.

Our model being a generic model can be implemented in similar advanced Grids such as Grid Gain, and Aneka and performance evaluation conducted in the context of other scheduling frameworks and algorithms. Improvements can be made to the model to allow dynamic selection of scheduling algorithms or schemes based on the scheduled application characteristics.

Further opportunities have arisen for additional research in Grid scheduling with natural heuristics. It would be interesting to see the application of machine learning algorithms in the scheduling. Additionally, the application of scheduling in tasks with dependencies remains a challenge to many researchers and requires more energy in terms of research effort.

REFERENCES AND BIBLIOGRAPHY

Ferreira, L. et al. (2005) *Grid Computing in Research and Education.* USA: IBM Press.

Foster, I. (2002) *What is the Grid? A Three Point Checklist*. [online] Available from: http://dlib.cs.odu.edu/WhatIsTheGrid.pdf. [Accessed: June 2012].

Buyya, R. & Murshed, M. (2002) *GridSim: A toolkit for the modeling and simulation of distributed resource management and scheduling for Grid computing*. Concurrency and computation: practice and experience. 14. p. 1175-1220.

Zhu, Y. (2003) *A Survey on Grid Scheduling Systems*. A Thesis Submitted in partial fulfilment of the Requirements of Hong Kong University of Science and Technology for the Degree of Doctor of Philosophy. Hong Kong: Hong Kong University of Science and Technology.

Foster, I., Kesselman, C. & Tuecke, S. (2001) The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *Journal International Journal of High Performance Computing Applications*. 15 (3). p. 200-222.

Schopf, J. (2004) Ten actions when Grid scheduling: the user as a Grid scheduler. In: Nabrzyski, J., Schopf, J. & Weglarz, J. (eds). *Grid Resource Management*. Norwell, MA, USA: Kluwer Academic Publishers.

Fernandez-Baca, D. (1989) Allocating Modules to Processors in a Distributed System, *IEEE Transactions on Software Engineering*. 15 (11). p. 1427-1436.

F. Dong & Akl S. G., (2004) Scheduling Algorithms for Grid Computing: State of the Art and Open problems. *Technical Report of the Open Issues in Grid Scheduling Workshop*. Ontario: University Kingston.

Berman, F. (1998) High-Performance Schedulers. In: Foster, I. & Kesselman, C. (eds). *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers.

El-Rewini, H., Lewis, T. G. & Ali, H., (1994). Task Scheduling in Parallel and Distributed Systems. PTR Prentice Hall.

Casavant, T.L. & Kuhl, J.G. (1988) A Taxonomy of Scheduling in General-Purpose Distributed Computing Systems. [online] Available from: http://ieeexplore.ieee.org/iel1/32/259/00004634.**pdf**? arnumber=4634. [Accessed: June 2012].

Buyya, R. et al (2002). Economic Models for Resource Management and Scheduling in Grid Computing. *Concurrency and Computation: Practice and Experience*. 14(13-15). p. 1507-1542.

Abraham A., Buyya, R. & Nath, B. (2000) Nature's Heuristics for Scheduling Jobs on Computational Grids. In *Proceeding of 8th IEEE International Conference on Advanced Computing and Communications (ADCOM 2000)*. Conchin, India. p. 45-52.

Braun, T. D. et al. (2001) A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems. *Journal of Parallel and Distributed Computing*. 61(6). p. 810-837.

Diot, C. & Seneviratne, A. (2003) Quality of Service in Heterogeneous Distributed Systems. In *Proceedings of the 30th International Conference on System Sciences (HICSS)*. Maui, Hawaii, USA. p. 238—253.