UNIVERSITEIT VAN AMSTERDAM

Graduate School of Informatics

Software Engineering Master Program

Master Project

A Grid Scheduling infrastructure for Smart Connect's performance monitoring calculations

Author:

Pantelis Zisimopoulos

Supervisor:

Tijs van der Storm

Host Organization:

Royal Dutch Shell via CGI Nederland

Host Organization's supervisors:

Alexander Stegehuis

Lennard Bakker

Table of Contents

1 Introduction

Smart Connect is a performance monitoring system that Shell uses in order to know how well its oil drilling equipment operate and thus control the oil production rate and avoid any loses caused by unexpected deferments. To do so, a group of engineers have set up a series of calculations that use values coming (for now) every one minute straight from the oil drilling sites. These calculations in order to provide correct results, should be executed with a known precedence. The calculations are currently hardwired to the servers they are to be performed. Each server runs a local, independent scheduler. For maintenance issues the precedence of calculations is not taken under consideration. This master project's main is concerned with making a first step towards developing a Grid Scheduling infrastructure tailored for the calculations Smart Connect uses.

The most distinguished grid scheduling systems are investigated in order our solution to be sketched. A Master-Slave infrastructure is common to all of them; where the master program orchestrates the work to be performed by the slave programs. It is also made quickly understood that the scheduling algorithm plays a major part to the performance of the system. For this reason, all main algorithms that are a good fit for the environment (i.e. heterogeneity of servers) at hand and the calculations of Smart Connect are considered. The best known currently known fundamental algorithm that performs best for the calculations of this problem is HEFT.

Due to the specifics of the calculations, independent scheduling algorithms are also considered as candidates. After assembling and ordering calculations in the smallest possible groups in a way that precedence dependencies are resolved (named here as jobs), the usage of independent scheduling algorithms was made possible. This could allow to bring the advantages (simplicity, adaptability, testability and so forth) of these algorithms to the new system. Nevertheless, this also requires the introduction of a small overhead. After analyzing the tasks that would be used for the independent algorithms, MaxMin shows that would perform the best.

As HEFT and MaxMin work on a different types of tasks, no literature study has been made to compare them. This is being done so to find which algorithm is more efficient and suits best to be used in a possible future grid scheduling system. A prototype that implements both algorithms is developed. In this master project, the algorithms are compared through and experimental quantitative research considering first a control setup Experiment Set I where the algorithms are executed by considering no parallelization and then two experimental setups Experiment Set II & III where parallelization is used and where parallelization is used and in addition separation of initialization costs.

The rest of the document is organized as follows: In chapter 2 the problem faced is defined and in chapter 3 the first step towards building a solution is given alongside with the analysis, contributions, research questions and the proposed methodology. Chapter 4 contains background information related to Grid Scheduling. In Chapter 5 the implemented algorithms are presented. Continuing, Chapter 6 and 7 contain all information regarding the experiments conducted during this project. Concluding the last chapter contains a proposed future word and a brief conclusion.

2 Defining the problem

2.1 The current system and its limitations

Shell is one of the biggest players in the oil industry. Its main business involves extracting oil from multiple sites across the globe. Extracting oil is a complex process that requires interdisciplinary knowledge (i.e. airplane turbines are used as electricity generators).

The assets (equipment) used for getting the oil to the surface is of high value for two major reasons. First, its high cost of acquirement, transportation and installation on the extraction sites. Second, the value it helps generate for the company (barrels of oil per day). On the top of that, the fact that possible problems in cases of failure of the assets may result in undesirable situations such as explosions, that may even put human life at risk or oil leaks, that may affect negatively the company's publicity and introduce additional cost of operation (e.g. fines, restoring the polluted environment), make real-time monitoring of the assets a very crucial process.

For that reason, Shell is gathering a great number of data, using sensors on assets, in a single database. The performance monitoring system (Smart Connect), which in turn is a set of subsystems, is responsible for processing and visualizing the data produced by the assets, giving a deeper understanding of the condition of the assets. In Figure 1 Smart Connect's levels of calculations are presented (starting from Level 1). It should be also mentioned that the calculations of following levels are dependent on the results of the previous. At the moment, Level, Level 2 and Level 3 calculations are being used for the majority of assets.

Figure 1 - Smart Connect's Levels of Calculations (starting from Level 1).
Each level's inputs have dependencies on the outputs of previous levels.
Currently Level 1, 2 and 3 are performed for the majority of assets.

The current system was initially conceptualized by engineers that were familiar and knew how the pumps worked about 10 years ago. As these engineers where not familiar with good programming practices, the result was a set monolithic programs as described by C. Lopes in [1]. As the years progressed and more and more assets were added to the monitoring mechanism; there was an increased need for modularization of the system. Thus, a scheduling mechanism was set up that would become a centralized management point for all calculations. This master project's concern is making a step towards improving this scheduling solution.

That is because, this scheduling mechanism was not properly designed and was put into place by people who were no experts in constructing complex software systems. Most of them were engineers that had knowledge of the assets to be monitored and simple understanding about programming concepts. Thus, no software architecture or implementation design was thoughtfully made; on the contrary, the design of the system was guided mainly by the chosen technology (solutions provided by OSISoft). Due to that, a great many of "quick fixes" were applied here and there, bending every time the design so to serve the needs of implementing the new change. As a result of choosing the easiest and quickest solution every time, the system now has an entangled design, misuses the available resources and it needs highly specialized employees to operate it, because of its unnecessary high complexity.

The current solution is illustrated in Figure 2 and is described in the following paragraphs. It is constructed by independent, local schedulers; each one deployed at every machine used for scheduling. Input files and calculations are scattered throughout a big number of servers. Managing and troubleshooting in a magnitude of a few servers was thought to an easy task at start, but now it requires a lot of effort and dedicated, specialized personnel. This of course, also makes it hard to resolve performance issues that derive from the I/O costs that may be added after some quick fixes.

Dependencies between calculations are not resolved. For example, there is no point of running performance monitoring calculations against assets that are not operational. Nevertheless, currently, all calculations are being executed, no matter if assets are being used, cause calculations are distributed to servers based on Level (i.e. all Level 1 calculations are performed only on Server 1).

Moreover, during the implementation, time concurrency was not considered to be much of importance. To deal with concurrency issues, time offsets are being applied from the start of the interval to mark the start of execution or by moving the calculation to another server by a user. This kind of static, local scheduling still exists and is managed by people that are constantly checking if the processed data seem to be correct and the users who take action accordingly. So, whenever a new task is added to a server, the possibility of running this additional task on that server is verified by deploying and running this new task and inspecting that all scheduled tasks are completed within one minute period. Of course this one minute mentioned earlier is not something that is based on a scientific research but rather something that the engineers that created the system agreed upon and thought as acceptable. As wear of assets as well as assets' equipment is happening with different rates, it would be more appropriate if the possibility to program tasks to be done with different intervals and still be easy to verify that the system can process these tasks successfully.

Additionally, the scheduling mechanism does not take availability and fault-tolerance into almost any consideration. For a real-time performance monitoring system though one should expect that availability of the output data is one of the most important aspects to be considered. However, there is no mechanism in place to keep calculations that are not being performed in case of hardware or software failures. The system now has no knowledge if computational machines are actually up and running because of the isolated and local nature of scheduling. When a machine is not functioning, values that were expected to be

calculated by it are left blank in the database. Also, if wrong input data exist in the database, at the time a calculation is about to be executed then wrong processed data will be provided back. This would be somewhat alright if there was a way to reschedule calculations done in the past, but the current solution does not provide this option; which in turn does not support the concept of determining the performance of assets on the long run. Missing or wrong processed data make impossible or unreliable any kind long term meta-processing that would provide trends and other useful insights to the organization.

Nowadays, the system is running at its limits, as hardware resources being used inefficiently. The Smart Connect team is yet again facing the need of acquiring new computers in order to deal with a set of new computations being added thought the spectrum of all assets.

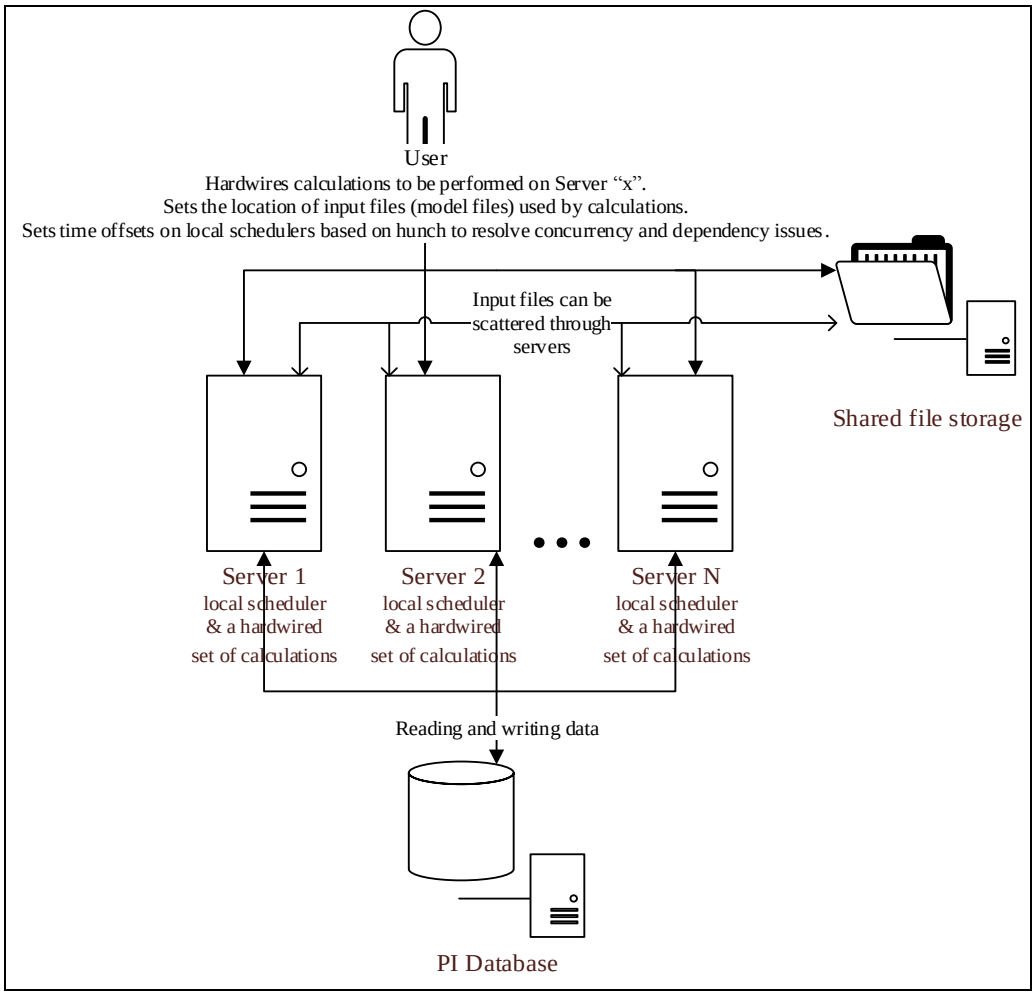


Figure 2 - An overview of the current system that performs the calculations of Smart Cornnect. At each server where calculations are executed there is a local, independent schedudler to which a user assigns calculations to be executed. When wrong values or congestion on a server is idendified by personel, a user will try to resolve this problem by adding a time offset, indicating the start time of the calculation for each period or putting it on a different server. Based on the write permitions a user has on the servers, some input files may be located on a different server than the one that will use them. Thin-headed arrows indicate that behavior. For maintainability reasons, each Server is given a specified Level of calculations or a subset of it to perfrom.

2.2 Framing the problem

A big amount of calculations has to be performed periodically. This period is currently set to 1 minute. Within this timeframe, it is already known that all these calculations cannot be performed from a single machine (that is currently available to the Smart Connect team). Furthermore, additional calculations are being added regularly to the system. Scalability is a major issue.

In respect of resources that are available for performing the calculations there nature is heterogeneous. This means that there are different types of machines with different specifications that are offered for the execution of the calculations. Heterogeneity of machines should be taken into account.

The results of some calculations are dependent upon the output of other calculations that are executed in the same period. In other words, in order to produce correct results calculations should be executed in a statically known, predefined order. An independent calculation (root calculation) produces input for a set of other calculations which in turn produce input for another set of calculations and so forth. The depth of the dependencies will not excide 6 (as shown in Figure 1) within the next few years. In these sets of calculations, the majority can be calculated in a very small amount of time, but there are some calculations that take significantly more time to be completed. In order to get correct results we should adhere to the precedence order of the calculations.

All calculations are being performed using VB.NET code, grouped in DLL files, which expose the exact same set of predefined public methods, adhering to an implementation contract of a specific interface. We can identify those methods in the following categories:

- Initialization methods, that determine specific constants and needed to be executed only once at the beginning of the calculation
- Execution methods, where arguments are retrieved from a database and calculations are performed
- Finalization methods that should be executed if the calculation is not to be executed anymore.

All programs written for the .NET framework, regardless of programming language, are executed by the Common Language Runtime (CLR). This frames the technology stack of this project to .NET languages.

Finally, it should be mentioned that currently, all execution methods are retrieving input from a PI database and after completion they insert an entry back in the PI database.

3 A first step towards resolving the problem

In this chapter the problem is analyzed and scoped down to creating a scalable infrastructure that it will have performance, described as the ability to execute a predefined set of tasks in the shortest time possible, as its main driver but will also favor other aspects such as simplicity and ease of adaptation to new needs that are going to arise in the future.

3.1 A general solution framework

From the previous chapter, it is obvious that a unified scheduling mechanism for a grid of computers known as Grid Scheduling (GS) is the solution to that problem. In order to design a grid scheduling mechanism tailored for the calculations used by Smart Connect, frameworks and infrastructures proposed in related scientific literature will examined and combine their strengths. One of the more widely used systems is UTOPIA, which is presented by Zhou et al in [2], Condor described by Basney in [3]. An additional basic notion of a high level architectural design of a system that schedules tasks to be executed in a grid of computers, is provided by J. Schopf in [4] which can provide insight on the components/modules of the new system.

All of these systems, compute all the required tasks, using a Master-Slave system[1]; since the problem they are solving is of indeterminate size and must be deployed on a large and unreliable workforce. This MS model is well-suited for our problem space since it can be examined independently (in the granularity of calculations or jobs), yet the progress has to be coordinated via intermediate results.

The capability of execution of all these systems is primarily based on the scheduling algorithm they implement which in turn is tightly coupled to the used tasks and the environment in which they schedule. The decisions a scheduler makes are only as good as the information it takes under consideration. This is easily understood as the place and the timing a task is executed, affects the overall time of execution. As a really illustrative example one may think of an algorithm that just distributes tasks equally on all available machines without taking under consideration the execution costs of tasks on the available servers. In this simple example it is both shown that such algorithm doesn't use resources efficiently and its ignorance of any precedence orders that should be followed so that the results produced by the algorithm are credible.

So, in this project, the most renowned algorithms (in their fundamental form) that are a good fit in the context of this problem are to be compared with performance as their primary criterion.

3.2 Framing the candidate Grid Scheduling algorithms

In this section the focus turns to minimizing the algorithms that will be considered, based on the specifics of the problem at hand. Algorithms that have the following characteristics are going to be considered as candidates for the new system: static & batch mode, since all the problem's tasks are known from the start and because we have to reschedule all of them with a predefined period. Additionally, algorithms will be scoped to only those that are capable of scheduling to a heterogeneous environment. Furthermore, the algorithms should be selected based on makespan (execution time of all calculations given in one period)

[1] In the literature this is also found as Master-Worker system, where workers are acting as slaves here do.

minimization as an optimality criterion. This is a logical decision for this situation since calculations are repeated on a standard, fixed interval.

Of course, choosing only from the class of the dependent algorithms would be the only way to go, one may argue, but due to the fact that even when the tasks at hand are grouped into their respective jobs[1], they are still computationally cheap, independent algorithms are also possible candidates. Thus, both, dependent and independent types of algorithms should be considered, in order to minimize makespan.

For static dependent task algorithms, list scheduling algorithms are regarded more highly in contrast to clustering, duplication and genetic algorithms. The algorithms in this category provide a good quality of schedules and their performance is comparable with the other categories at lower scheduling time. Consequently, only the list scheduling algorithms will be considered here.

3.3 Algorithm selection

Due to time constrains of the project, only one algorithm for independent tasks and one for dependent will be chosen, implemented and evaluated performance-wise. Algorithms that belong to the same category have been compared in several previous studies (usually the first such comparison is in the paper they are introduced) and hence, their performance can be compared through literature research. In this study, the most well-known algorithm from each category that is prominent to give the best results, in its category, is selected and is compared through a quantitative research.

The motivation on utilizing independent category algorithms lays into the following reasons:

- Independent algorithms are far more simple to implement, test, adopt and reason about
- Scheduling on a group of tasks (job) level requires less time than on a task level.

Thus, it will be assessed if the benefits of the independent scheduling algorithms can be utilized by the new scheduler.

In Table 1 the most popular algorithms are presented and are these that are taken under consideration as candidates. These algorithms satisfy the criteria discussed in the previous section [Section 3.2].

The computation costs of the tasks that will be scheduled by the independent algorithm are discussed in the section “The Tasks” of the next chapter and more precisely are presented in Figure 12.

In MET every task is given to the resource that minimizes the execution time for that task, no matter if the resources is available or not. MCT selects randomly a task each time and assigns it to the resource that minimizes the completion time. MinMin prioritizes the tasks based on their minimum completion time and then using this order, it assigns each of them to the resource that minimizes the overall completion time. MaxMin is the same with MinMin but it prioritizes the tasks based on the tasks’ maximum completion time. Lastly, Suffrage is based on the notion that a task should be assigned to a certain resource and if it does not

[1] A Directed Acyclic Graph (DAG) of calculations that is going to be carried out in a specified precedence along with its meta-properties (i.e. period of execution, order of the calculations) is going to be referred as “job” from now on. By introducing the concept of jobs to our design, the ability of treating the group of tasks included in a job as an independent entity is given, forming lets us say a bigger, but “independent task”; and thus make use of much simpler algorithms.

go to that resource, it will suffer the most. For each task, a suffrage value is defined as the difference between its best minimum completion time and its second-best minimum completion time. Tasks with high suffrage value take precedence during scheduling.

Table 1 - The candidate algorithms considered in this study after the applied refinement.
These algorithms have been selected as they are well suited, are studied the most in scientific literature.

1st level of categorization refinement	2nd level of categorization refinement	Algorithms
Static + batch mode	Independent	Minimum Execution Time (MET) Minimum Completion Time (MCT) MinMin MaxMin[1] Suffrage
	Dependent + List Scheduling	Dynamic Level Scheduling (DLS) Heterogeneous Earliest First Time (HEFT) Levelized-Min Time (LTM) Mapping Heuristic (MH)

It is evident that for independent tasks (in this case jobs), MaxMin is the best suitor, as it is:

- the most efficient when it comes to computationally cheap tasks with a few ones that are more intensive (an example of how the algorithm works is given in the next chapter where the algorithm is presented) along with Suffrage, but also
- simpler to implement and quicker to produce a schedule than Suffrage.

Now focus is turned to the category of dependent, list scheduling algorithms. The algorithms shown in Table 1 are picked so that all support heterogeneous processors. From all these algorithms, HEFT as introduced by Topcuoglu et al in [5], is the most efficient; both in terms of makespan[2] minimization and also the time it takes to produce a schedule (complexity).

Even though the running time of the algorithm is not going to be measured due to limitations of resources, it is of value to state that

- DLS as presented in [6] by Sih and Lee has a complexity is $O(u^3 r)$
- LTM is shown in [7] by Iverson et al to have a complexity of $O(u^2 r^2)$
- HEFT has a complexity of $O(u * r)$

[1] In literature can also be found with the name LTF/MFT which stands for Largest Task First / Minimizing Finish Time

[2] At this point it should be mentioned that Lookahead (a variation of HEFT), presented by Bittencourt et al in [28], proclaims to even improve HEFT's makespan and is positioned as the state of the art algorithm in its class. Nevertheless, since it an improvement of the original algorithm it should be considered as a future work, in case the results of HEFT are prominent.

- MH introduced by El-Rewini and Lewis in [8] has a complexity of $O(u(r^3u + e))$

where u are the number of tasks and r the number of resources.

To sum up, even though the traditional way to solve the problem at hand could be argued to choose only one of the dependent, list scheduling algorithms, because of the specifications of the tasks at hand, even independent scheduling algorithms can be utilized which consequently can have a lot of benefits. MaxMin and HEFT are the algorithms that their efficiency will be investigated in this project. Both of them are concerned with minimizing makespan, work very well with computationally cheap tasks, are widely used and are recognized even as benchmarking algorithms in their class.

3.4 Data locality - one additional concern

Because all of the calculations require the retrieval of initialization data (a model file), and due to the fact that they do not take a great amount of time to be executed, maintaining locally the instantiated type of the calculations may contribute in a lowering the overall execution time. Data locality is an issue that as shown by Ranganatha et al in [9], if not considered, even the best scheduling algorithms may be impeded by data transfer bottlenecks.

3.5 Contributions & Research Questions

Having discussed the context of Smart Connect, its limitations and based on knowledge on the problem of the scheduling mechanism the following contributions and research questions are presented:

Contribution 1: Development of a Grid Scheduling prototype that is able to scale up its calculation execution capabilities, by simply adding more computers to it.

Contribution 2: Considering two of the most efficient algorithms for this problem an implementation of the algorithms MaxMin on a group of calculations (job) granularity and HEFT on calculation granularity is developed.

HEFT is the traditional choice to solve this problem, whereas MaxMin can be utilized if calculations are grouped into jobs where their dependencies are resolved by setting the execution order of the calculations inside each job. The ability to use MaxMin is also reinforced by small amount of time even the groups of calculations require to be performed. Even though these algorithms have not been compared before in scientific literature, due to their different categorization, it is expected that MaxMin will yield comparable or even better makespan values (time required to complete all calculations at hand).

This hypothesis is based fundamentally on two facts. First, on the knowledge that even jobs are computationally cheap (as it will be shown in the next chapter) and can therefore be executed in a sequence without affecting significantly the overall makespan. Moreover, HEFT may leave some empty "processing time slots" throughout the schedule, where in the contrary, MaxMin may leave empty "processing time slots" only at the end.

Due to the small amount of time each calculation requires and the big number of them, an analytical comparison of the algorithms would only be based on models, assumptions and "roundings" that in the end would be far from reality. That kind of results can be easily obtained by just running the algorithms and considering only the schedule length each algorithm produces. Instead of considering the schedule length as the makespan, a thorough investigation of the actual makespan is going to be performed by measuring

the performance and the efficiency of the algorithms, measuring the actual execution makespan. This is believed to produce more reliable results on how the makespan of the algorithms react as load increases.

As an outcome of the above assertions, the following research question is conveyed.

Research Question 1: Which of the implemented algorithms perform better?

A logical improvement of the algorithms would be if they would take under consideration all the processors that reside within a machine. Computational machines that incorporate more than one processor are around for several years. These systems share a common, high level architecture, which is shown in Figure 3. This illustrates that a physical machine can perform in parallel more than one operations, as long as transferring data is not an issue. In other words, the performance of computationally intensive operations is improved linearly when adding more processors.

For this reason, the following question was formulated, which is going to illustrate how much execution performance is affected by the simultaneous usage of all processors within machines[1] and how much it is connected to I/O operations.

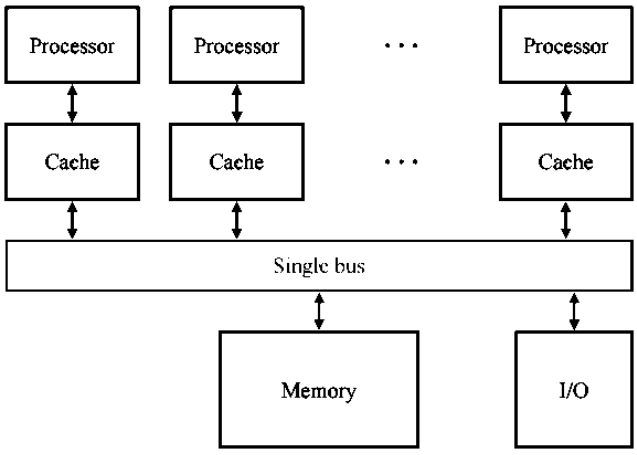


Figure 3 - The general architecture of a multiprocessor machine.
This clearly indicates that this architecture can utilize all of its processors as long as data transfer is not on the way.

Research Question 2: Can benefits be obtained if parallelization techniques are employed such that all processors of the machines can be utilized? Which algorithm performs better in this situation?

Continuing, due to the fact that calculations are being performed periodically, initialization costs of the calculations and jobs can be avoided if separated from the calculation part. Thus initialization is needed to be performed only once for each machine and only the calculation part (which is actually what we are concerned of) has to be performed each time. It is expected that by keeping the calculations' instances

[1]The performance results here are expected to differ to the performance results of the real calculations as the ones used in this study are not writing back to the database but in the model file stored in local storage. Nevertheless, any I/O bottlenecks are going to be exposed.

locally at each machine's memory, I/O costs caused by initializations of calculations' types can be significantly minimized. The following research question was formulated in respect to that though.

Research Question 3: How does separation of initialization costs affect the performance of the algorithms for the specified calculations? Which algorithm performs better in that occasion?

3.6 Methodology

For this project an experimental quantitative research methodology is followed to compare the performance of the selected algorithms.

This method is preferred instead of a simple analysis of the algorithms because of the following. The algorithms under comparison are of different classification and have never compared in literature as they work on different kind of tasks (dependent/independent). In order to be able to use independent algorithms dependencies were resolved by grouping and ordering tasks so that an independent task is formulated which in turn adds some additional overhead but nevertheless supports higher CPU utilization. Also, an experimental approach is going to provide more insight about potential bottlenecks that we are not aware of and may have to be mitigated in the near future.

In order to reason about the performance of the new, proposed system, a working part of it (prototype) is developed. For each research question proposed, an experiment will be performed. Each experiment should be repeated as many times as possible and in any case, enough times such that from its results can be argued that if the experiment is repeated again, it is going to yield a result close to a specific value point.

To minimize confounding variables while taking measurements, for each set of experiments presented here, the two algorithms are assessed by being deployed and then processing the same set of calculations on the same cluster of test machines in different times (algorithms are not run in parallel/competitively against one another).

4 Grid Scheduling

4.1 Definition

In order to achieve easy scalability and management of the calculations, a new hardware and software infrastructure must be developed; and it should provide an environment where physical and nonphysical resources (i.e. computers, data) can be shared and coordinated in order to achieve the desired goal. Such a kind of infrastructure is known as the Grid [10]. The development of a Grid Application that is going to be responsible for orchestrating the calculations to several machines is required.

Assigning tasks to machines (matching) and defining the execution order of the tasks assigned to each machine (scheduling), constitute a process called mapping [11]. Because of its obvious practical importance, it has been the subject of extensive research form the 50's and a big amount of literature has been created.

Augmenting on this idea, a more prevalent term that describes the whole procedure of executing tasks in a Grid infrastructure is Grid Scheduling (GS), and incorporates the discovery of the available resources, information gathering about the resources and the mapping of tasks to them as well as the execution of the tasks [4].

4.2 Frequently Used Terms

A specific terminology has been used throughout the relevant scientific literature regarding Grid Scheduling. For clarity, we specify a list of frequently used terms in Table 2:

Table 2 - Frequently used terms

Term	Description
Tasks[1]	are atomic units to be scheduled by the scheduler and assigned to a resource.
Properties of a task	are parameters like processor/memory cost, priority, etc.
Job	is a set of tasks that will be carried out on resources along with relevant properties.
Resource	is an entity that can perform at most one task at any time (i.e. a processor).
Slave	is an autonomous entity (machine) composed of one or more resources.
Scheduler/Master	An entity responsible for making the mapping and scheduling of tasks to resources or slave nodes.

[1] In the context of our problem, tasks are corresponding to calculations. Hence, in favor of understandability we are going to use the term calculations in the development of the prototype and the related experiment.

4.3 The three parameters of the Grid Scheduling problem

Let us say that we have m number resources R_i $(i = 1, \dots m)$ that have to process n tasks T_j $(j = 1, \dots n)$. A mapping is an allocation of one or more time intervals on one or more resources to each task. A Mapping is feasible if:

- no two time intervals on the same resource overlap,
- no two time intervals allocated to the same task overlap and in addition
- it meets specific requirements concerning the machine environment and task properties.

The machine environment, the task properties and the optimality criterion together define the problem.

For each of the three parameters of the grid scheduling problem we will now introduce the concepts that will be used in our prototype.

4.3.1 Environment under consideration

Our environment is constituted by a variety of heterogeneous machines. The technique of integrating and coordinating, non-homogeneous machines, networks and interfaces is known as Heterogeneous Computing (HC) and it has become popular due to its increased performance benefits in combination with its low cost as stated by I. Foster and C. Kesselman in [12].

4.3.2 Task properties (of the tasks at hand)

Each task has a computational cost per machine.

No preemption of tasks is allowed. That basically means that tasks cannot be divided into smaller units, as it is also mention in the definition of the term above.

One of the properties is if a precedence relation between tasks is specified. This is derived from a directed acyclic graph (DAG) (a simple example is shown in Figure 4).The DAG is notated as $G = (V, E)$ where V is the set of tasks and E the set of edges between the tasks. Edges $e = (i, j)$ where $i, j \epsilon V$ represent the precedence constrain that require task i to be completed before j can begin [13] [14] [15].

In our case, because there is a dependency amongst calculations, a workflow of multiple calculations should be created. The precedence constraints, are given in advance and known a priori. We will represent this information utilizing a DAG.

There are some heterogeneous systems that partition tasks in a DAG into levels, so that there is no dependency between tasks in the same level. Using domain knowledge of the calculations at hand, it is observed that DAGs used in this project have already the form of simple trees were every level is calculated sequentially after the calculations of the previous level have been completed. That means the design of any software can be simplified with trees that are being breadth-first traversed as illustrated in Figure 5, where every level can be executed in parallel. If used, this level-by-level scheduling that takes into account only a subset of ready tasks at a time; it can be reasoned that it would impede performance because of not considering all ready tasks. Nevertheless, because the dependencies of every next level use all outputs of the previous level, this does not impose an issue in our case.

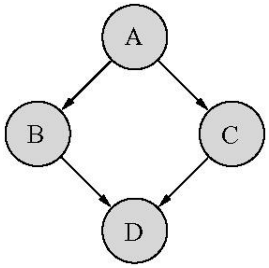


Figure 4 - A simple DAG. Where Nodes represent tasks and edges precedence constraints amongst them.

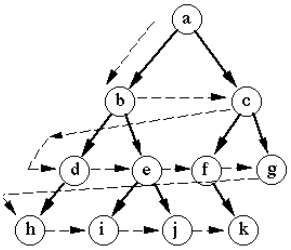


Figure 5 - Breadth-first traversal of a tree. In this case, a tree can be used to represent a DAG and all precedence constraints can be easily resolved if it is read in a breadth-first manner.

4.3.3 Optimality criterion

The objective when grid scheduling the execution of the calculations at hand is to minimize the overall finish time of the predefined set of tasks. This is to be done by properly mapping the tasks to the available machines and arranging their execution sequence. Thus, here makespan is referred as the overall time needed for all the given calculations to be executed (in literature this is may also be mentioned as schedule length); and it is equal with the elapsed time from the beginning of the first task till the end of the last task. Makespan is one of the most popular measures used for scheduling algorithms.

Other popular optimality criteria are throughput maximization and resource utilization. Focusing on minimizing the makespan, of one period where all the calculations are performed, will also guarantee throughput maximization. Finally, resource utilization is considered to be as of secondary priority here, since the ultimate goal is performing as much calculations as possible within a specified interval.

4.4 Complexity of the problem

For this project to be successful, it is of great importance to use algorithms that will be able to solve the scheduling problem within a reasonable amount of time. Complexity theory provides a mathematical framework in which computational problems can be studied so that they can be classified as 'easy' or 'hard'. A computational problem can be viewed as a function f that maps each input x in some given domain to an output $f(x)$. In this project, the interested part lays in investigating the time that it takes to compute $f(x)$ as a function of $|x|$.

The scheduling problem at hand is an optimization problem, where for an input x, the output $f(x)$ is the smallest value in a range of possible integral values. If this not associated to a decision problem where the output range is $\{yes, no\}$ then no know algorithm that solves this problem in polynomial time will be found. Here, a possible associated decision problem would be "Is there a feasible schedule that completes within the period T?" and these decisions can be taken within polynomial time.

NP denote the class of theses decision problems where the number of yes answers is bounded by a polynomial in $|x|$ and there is polynomial time algorithm to verify the correctness of this decision.

An NP-complete problem is the hardest problem in NP and the easiest of the NP-hard problems. In general, optimal multiprocessor scheduling is an NP-complete problem as shown by M. R. Garey and D. S. Johnson in [16] and worst case performance bounds are obtain for the algorithms that are used to solve this problem. Thus, it is already well known that the complexity of the DAG grid scheduling problem is NP-complete.

Algorithms for optimally scheduling a DAG in polynomial time are known only for three simple cases as stated by Coffman in [17] that does not match ours.

4.5 Taxonomies of Grid Scheduling Algorithms

As we dive deeper into the implementation specifics, we need to investigate available scheduling mechanisms and take them under consideration. GS can be categorized in *dynamic* or *static* depending if the complete set of tasks to be mapped is known beforehand [18]. Another kind of grouping can be done based on whether tasks are mapped onto the resources as they arrive, called *immediate mode*, or first collected into a set that is examined for mapping at specified events, *batch mode* [19]. Also, another categorization can be done, if having in mind whether the tasks are *independent* or *dependent* amongst them. When dependent tasks are considered, then there is a precedence order defined for the tasks.

An efficient way to do the independent calculations is described by Fujimoto and Hagihara in [20] while Yu et al describe how to deal with scheduling dependent tasks based on a workflow in [21]. A taxonomy of grid workflow scheduling is provided by Yu and Buyya in [22] and by Dong and Alk all in [23]; which in turn we can use as a guide about things that have to be considered when implementing a workflow grid scheduling mechanism.

Static dependent task algorithms, are subdivided into list scheduling, clustering and duplication algorithms. In list scheduling algorithms, an ordered list of tasks is constructed by assigning a priority to each one. Tasks are selected in the order of their priorities and each selected task is scheduled to a processor which minimizes a predefined cost function. The clustering algorithms are in generally for an unbounded number of processors and require a second phase to merge the task clusters onto a bounded number of processors and to order the task executions within each processor [24]. Similarly, task duplication-based are not considered as candidates due to their high complexity. Lastly, genetic algorithms even though provide good quality of schedules, their execution times (time they require in order to produce a schedule) are significantly higher in respect to other alternatives and tallying on that they are very difficult to be tested. Thus, list scheduling algorithms are regarded more highly in this context in contrast to clustering, duplication and genetic algorithms.

5 The algorithms

5.1 MaxMin

MaxMin as described by Ibarra et al in [25] is focused on scheduling independent tasks onto heterogeneous machines such that overall completion is minimized. To achieve that, MaxMin selects the task with the maximum completion time and assigns it to the resource with the minimum execution time.

```
for all tasks t_i in U
{
    for all resources r_j
    {
        calculate CT_ij
    }
}
do until all tasks in U are mapped
{
    for each task in U
    {
        find the CT_ij and the resource that obtains it
    }
    find the task n_k with the maximum CT_ij
    assign the task n_k to the resource r_l that gives the earliest completion time
    remove n_k from U
    update r_l
    update CT_il for all i
}
```

Figure 6 – The MaxMin algorithm

The pseudocode of MaxMin is shown in Figure 6. The complexity of MaxMin is $O(r * n^2)$ where r is the number of resources and n the number of the tasks to be scheduled.

MaxMin is a good choice when there are more short tasks than long tasks and this has been shown in several experiments as for instance shown in [19]. To show why this is the case, the following example is presented.

Example case: We have to schedule 6 tasks and we have 3 available resources. The execution cost of each task on each machine is given in Table 3. We assume that the machines are idle at start.

According to the MaxMin algorithm, the tasks are assigned to resources as shown in Table 3. The resulting scheduling is shown in Figure 7. So it is easy to understand why Max-Min gives a good makespan in such cases.

MaxMin begins with a set U of all unassigned tasks. Then, for all tasks in U, it determines the earliest (minimum) time that each task can be completed CT, given the projected idle times of each resource r_j and the estimated execution time of the task on each resource ET_{ij}.

$$CT_{ij} = ET_{ij} + r_j$$

Continuing, the task t with the highest CT is selected and assigned to the corresponding resource - hence the naming MaxMin. As a last step, the idle time for the projected machine is updated and the task removed from U. This step is repeated until U is the empty set.

Table 3 - Execution cost (in time units) of tasks on resources of the MaxMin example case

	Resource 1	Resource 2	Resource 3
Task 1	2	1	3
Task 2	7	5	12
Task 3	3	2	3
Task 4	2	1	4
Task 5	10	8	18
Task 6	2	1	4

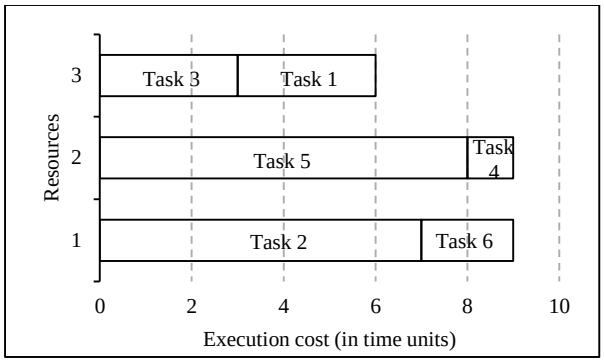


Figure 7 - Allocation of tasks to resources and makespan representation of the MaxMin example case

5.2 Heterogeneous Easiest Finish Time (HEFT)

HEFT was introduced by Topcuoglu et al. in [5]. It begins with computing the upward ranking $rank_u$ of every task. This is computed based on the average computation and communication costs and is defined as follows:

$$rank_u(n_i) = \overline{w_i} + \max_{n_j \in succ(n_i)} (\overline{c_{ij}} + rank_u(n_j))$$

Where:

- $\overline{w_i}$ is the average computation cost:

$$\overline{w_i} = \sum_{j=1}^{r} \frac{w_{ij}}{r}$$

- $\overline{c_{ij}}$ is the average communication cost. It represents the cost needed for transferring data from one resource to the other so that next dependent task can be executed. When both tasks are scheduled to the same resource, $c_{ij} = 0$. This is because it is assumed that the intra-resource cost is negligible.
- $succ(n_i)$ is the set of immediate successors of task n_i

Tasks are prioritized based on their upward ranking on a decreasing manner. That way precedence constraints are maintained.

After that the algorithm begins assigning the tasks, starting with the one that has highest priority, to resources that provide the earliest finish time (EFT) for that task, using insertion-based policy. This means that HEFT searches for the best possible idle time slot in all resources. Time slots thus between already scheduled tasks can be used. The appropriate time slot for the task n_i in every resource r_k, has of course to be found after the time all of the previous tasks that it is depended upon have been completed and all data are available on the resource r_k this time is known as ready time.

The pseudocode of the HEFT is shown in Figure 8. The complexity of HEFT is $O(r * e)$ where e is the number of edges and r is the number of resources.

For understandability reasons an example of how the algorithm works is given below.

Example case: The DAG of tasks shown in Figure 9 (this cannot be related to the MaxMin example case as it works on different types of tasks) have to be mapped and scheduled to the available resources. The execution cost of each task on each resource is known in advance and is presented in Table 4.

for all tasks
{
    compute $rank_u$
}
order the tasks in a *tasks list* in a decreasing order, based on their $rank_u$ values
for each task n_i in the *tasks list*
{
    for each resource r_k in the *resources*
    {
        compute $EFT(n_i, r_k)$ using insertion-based policy
    }
    assign n_i to the resource r_l that has minimum EFT for it
}

Figure 8 - The HEFT Algorithm

The algorithm starts with the prioritization of tasks, which means for each task the upward rank is being computed and then the tasks are shorted in a decreasing order of that value. In this case, the list of tasks after applying the prioritization order is [1, 2, 6, 3, 4, 5]. The result of scheduling the tasks to resources is illustrated in Figure 10.

For simplicity, we assume that there are no communication costs if a task is going to be scheduled at a different resource from its immediate ascendant. In other words, we assume that $c_{ij} = 0$.

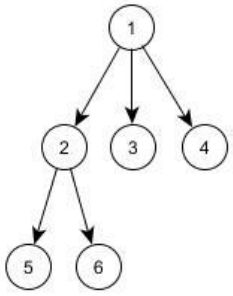


Figure 9 - The DAG used in the HEFT example case

Table 4 - Execution cost (in time units) of tasks on resources of the HEFT example case

Task	Resource 1	Resource 2	Resource 3	Resource 4
1	16	12	15	14
2	23	16	7	35
3	17	11	13	15
4	9	13	17	19
5	13	17	24	5
6	17	13	9	13

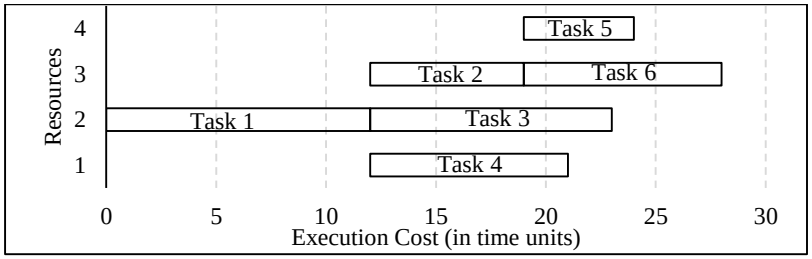


Figure 10 - Allocation of tasks to resources and makespan representation of the HEFT example case (once again this cannot be related to the MaxMin example case as it works on different types of tasks)

6 The components of the Experiments

In this chapter the components used in this project's experiments in order to obtain the results used to answer the research questions are presented.

6.1 The Tasks

In the experiment conducted a combination of simulation calculations and calculations that are used in the production environment were used. In Table 5 below the calculations that where provided and used to the experiment are described.

Table 5 – The calculations used in the experiments

Level	**Description of the calculations performed**
1st level (Root)	A calculation that simulates the real calculation was provided by the Smart Connect team. One 1st Level calculation is performed per site[1].
2nd level	The calculations for assets of type Pump Compressor (PCOMS) of all sites[1] monitored were provided. This type of calculations use: • a dll file were the code to be executed resides and • a csv file (model file) were are located input variables required for the initialization of the calculation. The PCOMS calculations provided are writing their results on a specified point of the csv file instead of the PI database.

The total number of all unique calculations used in this project is 1170 and they can be organized in 131 jobs. The total of these calculations are going to be referred as 1 Set of calculations. Thus, in order to simulate an increased workload, the Set of calculations is going to be increased.

Here a **threat to external validity** arises which is the population validity because only a subset of all the calculations (1st and 2nd level calculations) are considered. Significant changes may occur if the addition of more calculations show to transform the 100% histograms of the execution costs of calculations and jobs.

Another **thread to internal validity** lays in the calculations being used. The selected calculations may carry themselves the selection bias as the 1st level calculations here are only a simulation and the 2nd level calculations behave differently in that they write the results back in the model file; something that could add substantial I/O overhead.

Figure 11 illustrates the variance of the average execution costs of the calculations that we used. The values used for this histogram are the means of execution costs[2] of each calculation when they are executed without

[1] Site is referring to a location where one or more plants are. Each plant has physical assets (trains in the engineers' terminology) that are being monitored.

[2] The coefficient of variation of the execution cost of the calculations was $CV < 0{,}03$ out of a sample of 30 measurements taken for each calculation (without using any parallelism technique, e.g. threading).

parallelism. When those are grouped into their respective DAGs they formulate a combined cost that is shown in Figure 12 this illustrates why MaxMin is considered a good fit for this particular problem.

It should be mentioned at this point that the calculations are using just a few KBs of memory so there is no need to take memory into consideration, for this project, since all commodity machines, nowadays, have more than sufficient memory to run the calculations.

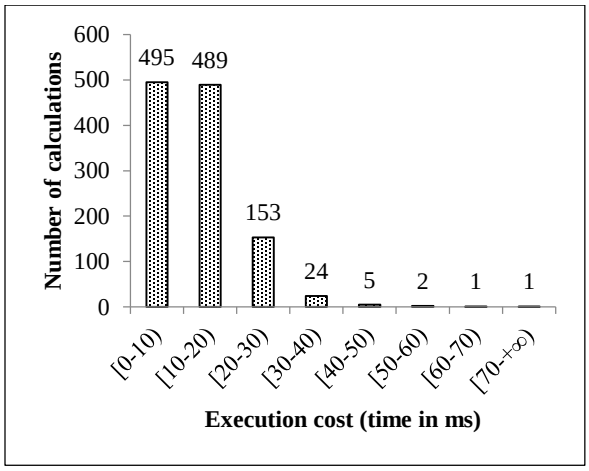


Figure 11 - Histogram showing the frequency of calculations (with a total number of 1170) in relation to their cost of execution

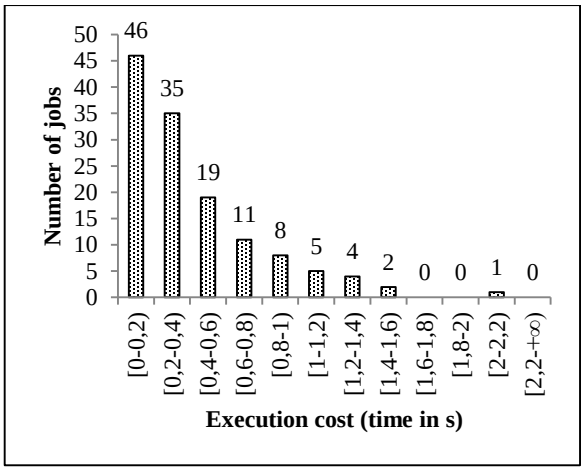


Figure 12 - Histogram showing the frequency of jobs (with a total number of 131) in relation to their cost of execution

6.2 The Machines

The machines, used for executing the tasks in the experiment, were two servers the specifications of which are shown in Table 6. All of the servers were using their highest possible clock speed at all times. No machine used had any kind of temporary overclocking mechanism; something that could interfere with the execution time of the calculations. Additionally, a database server in the same range of specifications that was running an instance of SQL Server 2012 R2 was used.

Table 6 - Specifications of the Servers used in execute calculations in the experiments

Properties of the Servers		**Server 1**	**Server 2**
Operating System		Windows Server 2008 R2 Enterprise with Service Pack 1	Windows Server 2008 R2 Enterprise with Service Pack 1
Processor	Type	Intel® Xeon®	Intel® Xeon®
	Model	E7-8837	E7-8837
	Clock frequency	2.66GHz	2.66GHz
	Number of processors	4	4
	Number of cores/processor	1	1
	Number of threads/core	1	1
Memory RAM		20GB	16GB
Hard Disk		Virtual VMware Disk (HDD) 35GB 7200rpm	Virtual VMware Disk (HDD) 100GB 7200rpm

Then small number of Servers used for performing the calculations brings forward a common **threat to validity** due to the low statistical power. In other words, it is impossible to make secure predictions of the makespan as more Servers are added in the infrastructure.

The scheduling was performed on a commodity laptop[1] in cooperation with the Database Server.

The SQL Server was used for "production purposes" and consequently was used simultaneously for other purposes, different of this experiment. Hence, it's CPU and Network usage was subject to great fluctuations even when it was not used for the experiment of this study. This is a potential **threat to validity** due to this experimental arrangement. Delays caused be congestion in the database may occur during the experiments that follow.

The laptop used had a great amount of services running in the background (enforced by Shell's policies).

6.3 The Network

All the Servers resided in the same cluster and were interconnected via a high speed intranet connection (with an average transfer rate of 35MB/s) and network latency <1ms and no firewall or antivirus system was existent in between them. The servers were located at Munich, Germany and were accessed from Rijswijk, The Netherlands using the laptop mentioned in the previous section.

6.4 The Prototype

6.4.1 A Master-Slave (MS) system

As described previously, all renowned grid scheduling systems found in the literature are using a Master-Slave setup, comprised by a set or slave nodes that are responsible for executing tasks and a master node that is orchestrating the slaves and schedules the tasks to them. In literature there are also systems that have the slaves organized in clusters and use an intermediary node between the slaves and the master, which abstracts a cluster of slaves from the master. In this context nevertheless this is quite unnecessary, as the number of servers used is currently 8 and it is not predicted to increase dramatically in the near future. In the following sections the slave and the master implementations are discussed.

6.4.2 Slaves

A Slave is a server program that waits to execute jobs or calculations upon request. It declares its existence to the system upon initialization via an entry to the database and provides all the needed information that the master will need so to access its services (endpoint information). A simple ping request has been implemented that the master can use in order to determine the availability of the slave.

In order not to interfere with execution performance of the work done, the prototype has independent optimized implementations for the execution of

- lists of jobs of calculations and

[1] Specifics of the laptop are not of importance as they do not interfere with the results presented in the experiments conducted for this study.

- lists of calculations.

Execution of job requires some additional overhead at the Slave level; Slaves have to retrieve information of the calculations (type and executable) of every job assigned to them following a breadth-first traversal beginning by the root calculation (which is already known). Interesting facts about the two implementations follow.

The Job Executor: receives as input an ordered list of the IDs of jobs to be executed. Then it initializes and executes each job. During the initialization of a job, the calculations that are part of it are retrieved and their respective types are instantiated and their "Initialize" method is called. After that the program continues with the execution of the "Calculate" method of every calculation of each job assigned to that slaves. To accomplice this functionality, the Job class of the Slaves is utilized. Within that class, the methods of the calculations are called sequentially[1] by going through the job's tree of calculations in a breadth-first manner. That way the order of execution is guaranteed. At the end, a function of the Logging class was called in order to save the interested properties of the execution in the database.

Slaves support the repeated execution of jobs, given an execution period has been defined. In the event that a job cannot be calculated within the specified interval, it is simply dropped and flagged in the database as one to be scheduled (pending). Even though a functionality for iterating execution of jobs on a specified interval is provided, it was not used during the experiments.

The Calculation Executor: receives a list of the calculations to be executed along with their respective Actual Start Time (AST). Based on that information, the program schedules the execution of the calculations methods accordingly. As in the previous implementation, a function of the Logging class was called in order to save the interested properties of the execution in the database. Both implementations rely on the same classes (Calculation and DLL) and code functions (Initialize and Calculate) for instantiating and performing the calculations.

6.4.3 Master or Scheduler

The Master or simply the Scheduler is the program that implements the grid scheduling algorithms and is responsible for retrieving the work that has to be performed, map it and schedule[2] it to Slaves. This program is referred to as Scheduler. It provides a single point of access and management of the system. The master upon initialization, uses the database and a simple ping request to search for possible connections to Slaves and after it has determined which of them are accessible it proceeds with accepting several commands:

- population of the database with tasks and grouping them into their respective jobs
- update of the average execution costs of tasks and jobs using logs
- retrieval of pending work from the database (all jobs marked with Status "False")
- execution of the retrieved worked
    - uses one of the implemented algorithms (the choice is make by the user during runtime)
    - sends execution requests to the slaves (the work to be done by each slave is determined by the implemented algorithm)

[1] It was considered that penalization at this point would not bring any significant benefit due to the amount of jobs scheduled. This may be researched as a future work.

[2] Processors in the same machines are considered as homogeneous and treated as of equal performance

The algorithm that will be used to schedule the work is selected upon runtime. The prototype abstracts the implementation of the algorithms used, by defining an abstract class "`GridScheduler`" that the concrete implementations of the algorithms should use. This abstract class receives the work that has to be completed and the available slaves and then leaves to the algorithm to choose how to prioritize the work and how to assign it to the slaves. In other words, the abstract class, has two abstract methods that the algorithms should implement. That are responsible for:

1. prioritization: prioritization of tasks using a specified function.
2. assignment: repeatedly select the task with the highest priority and assign it to the best resource possible (in accordance to the implementation of the respective algorithm).

Finally, after all assignments have been done, the `GridScheduler` abstract class orders the execution of the assigned work to the respective slaves by sending each one the schedule that they should perform.

By taking a look into the implementation of the algorithms themselves, it is very clear that HEFT is significantly more complex algorithm than MaxMin. Of course, this is because scheduling dependent tasks is far more difficult procedure. The ability of HEFT to use insertion-based policy generated the need of creating a class where the tasks can find the time slot that provides the earliest finish time on each available processor, and therefore select the one that is going to give the earliest finish time by booking a time slot on that processor.

6.4.4 Metacomputing Directory Service

The decisions a scheduler makes are only as good as the information provided to it. A high-throughput computing (HTC) scheduling mechanism that will be aware and capable of resource management such a framework is used by Condor. A resource aware system will enable us to better use computers based on preference (load balancing of tasks throughout the grid's resources or minimizing the use of required resources), as well as maintaining an agreed deadline standard and even making predictions on the impact of a calculation mapping.

Foster et al. in [26] has presented one of the most renowned metacomputing services known as Metacomputing Directory Service (MDS) and includes configuration details of all the Slaves (i.e. memory, CPU speed, number of CPUs), instantaneous information (i.e. CPU load, network bandwidth) and application specific information (i.e. memory requirements of the program structures). Other systems use some sort of a smaller variation of MDS usually referred as Load Indices that exist to offer the same functionality. Besides raw Slave information, calculation properties, these may include estimations of calculation, communication costs and memory consumption on specific Slaves and dependencies on other calculations. It easy to understand that an MDS is a collaboration of services that reside in both the master and the Slaves.

In our prototype, a simple MDS that will be used for the specific needs of this project has been developed. For testing of our prototype only the number of processors is required as each one represents a different resource. This information is used to instantiate representations of Slaves at the master and it is retrieved at start the start of execution, directly form the Slaves. Additionally, the MDS of the prototype should provide information about the execution costs which is something that both implemented algorithms require. This cost is defined by the master and uses the execution logs of the Slaves, which reside in an accessible from both parties database.

6.5 Deployment setup

The allocation of software to physical machines is presented in this section. The value of this section is great for two main reasons:

- machines are running services that in their turn impact their performance capabilities
- network bottlenecks that affect the overall performance are detectable

Thus, in order for one to reproduce the same performance results, of this experiment, the deployment of services to machines should be considered.

During the experiment the two servers available were used for executing the work. In other words the Slave version was executed on them. The grid scheduling was performed by the laptop using the Scheduler version. The database is also shown as it provides useful information about possible network bottlenecks. Figure 13 illustrates the deployment setup used during the experiments of this study.

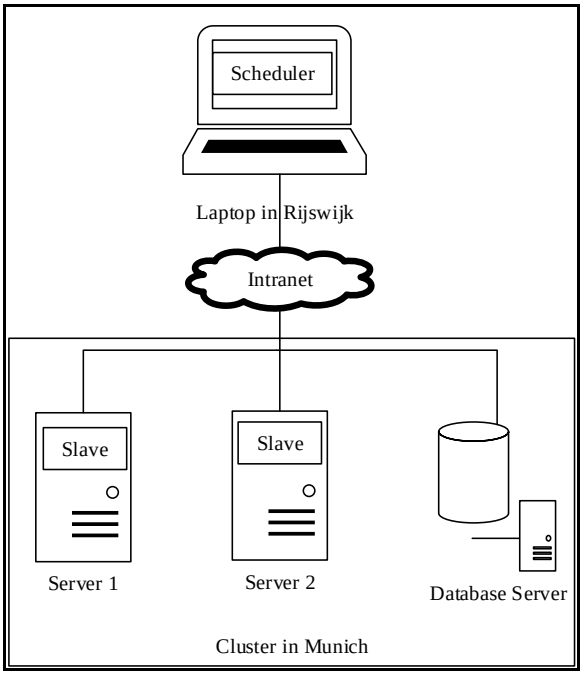


Figure 13 - Overview of allocation of programs to machines in the experiment. The available Servers are executing the Slave program used to perform the tasks whereas a commodity laptop is used to schedule the work to the Servers.

6.6 Comparison metrics

One of the major requirements of the system is that the calculations should be executed again and again with a specified period. That means that our scheduling algorithm should incorporate an objective function aiming to optimize the execution sequence of the predefined set of calculations in respect of makespan minimization.

In this section it is discussed what are the measurements used by relative scientific literature and why makespan has been chosen as the main measurement in this study. For each measurement, first, its definition will be presented and then a related discussion will follow where is appropriate.

6.6.1 Makespan

Makespan is defined as the time required for all tasks to be completed known also as schedule length. It's measurement is very easy since it requires only the start time of the entry task and the end time of the exit task. Makespan is considered as one of the main performance metrics of DAG scheduling. Thus, formally, makespan is defined as follows:

$$makespan = \max\{AFT(n_{exit}) - AST(n_{entry})\}$$

Where:

- $AFT(n_{exit})$ represents the Actual Finish Time of an exit task and
- $AST(n_{entry})$ the Actual Start Time of an entry task in the schedule.

In related literature, most of the times $AST(n_{entry})$ is equal to 0, since they consider it the start of the measurement point. So, the definition becomes $makespan = \max\{AFT(n_{exit})\}$.

6.6.2 Schedule Length Ration (SLR)

Another metric used frequently is Schedule Length Ration (SLR)**.** This metric is defined as:

$$SLR = \frac{makespan}{\sum_{n_i \in CP_{MIN}} \min_{p_j \in Q}\{w_{ij}\}}$$

Where:

- CP_{MIN} is the minimum Critical Path, meaning the minimum possible path from n_{entry} to n_{exit}

This means that SLR has always a value greater than 1, since the denominator is the lower bound of the makespan.

Because the exact set of calculations is used during this experiment, there is no need of calculating the SLR. The denominator is the same for the same set of tasks and only makespan will be subject to changes.

6.6.3 Speedup

Another popular metric is Speedup, which is the summation of the execution time of all tasks divided by the makespan. Formally,

$$speedup = \frac{\min_{p_j \in Q}\{\sum_{n_i \in V} w_{ij}\}}{makespan}$$

6.6.4 Efficiency

Efficiency is the ratio of speedup to the number of resources used. Hence, it is defined as:

$$efficiency = \frac{speedup}{number\ of\ resources}$$

6.6.5 Running Time of the Algorithm

Running Time of the Algorithm is the time required for each algorithm to provide the mapping and schedule of the given tasks to the respective machines. This is closely related to the complexity of the algorithm.

Because of the limited resources provided, it was not possible to take reliable measurements of the Running Time of the Algorithms. Nevertheless, it should be mentioned that HEFT, as expected, took a significant additional time to compute a schedule than MaxMin. This can be easily explained since HEFT, in this

experiment, is scheduling on a calculation granularity, whereas MaxMin, is scheduling groups of calculations.

6.7 Restrictions & Assumptions

In this project we are concerned with scheduling at the detailed, operational level. We pay no attention to tactical decisions, such as the determination of due dates, or to strategical decisions, such as the acquisition of machines.

Additionally, we assume the deterministic nature of the given calculations. In other words, all the information that defines a calculation instance, is known with certainty in advance.

This assumption also imposes the following **threat to validity;** the usage of the algorithms of this project is not feasible if the execution time of calculations used show extensive variability.

Moreover, it is indeed the case that some of the problem data are subject to random fluctuations, such as machine increased workload due to the influence of other applications. Nevertheless, those are not take this into account in the present study. This also presents a **threat to validity** if the results are generalized to Servers that have a substantial overhead of additional operations that they have to perform.

Furthermore, it should be mentioned that tie-breaking while using the algorithms is done randomly. This is because adding a tie-breaking policy will increase the time complexity of the algorithms without providing any significant improvements and it also such policy is not part of the algorithms as presented in literature.

7 The Experiments

In this chapter a description of the experiments, the procedure of performing them and retrieving their results of each algorithm separately for every set of experiment conducted presented in this project is defined. The performance of the algorithms was compared with respect to characteristics of the specified set of calculations and the available resources. The workload was increased by adding up the predefined set of calculations. The resources were increased by utilizing more processors. Before describing the experiments themselves, the procedure of how the results were obtained will be presented.

7.1 Data processing

7.1.1 Before the experiment

Before conducting any experiment, execution costs of each operation on each machine should become known. To obtain these costs a logging mechanism was used. This mechanism monitored the performance of the execution on two different levels of granularity. On a lower level, the execution time of every method of each calculation was monitored. Additionally, if jobs were scheduled, the initialization cost of all of them was also monitored (cost of retrieving from the database and creating the list of calculations that this job includes in the proper order). All of the durations monitored, were logged at the database. All jobs used were performed 30 times using these logging mechanisms described above. Then the costs of each calculation and job were estimated using the mean value of the execution costs of each calculation and job respectively this is how histograms in Figure 11 and Figure 12 were constructed. Logging of single calculations and jobs was not used during the experiments as it adds a substantial I/O overhead.

An investigation of spotting a regression towards the mean thread of validity was done here, but no extreme values were found. The Coefficient of Variation for each measurement was found to be $CV < 0,03$.

7.1.2 During the experiment

The start time and the duration of the execution of the set of calculations performed each time was monitored at each server and logged at the database.

For each experiment, a sample of 30 measurements was taken. This number was considered sufficient for the following reasons:

- the mean was not affected in a considerable extend after about the 15^{th} measurement for all experiments and the coefficient of variance of all the measurements for each experiment was observed to be below 0,05,
- the makespans observed help us safely recognize which algorithm performs better under every situation examined and
- the time the servers were available to be used for the purposes of this study.

For each measurement in every sample of each experiment the slaves were started and executed only one master's request. After the end of request, the slaves were terminated and restarted so another measurement can be taken. In that way there is no need to be concerned at this time with garbage collection operations that may affect execution times.

7.1.3 After the experiment

In order to avoid a possible experimenter bias during the extraction of the results, a systematic approach was followed.

1. For each execution of the specified calculations, the makespan was calculated as the highest overall execution time on all servers for each execution.
2. From those results, the respective mean makespan was calculated for each experiment conducted.
3. Additionally, the Coefficient of Variation (CV) was also obtained so that the extent of the distance of makespans from the mean makespan of each experiment can be depicted.
4. Finally, Speedup and Efficiency values were calculated.

7.2 Common parameters over all experiments

As discussed above all experiments share a same collection of parameters that for readability reasons have been collected in the following table.

Table 7 - Common parameters shared over all experiments

Description	**Value**
Number of runs of each unique experiment (every makespan value shown represents an experiment)	30
1 Set of Calculations contains	1170 unique Calculations (used for HEFT) or 131 Jobs (used for MaxMin) that contain the 1170 unique Calculations mentioned above
Makespan value point of each experiment	The mean value over the results taken from the experiment
The coefficient of variance of each makespan value point presented	CV < 0,05

7.3 Experiment Set I - Parallelization Off

7.3.1 Description

In the first set of experiments (Parallelization Off), the algorithms scheduled work to slaves taking no parallelism capabilities of the servers on which the slaves run into consideration. Additionally, the slaves executed all functions sequentially (using only one processor). For each calculation performed, the methods "Initialize" and "Calculate" were executed one after the other.

This experiment set addresses the first research question.

7.3.2 Results

As it is shown in Figure 14 and Figure 15 MaxMin provides steadily better performance results over HEFT. This is also easily depicted on the respective graphs of the speedup and efficiency.

The trend lines shown in Figure 14 and Figure 15 can be considered as a rough estimation of growth of makespan as workload increases and they follow the polynomial functions that are shown in Table 8 and are subject to criticism due to the law of small numbers.

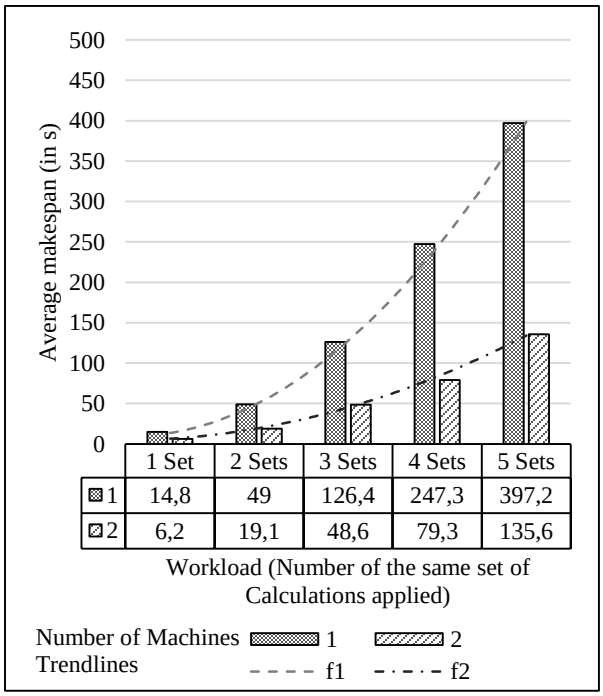


Figure 14 - Average makespan of MaxMin for different sets of loads when executed on a specified number of machines with no parallelization capabilities

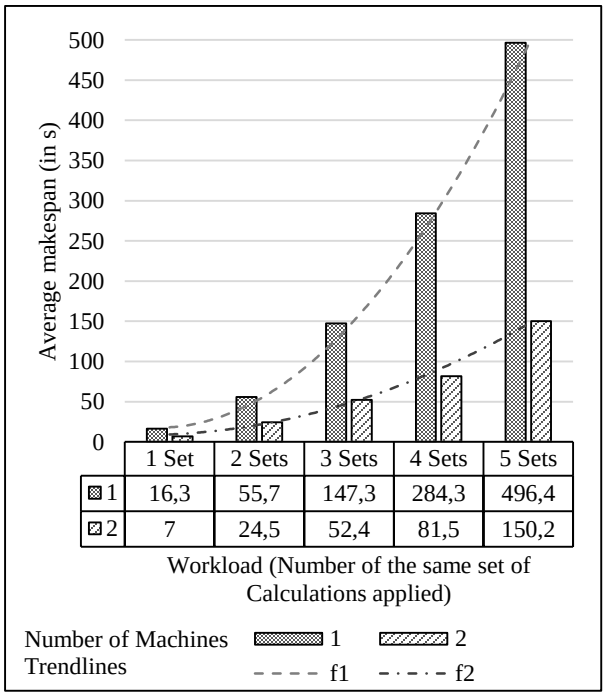


Figure 15 - Average makespan of HEFT for different sets of loads when executed on a specified number of machines with no parallelization capabilities

Table 8 - Functions of trendlines shown in Figure 14 and Figure 15

MaxMin	**HEFT**
$f1(x) = 19{,}636x^2 - 21{,}504x + 15{,}46$ $R^2 = 0{,}9998$	$f1(x) = 27{,}914x^2 - 48{,}606x + 38{,}76$ $R^2 = 0{,}9995$
$f2(x) = 6{,}2857x^2 - 5{,}8143x + 6{,}06$ $R^2 = 0{,}9971$	$f2(x) = 7{,}4x^2 - 10{,}06x + 11{,}9$ $R^2 = 0{,}9907$

Observation: During this set of experiments, CPU utilization (for all processors in the machine combined) was 24-25% for MaxMin and 23-25% for HEFT. In both cases, memory used was gradually increased and for the biggest workload reached about 55MB.

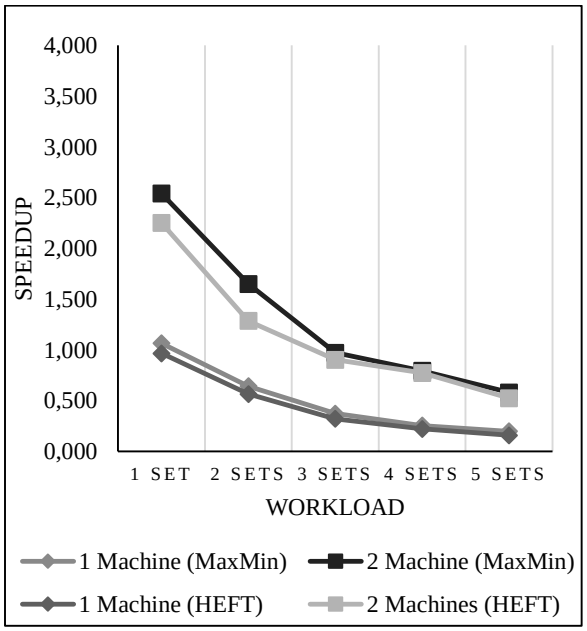


Figure 16 - Speedup of algorithms in association to workload represented as sets of calculations when no parallelization is used

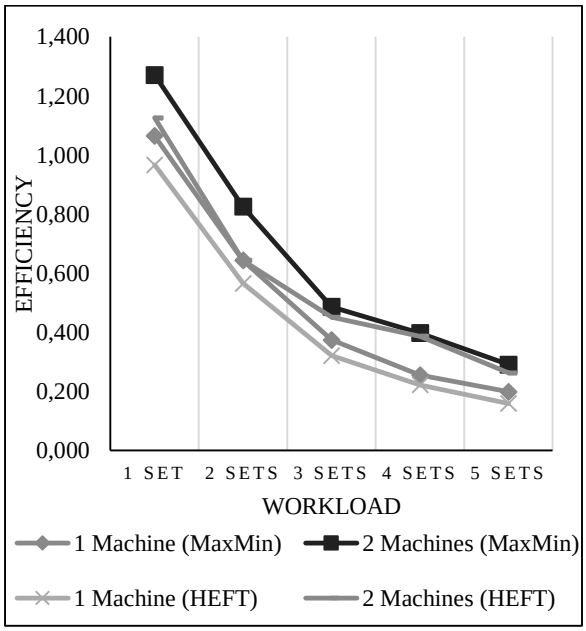


Figure 17 - Efficiency of the algorithms in association to workload represented as sets of calculations when no parallelization is used

7.4 Experiment Set II - Parallelization On

7.4.1 Description

In the second set of experiments (Parallelization On) parallelism on slaves was considered. The algorithms scheduled work to slaves while knowing the number of available processors each slave machine had. Slaves used their threading capabilities in order to utilize all available processors. Here again, for each calculation performed, the methods "Initialize" and "Calculate" were executed one after the other.

This experiment set along with the previous one addresses the second research question.

For this experiments System.Threading.Tasks.Task and System.Timers.Timer class were used for the MaxMin and HEFT implementations respectively. The Task object is executed asynchronously using a thread pool. The Timer object raises an event after an interval has passed.

7.4.2 Results

When parallelism is employed things turn to be better for HEFT. As presented in Figure 18 and Figure 18Figure 19, HEFT produces lower values of makespan at all times. Additionally, considerably higher performance of HEFT was observed as the load increases. This is better shown in the efficiency chart in Figure 21.

The trend lines shown in Figure 18 and Figure 19 can be considered as a rough estimation of growth of makespan as workload increases and they follow the polynomial functions that are shown in Table 9 and are subject to criticism due to the law of small numbers.

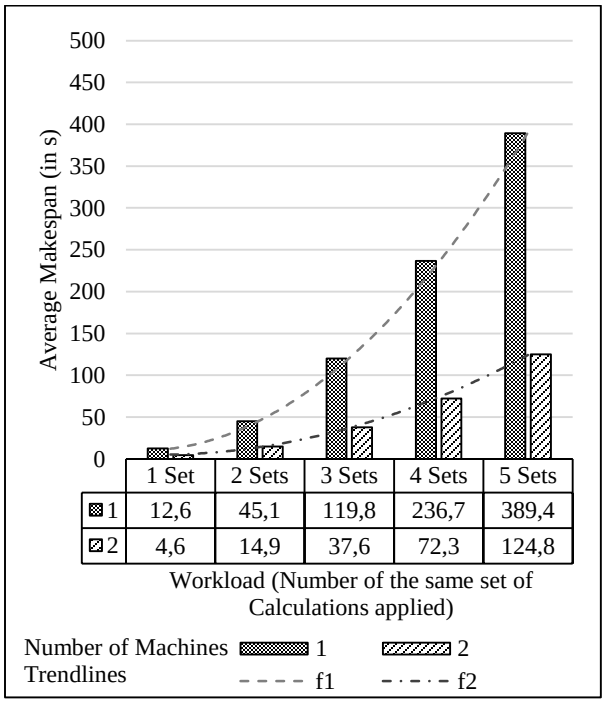


Figure 18 - Average makespan of MaxMin for different sets of loads when executed on a specified number of machines with parallelization capabilities

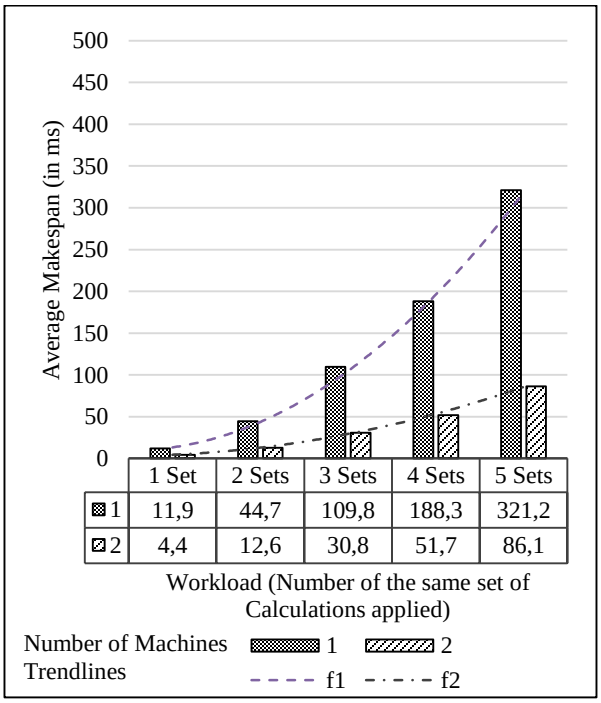


Figure 19 - Average makespan of HEFT for different sets of loads when executed on a specified number of machines with parallelization capabilities

Table 9 - Functions of trendlines shown in Figure 18 and Figure 19

MaxMin	**HEFT**
$f1(x) = 20{,}186x^2 - 26{,}594x + 18{,}46$ $R^2 = 1$	$f1(x) = 15{,}257x^2 - 15{,}323x + 13{,}32$ $R^2 = 0{,}9984$
$f2(x) = 6{,}8857x^2 - 11{,}534x + 9{,}7$ $R^2 = 0{,}9996$	$f2(x) = 3{,}9357x^2 - 3{,}3643x + 3{,}92$ $R^2 = 0{,}9986$

Observation: During this set of experiments, CPU utilization (for all processors in the machine combined) was 25-27% for MaxMin and 28-32% for HEFT. In both cases, memory used was gradually increased and for the biggest workload reached about 55MB.

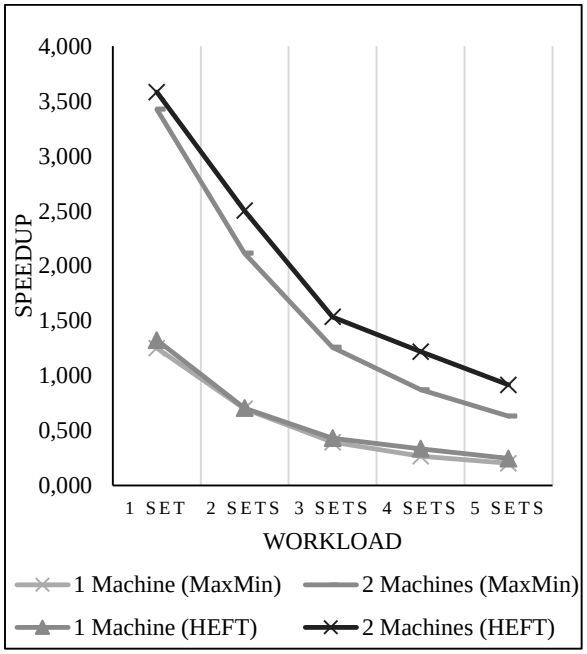


Figure 20 - Speedup of algorithms in association with workload represented as sets of calculations when parallelization is used.

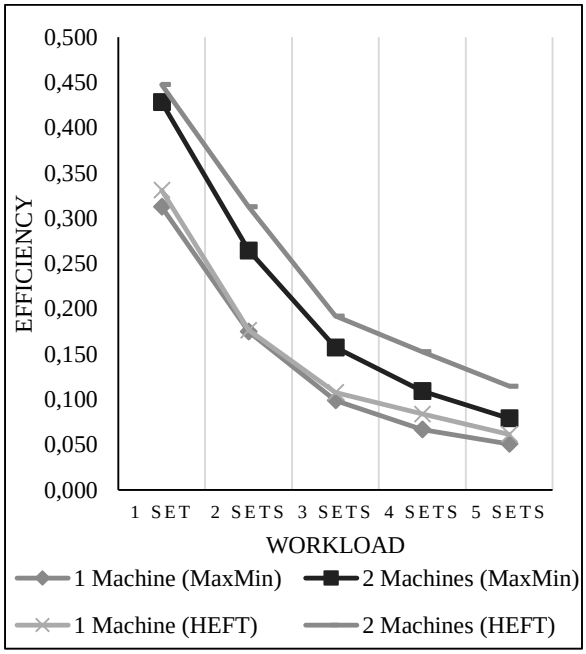


Figure 21 - Efficiency of the algorithms in association to workload represented as sets of calculations when parallelization is used.

7.5 Experiment Set III - Parallelization On and Separation of Initialization

7.5.1 Description

In the last set of experiments (Parallelization On and Separation of Initialization), the second set of experiments was repeated; but this time, initialization of calculations was separated from the calculation part. This was considered because:

- initialization of calculations is mainly an IO intensive part that has to be performed only once per machine, per calculation.
- calculations have to be executed periodically.

The reasons above justify the belief that significant benefits could be obtained if initialization could be separated from the computation of the calculation and thus occur only once. Thus, during this set of experiments, the time required for the initialization of each job and calculation is not taken under consideration.

So, after performing this Experiment Set too, the last research question can be addressed.

7.5.2 Results

When separating initialization from the calculation, the results are not showing any noteworthy performance differences of the schedules of the algorithms when only one machine is used. Nevertheless, when both machines were used MaxMin showed that it outperforms HEFT substantially as more workload is applied as depicted in Figure 25.

The trend lines shown in Figure 22 and Figure 23 and can be considered as a rough estimation of growth of makespan as workload increases and they follow the polynomial functions that are shown in Table 10 and are subject to criticism due to the law of small numbers.

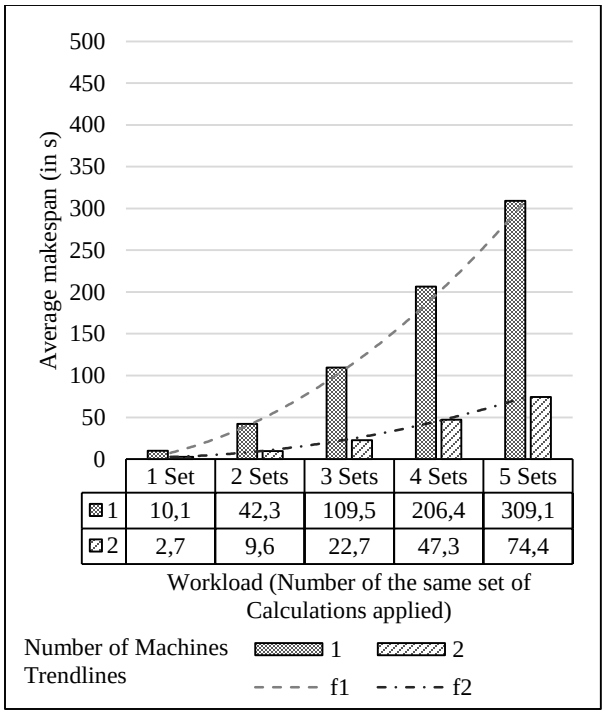


Figure 22 - Average makespan of MaxMin for different sets of loads when executed on a specified number of machines with parallelization capabilities and separation of initialization

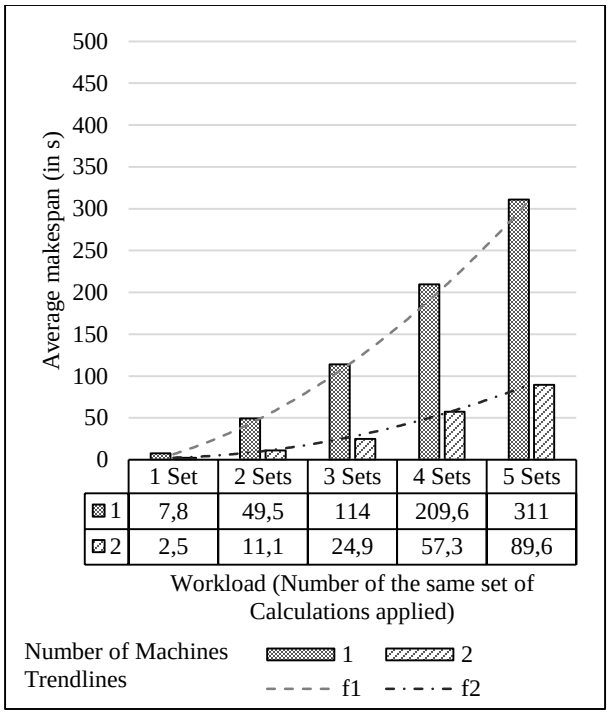


Figure 23 - Average makespan of MaxMin for different sets of loads when executed on a specified number of machines with parallelization capabilities and separation of initialization

Table 10 - Functions of trendlines shown in Figure 22 and Figure 23

MaxMin		**HEFT**	
$f1(x) = 12{,}193x^2 + 3{,}0529x - 7{,}8$	$R^2 = 0{,}9985$	$f1(x) = 10{,}75x^2 + 12{,}15x - 16{,}32$	$R^2 = 0{,}9993$
$f2(x) = 3{,}7071x^2 - 4{,}1329x + 2{,}96$	$R^2 = 0{,}9988$	$f2(x) = 4{,}7143x^2 - 6{,}2457x + 3{,}96$	$R^2 = 0{,}9966$

Observation: During this set of experiments, CPU utilization (for all processors in the machine combined) was 31-34% for MaxMin and 29-33% for HEFT. In both cases, memory used was increased rapidly in the initialization part and maintained that level on the execution part. For the biggest workload memory used was about 55MB.

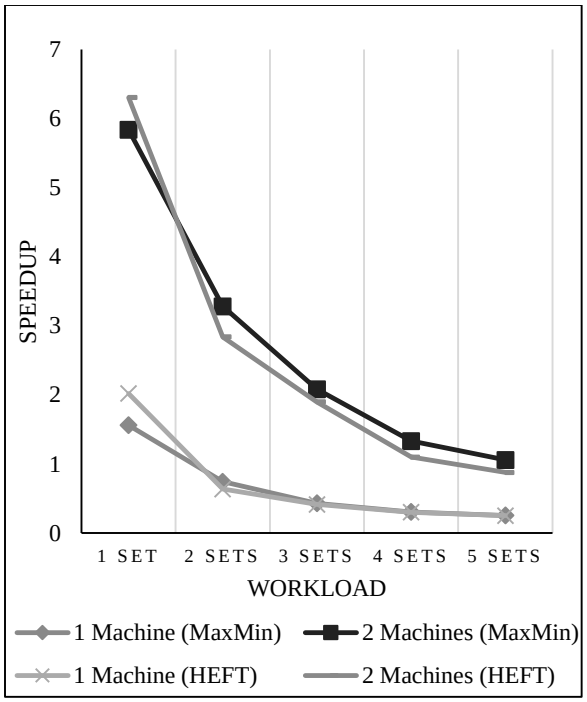


Figure 24 - Speedup of algorithms in association with workload represented as sets of calculations when parallelization is used and initialization is separated

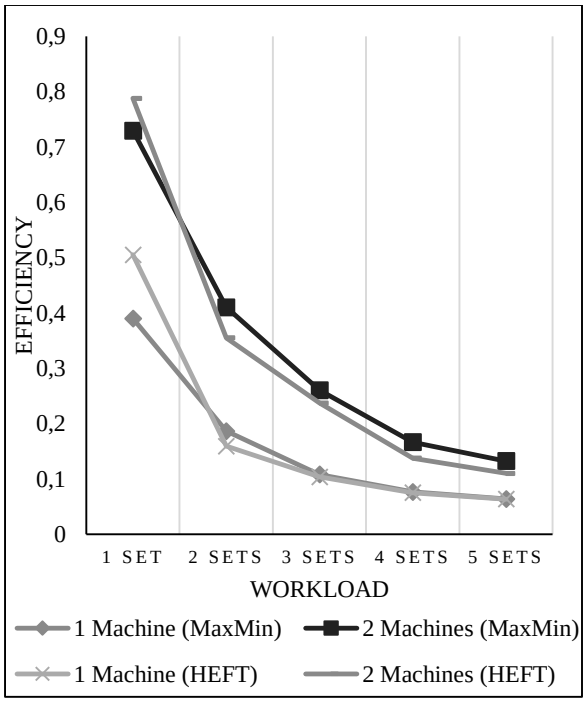


Figure 25 - Efficiency of the algorithms in association to workload represented as sets of calculations when parallelization is used and initialization is separated

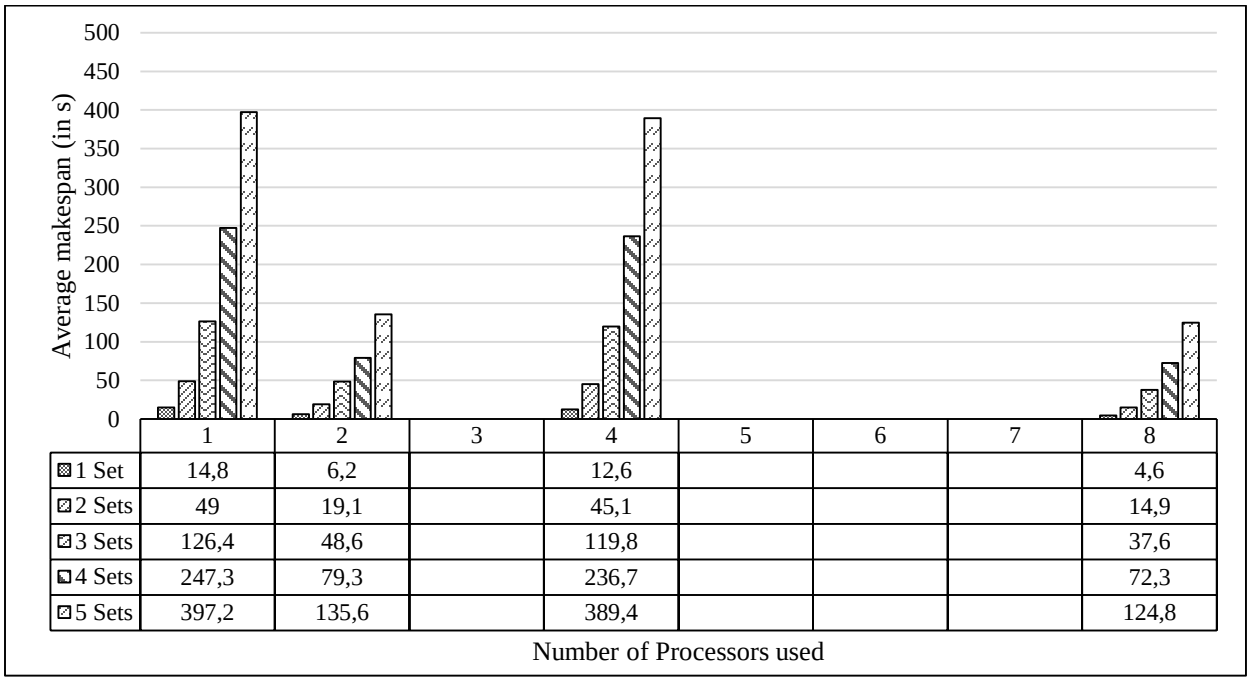
7.6 Further analysis of the results & additional observations

In Figure 26 and Figure 27 the results of the first and second sets of experiments are combined in respect to the number of processors used. For number of processors 1 and 2 the results from the Experiment Set I is used where no parallelization technique was used while for the number of processors 4 and 8 the Experiment Set II was used as the machines where able to execute calculations in parallel whenever possible. Although, there is no linear decrease of time when parallelization techniques as one could expect, the results still illustrate a small improvement.

Another extrapolation that can be made through the comparison of Figure 26 and Figure 27 is that HEFT is more affected by the introduction of parallelization capabilities than MaxMin. This can be explained because average values of the calculations were obtained to populate the cost of execution of each calculation on each machine and HEFT used that information of schedule calculations on the machines on a specific time. Because of that, there were times that a processor was not executing any calculation at all. MaxMin is not affected by that since it just receives the work that has to perform, without any predefined start time; in the MaxMin implementation, jobs of calculations are executed as soon as possible based on the ordered received.

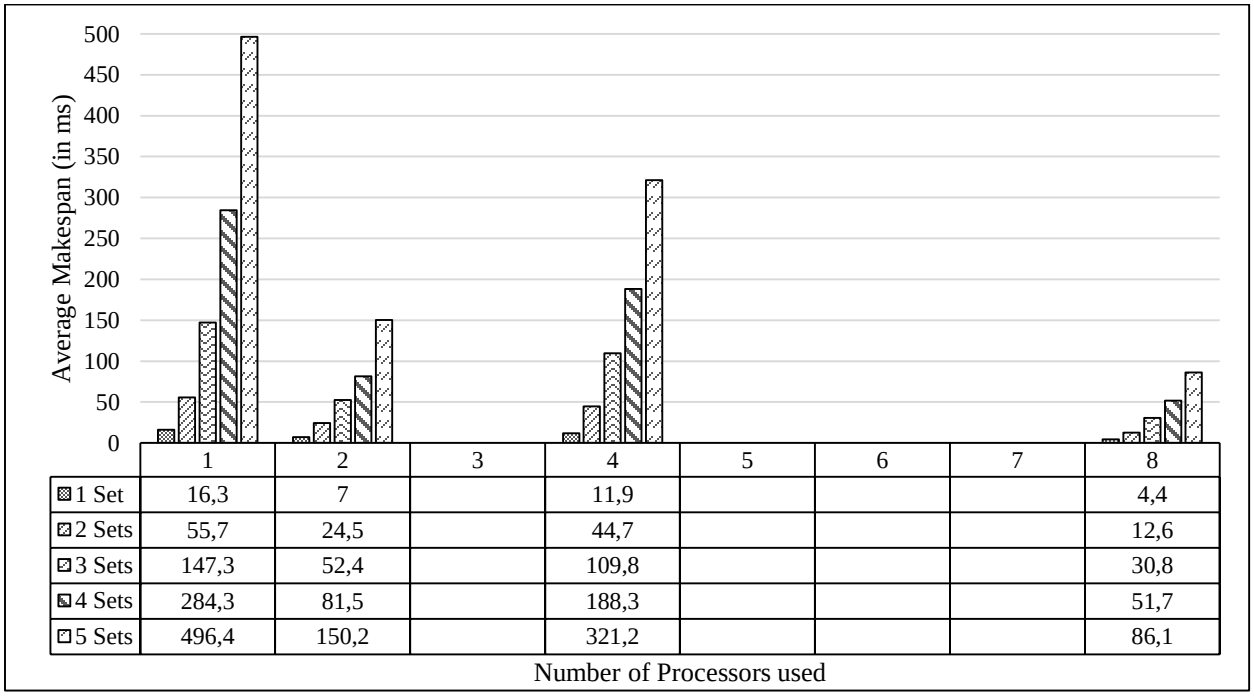
As a great resilience of the algorithms to show a somewhat linear correlation of processors used to makespan was witnessed, I/O bottlenecks that can occur within a machine should be investigated and mitigated. Of course, this has to do with the actual implementation code of the calculations as data retrieval, execution and write back of the output values are all performed within the “Calculate” function of every calculation.

As a final remark, in all cases of all experiments, as workload was added up, efficiency of the algorithms was greatly reduced.



	1	2	3	4	5	6	7	8
1 Set	14,8	6,2		12,6				4,6
2 Sets	49	19,1		45,1				14,9
3 Sets	126,4	48,6		119,8				37,6
4 Sets	247,3	79,3		236,7				72,3
5 Sets	397,2	135,6		389,4				124,8

Figure 26 - Projection of the makespan results of MaxMin on number of processors
(Data for Number of processors 1 and 2 come from the Experiment Set I while 4 and 8 from the Experiment Set II.)



	1	2	3	4	5	6	7	8
1 Set	16,3	7		11,9				4,4
2 Sets	55,7	24,5		44,7				12,6
3 Sets	147,3	52,4		109,8				30,8
4 Sets	284,3	81,5		188,3				51,7
5 Sets	496,4	150,2		321,2				86,1

Figure 27 - Projection of the makespan results of HEFT on number of processors
(Data for Number of processors 1 and 2 come from the Experiment Set I while 4 and 8 from the Experiment Set II.)

7.7 Conclusions from the results

Those algorithms were chosen amongst others found in scientific literature because of how well they fit into the context of scheduling the tasks at hand. After conducting a performance experiment of MaxMin showed steadily that outperforms HEFT, confirming the hypothesis presented in Section 3.5.

Continuing, the prototype was improved with parallelization capabilities for the implemented algorithms. Servers used were comprised by a CPU with multiple processors, which gives the possibility of performing more work in the same time. After conducting a performance experiment, things are in favor of HEFT. Especially as the load increases HEFT presents significantly better results.

As a last optimization attempt of the prototype, during this project, initialization costs were avoided since this operation has to be performed only once for each calculation on each machine. After conducting a performance experiment, the results are not in favor of one algorithm at all times. Nevertheless, MaxMin demonstrates better efficiency than HEFT as load grows.

In addition to those results, it was observed that the capability of execution of calculations is not any close to linear as more processors are added to the system and machines were unable to achieve high utilization of CPU. Efficiency was significantly more related to the physical machines than the utilization of more processors, which indicates that I/O operations performed by the code of the calculations, is a phenomenon to be investigated and if it also stands when the real calculations are being performed it should be mitigated.

By parallelizing execution of calculations on machines that performed the work and avoiding initialization costs by separating them from the execution sequence and performing them only once per calculation, per machine, it was managed to extensively reduce the time required for all calculations to be performed. In this particular case HEFT's makespan trend line was witnessed to be slightly steeper than MaxMin's and hence even if HEFT starts with better results, things quickly turn in favor of MaxMin as workload increases.

Furthermore, the results in all experiments conducted, support the claim that an experimental approach would be more valuable, as makespan follows a polynomial growth instead of a linear as load increases.

Finally, the results show that independent scheduling algorithms can be employed in the situation at hand. Therefore, the benefits that come along them (simplicity of implementation, adaptation, reason about its results, test and so forth) can be brought here. Especially, in respect to the independent scheduling algorithm implemented for this project, MaxMin it was observed that it is able to provide schedules more quickly as it works on groups of calculations in contrast to HEFT that performs schedules of calculations.

7.8 Threats to validity

The results presented in this project are subject to the limitations of the experiments conducted and hence cannot give sufficient predictions about the makespan of the algorithms as more machines are added into the system. Experiments with more Slave Servers should be performed.

Also, more calculations have to be added in the structure in order to complete the set of calculations that are currently performed. At this study only a subset of the first two levels of calculations were considered that their specifics could incorporate a potential selection bias. Additionally, as more sets of calculations are being created and incorporated in the monitoring performance values of the algorithms presented here may be altered especially if the 100% histograms of the execution costs of calculations and jobs are

changed. The population threat to validity should be thought of before generalizations to the population can be made.

Furthermore, our assumption of the deterministic nature of the calculations is subject to criticism as the results show. Nevertheless, the usage of the algorithms of this project is not disproved as the execution time of calculations used did not show extensive variability. Yet, this has an impact on the validity of the outputs produced by HEFT, as there is no mechanism of knowing if execution of all predecessor calculations has already been completed at the time a calculation starts. Still, it should be mentioned at this point, that calculations at the present time are completely separated one from each other and executed that way, so that kind of mismatches are happening all the time and for the time being are acceptable. Background operations of the operating system can also affect the availability of the processors and thus the time a calculation needs to be executed or starts its execution. This disproof of our assumption could be well based on the following.

In addition, the results presented in this project are relied upon simulation for calculations of Level 1. It may be the case that the operations done by those calculations affect the execution in a different manner. Moreover, the PCOMS calculations provided are writing their results into the csv file (model file) used for initialization purposes. The I/O bottleneck observed may be tightly connected to this fact mainly because the amount of writes to be performed are many within a small amount of time and HDD drives were used.

What is more, the trendlines of makespan growth, speedup and efficiency of the algorithms presented in the experiments may be biased because of the law of small numbers (workload applied) should only be used for rough estimations.

Moreover, always some of the data produces were affected by random fluctuations, such as machine increased workload due to the influence of other applications. In this study, those fluctuations were not taken into account since running application were minimized. This represents a threat to validity if the results are generalized to Servers that have a substantial overhead of additional operations that they have to perform.

Finally, the SQL Server utilized was also used for “production purposes” and consequently was used simultaneously for other purposes, different of this experiment. Hence, the results presented here may have been influenced by that usage. Delays caused be congestion in the database may have been occurred during the experiments.

8 Closing Chapter

8.1 Future work

Developing a new scheduling mechanism for computations in a grid is by itself a tough problem. Tallying on it the needs for reliability, expandability, scalability, a single point of management while delivering before specified deadlines increase the complexity of the system to be developed significantly. During this project, a small aspect of a system was created and two algorithms were implemented. In this section future work is proposed.

To begin with, experiments should be continued in order to test the performance as more machines are added. Furthermore, a controlled environment should be established where a full set of the production calculations can be tested and the performance of the algorithms can be monitored. All possible improvements of the algorithms should also be considered (i.e. the Lookahead variation of HEFT as presented by Bittencourt et al in [28]).

Additionally, because the scheduler will be used for monitoring purposes, one of the most important aspects that has to be considered is to deliver calculations based on a deadline. Takefusa et al. have provided a methodology, described in [29], that would be able to recognize with high prediction rate weather a task will meet the deadline or not when performed on a specific machine. For this prediction to take place, several techniques could be used that could enrich the MDS of the system. For instance, Wolski et al. in [30] assert that prediction can be done using a commodities markets strategy. Nevertheless, the ability to feasibly schedule preemptive tasks is always higher than the ability to feasibly schedule corresponding non-preemptive tasks. In [31] Doulamis and Varvarigos describe rules that augment the principle of deadlines with fairness all over the phases a task will go through in order to be executed.

As mentioned before, Ranganathan and Foster state in [9] that jobs could be performed faster when minimizing movement of data since it has to do with network performance. Because thousands of computations are to be executed every minute in the grid, each of them requiring a set of inputs, we should build the system in a way that the least amount of data is needed to be communicated for every calculation and have as much of them already locally available at the machine the job will be performed. This information, is showing how to improve a scheduling solution derivative of Condor and Isard et al. have introduced in [32] the “Quincy” algorithms in order to improve locality of data when scheduling in a grid environment, while still have a central point of management for all the system. Moreover, based on the same principal, as IO seems to be an issue to be addressed even within the “Calculate” methods of the calculations, separating it from them and using batching could provide significal improvements.

Another aspect that must be addressed is the quality of Service (QoS) that a task may request. For instance, it is expected that high importance calculations that have to do with raising alarms on oil platform sites will request higher QoS than others. As stated by Kounev et al. in [33] a QoS-aware grid resource manager can help on providing a higher Service Level Agreement (SLA).

Finally, fault-tolerance should be considered both in case of a computation failures and in case of wrong inputs in time of computation especially for algorithms such as HEFT. In those cases, a fault-tolerant scheduling mechanism like the one that has been described in [34] by Wrzesinska et al. whose basic concept is to maintain a global variable table could be employed. Also, a mechanism that allows the recalculation of values without affecting the QoS of the real-time procedure of the calculations should be developed. Thus, it will enable the rescheduling of calculations automatically if the system detects that calculations

have not been performed or have failed and also manually in case inputs have changed recursively (e.g. an improved model is given for a set of calculations).

8.2 Conclusion

Smart Connect is a performance monitoring system of the oil drilling equipment of Shell. For this monitoring to take place, a number of calculations are being performed on a regular basis. At the moment all these calculations are being hardcoded on servers that run local, independent schedulers in order for them to be performed. The development of an integrated system is of high importance.

During this project, a prototype for grid scheduling implementing two different algorithms was developed. The implantation of the prototype borrowed a lot of ideas from well-known and respected systems such as the one presented by Fitzgerald et al. in [27], the Condor-G by Frey et al. in [14], Utopia by Zhou et al in [2] and Globus in [26] by Foster and Karl.

The performance of this system, measured as its ability to perform the calculations at the lowest time possible was our primary concern. Consequently, choosing a good scheduling algorithm is of the essence. Because there are dependencies amongst the calculations and the servers used are not homogeneous, dependent list based algorithms were classified as the traditional solution. After an investigation of all the possible algorithms that could be used in this context, as such, HEFT was showed to be the most prominent.

Nevertheless, because of the small amount of time the calculations required to be performed independent scheduling algorithms were also put into perspective, because of the advantages they could bring along. Therefore, calculations were assembled and ordered into groups, called jobs. These jobs presented tasks that the independent algorithms could use. After an analysis of the jobs that were created, MaxMin was shown to be the best fit.

As a consequence a prototype, implementing two algorithms, MaxMin on groups of tasks (calculations) and HEFT on tasks granularity was developed. The algorithms were compared on three different scenarios with the first one being the control setup, where the algorithms were in their simplest form. In this occasion MaxMin showed better results. In the second scenario, the experiment set was augmented with the possibility of utilizing all the processors the servers have. Here results were in favor of HEFT. Additionally, by extrapolating the results of both scenarios, it was shown that using parallelization techniques yield better makespan results but at the same time efficiency dropped dramatically implying of a possible I/O bottleneck. At the last scenario, the initialization costs of the calculations were excluded (by being performed in advance). This is because its calculation needs to be initialized only once per server. Due to the small amount the calculations require in memory, this improvement is feasible. The performance results showed a great improvement in makespan and efficiency. Here HEFT only outperformed MaxMin for the workload of 1 set of calculations, in all other situations MaxMin had better results. Furthermore, for both algorithms as workload increased efficiency dropped greatly again. A possible explanation is that every 2nd level calculation used in the experiments, writes values to a file in the HDD.

This study showed that MaxMin may be used for the calculations of Smart Connect but still, experiments with more servers should be undertaken and a greater number of more realistic calculations should be used in the future. Nevertheless, regardless of the problems faced during this master project, it can be argued that this is the first step towards developing a Grid Scheduling system for the calculations of Smart Connect.

9 References

[1] C. V. Lopes, Exercises in Programming Style, CRC Press, 2014.

[2] S. Zhou, X. Zheng, J. Wang and P. Delisle, "UTOPIA: a Load Sharing Facility for Large, Heterogeneous Distributed Computer Systems," *Software—Practice & Experience*, vol. 23, no. 12, pp. 1305 - 1336, 1993.

[3] J. Basney, M. Linvy and T. Tannenbaum, "High Throughput Computing with Condor," *High Performance Computer Unit (HPCU) News,* vol. 1, no. 2, 1997.

[4] J. Schopf, "A general architecture for scheduling on the grid," *Journal of Parallel and Distributed Computing*, 2002.

[5] H. Topcuoglu, S. Hariri and M.-Y. Wu, "Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing," in *IEEE Transactions on Parallel and Distributed Systems*, 2002.

[6] G. Sih and E. Lee, "Declustering: a new multiprocessor scheduling technique," *IEEE Transactions on Parallel and Distributed Systems* , vol. 4, no. 6, pp. 625 - 637, 1993.

[7] M. A. Iverson, F. Özgüner and G. J. Follen, "Parallelizing Existing Applications in a Distributed Heterogeneous Environment," *4th Heterogeneous Computing Workshop*, 1995.

[8] H. El-Rewini and T. G. Lewis, "Scheduling parallel program tasks onto arbitrary target machines," *Journal of Parallel and Distributed Computing*, vol. 9, no. 2, pp. 138 - 153, 1990.

[9] K. Ranganathan and I. Foster , "Decoupling Computation and Data Scheduling in Distributed Data-Intensive Applications," in *Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing*, 2002.

[10] I. Foster and C. Kesselman, "The grid in a nutshell," in *Grid Resource Management*, vol. 64, Kluwer Academic Publishers, 2004, pp. 3-13.

[11] T. Braun, H. Siegel, N. Beck, L. Boloni, M. Maheswaran, A. Reuther, J. Robertson, M. Theys and B. Yao, "A taxonomy for describing matching and scheduling heuristics for mixed-machine heterogeneous computing systems," in *Seventeenth IEEE Symposium on Reliable Distributed Systems*, West Lafayette, IN, 1998.

[12] I. Foster and C. Kesselman, The grid: blueprint for a new computing infrastructure, San Francisco: Morgan Kaufmann Publishers Inc., 1999.

[13] M. Isard , M. Budiu , . Y. Yu , . A. Birrell and . D. Fetterly , "Dryad: distributed data-parallel programs from sequential building blocks," in *Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems*, New York, NY, 2007.

[14] J. Frey, T. Tannenbaum, M. Livny, I. Foster and S. Tuecke, "Condor-G: a computation management agent for multi-institutional grids," in *10th IEEE International Symposium on High Performance Distributed Computing*, 2001.

[15] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, S. Patil, M.-H. Su, K. Vahi and M. Livny, "Pegasus: Mapping Scientific Workflows onto the Grid," in *Grid Computing: Second European AcrossGrids Conference (AxGrids 2004)*, Nicosia, Cyprus, 2004.

[16] M. R. Garey and D. S. Johnson, "Computers and Intractability: A Guide to the Theory of NP-Completness," W.H. Freeman and Company, New York, NY, 1979.

[17] E. G. Coffman, "Computer and Job-Shop Scheduling Theory," John Wiley and Sons, New York, NY, 1976.

[18] L. Wang, H. J. Siegel, V. R. Roychowdhury and A. A. Maciejewski, "Task matching and scheduling in heterogeneous computing environments using a genetic-algorithm-based approach," *Journal of Parallel and Distributed Computing - Special issue on parallel evolutionary computing*, vol. 47, no. 1, pp. 8-22, 1997.

[19] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen and R. F. Freund, "Dynamic Mapping of a Class of Independent Tasks onto Heterogeneous Computing Systems," *Journal of Parallele and Distriuted Computing*, vol. 59, pp. 107-131, 1999.

[20] N. Fujimoto and K. Hagihara, "Tasks, A Comparison among Grid Scheduling Algorithms for Independent Coarse-Grained," in *Proceedings of the 2004 Symposium on Applications and the Internet-Workshops* , 2004.

[21] J. Yu, R. Buyya and K. Ramamohanarao, "Workflow Scheduling Algorithms for Grid Computing," in *Metaheuristics for Scheduling in Distributed Computing Environments Studies in Computational Intelligence*, Springer, 2008, pp. 173-214.

[22] J. Yu and R. Buyya , "A taxonomy of scientific workflow systems for grid computing," *ACM SIGMOD Record*, vol. 34, no. 3, pp. 44 - 49, 2005.

[23] F. Dong and S. . G. Akl, "Scheduling Algorithms for Grid Computing: State of the Art and Open Problems," 2006.

[24] J. Liou and M. A. Palis, "A Comparison of General Approaches to Multiprocessor Scheduling," in *Proceedings 11th International Parallel Processing Symposium*, Middletown, NJ, 1997.

[25] O. H. Ibarra and C. E. Kim, "Heruristic algorithms for scheduling independet tasks on nonidentical processosrs," *Journal of the Association for Computing Machinery*, vol. 24, no. 2, pp. 280-298, 1977.

[26] I. Foster and K. Carl, "Globus: A Metacomputing Infrastructure Toolkit," *International Journal of High Performance Computing Applications*, vol. 11, no. 2, 1997.

[27] S. Fitzgerald, I. Foster, C. Kasselman, G. v. Laszewski, W. Smith and S. Tuecke, "A Directory Service for Configuring High-Performance Distributed Computations," in *Proceedings of the 6th IEEE International Symposium on High Performance Distributed Computing*, 1997.

[28] L. Bittencourt, R. Sakellariou and E. Madeira, "DAG Scheduling Using a Lookahead Variant of the Heterogeneous Earliest Finish Time Algorithm," in *18th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*, 2010.

[29] A. Takefusa, H. Casanova, S. Matsuoka and F. Berman, "A study of deadline scheduling for client-server systems on the Computational Grid," in *Proceedings of the 10th IEEE International Symposium on High Performance Distributed Computing*, 2001.

[30] R. Wolski, J. Plank, J. Brevik and T. Bryan, "Analyzing Market-based Resource Allocation Strategies for the Computational Grid," *International Journal of High Performance Computing Applications*, vol. 15, no. 3, pp. 258-281, 2001.

[31] N. Doulamis and E. Varvarigos, "Fair Scheduling Algorithms in Grids," in *IEEE Transactions on parallel and distributed systems*, 2007.

[32] M. Isard, V. Prabhakaran, J. Currey, U. Wieder, K. Talwar and A. Goldberg, "Quincy: fair scheduling for distributed computing clusters," in *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, 2009.

[33] S. Kounev , R. Nou and J. Torres, "Autonomic QoS-Aware resource management in grid computing using online performance models," in *Proceedings of the 2nd international conference on Performance evaluation methodologies and tools*, 2007.

[34] G. Wrzesinska, R. V. van Nieuwpoort, J. Maassen, T. Kielmann and H. E. Bal, "Fault-tolerant Scheduling of Fine-grained Tasks in Grid Environments," *International Journal of High Performance Computing Applications*, vol. 20, no. 1, pp. 103-114, 2006.

APPENDIX I. THE DATABASE

Here the entities of the database that was used by the prototype is presented in Figure 28. The value types can easily be infered by a quick look at the code which follows in the following appendix.

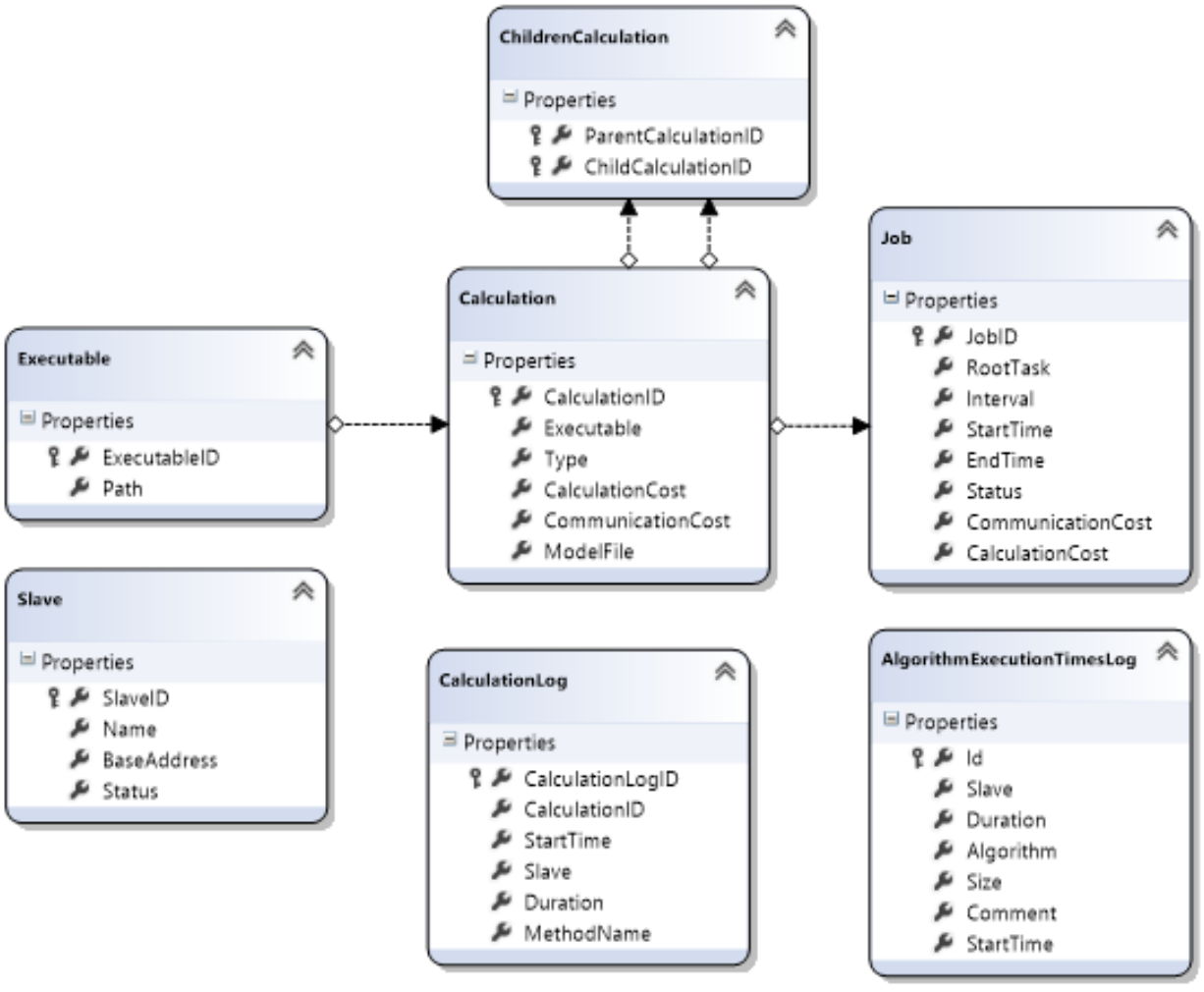


Figure 28 - Overview of the entities of the databased used (Arrows indicate Foreign Key assosiations)

In this section the code that was used for this project is provided. The two programs used are the Scheduler and the Slave.

Entry point of the slaves

```
using System;

namespace Scheduler
{
    class Program
    {
        static void Main(string[] args)
        {
            DataAccess.DatabaseOperator updater = new DataAccess.DatabaseOperator();

            //start the local slave node
            Slaves.SlaveNode slave = null;
            try
            {
                slave = new Slaves.SlaveNode();
            }
            catch (System.ServiceModel.AddressAccessDeniedException)
            {
                Console.WriteLine("Access denied. You need elevated rights to start the WCF
Service of the Slave Node.");
                Console.WriteLine("Press any key to exit.");
                Console.ReadKey();
            }
            if (slave != null)
            {
                slave.stopService();
            }
        }
    }
}
```

Entry point of the Scheduler

```
using Scheduler.Master;
using System;

namespace Scheduler
{
    class Program
    {
        static DataAccess.DatabaseOperator databaseUpdater = new
DataAccess.DatabaseOperator();

        static void Main(string[] args)
        {
            MasterNode master = new MasterNode();
            master.CheckSlaveConnectivity();
            while (true)
            {
                SelectCommand(master);
            }
        }

        private static void SelectCommand(MasterNode master)
        {
            Console.Write("Type your command and press enter: ");
            string command = Console.ReadLine();
            switch (command)
            {
                case "clear jobs": databaseUpdater.DeleteJobs();
                    break;
                case "populateDB": databaseUpdater.GenerateCalculationsAndJobs();
                    break;
                case "updateDB": databaseUpdater.UpdateCosts();
                    break;
                case "addwork": master.AddPendingJobs();
                    break;
                case "execute": master.StartExecution();
                    break;
                case "exit": master.CloseConnections(); Environment.Exit(0);
                    break;
                case "stop": master.Stop();
                    break;
                default: Console.WriteLine("Unknown command. Try again.");
                    break;
            }
            Console.WriteLine();
        }
    }
}
```

Configuration file

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <configSections>
    </configSections>
    <connectionStrings>
        <add name="Scheduler.Properties.Settings.GridSchedulerConnectionString"
            connectionString="Data Source=[redacted].europe.shell.com;Initial
Catalog=GridScheduler;Integrated Security=True"
            providerName="System.Data.SqlClient" />
    </connectionStrings>
    <startup>
        <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5" />
    </startup>
    <system.serviceModel>
      <client>
        <endpoint
          binding="basicHttpBinding" bindingConfiguration="BasicHttpBinding_ICommunicationService"
          contract="SlaveService.ICommunicationService"
name="BasicHttpBinding_ICommunicationService" />
      </client>
      <bindings>
        <basicHttpBinding>
          <binding name="myXmlHttpBinding" maxBufferSize="10485760"
maxReceivedMessageSize="10485760">
            <readerQuotas maxDepth="2147483647" maxStringContentLength="2147483647"
              maxArrayLength="2147483647" maxBytesPerRead="2147483647"
maxNameTableCharCount="2147483647" />
            <security mode="None" />
          </binding>
          <binding name="BasicHttpBinding_ICommunicationService" />
        </basicHttpBinding>
      </bindings>
        <behaviors>
            <serviceBehaviors>
                <behavior name="">
                    <serviceMetadata httpGetEnabled="true" httpsGetEnabled="true" />
                    <serviceDebug includeExceptionDetailInFaults="false" />
                </behavior>
            </serviceBehaviors>
        </behaviors>
        <services>
            <service name="Scheduler.Slaves.CommunicationService">
                <endpoint address="" binding="basicHttpBinding"
bindingConfiguration="myXmlHttpBinding" contract="Scheduler.Slaves.ICommunicationService" >
                    <identity>
                        <dns value="localhost" />
                    </identity>
                </endpoint>
                <endpoint address="mex" binding="mexHttpBinding" contract="IMetadataExchange" />
                <host>
                    <baseAddresses>
                        <add baseAddress="http://localhost:8733/Slave/Communication/" />
                    </baseAddresses>
                </host>
            </service>
        </services>
    </system.serviceModel>
</configuration>
```

Scheduler.DataAccess namespace

```
using System.Collections.Generic;
using System.Linq;

namespace Scheduler.DataAccess
{
    partial class Calculation
    {
        public double rankU { get; private set; }

        public void CalculateRankU()
        {
            rankU = CalculationCost;
            List<Calculation> childrenCalculations = GetChildrenCalculations();
            double maxChildRankU = 0;
            maxChildRankU = MaxRankUOfChildren(childrenCalculations, maxChildRankU);
            rankU += maxChildRankU;
        }

        private List<Calculation> GetChildrenCalculations()
        {
            List<Calculation> childrenCalculations = new List<Calculation>();
            using (DataAccessDataContext dataContext = new DataAccessDataContext())
            {
                childrenCalculations = (from calculations in dataContext.Calculations
                                        join childCalcs in dataContext.ChildrenCalculations
                                        on calculations.CalculationID equals
childCalcs.ParentCalculationID
                                        where CalculationID == calculations.CalculationID
                                        select calculations).ToList();
            }
            return childrenCalculations;
        }

        private static double MaxRankUOfChildren(ICollection<Calculation>
childrenCalculations, double maxChildRankU)
        {
            foreach (Calculation c in childrenCalculations)
            {
                c.CalculateRankU();
                if (maxChildRankU < c.rankU)
                {
                    maxChildRankU = c.rankU;
                }
            }
            return maxChildRankU;
        }
    }
}
```

```
using System;
using System.Collections.Generic;
using System.Linq;

namespace Scheduler.DataAccess
{
    public class DatabaseOperator
    {
        #region fields

            DataAccessDataContext dataContext = new DataAccessDataContext();
            string modelsPath = @"..\Shell SmartConnect IO Files\";
            string pcomsRelativePath = @"L2\PCOMS\";

        #endregion fields

        public DatabaseOperator()
        {
            CheckDatabaseConnection();
        }

        #region public methods

        public void DeleteJobs()
        {
            dataContext.ExecuteCommand("DELETE FROM Jobs");
            dataContext.ExecuteCommand("DELETE FROM ChildrenCalculations");
            dataContext.ExecuteCommand("DELETE FROM Calculations");
        }

        public void GenerateCalculationsAndJobs()
        {
            AddPCOMSCalculations();
            GenerateJobs();
        }

        public void UpdateCosts()
        {
            UpdateCalculationsCosts();
            UpdateJobsCosts();
        }

        public void UpdateCalculationsCosts()
        {
            foreach (Calculation c in dataContext.Calculations)
            {
                c.CalculationCost = CalculationCost(c);
            }
            dataContext.SubmitChanges();
        }

        #endregion public methods

        #region private methods

        private void AddJob(Calculation l1Calculation)
        {
            dataContext.Jobs.InsertOnSubmit(new Job()
            {
                CalculationCost = 0,
```

```
                CommunicationCost = 0,
                Interval = 0,
                Status = false,
                RootTask = l1Calculation.CalculationID
            });
            dataContext.SubmitChanges();
        }

        private Calculation AddL1Calculation()
        {
            var l1Calculation = new Calculation()
            {
               //TODO add a mechanism that finds the executable number and its respective type
                CalculationCost = 0,
                Executable = 2,
                ModelFile = "",
                Type = "TestCCEInterface.TestInterface"
            };
            dataContext.Calculations.InsertOnSubmit(l1Calculation);
            dataContext.SubmitChanges();
            return l1Calculation;
        }

        private void AssosiateCalculations(IQueryable<Calculation> calculations, Calculation
l1Calculation)
        {
            foreach (Calculation c in calculations)
            {
                dataContext.ChildrenCalculations.InsertOnSubmit(
                new ChildrenCalculation()
                {
                    ParentCalculationID = l1Calculation.CalculationID,
                    ChildCalculationID = c.CalculationID
                });
            }
        }

        private void UpdateJobsCosts()
        {
            foreach (Job j in dataContext.Jobs)
            {
                j.CalculationCost = CalculationCostJob(j);
            }
            dataContext.SubmitChanges();
        }

        private double CalculationCostJob(Job j)
        {
            double cost = 0;
            Calculation rootC = dataContext.Calculations.Where(x => x.CalculationID ==
j.RootTask).SingleOrDefault();
            cost = CalculationCostJobHelper(rootC);
            return cost;
        }

        private double CalculationCostJobHelper(Calculation c)
        {
            double cost = c.CalculationCost;
            var children = from calculations in dataContext.Calculations
                           join childCalculations in dataContext.ChildrenCalculations
```

```
                            on calculations.CalculationID equals
childCalculations.ChildCalculationID
                            where childCalculations.ParentCalculationID == c.CalculationID
                            select calculations;
            if (children.Count() > 0)
            {
                foreach (Calculation child in children)
                {
                    cost += CalculationCostJobHelper(child);
                }
            }
            return cost;
        }

        private void CheckDatabaseConnection()
        {
            DataAccess.DataAccessDataContext context = new DataAccess.DataAccessDataContext();
            if (!context.DatabaseExists())
            {
                Console.WriteLine("No database connection...");
            }
        }

        private void GenerateJobs()
        {
            int numberOfJobs = 0;
            bool jobAdded;
            Console.Write("Generating jobs...");
            IList<string> sites = new List<string>();
            AddSites(sites);

            foreach (string site in sites)
            {
                CreateJob(site, out jobAdded);
                if (jobAdded)
                {
                    numberOfJobs++;
                }
            }
            Console.WriteLine("done! Generated: " + numberOfJobs + " jobs.");
        }

        private void CreateJob(string siteName, out bool jobAdded)
        {
            jobAdded = false;
            var l2calculations = from l2calcs in dataContext.Calculations
                                 where l2calcs.ModelFile.Contains(siteName)
                                 select l2calcs;
            if (l2calculations.Count() > 0)
            {
                var l1Calculation = AddL1Calculation();
                AssosiateCalculations(l2calculations, l1Calculation);
                AddJob(l1Calculation);
                jobAdded = true;
            }
        }



        private void AddSites(IList<string> sites)
```

```
        {
            /*code that adds the names of all sites to the sites list*/
        }

        private double CalculationCost(Calculation c)
        {
            double calcCost = 0;
            calcCost += AverageCalculationCostOfMethod(c, "Initialize");
            double AverageCalculationCost = AverageCalculationCostOfMethod(c, "Calculate");
            calcCost += AverageCalculationCost == 0 ? 1 : AverageCalculationCost;
            calcCost += AverageCalculationCostOfMethod(c, "Termination");
            return Math.Round(calcCost, 0);
        }

        private double AverageCalculationCostOfMethod(Calculation c, string methodName)
        {
            double cost = 0;
            var durations = from logs in dataContext.CalculationLogs
                            where logs.CalculationID == c.CalculationID && logs.MethodName ==
methodName
                            select logs.Duration;
            if (durations.Count() > 0)
            {
                cost = durations.Average();
            }
            return cost;
        }

        private void AddPCOMSCalculations()
        {
            int numberOfPCOMS = 0;
            Console.Write("Adding PCOMS calculations...");
            FileAccess fileAccess = new FileAccess();
            List<string> modelfiles = fileAccess.GetAllFileNamesFromDirectory(modelsPath +
pcomsRelativePath).ToList();
            foreach (string m in modelfiles)
            {
                numberOfPCOMS = AddPCOMSCalculation(numberOfPCOMS, m);
            }
            dataContext.SubmitChanges();
            Console.WriteLine("done! Added: " + numberOfPCOMS + " calculations.");
        }

        private int AddPCOMSCalculation(int numberOfPCOMS, string modelFile)
        {
            Calculation caclculation = dataContext.Calculations.Where(x => x.ModelFile ==
pcomsRelativePath + modelFile).SingleOrDefault();
            if (caclculation == default(Calculation))
            {
                dataContext.Calculations.InsertOnSubmit(new Calculation()
                {
                    Executable = 1,
                    Type = "PIACE.PcomsPiInterface.PcomsPantelisScheduler",
                    CalculationCost = 0,
                    CommunicationCost = 0,
                    ModelFile = pcomsRelativePath + modelFile
                });
                numberOfPCOMS++;
            }
            else
```

```
            {
                caclculation.CalculationCost = CalculationCost(caclculation);
            }
            return numberOfPCOMS;
        }

        #endregion private methods
    }
}
```

```
using System;
using System.Collections.Generic;
using System.IO;

namespace Scheduler.DataAccess
{
    public class FileAccess
    {
        public ICollection<string> GetAllFileNamesFromDirectory(string targetDirectory)
        {
            ICollection<string> fileNames = new List<string>();
            string[] fileEntries = Directory.GetFiles(targetDirectory);
            foreach (string fileName in fileEntries)
            {
                fileNames.Add(GetRelativePath(fileName, targetDirectory));
            }
            return fileNames;
        }


        public string GetRelativePath(string filespec, string folder)
        {
            Uri pathUri = new Uri(filespec);
            // Folders must end in a slash
            if (!folder.EndsWith(Path.DirectorySeparatorChar.ToString()))
            {
                folder += Path.DirectorySeparatorChar;
            }
            Uri folderUri = new Uri(folder);
            return
Uri.UnescapeDataString(folderUri.MakeRelativeUri(pathUri).ToString().Replace('/',
Path.DirectorySeparatorChar));
        }
    }
}
```

Scheduler.MDS namespace

```
using System;
using System.Runtime.Serialization;

namespace Scheduler.MDS
{
    [DataContract]
    public partial class LoadIndex
    {
        [DataMember]
        public int numberOfProcessors { get; private set; }

        public LoadIndex()
        {
            numberOfProcessors = Environment.ProcessorCount;
        }
    }
}



namespace Scheduler.MDS
{
    public class LoadMonitoring
    {
        LoadIndex loadIndex;

        public LoadMonitoring()
        {
            loadIndex = new LoadIndex();
        }

        public LoadIndex GetLoadIndex()
        {
            return loadIndex;
        }
    }
}
```

Sheduler.Master namespace

```
using Scheduler.Master.GridSchedulingAlgorithms;
using Scheduler.Slaves.Jobs;
using System;
using System.Collections.Generic;
using System.Linq;

namespace Scheduler.Master
{
    public class MasterNode
    {
        #region fields

            GridScheduler scheduler;
            IList<SlaveNode> slaves = new List<SlaveNode>();
            ICollection<Job> jobs = new List<Job>();
            DataAccess.DataAccessDataContext dataContext = new
DataAccess.DataAccessDataContext();

        #endregion fields

        public MasterNode()
        {
            slaves = SetSlavesNodes();
        }

        #region public methods

            public void StartExecution()
            {
                if (slaves.Count == 0)
                {
                    Console.WriteLine("No available slaves. Terminating...");
                }
                else
                {
                    SelectSchedulingAlgorithm();
                    scheduler.ExecuteJobs(slaves, jobs);
                }
            }

            public void CheckSlaveConnectivity()
            {
                Console.WriteLine("\nChecking connectivity status of all slaves...");
                for (int i = slaves.Count() - 1; i >= 0; i--)
                {
                    try
                    {
                        slaves[i].ping();
                    }
                    catch
                    {
                        Console.WriteLine("Could not connect to slave: " +
slaves[i].ToString());
                        slaves.Remove(slaves[i]);
                    }
                }
            }
```

```
            public void AddPendingJobs() { jobs = GetAllJobs(); }

            public void Stop()
            {
                foreach (SlaveNode s in slaves)
                {
                    s.StopExecution();
                }
            }

            public void CloseConnections()
            {
                foreach (SlaveNode s in slaves)
                {
                    s.CloseConnection();
                }
            }

        #endregion public methods

        #region private methods

            private void SelectSchedulingAlgorithm()
            {
                int selection = 0;
                while (selection == 0)
                {
                    Console.WriteLine(
                    @"Select a scheduling mechanism :
                        1. MaxMin
                        2. HEFT");
                    int.TryParse(Console.ReadLine(), out selection);
                }
                scheduler = new GridSchedulerFactory().MakeGridScheduler(selection);
                Console.WriteLine(scheduler.GetName() + " has been selected.");
            }

            private IList<SlaveNode> SetSlavesNodes()
            {
                IList<SlaveNode> slaveNodes = new List<SlaveNode>();
                var slaveMachines = from slaveNodeTable in dataContext.Slaves select
slaveNodeTable;
                foreach (DataAccess.Slave s in slaveMachines)
                {
                    slaveNodes.Add(new SlaveNode(s.SlaveID, s.Name, s.BaseAddress));
                }
                return slaveNodes;
            }

            private ICollection<Job> GetAllJobs()
            {
                Console.Write("\nRetrieving jobs...");
                DataAccess.DataAccessDataContext dataContext = new
DataAccess.DataAccessDataContext();
                ICollection<Job> jobs = new List<Job>();
                var pendingJobs = dataContext.Jobs.Where(x => x.Status == false);
                foreach(var job in pendingJobs)
                {
                    jobs.Add(new Job(job.JobID, job.RootTask));
                }
```

```
                Console.WriteLine("done! Retrieved " + jobs.Count() + " jobs.");
                return jobs;
            }

        #endregion private methods
    }
}
```

```
using Scheduler.DataAccess;
using Scheduler.SlaveService;
using System;
using System.Collections.Generic;
using System.Linq;
using System.ServiceModel;

namespace Scheduler.Master
{
    public class SlaveNode
    {
        public MDS.LoadIndex loadIndex;

        #region fields

            private int slaveId;
            private string slaveName;
            private string baseAddrress;
            private ICollection<Slaves.Jobs.Job> pendingJobs = new List<Slaves.Jobs.Job>();

            private CommunicationServiceClient client;
            private DataAccessDataContext dataContext = new DataAccessDataContext();

            private double[] freeTimeAt;         //used from MaxMin
            private IList<Processor> processors; //used from HEFT

        #endregion fields

        public SlaveNode(int id, string computerName, string baseAddr)
        {
            slaveId = id;
            slaveName = computerName;
            baseAddrress = baseAddr;
            client = new SlaveService.CommunicationServiceClient();
            client.ChannelFactory.Endpoint.Address = new EndpointAddress("http://" +
baseAddrress + ":8733/Slave/Communication");
            client.executeJobsCompleted += cl_executeJobsCompleted;
            client.executeCalculationsCompleted += cl_executeCalculationsCompleted;
        }

        #region public methods

        public void CloseConnection()
        {
            try
            {
                client.Close();
            }
            catch
            {
                if (client != null)
                {
                    client.Abort();
                }
            }
        }

        public override string ToString()
        {
            return slaveName;
```

```
}

public bool ping()
{
    return client.ping();
}

public void RetrieveLoadInformation()
{
    loadIndex = client.getLoadInformation();
    if (freeTimeAt == null && processors == null)
    {
        freeTimeAt = new double[loadIndex.numberOfProcessors];
        processors = new List<Processor>();
        for (int i = 0; i < loadIndex.numberOfProcessors; i++)
        {
            processors.Add(new Processor());
        }
    }
}

#region public methods (Used from MaxMin)

    public void Assign(Slaves.Jobs.Job assignedJob)
    {
        pendingJobs.Add(assignedJob);
        UpdateJobEST(assignedJob);
    }

    public void Assign(List<Slaves.Jobs.Job> list)
    {
        pendingJobs = list;
    }

    public void ExecutePendingJobs()
    {
        if (pendingJobs.Count() > 0)
        {
            Console.WriteLine("Sending calculations to..." + slaveName);
            client.executeJobsAsync(pendingJobs.ToList());
        }
        pendingJobs.Clear();
    }

    public void StopExecution()
    {
        client.stopExecution(pendingJobs.ToList());
    }

    public double CalculateEFT(Slaves.Jobs.Job pendingJob)
    {
        return MaxCalculationTime(pendingJob) + JobEST();
    }

#endregion public methods (Used from MaxMin)

#region public methods (Used from HEFT)

    public void Assign(Calculation calculation)
    {
```

```
                ProcessorWithMinEFT(calculation).Assign(calculation);
            }

            public void ExecutePendingCalculations()
            {
                ICollection<Slaves.Jobs.Calculation> calculations = new
List<Slaves.Jobs.Calculation>();
                foreach (Processor p in processors)
                {
                    foreach (Calculation c in p.GetCalculations())
                    {
                        calculations.Add(new Slaves.Jobs.Calculation(c.GetCalculationID(),
c.GetAST()));
                    }
                    p.RemoveAllCalculations();
                }
                if (calculations.Count() > 0)
                {
                    calculations = calculations.OrderBy(x => x.AST).ToList();
                    client.executeCalculationsAsync(calculations.ToList());
                    calculations.Clear();
                }
            }

            public double ComputeEFT(Calculation calculation)
            {
                double eft = double.MaxValue;
                for (int i = 0; i < processors.Count; i++)
                {
                    double eftOnProcessor = processors[i].ComputeEFT(calculation);
                    if (eft > eftOnProcessor)
                    {
                        eft = eftOnProcessor;
                    }
                }
                return eft;
            }

            public void UpdateCalculationCost(Calculation c)
            {
                double averageInitializationCost = AverageCost(c, "Initialize");
                double averageCalculationCost = AverageCost(c, "Compute");
                double averageTerminationCost = AverageCost(c, "Termination");
                c.calculationCost = averageInitializationCost + averageCalculationCost +
averageTerminationCost;
            }

        #endregion public methods (Used from HEFT)

        #endregion public methods

        #region private methods (Used from MaxMin)

            private void UpdateJobEST(Slaves.Jobs.Job assignedJob)
            {
                int index = GetIndexOfMinFreeTimeAt();
                freeTimeAt[index] += MaxCalculationTime(assignedJob);
            }

            private int GetIndexOfMinFreeTimeAt()
```

```
            {
                int index = 0;
                double est = double.MaxValue;
                for (int i = 0; i < freeTimeAt.Count(); i++)
                {
                    if (est > freeTimeAt[i])
                    {
                        est = freeTimeAt[i];
                        index = i;
                    }
                }
                return index;
            }

            private double MaxCalculationTime(Slaves.Jobs.Job pendingJob)
            {
                return (from jobs in dataContext.Jobs
                        where pendingJob.GetRootCalculation() == jobs.RootTask
                        select jobs.CalculationCost).Max();
            }

            private double JobEST()
            {
                return freeTimeAt[GetIndexOfMinFreeTimeAt()];
            }

            private void cl_executeJobsCompleted(object sender,
System.ComponentModel.AsyncCompletedEventArgs e)
            {
                if (e.Cancelled)
                {
                    Console.WriteLine("Jobs request has been cancelled from " + slaveName);
                }
                else if (e.Error != null)
                {
                    Console.WriteLine("Error " + e.Error.Message + " has been returned from
the request to perform jobs to " + slaveName);
                }
                else
                {
                    Console.WriteLine("Jobs successfully submitted to " + slaveName);
                }
            }

        #endregion private methods (Used from MaxMin)

        #region private methods (Used from HEFT)

            private Processor ProcessorWithMinEFT(Calculation calculation)
            {
                int processorIndex = 0;
                double eft = double.MaxValue;
                for (int i = 0; i < processors.Count; i++)
                {
                    double eftOnProcessor = processors[i].ComputeEFT(calculation);
                    if (eft > eftOnProcessor)
                    {
                        eft = eftOnProcessor;
                        processorIndex = i;
                    }
```

```
                }
                return processors[processorIndex];
            }

            private double AverageCost(Calculation c, string method)
            {
                var cost = (from logs in dataContext.CalculationLogs
                            where logs.CalculationID == c.GetCalculationID() &&
                            logs.Slave == slaveId &&
                            logs.MethodName == method
                            select logs.Duration);
                return cost.Count() > 0 ? cost.Average() : 0;
            }

            private void cl_executeCalculationsCompleted(object sender,
System.ComponentModel.AsyncCompletedEventArgs e)
            {
                if (e.Cancelled)
                {
                    Console.WriteLine("Calculations request has been cancelled from " +
slaveName);
                }
                else if (e.Error != null)
                {
                    Console.WriteLine("Error " + e.Error.Message + " has been returned from
the request to perform calculations to " + slaveName);
                }
                else
                {
                    Console.WriteLine("Calculations successfully submitted to " + slaveName);
                }
            }
        #endregion private methods (Used from HEFT)
    }
}
using Scheduler.DataAccess;
using System.Collections.Generic;
using System.Linq;

namespace Scheduler.Master
{
    public class Calculation
    {
        #region fields

            private int calculationID;
            private double readyTime;
            private double AST;
            private IList<Calculation> childrenCalculations;

        #endregion fields

        #region properties

            public double AFT { get { return AST + calculationCost; } }
            public double calculationCost;
            public double rankU { get; private set; }

        #endregion properties
```

```
public Calculation(int id, double calcCost)
{
    calculationID = id;
    calculationCost = calcCost;
    PopulateChildrenCalculations();
}

#region public methods

    public double GetAST()
    {
        return AST;
    }

    public void updateAST(double value)
    {
        AST = value;
    }

    public ICollection<Calculation> GetChildrenCalculations()
    {
        return childrenCalculations;
    }

    public double CalculateRankU()
    {
        rankU = calculationCost + MaxRankUOfChildren(childrenCalculations);
        return rankU;
    }

    public double GetCalculationTime()
    {
        return this.calculationCost;
    }

    public int GetCalculationID()
    {
        return calculationID;
    }

    public void UpdateReadyTime(double eft)
    {
        readyTime = eft;
        AST = eft;
    }

    public double GetReadyTime()
    {
        return readyTime;
    }

#endregion public methods

#region private methods

    private void PopulateChildrenCalculations()
    {
        childrenCalculations = new List<Calculation>();
        DataAccessDataContext dataContext = new DataAccessDataContext();
```

```
                var calcs = (from calculations in dataContext.Calculations
                             join childCalcs in dataContext.ChildrenCalculations
                             on calculations.CalculationID equals
childCalcs.ChildCalculationID
                             where calculationID == childCalcs.ParentCalculationID
                             select new
                             {
                                 childID = childCalcs.ChildCalculationID,
                                 childCalcCost = calculations.CalculationCost
                             }).ToList();
                foreach (var c in calcs)
                {
                    childrenCalculations.Add(new Calculation(c.childID, c.childCalcCost));
                }
            }

            private double MaxRankUOfChildren(ICollection<Calculation> childrenCalculations)
            {
                double maxChildRankU = 0;
                foreach (Calculation c in childrenCalculations)
                {
                    double chilidRacnkU = c.CalculateRankU();
                    if (maxChildRankU < chilidRacnkU)
                    {
                        maxChildRankU = chilidRacnkU;
                    }
                }
                return maxChildRankU;
            }

        #endregion private methods
    }
}
```

```
using System.Collections.Generic;
using System.Linq;

namespace Scheduler.Master
{
    public class Processor
    {
        IList<Calculation> calculations;

        public Processor()
        {
            calculations = new List<Calculation>();
        }

        public double GetAST(Calculation calculation)
        {
            double est = 0;
            var sortedCalculations = calculations.OrderBy(x => x.GetAST());
            for (int i = 0; i < calculations.Count; i++)
            {
                if (est <= calculations[i].GetAST() - calculation.GetCalculationTime())
                {
                    est = calculations[i].GetAST() - calculation.GetCalculationTime();
                }
            }
            return est;
        }

        public ICollection<Calculation> GetCalculations()
        {
            return calculations.OrderBy(x => x.GetAST()).ToList();
        }

        public double ComputeEFT(Calculation c)
        {
            double eft = c.GetReadyTime() + c.GetCalculationTime();
            if (calculations.Count == 1)
            {
                if (calculations[0].GetAST() < eft)
                {
                    eft = calculations[0].AFT + c.GetCalculationTime();
                }
            }
            else
            {
                for (int i = 1; i < calculations.Count; i++)
                {
                    if (calculations[i].GetAST() < eft)
                    {
                        eft = calculations[i].AFT + c.GetCalculationTime();
                    }
                    else
                    {
                        if (ThereIsAGap(c, i))
                        {
                            //found it!
                            break;
                        }
                        else
                        {
```

```
                            eft = calculations[i].AFT + c.GetCalculationTime();
                        }
                    }
                }
            }
            return eft;
        }

        public void Assign(Calculation c)
        {
            calculations.Add(c);
        }

        public void RemoveAllCalculations()
        {
            calculations.Clear();
        }

        private bool ThereIsAGap(Calculation c, int i)
        {
            return calculations[i].GetAST() - calculations[i - 1].AFT >
c.GetCalculationTime();
        }
    }
}
```

Scheduler.Master.GridSchedulingAlgorithms namespace

```
using Scheduler.Slaves.Jobs;
using System;
using System.Collections.Generic;
using System.Diagnostics;

namespace Scheduler.Master.GridSchedulingAlgorithms
{
    public abstract class GridScheduler
    {
        protected ICollection<Job> jobs;
        protected ICollection<SlaveNode> slaves;

        public void ExecuteJobs(ICollection<SlaveNode> slaveList, ICollection<Job> jobList)
        {
            slaves = slaveList;
            jobs = jobList;
            foreach (SlaveNode slave in slaves)
            {
                slave.RetrieveLoadInformation();
            }
            Stopwatch stopwatch = Stopwatch.StartNew();
            PrioritizeJobs();
            AssignJobs();
            stopwatch.Stop();
            Console.WriteLine(GetName() + " has completed scheduling in " +
stopwatch.ElapsedMilliseconds + "ms.");
            while (true)
            {
                foreach (SlaveNode slave in slaves)
                {
                    slave.ExecutePendingJobs();
                    slave.ExecutePendingCalculations();
                }
                Console.ReadLine();
            }
        }

        public string GetName()
        {
            int index = this.ToString().LastIndexOf('.') + 1;
            return this.ToString().Substring(index);
        }

        public abstract void PrioritizeJobs();

        public abstract void AssignJobs();

    }
}
```

```
namespace Scheduler.Master.GridSchedulingAlgorithms
{
    class GridSchedulerFactory
    {
        public GridScheduler MakeGridScheduler(int selection)
        {
            GridScheduler scheduler;
            if (selection == 1)
            {
                scheduler = new MaxMin();
            }
            else if (selection == 2)
            {
                scheduler = new HEFT();
            }
            else
            {
                scheduler = new EqualAllocation();
            }
            return scheduler;
        }
    }
}
```

```
using Scheduler.Slaves.Jobs;
using System.Linq;

namespace Scheduler.Master.GridSchedulingAlgorithms
{
    public class MaxMin : GridScheduler
    {
        DataAccess.DataAccessDataContext dataContext = new DataAccess.DataAccessDataContext();

        public override void PrioritizeJobs()
        {
            // put the most expensive jobs first
            jobs = jobs.OrderByDescending(x => x.ExecutionCost).ToList();
        }

        public override void AssignJobs()
        {
            // assign to the machine with the minimum EFT
            foreach (Job pendingJob in jobs)
            {
                AssignToTheMachineWithTheMinimumEFT(pendingJob);
            }
        }

        private void AssignToTheMachineWithTheMinimumEFT(Job pendingJob)
        {
            SlaveNode selectedSlave = slaves.First();
            double minEFT = double.MaxValue;
            foreach(SlaveNode slave in slaves)
            {
                double currentSlaveEFT = slave.CalculateEFT(pendingJob);
                if (minEFT > currentSlaveEFT)
                {
                    minEFT = currentSlaveEFT;
                    selectedSlave = slave;
                }
            }
            selectedSlave.Assign(pendingJob);
        }
    }
}
```

```
using Scheduler.DataAccess;
using System.Collections.Generic;
using System.Linq;

namespace Scheduler.Master.GridSchedulingAlgorithms
{
    class HEFT : GridScheduler
    {
        IList<Calculation> calculations = new List<Calculation>();
        DataAccessDataContext dataContext = new DataAccessDataContext();

        #region overrided methods

        public override void PrioritizeJobs()
        {
            jobs = jobs.Concat(jobs)
                .Concat(jobs).Concat(jobs).Concat(jobs)
                .ToList();
            PopulateListOfRootCalculations();
            AddChildrenCalculations();
            CalculateRankUForAllCalculations();
            calculations = calculations.OrderByDescending(x => x.rankU).ToList();
        }

        public override void AssignJobs()
        {
            foreach(Calculation calculation in calculations)
            {
                AssignToTheSlaveWithThemMinimumEFT(calculation);
            }
        }

        #endregion overrided methods

        #region private methods

        private void PopulateListOfRootCalculations()
        {
            foreach (Slaves.Jobs.Job job in jobs)
            {
                var c = dataContext.Calculations.Where(x => x.CalculationID ==
job.GetRootCalculation()).First();
                calculations.Add(new Calculation(c.CalculationID, c.CalculationCost));
            }
        }

        private void AddChildrenCalculations()
        {
            for (int i = 0; i < calculations.Count; i++)
            {
                calculations =
calculations.Concat(calculations[i].GetChildrenCalculations()).ToList();
            }
        }

        private void CalculateRankUForAllCalculations()
        {
            foreach (Calculation c in calculations)
            {
                c.CalculateRankU();
```

```
            }
        }

        private void AssignToTheSlaveWithThemMinimumEFT(Calculation calculation)
        {
            double eft;
            SlaveNode slaveWithMinEFT;

            FindSlaveWithMinEFT(calculation, out eft, out slaveWithMinEFT);
            slaveWithMinEFT.Assign(calculation);

            UpdateReadyTimeOfChildren(calculation, eft);
        }

        private void UpdateReadyTimeOfChildren(Calculation calculation, double eft)
        {
            var children = (from calcs in calculations
                            join childs in calculation.GetChildrenCalculations().ToList()
                            on calcs.GetCalculationID() equals childs.GetCalculationID()
                            select calcs);
            foreach (Calculation c in children)
            {
                c.UpdateReadyTime(eft);
            }
        }

        private void FindSlaveWithMinEFT(Calculation calculation, out double eft, out
SlaveNode slaveWithMinEFT)
        {
            eft = double.MaxValue;
            slaveWithMinEFT = null;
            foreach (SlaveNode slave in slaves)
            {
                slave.UpdateCalculationCost(calculation);
                double slaveEFTforCalculation = slave.ComputeEFT(calculation);
                if (eft > slaveEFTforCalculation)
                {
                    eft = slaveEFTforCalculation;
                    calculation.updateAST(slaveEFTforCalculation -
calculation.calculationCost);
                    slaveWithMinEFT = slave;
                }
            }
        }

        #endregion private methods

    }
}
```

Scheduler.Slaves namespace

```
using Scheduler.DataAccess;
using Scheduler.MDS;
using System;
using System.Linq;
using System.ServiceModel;

namespace Scheduler.Slaves
{
    public class SlaveNode
    {
        #region fields

            static int slaveID;
            string slaveName = Environment.MachineName;
            ServiceHost host;
            JobExecutor jobExecutor;
            CalculationExecutor calculationExecutor;
            LoadMonitoring loadMonitoring = new LoadMonitoring();
            DataAccessDataContext dataContext = new DataAccessDataContext();

        #endregion fields

        public SlaveNode()
        {
            SetSlaveID();
            jobExecutor = new JobExecutor();
            calculationExecutor = new CalculationExecutor();

            //start WCF Service (needs admin permissions)
            startService();
        }

        public static int GetSlaveID()
        {
            return slaveID;
        }

        public void stopService()
        {
            Console.WriteLine("Press <enter> to stop the service.");
            Console.ReadLine();
            // Close the ServiceHost.
            host.Close();
        }

        public void startService()
        {
            host = new System.ServiceModel.ServiceHost(new CommunicationService(jobExecutor,
calculationExecutor, loadMonitoring));

            //Enable metadata publishing.
            //ServiceMetadataBehavior smb = new ServiceMetadataBehavior();
            //smb.HttpGetEnabled = true;
            //smb.MetadataExporter.PolicyVersion = PolicyVersion.Policy15;
            //host.Description.Behaviors.Add(smb);

            // Open the ServiceHost to start listening for messages. Since
            // no endpoints are explicitly configured, the runtime will create
```

```
            // one endpoint per base address for each service contract implemented
            // by the service.
            host.Open();

            Console.WriteLine("The service is ready at {0}", host.BaseAddresses.Single());
        }

        private void SetSlaveID()
        {
            try
            {
                var slave = dataContext.Slaves.Where(x => x.Name == slaveName).First();
                slaveID = slave.SlaveID;
            }
            catch (InvalidOperationException)
            {

                dataContext.Slaves.InsertOnSubmit(new DataAccess.Slave
                {
                    Name = slaveName,
                    BaseAddress = slaveName
                });
                dataContext.SubmitChanges();
                slaveID = dataContext.Slaves.Where(x => x.Name == slaveName).First().SlaveID;
                Console.WriteLine(
                    "The slave with name " + slaveName + " was not found, \n"
                    + "and added to the Database with id " + slaveID);
            }
        }
    }
}
```

```
using Scheduler.DataAccess;
using System;

namespace Scheduler.Slaves
{
    public class Logging
    {
        int slaveID;
        public Logging()
        {
            slaveID = SlaveNode.GetSlaveID();
        }

        public void LogCalculationInformation(int calculationID, string methodName,
DateTimeOffset startTime, double duration)
        {
            using (DataAccessDataContext dataContext = new DataAccessDataContext())
            {
                dataContext.CalculationLogs.InsertOnSubmit(new CalculationLog()
                {
                    CalculationID = calculationID,
                    Duration = duration,
                    StartTime = startTime,
                    MethodName = methodName,
                    Slave = slaveID
                });
                dataContext.SubmitChanges();
            }
        }

        public void LogExecutionTime(int duration, string algoritm, int size, DateTimeOffset
startTime, string comment)
        {
            using (DataAccessDataContext dataContext = new DataAccessDataContext())
            {
                dataContext.AlgorithmExecutionTimesLogs.InsertOnSubmit(new
AlgorithmExecutionTimesLog()
                {
                    Slave = slaveID,
                    Duration = duration,
                    Algorithm = algoritm,
                    Size = size,
                    StartTime = startTime,
                    Comment = comment
                });
                dataContext.SubmitChanges();
            }
        }
    }
}
```

```
using Scheduler.Slaves.Jobs;
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Threading.Tasks;

namespace Scheduler.Slaves
{
    public class JobExecutor
    {
        ICollection<Job> jobs;
        Logging logs = new Logging();
        List<Task> tasks;

        public JobExecutor()
        {
            jobs = new List<Job>();
        }

        public void StopExecutionOfAllJobs(ICollection<Job> stoppingJobs)
        {
            foreach (Job job in stoppingJobs)
            {
                if (jobs.Contains(job))
                {
                    job.StopExecution();
                }
            }
            Console.WriteLine("STOPPING EXECUTION OF REQUESTED JOBS");
        }

        public void StartExecution(ICollection<Job> pendingJobs, bool
separationOfInitializationFromExecution)
        {
            tasks = new List<Task>();
            jobs = jobs.Concat(pendingJobs).ToList();

            if (separationOfInitializationFromExecution)
            {
                Initialize(pendingJobs);
                ExecuteJobs(pendingJobs);
            }
            else
            {
                InitializeAndExecuteJobs(pendingJobs);
            }
        }

        public void InitializeAndExecuteJobs(ICollection<Job> pendingJobs)
        {
            Console.WriteLine("\nInitializing and executing " + pendingJobs.Count() + "
jobs...");
            DateTimeOffset startTime = DateTimeOffset.UtcNow;
            foreach (Job job in pendingJobs)
            {
                Task t = Task.Factory.StartNew(() =>
                {
                    job.Initialize();
                    job.Execute();
```

```
                });
                tasks.Add(t);
            }
            Task.WaitAll(tasks.ToArray());
            DateTimeOffset endTime = DateTimeOffset.UtcNow;
            int totalTime = (int)((endTime - startTime).TotalMilliseconds);
            logs.LogExecutionTime(totalTime, "MaxMin", pendingJobs.Count(), startTime, "Total
time");
            Console.WriteLine("Total time:  " + totalTime + "ms.");
        }

        private void Initialize(ICollection<Job> pendingJobs)
        {
            DateTimeOffset startTime = DateTimeOffset.UtcNow;
            Console.Write("\nInitializing execution of " + pendingJobs.Count() + " jobs...");
            Stopwatch stopwatch = Stopwatch.StartNew();
            foreach (Job job in pendingJobs)
            {
                Task t = Task.Factory.StartNew(() =>
                {
                    job.Initialize();
                });
                tasks.Add(t);
            }
            Task.WaitAll(tasks.ToArray());
            stopwatch.Stop();
            int elapseTime = (int)stopwatch.ElapsedMilliseconds;
            Console.WriteLine("Done! in " + stopwatch.ElapsedMilliseconds + "ms.");
            logs.LogExecutionTime(elapseTime, "MaxMin", pendingJobs.Count(), startTime,
"Initialize");
        }

        private void ExecuteJobs(ICollection<Job> pendingJobs)
        {
            DateTimeOffset startTime = DateTimeOffset.UtcNow;
            Console.Write("\nStarting execution of " + pendingJobs.Count() + " jobs...");
            Stopwatch stopwatch = Stopwatch.StartNew();
            foreach (Job job in pendingJobs)
            {
                Task t = Task.Factory.StartNew(() =>
                {
                    {
                        job.Execute();
                    }
                });
                tasks.Add(t);
            }
            Task.WaitAll(tasks.ToArray());
            stopwatch.Stop();
            int elapseTime = (int)stopwatch.ElapsedMilliseconds;
            Console.WriteLine("Done! in " + stopwatch.ElapsedMilliseconds + "ms.");
            logs.LogExecutionTime(elapseTime, "MaxMin", pendingJobs.Count(), startTime,
"Execute");
        }
    }
}
```

```
using Scheduler.MDS;
using Scheduler.Slaves.Jobs;
using System.Collections.Generic;
using System.ServiceModel;

namespace Scheduler.Slaves
{
    [ServiceContract]
    public interface ICommunicationService
    {
        [OperationContract]
        void stopExecution(ICollection<Job> jobs);

        [OperationContract]
        bool ping();

        [OperationContract]
        LoadIndex getLoadInformation();

        [OperationContract]
        void executeJobs(ICollection<Job> jobs);

        [OperationContract]
        void executeCalculations(ICollection<Calculation> collection);
    }
}
```

```
using Scheduler.MDS;
using Scheduler.Slaves.Jobs;
using System.Collections.Generic;
using System.ServiceModel;

namespace Scheduler.Slaves
{
    [ServiceBehavior(InstanceContextMode=InstanceContextMode.Single)]
    public class CommunicationService : ICommunicationService
    {
        JobExecutor jobExecutor;
        CalculationExecutor calculationExecutor;
        LoadMonitoring loadMonitoring;

        public CommunicationService() { }

        public CommunicationService(JobExecutor jExecutor, CalculationExecutor cExecutor,
LoadMonitoring monitoring)
        {
            jobExecutor = jExecutor;
            calculationExecutor = cExecutor;
            loadMonitoring = monitoring;
        }

        public void stopExecution(ICollection<Job> jobs)
        {
            jobExecutor.StopExecutionOfAllJobs(jobs);
        }

        public bool ping()
        {
            return true;
        }


        public void executeJobs(ICollection<Job> jobs)
        {
            // flag for first initializing and the executing calculation
            bool separationOfInitializationFromExecution = true;
            jobExecutor.StartExecution(jobs, separationOfInitializationFromExecution);
        }

        public LoadIndex getLoadInformation()
        {
            return loadMonitoring.GetLoadIndex();
        }

        public void executeCalculations(ICollection<Calculation> calculations)
        {
            calculationExecutor.StartExecution(calculations);
        }
    }
}
```

```
using Scheduler.Slaves.Jobs;
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Threading;

namespace Scheduler.Slaves
{
    public class CalculationExecutor
    {
        ICollection<Calculation> calculations;
        Logging logs = new Logging();
        List<Timer> timers = new List<Timer>();
        Object lockObj = new Object();

        public CalculationExecutor()
        {
            calculations = new List<Calculation>();
        }

        public void StopExecutionOfAllCalculations()
        {
            calculations = new List<Calculation>();
            timers = new List<Timer>();
            Debug.WriteLine("STOPPING EXECUTION");
        }
        public void StartExecution(ICollection<Calculation> calculations)
        {
            Console.WriteLine("Starting execution of " + calculations.Count() + "
calculations...");
            int calcsCompleted = 0;
            DateTimeOffset endTime = DateTimeOffset.MaxValue;
            DateTimeOffset startTime = DateTimeOffset.UtcNow;

            foreach (Calculation calculation in calculations)
            {
                Timer timer = new Timer(x =>
                {
                    {
                        calculation.Initialize();
                        calculation.Calculate();
                        Interlocked.Increment(ref calcsCompleted);
                        endTime = DateTimeOffset.UtcNow;
                    }
                }, null, (int)(calculation.AST), Timeout.Infinite);
                timers.Add(timer);
            }

            while (calcsCompleted != calculations.Count())
            {
                Thread.Sleep(10000);
            }
            int executionTime = ((int)(endTime - startTime).TotalMilliseconds);
            Console.WriteLine("Successful completion of " + (calculations.Count()) + "
calculations in " + executionTime + "ms.");
            logs.LogExecutionTime(executionTime, "HEFT", calculations.Count(), startTime, "");
        }
    }
}
```

Scheduler.Slaves.Jobs namespace

```
using Scheduler.DataAccess;
using Scheduler.Slaves.Jobs.Statuses;
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Runtime.Serialization;
using System.Timers;

namespace Scheduler.Slaves.Jobs
{
    [DataContract]
    public partial class Job
    {
        public double ExecutionCost;

        #region fields

            [DataMember] private int jobId;
            [DataMember] private int rootCalculation;
            private DataAccessDataContext dataContext;
            private double interval;
            private Status status;
            private DateTimeOffset startTime, endTime;
            private List<Calculation> calculations;
            private Object lockObject;
            private Timer timer;

        #endregion fields

        public Job(int id, int rootTask)
        {
            jobId = id;
            rootCalculation = rootTask;
        }

        #region public methods

            public void Initialize()
            {
                /* --- Commented out code in this function was used for logging purposes ---*/
                //Stopwatch initStopwatch = new Stopwatch();
                //initStopwatch.Start();
                {
                    jobInitialization();
                    initializationOfCalculations();
                }
                //initStopwatch.Stop();
                //var j = dataContext.Jobs.Where(x => x.JobID == jobId).SingleOrDefault();
                //j.CommunicationCost = initStopwatch.ElapsedMilliseconds;
                //dataContext.SubmitChanges();
            }

            public void Execute()
            {
                status = new Running();
                {
                    PerformCalculations();
```

```
                }
            }

            public void createCalculationsList()
            {
                List<int> pendingCalculationsIDs = new List<int>();

                calculations = new List<Calculation>();
                pendingCalculationsIDs.Add(rootCalculation);
                calculations.Add(new Calculation(pendingCalculationsIDs[0]));

                for (int i = 0; i < pendingCalculationsIDs.Count; i++)
                {
                    var childrenCalculations = from childCalcs in
dataContext.ChildrenCalculations
                                               where childCalcs.ParentCalculationID ==
pendingCalculationsIDs[0]
                                               select childCalcs.ChildCalculationID;
                    pendingCalculationsIDs.Concat(childrenCalculations.ToList());
                    foreach (var c in childrenCalculations)
                    {
                        calculations.Add(new Calculation(c));
                    }
                    pendingCalculationsIDs.RemoveAt(0);
                    i = 0;
                }
            }

            public void StopExecution()
            {
                status = new Stopped();
                Debug.WriteLine("Stoping " + jobId);
            }

            public bool IsRecuring()
            {
                return (interval > 0);
            }

            public int GetRootCalculation()
            {
                return rootCalculation;
            }

        #endregion public methods

        #region private methods

            private void jobInitialization()
            {
                lockObject = new object();
                dataContext = new DataAccessDataContext();
                Scheduler.DataAccess.Job job = dataContext.Jobs.First(j => j.JobID == jobId);
                interval = (double)job.Interval;
                createCalculationsList();
                ConfigureTimer();
            }

            private void ConfigureTimer()
            {
```

```
            if (IsRecuring())
            {
                timer = new Timer(interval);
                timer.AutoReset = true;
                timer.Elapsed += timer_Elapsed;
                timer.Enabled = true;
                timer.Start(); // calculate next interval
            }
        }

        private void initializationOfCalculations()
        {
            foreach (Calculation c in calculations)
            {
                c.Initialize();
            }
        }

        private void PerformCalculations()
        {
            startTime = DateTimeOffset.UtcNow;
            foreach (Calculation c in calculations)
            {
                ExeucteMethods(c);
            }
        }

        private void ExeucteMethods(Calculation calculation)
        {
            calculation.Calculate();
        }

        private void timer_Elapsed(object sender, ElapsedEventArgs e)
        {
            //prevent from using the calculation while the previous is still calculating
            lock (lockObject)
            {
                endTime = DateTimeOffset.UtcNow;
                if (new Running().Equals(status))
                {
                    TimeSpan elapsed = endTime.Subtract(startTime);
                    if (elapsed.TotalMilliseconds > interval * 1.1)
                    {
                        // miss
                        Finalizing();
                        Console.WriteLine("Missed by " + (int)elapsed.TotalMilliseconds +
"ms.");
                    }
                    else
                    {
                        PerformCalculations();
                    }
                }
                else
                {
                    Finalizing();
                }
            }
        }
```

```
            private void Finalizing()
            {
                timer.Stop();
                foreach(Calculation c in calculations)
                {
                    c.Termination();
                }
                var job = dataContext.Jobs.First(x => x.JobID == jobId);
                job.Status = false;
                dataContext.SubmitChanges();
            }

        #endregion private methods
    }
}
```

```
using Scheduler.DataAccess;
using Slave.Jobs.Executables;
using System;
using System.Diagnostics;
using System.Linq;
using System.Runtime.Serialization;

namespace Scheduler.Slaves.Jobs
{
    [DataContract]
    public partial class Calculation
    {
        [DataMember] public double AST { get; set;}

        #region fields

            [DataMember] private int id;
            private DLL dll;
            private string modelfile;
            private Stopwatch stopwatch;
            private DateTimeOffset startTime;
            private DataAccessDataContext dataContext;

        #endregion fields

        #region constructors

            public int getCalculationID()
            {
                return id;
            }

            public Calculation(int calculationID)
            {
                id = calculationID;
            }

            public Calculation(int calculationID, double ast)
            {
                id = calculationID;
                AST = ast;
            }

        #endregion constructors

        #region public methods

            public object Initialize()
            {
                startTime = DateTimeOffset.UtcNow;
                stopwatch = Stopwatch.StartNew();
                instanciateType();
                object result = executeMethod("Initialize", new object[] { (object)modelfile
});
                stopwatch.Stop();
                //logCalculationInformation("Initialize");
                return result;
            }

            public object Calculate()
```

```
            {
                startTime = DateTimeOffset.UtcNow;
                stopwatch = Stopwatch.StartNew();
                object result = executeMethod("Calculate", null);
                stopwatch.Stop();
                //logCalculationInformation("Calculate");
                return result;
            }

            public object Termination()
            {
                startTime = DateTimeOffset.UtcNow;
                stopwatch = Stopwatch.StartNew();
                object result = executeMethod("Termination", null);
                stopwatch.Stop();
                //logCalculationInformation("Termination");
                return result;
            }

        #endregion public methods

        #region private methods

            private void instanciateType()
            {
                dataContext = new DataAccessDataContext();
                Scheduler.DataAccess.Calculation c = dataContext.Calculations.First(cu =>
cu.CalculationID == id);
                dll = new DLL(c.Executable1.Path, c.Type);
                string modelFolder = @"..\Shell SmartConnect IO Files\"; //relative location
of the root model folder
                modelfile = modelFolder + c.ModelFile;
                dll.CreateInstance();
            }

            private void logCalculationInformation(string methodName)
            {
                Logging log = new Logging();
                log.LogCalculationInformation(id, methodName, startTime,
stopwatch.ElapsedMilliseconds);
            }

            private object executeMethod(string method, object[] args)
            {
                return dll.ExecuteMethod(method, args);
            }

        #endregion private methods
    }
}
```

```
using System;
using System.Collections.Generic;
using System.Reflection;

namespace Slave.Jobs.Executables
{
    public class DLL
    {
        #region fields

            private string _FilePath;
            private Assembly _Assembly;
            private Type _Type;
            private object _Obj;

        #endregion fields

        #region public methods

            public DLL(string filePath, string typeName)
            {
                _FilePath = filePath;
                LoadAssembly();
                SetType(typeName);
                CreateInstance();
            }

            public void CreateInstance()
            {
                _Obj = Activator.CreateInstance(_Type);
            }

            public object ExecuteMethod(string methodName, object[] args)
            {
                object result = null;
                MethodInfo method = getMethod(methodName);
                // Execute the method.
                try
                {
                    result = method.Invoke(_Obj, args);
                }
                // TODO Implement an exception handler
                catch (Exception e)
                {
                    System.Diagnostics.Debug.WriteLine(e.Message);
                }
                return result;
            }

        #endregion public methods

        #region private methods

            private void LoadAssembly()
            {
                // Use 'UnsafeLoadFrom' load if assembly has been loaded without full trust
                // (i.e. downloaded from the intranet)
                _Assembly = Assembly.UnsafeLoadFrom(_FilePath);
                //_Assembly = Assembly.Load(_FilePath);
            }
```

```
            public ICollection<Type> GetTypes()
            {
                return _Assembly.GetTypes();
            }

            private void SetType(string typeName)
            {
                _Type = _Assembly.GetType(typeName);
            }

            // Get the available methods for that type
            public ICollection<MethodInfo> GetMethods(Type type)
            {
                return type.GetMethods();
            }

            private MethodInfo getMethod(string methodName)
            {
                return _Type.GetMethod(methodName);
            }

        #endregion private methods
    }
}
```

Scheduler.Slaves.Jobs.Statuses namespace

```
namespace Scheduler.Slaves.Jobs.Statuses
{
    public abstract class Status
    {
        public abstract string toString();

        // override object.Equals
        public override bool Equals(object obj)
        {
            if (obj == null || GetType() != obj.GetType())
            {
                return false;
            }
            return true;
        }

        // override object.GetHashCode
        public override int GetHashCode()
        {
            return toString().GetHashCode();
        }
    }
}

namespace Scheduler.Slaves.Jobs.Statuses
{
    public class Running : Status
    {
        public override string toString()
        {
            return "Running";
        }
    }
}

namespace Scheduler.Slaves.Jobs.Statuses
{
    class Stopped : Status
    {
        public override string toString()
        {
            return "Stopped";
        }
    }
}
```