AN INVESTIGATION INTO PARALLEL JOB SCHEDULING USING SERVICE LEVEL AGREEMENTS

A thesis submitted to the University of Manchester
for the degree of Master of Philosophy
in the Faculty of Engineering and Physical Sciences

2014

By
Syed Zeeshan Ali
School of Computer Science

Contents

Word Count: 50,231

List of Tables

List of Figures

Abstract

A *scheduler*, as a central components of a computing site, aggregates computing resources and is responsible to distribute the incoming load (jobs) between the resources. Under such an environment, the optimum performance of the system against the *service level agreement (SLA)* based workloads, can be achieved by calculating the priority of SLA bound jobs using integrated heuristic. The SLA defines the service obligations and expectations to use the computational resources. The integrated heuristic is the combination of different SLA terms. It combines the SLA terms with a specific weight for each term. The weights are computed by applying parameter sweep technique in order to obtain the best schedule for the optimum performance of the system under the workload. The sweeping of parameters on the integrated heuristic observed to be computationally expensive. The integrated heuristic becomes more expensive if no value of the computed weights result in improvement in performance with the resulting schedule. Hence, instead of obtaining optimum performance it incurs computation cost in such situations. Therefore, there is a need of detection of situations where the integrated heuristic can be exploited beneficially. For that reason, in this thesis we propose a metric based on the concept of utilization, to evaluate the SLA based parallel workloads of independent jobs to detect any impact of integrated heuristic on the workload.

Declaration

No portion of the work referred to in this thesis has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.

Copyright

Acknowledgements

I have been supported by many people throughout my PhD but most notable and obvious person is my supervisor Dr. Rizos Sakellariou. This thesis is a result of his magnanimous support, continuous encouragement and insightful guidance without that the completion of this thesis would not have been possible. I am really grateful for his thoughtful advices, his support and encouragement (especially when I was struck by family and visa crisis) and for everything that I learnt from him.

I am deeply indebted to my parents, Mr. Syed Anwer Ali (Late) and Mrs. Rehana Begum, my wife Mrs. Sana Khalid and all of my extended family members for their invaluable love, understanding and support all the time especially when I lost my most precious gem of my life that is, my dad.

I like to thank to all members in School of Computer Science, especially Farhana, Kashif, Sadia, Seemab, Sohel, Viktor, Wei and Ye for their help during my PhD Study.

I also like to mention the support of my house mates Ali, Naveed, Saqib, Bilal and Waleed that lived with me during my stay in UK. Also, thankful to the mora support of my friends Adnan and Sohaib.

I would like to thank National University of Sciences and Technology, Pakistan for supporting me with the scholarship.

Above all, I am grateful to almighty ALLAH for associating me with such nice people and giving me the strength through them to complete this thesis.

Chapter 1

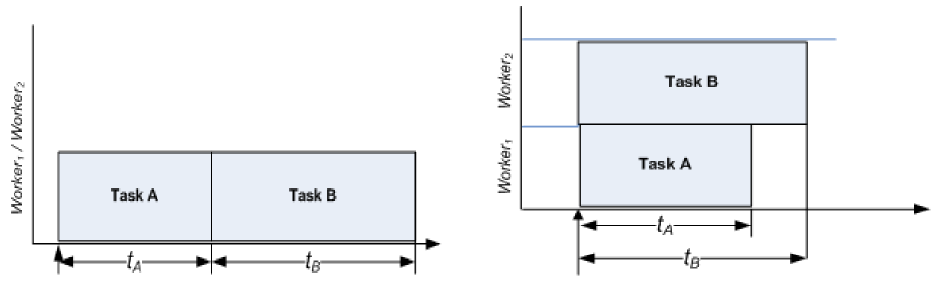
Introduction

This chapter presents a high-level overview of the whole thesis. Section 1.1 illustrates *planning and scheduling* in general terms. Section 1.2 first discusses the simple scheduling concept in the context of computing environments and then, more specifically, under constraints. Then, in Section 1.3 the details of the *Service Level Agreement (SLA)* based heuristic of the work in [YS06], which, to a large extent, motivates the work in this thesis, are briefly discussed. Further, in the context of a metacomputing environment, Section 1.4 states the overall problem and highlights the relevant issues associated with it. Section 1.5 discusses the performance metric for the evaluation of different scheduling heuristics. Then, Section 1.6 presents the aims & contribution of the research . Finally, the chapter is concluded with the structure of the remainder of this thesis in Section 1.7.

1.1 Planning and Scheduling

The Oxford Dictionary on-line [sch] defines the term "scheduling" as *to arrange or plan (an event) to take place at a particular time.* Furthermore, it defines "plan" as *an intention or decision about what one is going to do.* Therefore, we can derive that *scheduling is a decision making process which tells what one is going to do at a particular time.*

Hence, scheduling involves *decision making, assigning* and *the time to carry out the assignment.* Let us consider a simple example before moving to scheduling in the computing environment context. Let us assume that we have two tasks A and B to perform by the workers w_1 and w_2. Tasks A and B require resources r_A and r_B to be carried out, respectively. Now, the decision making process will involve which task will be performed by which worker. Once this is decided, then at the time of assignment of a task to a worker, the required resource should also be provided to carry out the task. Let us further assume that tasks A and B will need t_A and t_B time to finish, respectively. One of the possible scheduling decisions can be to assign both tasks with the respective resources to either worker w_1 or worker w_2. In this case, the two tasks will take a time $t_A + t_B$ to complete, as depicted in Figure 1.1(a). Alternatively, another possible decision can be, to assign each task with the required resources to an individual worker i.e., assign task A with resource r_A to w_1 and task B with resource r_B to w_2. This type of assignment allows the workers to start the tasks at the same time, as depicted in Figure 1.1(b). Therefore, both tasks can be completed in time $max(t_A, t_B)$, that is, the time it takes to complete the longest of the two tasks. If, for example, $t_B > t_A$ (as shown in Figure 1.1(b)) then $max(t_A, t_B)$ returns t_B.



(a) Either of the Worker Performing Tasks (b) Performing Tasks Simultaneously

Figure 1.1: Two Possible Ways of Tasks Completion

From the above example, and the two possible scheduling decisions discussed, we can make some interesting observations.

- The first point is that if the number of workers is less than the number of tasks then the time needed to complete the tasks will increase as some tasks are performed sequentially, i.e., one after the other. Conversely, if the number of tasks and the number of workers are equal then leaving one worker idle and burdening the other worker(s) with (more) tasks results in higher completion time; one can say that this is a bad scheduling decision.
- The second point to observe is, the tasks can be performed by the workers independently at the same time. Therefore, assigning the tasks equally and performing them *simultaneously* keeps all the workers busy. Hence, a scheduling decision should take into the account the assignment of tasks to workers to achieve some balance. An unbalanced assignment may result in a longer time for the completion of tasks; such a delay may suggest that a scheduling decision is not good.
- The third point to observe is the simultaneous execution of tasks, which suggests that the tasks can be performed in *parallel* of each other. The *independent* nature of tasks means none of the tasks is required to finish before starting the other.

A simple, real-world example of the above can be found when painting the side walls of a room. If there are four workers, one assigned to each side wall with the required resources available i.e., paint brush/roller and desired paint, then all of the workers can paint the walls independently and *in parallel* and finish painting in less time if compared to the painting of all walls by one worker.

1.2 Scheduling in Computing Environments

In the context of scheduling jobs onto computing resources the term *scheduling* alone refers to the *planning and scheduling* of a number of (computational) jobs (or tasks) onto some available resources. Similar to the ideas presented above, the key objective of scheduling is to balance the load distribution among the computing resources, keep all resources busy, and, thus, minimize the time to complete all the assigned jobs. In order to achieve this objective, assuming that the number of jobs is greater than the number of resources and only one job can be executed at a time by each resource, jobs have to be prioritized somehow; the resulting pattern or prioritization is considered as a plan for scheduling the jobs. In general, a *scheduler* is responsible for planning and allocation of resources to a number of jobs submitted to it. Usually, the scheduler is the central component of any system required to execute multiple jobs. As a result, it needs some knowledge of the state of the jobs as well as the state of the available resources in order to prioritize the jobs to meet the requirements as set by the system administrators.

Although scheduling is an issue faced by every computing environment, most of the interesting research (and the focus of the work in this dissertation) relates to parallel job scheduling for parallel computers. Parallel jobs consist of a set of independent jobs (that can be executed in parallel, hence they have no dependency with each other) and parallel computers provide multiple CPUs or, simply, processors onto which jobs can be executed. Parallel computers have been associated with *high performance computing*, which aims at providing increased performance for applications with high computational (and not only) requirements.

Traditionally, the scheduling mechanism in high performance computing has been essentially a simple one: First-In-First-Out. Jobs are submitted to a queue from where the jobs are executed when they reach the head of the queue [SY08]. As parallel jobs may require a different number of processors, some jobs may also be selected from the queue depending on how many processors they require, as a key objective, from the system's point of view, is to keep all processors busy. A queue-based simple scheduling approach may be acceptable for an environment where all the computing resources are governed by a central scheduler. But scheduling parallel jobs in an environment where the resources may belong to and govern by different schedulers, as it is the case in a Grid [BFH03] where resources (and their schedulers) may belong to (and managed by) different administrative domains, needs some collaboration across the domains. With that collaboration, the schedulers are coordinated with each other in order to schedule parallel independent jobs efficiently. The coordination mechanism is called *coordinated resource sharing* [Fos01]. In the absence of coordination mechanism, the benefits of parallelism may not be fully exploited as noted in [SY08]. To address this issue, *advance reservation* has been suggested to run the jobs at the same time [SFT00]. This allows the users to specify an exact time when a job will start running on the resources, by making an advance reservation.

However, as noted in [MSK$^+$04], advance reservation is again an 'extreme' level of service, which places several constraints on resources. For example, any running jobs must be pre-empted at the time when the advance reservation starts, something that may be too costly. In addition, it is almost impossible to guarantee that any jobs will terminate and release the resources when the advance reservation starts. As a result, a new approach that has been proposed and investigated by several researchers in the past decade (e.g., [MSK$^+$04]) is to offer users the possibility to specify some constraints on acceptable start time and finish time of their jobs and leave the scheduler to schedule the jobs in any way the scheduler likes, as long as it meets the user constraints.

The specification of information such as the acceptable earliest start time and latest finish time of a job can be a part of a contract, which is termed as *Service Level Agreement (SLA)* [ACD+04]. Essentially, an SLA is a contract between two parties, the user and the resource provider, or, more generally, a service consumer and a service provider, respectively. It includes the terms that describe an agreed expected level of service by the user; this is the level of service within which the service will be provided by the service provider [SY08]. In other words, an SLA defines the service obligations and expectations to use the resources. As an incentive to meet these obligations, an SLA may include some monetary parameters, that stipulate a payment to the resource provider upon successful completion of a job, or a penalty that has to be paid to the user if the provider fails to meet the user's expectations. SLAs have been used in a different context in distributed and e-business environments [HQK+06] [PBM08] [BMG+09]. Recent work [WGB11] [Kub11] [AJY10] [AD09] [YS06], shows the need and growing interest towards SLA based job management in different forms of metacomputing. However, research on using SLAs in high performance computing, when executing parallel jobs, is still ongoing [YS06] [AJY10].

In the context of high performance computing, job submission using SLAs requires at least four different pieces of information. As the domain relates to parallel jobs, it is not possible to agree on an SLA without knowing (in addition to earliest start time and latest finish time), the time required to execute the job and the number of resources (or simply processors (CPUs)) needed. Clearly, the difference between latest finish time and earliest start time should be greater than or equal to the time required to execute the job. In fact, the equality makes the whole process equivalent to advance reservation. It is the flexibility provided by setting the difference finish time minus start time significantly longer than the required execution time that can be particularly useful from the resource provider's point of view.

Therefore, in the context of using SLAs for parallel job scheduling, every parallel job is bound by an SLA, agreed between the user (owner of the job) and the resource provider (owner of the resources). This SLA can be represented as a quadruple, which defines the required level of service, such as $\{T_s, t_D, T_f, N_{cpu}\}$ [YS06], where:

- T_s: the earliest acceptable start time for the job, or, simply, start time
- t_D: the expected runtime or execution time of the job
- T_f: the latest acceptable finish time for the job (or simply the deadline)
- N_{cpu}: the number of CPUs required by the job.

Clearly, for every SLA, it must be that $T_f - T_s \geq t_D$. It is also assumed, without loss of generality, that t_D refers to the expected runtime using the number of processors required, i.e., N_{cpu}.

Scheduling parallel jobs specified by an SLA can be considered a problem related to scheduling tasks with deadlines or time critical constraints, as the option of latest acceptable finish time is one of the components of the SLA. The problem of scheduling parallel jobs with time critical service levels is considered to be an NP-hard problem [Joh06]. It becomes more difficult to make optimal scheduling decisions when jobs have conflicting requirements such as the number of resources (i.e., processors (CPUs)) needed, start time and deadline time, and everything under a resource-constrained environment. In such a resource-constrained and deadline-based environment, a common solution is based on prioritizing jobs and then scheduling jobs one after the other in the order of priority. The whole problem becomes that of the choice of a prioritization scheme that gives good results. This is the approach that has been used by the relevant literature on *real-time scheduling* for several decades.

Two commonly used and cited approaches for prioritizing jobs in deadline-constrained environments are the *earliest deadline first (EDF)* and the *least laxity first (LLF)*. As the name suggests, with EDF the job with approaching deadline gets highest priority, i.e., EDF prioritizes the jobs according to their deadline times starting with the jobs with the soonest deadline. In another approach, where it is assumed that both execution time and deadline of the jobs are known in advance, the jobs can be prioritized based on the *laxity*. The laxity is defined as the difference between the deadline time and the latest possible completion time. Then, jobs are prioritized according to laxity; job with the least laxity gets the highest priority.

It can easily be realized that many approaches for job prioritization can be chosen. For parallel jobs, in particular, there is another parameter that can be used to prioritize the jobs, i.e., size of a job (or simply *jobsize*) which is computed by multiplying the number of resources (CPUs) required by a job with it's runtime/execution time. Jobs can be prioritized with either least value or highest value of jobsize. Similarly, the counter part of EDF and LLF can be drawn that is latest/highest deadline first (HDF) and highest laxity first (HLF). In the context of SLA-based resource constrained environments, different methods have been proposed to prioritize the jobs, such as [YS06] [SAL+02] [SAL+04]. Most commonly used algorithms are based on heuristics; such algorithms may not have good worst case behaviour but results are often quite good and easy to implement [GP99]. There are different heuristics evaluated by Yarmolenko and Sakellariou in [YS06] to schedule SLAs and shown an improvement in performance of the system using a proposed *integrated heuristic*, discusses in Section 1.3.

Table 1.1: Example SLAs or Jobs

SLA	Start Time (T_s)	Runtime (t_D)	No. of CPUs (N_{cpu})	Deadline Time (T_f)	Jobsize ($A = t_D * N_{cpu}$)	Laxity ($t_L = T_f - T_s - t_D$)
A	1	1	8	6	8	4
B	1	2	5	5	10	2
C	1	8	2	10	16	1
D	1	2	3	6	6	3

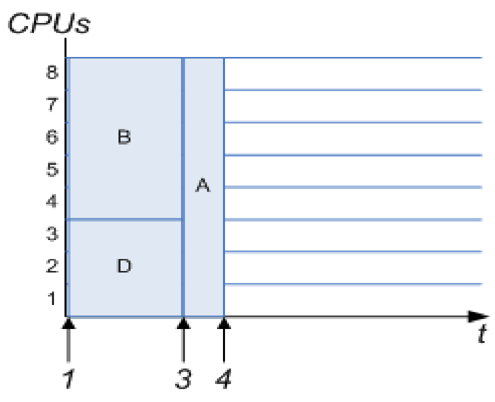
Before moving on to the integrated heuristic, an understanding is needed to be established for the scheduling of SLAs using the above mentioned simple heuristics i.e., earliest deadline first (EDF), latest/highest deadline first (HDF), least laxity first (LLF), highest laxity first (HLF), least jobsize first (LAF) and highest jobsize first (HAF). Let us consider the four different SLAs given in Table 1.1 to understand the effect of priority based on the above mentioned heuristics. In addition to the four SLA parameters, Table 1.1 includes the jobsize (computed as the product of runtime and number of CPUs) and laxity (computed as the difference between deadline and sum of runtime & start time) against each of the given SLA. It is assumed that a maximum of 8 resources are available onto which the SLAs given in the example table, are needed to be scheduled. The prioritization of the SLAs based on the heuristics is presented in Table 1.2 and the possible mapping of SLAs onto resources is graphically shown in Figure 1.2.

Table 1.2 is divided into six sub-tables. In each sub-table the SLAs are ordered according to the priority from highest to lowest. For example, Table 1.2(a) represents the prioritization of SLAs from highest to lowest based on the least jobsize first heuristics; if the SLAs are scheduled onto the system with maximum capacity of 8 CPUs then SLA C will not be able to schedule in order to finish by the mentioned deadline time. The mapping of SLAs onto resources under least jobsize heuristics is presented in Figure 1.2(a). It can be seen that after scheduling the highest priority SLA i.e., D, not enough resources would be available for the next high priority SLA i.e., A, therefore, SLA B is picked up to schedule. The runtime of SLA B is same as SLA D therefore, SLA B is scheduled at the same time as SLA D instead of leaving the resources idle. Then, SLA A is scheduled as soon as all the resources become available i.e., at the end of the execution of SLA D and B which is at t = 3 units. After scheduling SLA A, it is not possible to schedule SLA C as it will run over its deadline time. Likewise, the prioritization based on the *highest jobsize first (HAF)* is given in Table 1.2(b) and the mapping of SLAs based on this prioritization is shown in Figure 1.2(b). The prioritization of SLAs is changed now and SLA A is unable to get scheduled to finish by the deadline time. Similarly, the prioritization of SLAs based on the heuristics *least laxity first (LLF), highest laxity first (HLF), earliest deadline first (EDF)* and *latest/highest deadline first (HDF)* is shown in Table 1.2(c), 1.2(d), 1.2(e) & 1.2(f) and the mapping of the schedule based on these heuristics is given in Figure 1.2(c), 1.2(d), 1.2(e) & 1.2(f), respectively.

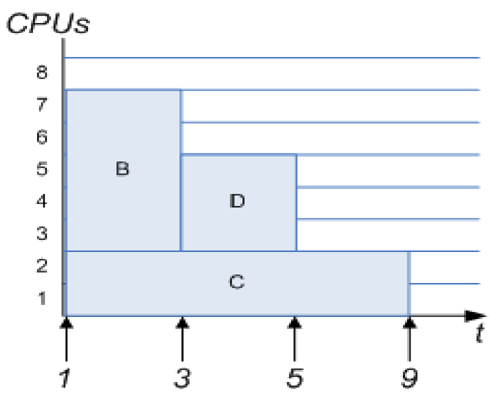
From the mapping given in Figure 1.2, it can be seen that the number of successfully served SLAs/jobs may be same under all mentioned simple heuristics but the mapping pattern solely based on the heuristics used which is why the performance of the system differ anomalously under all heuristics if a large number of SLAs are required to schedule.

1.3 An Integrated SLA Scheduling Heuristic

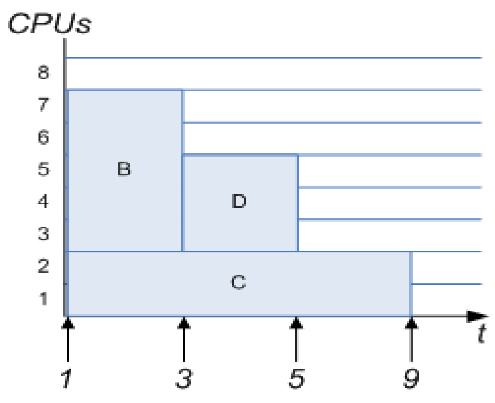
Scheduling SLAs based on the terms of agreement is addressed using integrated heuristic in [YS06] in an attempt to achieve the maximum performance. The proposed scheme finds the best solution for a set of SLAs by scheduling them with a priority which is calculated heuristically by applying parameter sweep. For the heuristics, it is suggested that the multiple terms from the SLA can be added together by multiplying with some weight factor w in order to calculate the priority of the SLAs. For a set of SLAs, a simulation is executed again and again with different values of weight (w) parameters using parameter sweep. Those values of weights that result in maximum performance are recorded and used for the scheduling decision. It is concluded that by increasing the number of SLA terms in the integrated heuristic results in increased performance. As the value of weights (i.e., w parameters) is evaluated using parameter sweep simulation therefore, this evaluation results in long computation time depending on the range of values used for the weights, number of SLA terms used in integrated heuristic, number of



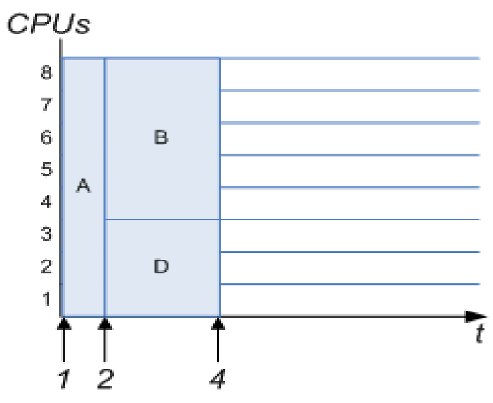
(a) Least Jobsize First (LAF)



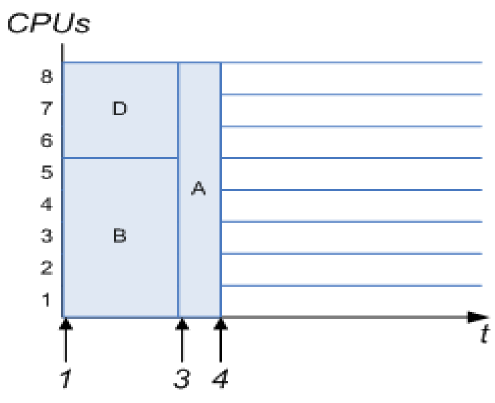
(b) Highest Jobsize First (HAF)



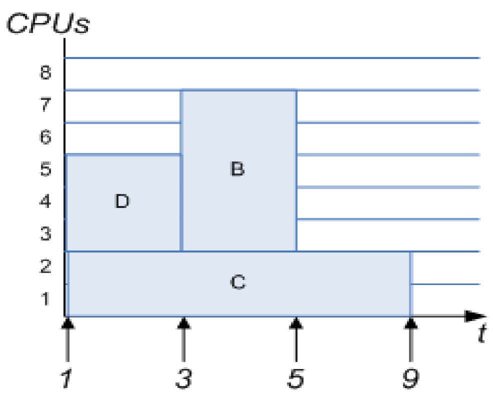
(c) Least Laxity First (LLF)



(d) Highest Laxity First (HLF)



(e) Earliest Deadline First (EDF)



(f) Latest/Highest Deadline First (HDF)

Figure 1.2: Scheduling Map under Simple Heuristics

Table 1.2: Example of Priority Based on Simple Scheduling Heuristics

(a) Least Jobsize First (LAF)

SLA	T_s	t_D	N_{cpu}	T_f	A	t_L
D	1	2	3	6	6	3
A	1	1	8	6	8	4
B	1	2	5	5	10	2
C	1	8	2	10	16	1

(b) Highest Jobsize First (HAF)

SLA	T_s	t_D	N_{cpu}	T_f	A	t_L
C	1	8	2	10	16	1
B	1	2	5	5	10	2
A	1	1	8	6	8	4
D	1	2	3	6	6	3

(c) Least Laxity First (LLF)

SLA	T_s	t_D	N_{cpu}	T_f	A	t_L
C	1	8	2	10	16	1
B	1	2	5	5	10	2
D	1	2	3	6	6	3
A	1	1	8	6	8	4

(d) Highest Laxity First (HLF)

SLA	T_s	t_D	N_{cpu}	T_f	A	t_L
A	1	1	8	6	8	4
D	1	2	3	6	6	3
B	1	2	5	5	10	2
C	1	8	2	10	16	1

(e) Earliest Deadline First (EDF)

SLA	T_s	t_D	N_{cpu}	T_f	A	t_L
B	1	2	5	5	10	2
A	1	1	8	6	8	4
D	1	2	3	6	6	3
C	1	8	2	10	16	1

(f) Latest Deadline First (HDF)

SLA	T_s	t_D	N_{cpu}	T_f	A	t_L
C	1	8	2	10	16	1
A	1	1	8	6	8	4
D	1	2	3	6	6	3
B	1	2	5	5	10	2

resources within the system and the number of SLAs within a workload (or SLAs waiting to get scheduled).

Using the maximum number of served SLAs as a performance objective, the best heuristic of the work [YS06] is given in Equation 1.1 which consists on the integration of three parameters i.e., deadline (T_f), jobsize (A) and laxity (t_L). It is called *integrated heuristic* throughout in this thesis. Integrated heuristic requires to sweep through a number of values for w_1 and w_2 one at a time. For each value of w_1, the scheme exhausts all the values defined for w_2. The H is the computed priority based on the minimum value of the combination $T_f + w_1A + w_2t_L$ for each SLA against the initialized/set values of w_1 and w_2. If, for example, the sweeping values for w_1 and w_2 are started from -5.0 and incremented with 0.1 and ended at +5.0 then, a total of 10201 values (or combination) are required to be evaluated (i.e., 101 steps from -5.0 to +5.0 with the increment of 0.1 and for both w_1 and w_2, the possible combination to evaluate will be $101 \times 101 = 10201$). Against the performance objective of maximum number of served SLAs those values of w_1 and w_2 are selected which results in maximum performance.

$$H = min(T_f + w_1A + w_2t_L) \tag{1.1}$$

where

T_f = deadline time
A = jobsize; which is the product of runtime (t_D) and number of processors (N_{cpu})
t_L = Laxity; the difference between the deadline time and execution finish time.
and w_1, w_2 are sweeping parameters.

To understand the behaviour of the integrated heuristic under parameter sweep scheme, let us now schedule the SLAs given in Table 1.1. In this particular scenario, different combinations of w_1 and w_2 values can attain maximum performance but for the sake of simplicity, the values -2 and -5 for w_1 and w_2 are selected, respectively. Assume that these values have been found after evaluating number of different values for w_1 and w_2 and among the satisfying values, this set of values is picked up. Now, the value of H is based on the picked up w_1 and w_2 values; the SLAs are ordered by minimum value of H. This order of SLAs is considered as prioritization order which is given in Table 1.3. A scheduling map against this prioritization order is shown in Figure 1.3. It can be seen that all four SLAs are scheduled to finish successfully by their mentioned deadline times according to the priority calculated using integrated heuristic.

Table 1.3: Example of Priority under Parameter Sweep Heuristics

SLA	T_s	t_D	N_{cpu}	T_f	A	t_L	w_1	w_2	$H = min(T_f + w_1 A + w_2 t_L)$
A	1	1	8	6	8	4	-2	-5	-30
C	1	8	2	10	16	1	-2	-5	-27
B	1	2	5	5	10	2	-2	-5	-25
D	1	2	3	6	6	3	-2	-5	-21

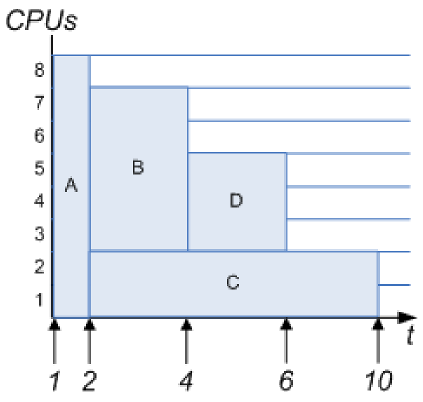


Figure 1.3: Scheduling Map under Integrated Heuristics (H)

Although, the scheme gives the best possible solution under the given objective of maximizing the number of served SLAs, the cost of computation is high due to the evaluation of a complete set of workload (i.e., set of SLAs) iteratively for each combination of w_1 and w_2 values. This time complexity is considerably increased with the number of SLAs included in the workload and therefore, not suitable for a system where not enough time or resources available to compute the best possible values of w_1 and w_2 exist. The situation can become worse if no value of w_1 and w_2 results in the best schedule. This suggests that there is a need to identify whether the use of different parameter values for the integrated heuristic will really make a significant impact justifying the high computation cost associated with it.

1.4 Problem

The problem considered in this thesis concerns the allocation of jobs bound by the SLAs onto a set of resources so that some performance objective is met. For example, such an objective could be the maximization of the number of SLAs that can meet their constraints. This problem can be of interest in a metacomputing [SC92] environment such as high performance Grid [BFH03], in which multiple computing sites are aggregated to execute a job whose execution requires multiple resources. A component, called *scheduler*, is used to schedule the incoming jobs/SLAs onto the resources. Our focus in this environment is on the scheduling aspect, that is, how to allocate the jobs onto resources to get the maximum advantage.

Different approaches for scheduling such SLA-bound jobs are based on job prioritization. As demonstrated with the examples in the previous section, different prioritization schemes may lead to different system performances. In other words, there is not a single heuristic from the heuristics discussed in Section 1.2, which is able to produce optimal performance under all scenarios. The problem of scheduling becomes more complicated if some expected level of service is desired to maintain. To cope with this problem, an integrated heuristic proposed in [YS06]. But as this integrated heuristic requires parameter sweeping to find the best possible solution or priorities therefore, there is a need to identify whether the sweeping values will really make a positive impact on the quality of the schedule. As, without using parameter sweep simulation on integrated heuristic there is no way to predict the weights to yield the best results [HSR09].

To motivate this, imagine that there are only two SLA-bound jobs to be scheduled onto resources and there is an unlimited number of resources, which far exceed the requirements of these two jobs. Then, it is easy to realize that even the simplest scheduling heuristic will easily meet the requirements of the two jobs. There is no need to resort to computationally expensive heuristics, such as the heuristic proposed in [YS06].

Thus, the question that this thesis sets to investigate is to what extent the power of the integrated heuristic for SLA scheduling is really beneficial and to what extent equally good solutions can be obtained using simple heuristics. An answer to this problem will allow the development of an adaptive scheduling scheme, which resorts to the integrated heuristic only when the need arises and some quantifiable benefits can be obtained.

1.5 Performance Metric

The performance of the system for the respective scheduling heuristics is measured in terms of served number of SLAs. The comparison of obtained performance under simple heuristics and the integrated heuristics (both discussed above) is done using *performance difference (PD)*. PD is the percentage difference between the maximum number of served SLAs by any simple heuristic and by integrated heuristic and is calculated in terms of percentage using the formulae given in Equation 1.2.

$$PD = \frac{(P_{Int} - Max\_P_S)}{Total\ Number\ of\ Submitted\ SLAs} * 100 \tag{1.2}$$

Where

P_{Int} = Performance achieved by Integrated Heuristic
Max_P_S = Maximum Performance achieved by any Simple Heuristics

Although, the performance metric is the number of served SLAs in terms of PD, but the utilization of the system is also calculated and presented. The utilization (U) of the system is calculated against sum of served jobsize(A) and overall system capacity (S_C) using Equation 1.3. The overall system capacity (S_C) is defined as the product of number of system CPUs (S_{CPU}) and the difference of finish time of last accepted SLA/job & start time of first submitted SLA/job (i.e., $T_{f_{last}} - T_{s_{first}}$) and is calculated using Equation 1.4.

$$U = \frac{\sum A}{S_C} * 100 \tag{1.3}$$

Where
$\sum A$ = Sum of served jobsize (A)
S_C = Over all system capacity

$$S_C = (T_{f_{last}} - T_{s_{first}}) * S_{CPU} \tag{1.4}$$

Where
$T_{s_{first}}$ = Start time of first submitted job/SLA
$T_{f_{last}}$ = Finish time of last accepted job/SLA
S_{CPU} = Total system CPUs

As mentioned, the utilization (U) of the system is not a measure of performance but presented for the sake of clarity and to show that, in a system where characteristics of next arriving jobs are not known apriori, the utilization of the system may not be increased with any particular heuristic even obtaining the maximum served SLAs. For example, the computed system utilization for the scenario presented in Section 1.2 and 1.3, is shown in Figure 1.4. Even though, the heuristics least jobsize first (LAF), highest laxity first (HLF) and earliest deadline first (EDF) resulted in 100% system utilization but served only three jobs/SLAs for shorter period of time. On the contrary, the heuristics highest jobsize first (HAF), least laxity first (LLF) and latest/highest deadline first (HDF) resulted in 50% of system utilization but again served only three jobs/SLAs for longer period of time leaving gaps within the schedule (as shown in Figure 1.2). In contrast to all mentioned simple heuristics, the integrated heuristics (H) resulted in 55.56% of system utilization but served successfully all four jobs/SLAs, as shown in Figure 1.3. Although, integrated heuristic served all four jobs/SLAs but if compared in terms of utilization with all simple heuristics, it utilized the system 5.56% higher than the highest jobsize first (HAF), least laxity first (LLF) & latest/highest deadline first (HDF) and 44.44% lower than the least jobsize first (LAF), highest laxity first (HLF) & earliest deadline first (EDF).

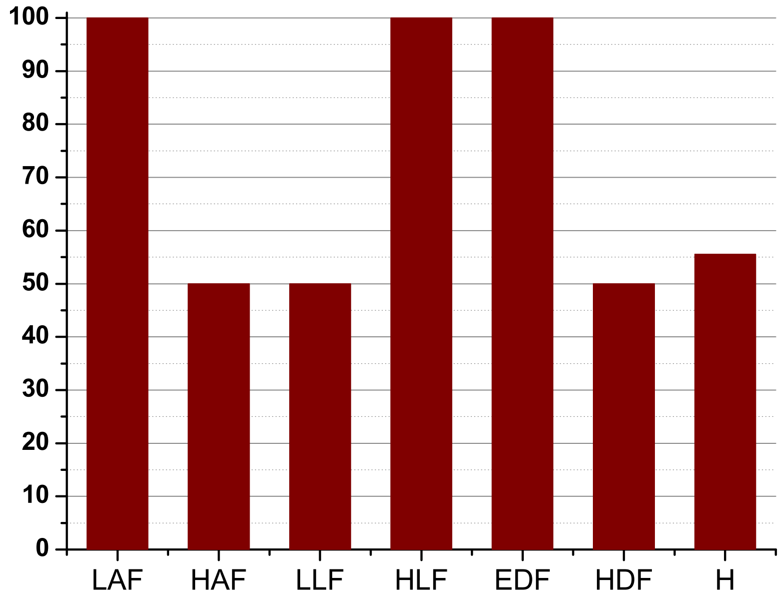


Figure 1.4: System Utilization under all Heuristics

1.6 Aims & Contribution

The increased expressiveness in the form of SLAs allows the efficient reservation of resources and could also lead to an optimized schedule using an integrated heuristic such as in [YS06]. The aim is to analyse the problem of identifying the need of parameter sweep on the integrated heuristic for the SLA based workload. For this purpose, we need to evaluate all the discussed simple heuristics as well as the integrated heuristic separately against different SLA based workloads to observe the performance of the system. It is desired to quantify the workload against the proposed metric which uses the concept of utilization for its calculation. It is then investigated further, whether the integrated heuristic is always required for the maximum performance or not. Therefore, a scheme is required to distinguish and characterize the SLA based workload for which a metric is proposed. Briefly, this thesis contributes to the knowledge on scheduling parallel jobs bound by SLAs in the following context:

- It proposes an approach to identify the impact of scheduling heuristics for SLA-bound workloads.
- It suggests a novel approach to characterize an SLA-bound workload.
- A novel metric-based approach is proposed to identify the improvement in performance with integrated heuristic.

- Based on the above observations, an adaptive scheme to adapt the scheduling approach to use the integrated heuristic for maximum performance (in situations where this is necessary) is proposed and evaluated.

1.7 Thesis Organization

This section gives a brief overview of the individual chapters of the thesis in itemized form.

Chapter 2: Presents a discussion on Service Level Agreement (SLA) in the context of its structure and life-cycle together with a review and taxonomy for the classification of the reviewed scheduler and resource manager.

Chapter 3: Presents the methodology to carry out experiments. It discusses the architecture of the designed simulator, scheduling heuristics and simulator execution scheme. Further, it details about SLA-based workload generator, assumptions made to perform the experiments, methods to compute and compare the results, and limitations of the designed simulator.

Chapter 4: A metric based workload characterization scheme is proposed and evaluated, which checks the feasibility of the application of an integrated heuristic on the SLA based workloads. A simulation based evaluation suggests that the high percentage of workloads can be checked in real time for the suitability of scheduling the workload with the exhaustive scheme, which uses the integrated heuristic, to extract better performance than the simple scheme that uses simple heuristics.

Chapter 5: An adaptive scheme is proposed and evaluated to schedule the SLAs using the metric and its characterization scheme, explored in Chapter 4. The adaptive scheme suggests whether to use the exhaustive scheme or the simple scheme for the arriving SLAs. Again, the simulation based evaluation is conducted which shows that how the computation cost can be reduced by avoiding the exhaustive scheme without the major loss in performance.

Chapter 6: It is the final chapter and concludes the dissertation by reviewing the contribution and discussing the direction for the future work.

Chapter 2

Review of State of the Art

In the context of distributed computing on networked virtual supercomputer, constructed dynamically from geographically distributed resources linked by high-speed networks [FK96], *metacomputing* concept was proposed in 1992 by Larry Smarr and Charles E. Catlett [SC92]. The idea of metacomputing was transparent provision of collection of heterogeneous computing resources connected by high-speed network to the user. The resource could be of any type such as cluster of workstations, supercomputer, networked virtual supercomputers or simply meta-computer. Metacomputing system is distinguished from a simple collection of computers by a software layer, often called *middleware*, which transforms a collection of independent resources into a single, coherent, virtual machine [LGFH95]. It consists of a set of services that enables the system to execute jobs on number of resources seamlessly.

In metacomputing environments for example *Grid* [BFH03] and *Cloud* [WvLKT08], services are composed at run time to fulfil users' requirements and the composition of a service may be based on large distributed, loosely coupled applications such as in Web services [BCT$^+$05]. Under such an environment users may want to set an upper limit or threshold for the composed service response time or set a deadline time by which a user needs the results from the composed service which is termed as *quality of service (QoS)*; an important issue when service is built out of independent components [BCT$^+$05]. To establish the quality of service criterion between a user and a service provider, a *Service Level Agreement (SLA)* is required. The SLA is then considered as a contract between two parties i.e., service provider and service consumer. It includes the terms that describe the expected level of service within which the service will be provided by service provider [SY08].

In this chapter, a discussion on Service Level Agreement (SLA) in the context of its structure and life-cycle is presented in Section 2.1. Then, taxonomy is presented for the classification of the surveyed scheduler and resource manager in Section 2.2. Finally, the survey is conducted and presented in Section 2.3, based on the taxonomy for the few renowned scheduler and resource management system.

2.1 Service Level Agreements

The Web Services (WS) are a set of technologies based on which a large distributed and loosely coupled applications can be composed automatically [BCT$^+$05]. The composition of such type of applications is defined by two types of properties i.e., *functional* and *non-functional*. Functional properties focus on how does a user automatically compose a 'correct' service whereas, non-functional properties focus on the 'quality' of a composed service. The quality of service (QoS) of the composed services is also of paramount importance which refers to the non-functional propoerties [BCT$^+$05]. Therefore, in order to maitain the quality of the composed services, the concept of service level agreement (SLA) was proposed for the Web Services (WS) which has been adopted from the domain of telecommunications [Kub11]. To address the issues associated with QoS, number of standards have been proposed such as, IBM Web Service Level Agreement (WSLA) [KL03], HP's Web Service Management Language (WSML) [SMS$^+$02] and GRAAP (Grid Resource Allocation and Agreement Protocol) Working Group WS-Agreement [ACD$^+$04].

Different aspects of QoS have been considered in different researches, such as, a QoS based negotiation addressed in G-QoSM framework [AaRW$^+$02] in which QoS requirement for a service can only be expressed in low-level and hence a user should be well versed in order to write the SLA for the required services. Whereas, in the work [KL03], the need of formal language to express SLAs and a runtime architecture to interpret the language, from the web service provider perspective has been addressed. It measures different services using number of different defined metrics and the metrics are defined by technical experts. The requirement of technical expertise to define such metrics is addressed in [TP11], a web service monitoring framework to specify the SLAs, which claims to reduce the manual efforts to guarantee the SLAs without the need of the technical skills. An Open Grid Services Architecture (OGSA) based scheme to manage the QoS attributes specified within the SLA, is introduced in [LKA$^+$08]. It discussed the OGSA compliant Execution Management Service, to enable the management and enforcement of SLAs. A genetic algorithm based approach for the SLA based service composition in Cloud computing environment, proposed in [AFAS11]. It claimed that as Cloud providers offer SLAs based on the history of their performance which is unreliable and may offer SLAs with incorrect QoS measures because there are no means of SLAs validation. Thus, from user perspective, the automatic discovery of the services based on the formal specification of user needs, is required. This allows user to compare and evaluate different SLAs offered by different Cloud providers.

We discuss here the GRAAP WS-Agreement [ACD$^+$04] in brief detail in the next section to understand the purpose and need of such standards.

2.1.1 WS-Agreement

The WS-Agreement [ACD$^+$04] is a recent proposal for GRAAP working group to advertise the services or capabilities of service providers, creating agreements and monitoring the agreement compliance. It is an XML-based document containing descriptions of the functional and non-functional properties. The agreement creation process typically starts with a pre-defined agreement template specifying customizable aspects of the documents and rules that must be followed in creating an agreement.

The general structure of SLA is depicted in Figure 2.1 as specified in WS-Agreement specification [ACD+04]. The agreement may optionally have a *name* wherease, the *context* contains mandatory information about the initiator, the responder and the provider of the agreement; expiration time of the agreement; and its template Id. Functional and non-functional requirements are specified in the *Terms* section that is divided into *Service Description Terms (SDTs)* and *Guarantee Terms*. SDTs define the functional attributes of the agreement whereas the guarantee terms contain the non functional attributes. Guarantee terms further describe the conditions, Service Level Objectives (SLO) and Business Value List (BVL) related to the agreement. SLO terms use to express the objective to meet e.g., earliest job start time, latest job finish time (or deadline time), execution time (or reserved time for job execution) and number of CPUs. BVL may express the importance of meeting an objective as well as information regarding penalty or reward [UHHS11]. An agreement may contain any number of SDTs. An agreement can refer to multiple components of functionalities within one service, and can refer to several services. Guarantee terms define an assurance on service quality associated with the service described by the SDTs. An agreement may contain zero or more guarantee terms.

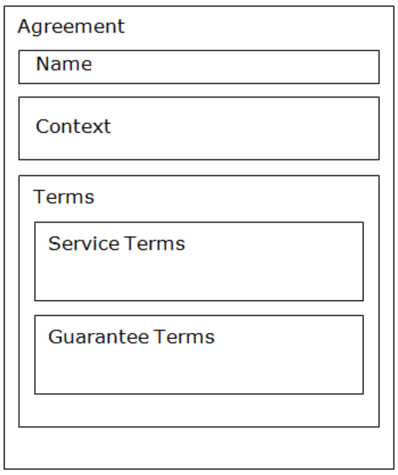


Figure 2.1: SLA Structure [ACD+04]

SLA Life-cycle

SLA concept in service oriented computing, is inspired from the telecommunication therefore, the life-cycle of an SLA can be divided into six distinct phases i.e., *development, negotiation, implementation, execution, assessment,* and *decommission* [HQK+06]. But it has been noted in [PBM08] that the life-cycle of an SLA is confused with the life-cycle of a service and hence, a better model of an SLA life-cycle suggested which consisted of eight phases i.e., *template created, template advertised, SLA negotiation, agree, execute, assess, settle,* and *archive.* The two life-cycles are compared and depicted in Figure 2.2. In the work [PBM08], the development phase replaced by two different phases that are, template created and template advertised. First an SLA template is formed and then advertised by the provider. Then, an individual SLA is negotiated then agreed on the basis of this template. The negotiation phase is separate to the agreement phase as it is possible that a negotiation may not always lead to an agreement. In execution phase the resources are actually commissioned and deployed to complete the agreed service. Once the execution of a service based on an agreed SLA is started, in the assessment stage where, the performance of the service is checked periodically against the agreed SLA. Finally, when the service has completed, the final settlement phase takes place where it is determined if the SLA was met and what the costs to the customer are and what penalties may apply to the provider for breaching the terms of the SLA.

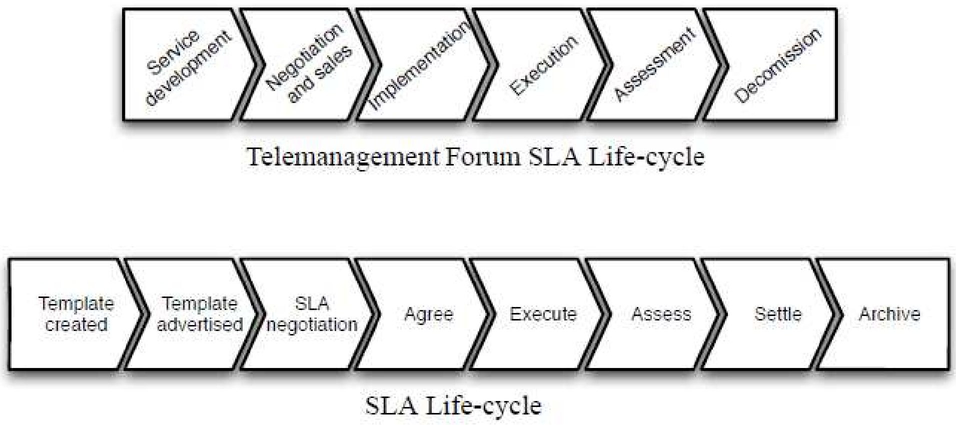


Figure 2.2: Comparison of Two Life-cycles [PBM08]

The objective of the WS-Agreement specification is to define a language and a protocol with a goal of standardized methods for complete SLA life-cycle. It needed *Web Service Description Language (WSDL)* [CMRW07], a XML based interface description language, to express the message exchanges and the state of the resources. It describes a Web service, specifies the location of the service and the operations offer by the service. The term *service* typically refers to the piece of code implementing the XML interface to a resource and resource could be of any type of application or data store within an organization [Sko03]. XML schema defines 'what' XML messages may be used and WSDL group these messages into operations and operations into interface. WSDL service definitions provide documentation for distributed systems and serve as a recipe for automating the details involved in applications communication [CCMW00]. At the time of writing, the latest version of WSDL is 2.0 like its' previous version it is also described into three layers i.e., *abstract interface*, *protocol binding* and *service endpoints* [Liu05] also depicted in Figure 2.3.

As detailed in [Liu05], an *abstract interface* defines what a service is i.e., the messages to be exchanged with a service which correspond to expected input and output. A *protocol binding* defines how to access a service with a concrete protocol. For example, when SOAP messages are exchanged over HTTP, the binding defines the parts of messages for the SOAP header and body. The *service endpoint* describes the exact address where a service can be activated which in turn largely depends on the binding in use such as URL

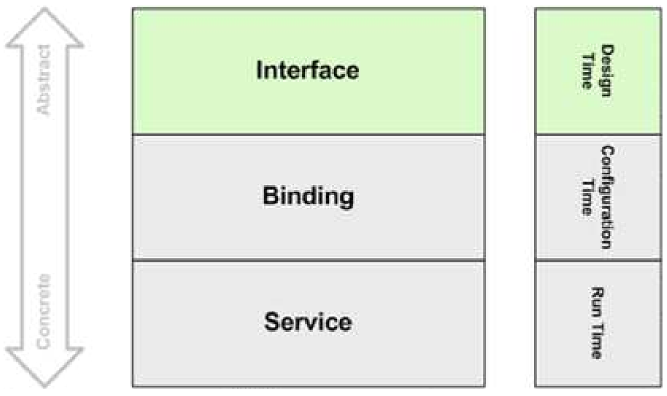


Figure 2.3: Three Layered Description of WSDL [Liu05]

in an HTTP scheme. Figure 2.3 also shows the timing at which the work of all three layer is carried out as well as the level of service from abstract to concrete. Usually, at design time, service is defined using abstract interface at abstract level. The endpoint refers to the real implementation of the service which is available to run at run time, of course a concrete object. Whereas, binding definition is typically defined at configuration time to connect the abstract level to concrete level for the provision of required service.

Besides WSDL, the WS-Agreement protocol is dependent on *WS-Addressing* [BCC+04], *WS-ResourceProperties* [GT04], *WS-ResourceLifetime* [SB06], and *WS-BaseFaults* [LM06] [CMRW07].

2.1.2 Web Service Resource Framework (WSRF)

To fulfil the need to define conventions for managing the state of services so that applications discover, inspect, and interact with stateful resources in standard and interoperable ways, the WS-Resource Framework is then defined to address these conventions [CFF+04a]. Web Service Resource Framework (WSRF) is an approved OASIS [oas] standard [glob]. It is inspired by the work of the Global Grid Forum's Open Grid Services Infrastructure (OGSI) Working Group [glob]. Two different technical committees were formed to work on the specifications [glob]: the WSRF technical committee; to work on WS-ResourceProperties, WS-ResourceLifetime, WS-ServiceGroup, and WS-BaseFaults specifications and the WSN technical committee; to work on WS-BaseNotification, WS-Topics, and WS-BrokeredNotification specifications. The WSN specifications are build on WSRF but still they are not considered as part of WSRF proper [glob].

The WS-Resource Framework builds on the WS-Addressing [BCC+04] specification. It adopts the endpoint reference construct defined in the WS-Addressing specification as an XML syntax for identifying Web service endpoints and endpoint reference contains an identifier of a specific stateful resource associated with the Web service [CFF+04a].

WS-Addressing [BCC+04]

WS-Addressing provides transport-neutral mechanisms to address Web services and messages. This specification enables messaging systems to support message transmission through networks that include processing nodes such as endpoint managers, firewalls, and gateways in a transport-neutral manner. It defines two interoperable constructs that are, *endpoint reference* and *message information headers*. Endpoint references convey the information needed to identify/reference a Web service endpoint and message information headers convey end-to-end message characteristics including addressing for source and destination endpoints as well as message identity. Both of these constructs are designed to be extensible and re-usable so that other specifications can build on and leverage endpoint references and message information headers.

WS-ResourceProperties [GT04]

The composition of a *stateful resource* and a Web service that participates in the *implied resource pattern* is termed as *WS-Resource*. A stateful resource have a specific set of state data expressible as an XML document, a well-defined lifecycle and be known to, and acted upon, by one or more Web services [FCF$^+$05]. The implied resource pattern refers to the mechanisms used to associate a stateful resource with the execution of message exchanges implemented by a Web service [FCF$^+$05]. WS-ResourceProperties describes the WS-Resource definition, and describes how to make the properties of a WS-Resource accessible through a Web service interface, and to manage and reason about a WS-Resources lifetime [CFF$^+$04a]. WS-ResourceProperties standardized the terminology, concepts, operations, WSDL and XML needed to express the resource properties projection, its association with the Web service interface, and the messages defining the query and update capability against the properties of a WS-Resource. It also standardized the definition of the properties of a WS-Resource to be declared as part of a Web service interface. The declaration of the WS-Resources properties represents a projection of or a view on the WS-Resources state. This projection is defined in terms of a resource properties document. This resource properties document serves to define a basis for access to the resource properties through Web service interfaces.

WS-ResourceLifetime [SB06]

WS-ResourceLifetime standardized the terminology, concepts, message exchanges, WSDL and XML needed to monitor the lifetime of, and destroy, WS-Resources. The lifetime of a WS-Resource is defined as the period between its instantiation and its destruction. The WS-ResourceLifetime specification standardized the means of WS-Resource destruction. It defined two means of destroying a WS-Resource: *immediate destruction* and *time-based or scheduled destruction*. The immediate destruction of a WS-Resource may be accomplished using the defined message exchanges. However, a resource may be destroyed after a defined period of time. If that time period is elapsed, the WS-Resource may self-destruct without the need for an explicit destroy request message from a client. WS-ResourceLifetime also defines a standard message exchange by which a service requestor can establish and renew a scheduled termination time for the WS-Resource, and the circumstances under which a service requestor can determine that this termination time has elapsed.

In particular, WS-ResourceProperties and WS-ResourceLifetime are used to represent Agreements as Resources [CMRW07].

WS-BaseFaults [LM06]

The WS-BaseFaults standardized the terminology, concepts, XML types, and WSDL usage of a base fault type for Web service interfaces. Problem determination in a Web services setting is simplified by standardizing a base set of information that may appear in fault messages. WS-BaseFaults defines an XML Schema type for base faults, along with rules for how this base fault type is used and extended by Web services. WS-BaseFaults defines an XML Schema type for a base fault, along with rules for how a fault type is used by Web services.

WS-ServiceGroup [MS04]

The WS-ServiceGroup defines a mean by which Web services and WS-Resources can be aggregated or grouped together for a domain specific purpose. The term service group refers to a standard mechanism for creating a heterogeneous by-reference collection of Web services [CFF$^+$04a]. WS-ResourceProperties are used to express the ServiceGroup membership rules, membership constraints and classifications. Groups are defined as a collection of members that meet the constraints of the group. The ServiceGroupRegistration interface extends the basic ServiceGroup capabilities with message exchanges for managing the membership of a ServiceGroup. Service groups can form a wide variety of collections of services, including building registries of services and they are particularly important when dealing with stateful entities such as WS-Resource [CFF$^+$04a].

WS-Notification [GHM06]

In an environment where stateful resources may be created and destroyed, and may change their state, dynamically, it becomes important to provide support for asynchronous notification of changes in the state of individual resources [CFF$^+$04a]. Such support is provided using WS-Notification family that define a standard Web services approach to notification using a topic-based publish/subscribe pattern. From the perspective of the WS-Resource framework, the WS-Notification family of specifications extends the utility of WS-Resource by allowing requestors to ask to be asynchronously notified of changes to resource property values [CFF$^+$04b]. WS-Notification includes: standard message exchanges to be implemented by service providers that wish to participate in Notifications, standard message exchanges for a notification broker service provider (allowing publication of messages from entities that are not themselves service providers), operational requirements expected of service providers and requestors that participate in notifications, and an XML model that describes topics. *WS-BaseNotification, WS-BrokeredNotification* and *WS-Topics* are part of WS-Notification family.

WS-BaseNotification [GHM06] [CFF$^+$04b]: describes the basic roles, concepts, and patterns required to allow a subscriber to register interest in receiving notification messages from a notification producer. An notification can concern anything, a change in the value of a resource property, some other internal change in the state of the notification producer, or some other 'situation' within the environment. A subscriber registers interest in receiving notification messages on one or more topics by issuing a 'subscribe' message. In response, the subscriber receives a WS-Resource-qualified endpoint reference to a 'subscription' WS-Resource. The subscription WS-Resource models this relationship between the subscriber and the producer, and it uses WS-ResourceProperties and WS-ResourceLifetime to help manage this relationship.

WS-Topics [GNC+04] [CFF+04b]: presents an XML description of topics and associated meta data. Topics are a mechanism for organizing notification messages so that subscribers can conveniently understand what types of notification are available for subscription. Topics can be organized hierarchically; one topic can be further decomposed with child topics. Topics are also scoped by namespace, much as XML types and elements are scoped by XML namespaces.

WS-BrokeredNotification [CL06] [CFF+04b]: defines the interface to a NotificationBroker that implements an intermediary service to manage subscriptions for other entities in the system that produce notification messages.

2.1.3 JSDL and WS-Agreement

Job Submission Description Language (JSDL) [AEBCDF+] is an extensible XML specification from the Global Grid Forum for describing the requirements of computational jobs for submission to resources, particularly in Grid environments but not restricted to it. The motivations behind the JSDL specifications are ineteroperability between the systems and ease of interactions between the grid systems. In order to utilize different systems within an organization without preparing and maintaining a number of different job submission documents (one for each system), a standardized language JSDL proposed to alleviate this problem. Further, as grid systems involve in number of interactions between different types of job submission system it is therefore required, that interactions can be undertaken automatically, facilitated by a standard language that can be easily translated to each system's own language. JSDL elements can generally be categorized into following three categories:

- Job identification requirements,
- Resource requirements and
- Data requirements

These elements are restricted to the description of requirements of jobs at submission time and no element can be defined or altered after it has been submitted. Job management system responsible for maintaining the unique job identifiers or job status that may be available in separate documents. As JSDL describes submission requirements of a job only and hence, it does not address the management of entire lifecycle of a job or the relationship between individual jobs. It is therefore anticipated that, number of other languages and protocols are also required to manage entire lifecycle of a job and/or to enable communication between the jobs. For example, a workflow language is required to use in order to make relationships for the consumption and production of data, between individual jobs described using JSDL. WS-Agreement specification is required to integrate into system together with JSDL so that appropriate negotiation and agreement protocol is used to put a job in an optimal resource environment according to its requirements.

2.2 Taxonomy of Scheduling and Resource Management Systems

A taxonomy and a survey of the published scheduling and resource management is presented in this section. Generally, a market-based metacomputing environment such as Grid, consisted of three entities i.e., user, broker/meta-scheduler, and service provider. A user is considered as a consumer of the services (i.e., computing resources) provided by the service providers. The service providers own and manage the computing resources in order to satisfy the user demands. Therefore, service provider is also termed as Local Resource Manager (LRM). It is responsible for the actual planning and scheduling of user request on to the available resources. The broker/meta-scheduler acts as a middle man to help user to find suitable resources (or service provider) for the user request. A broker provides matchmaking service between end-user and resource provider based on the requirement of end-user and features of resources available from resource provider. Clearly, broker/meta-scheduler does not actually execute the user request on to resources instead, it presents the suitable resources to the user using the information published by the service providers to the broker/meta-scheduler and then LRM is responsible to execute the user request.

The taxonomies presented in [KBM02] classify resource management system with respect to organization of machines within Grid, resource model, resource namespace organization, quality of service (QoS) support, information store organization, resource discovery and dissemination mechanism. Based on the marketing principles [AHR01], the survey can be classified into two broad categories: Market-oriented or System(production)-oriented. The similar principle for the classification of scheduling and resource management systems adopted in [GB11] to present several other taxonomies. In this thesis, the classification of the surveyed literature is done using the taxonomy followed by the work [GB11] [Zhe10] [YB06b] [HML10]. Therefore, classification from six perspectives: *Resource Access Mechanism, Architectural Model, Application Model & Scheduling Domain, QoS* and *Scheduling Perspective*, is done for the surveyed literature.

Scheduling Domain & Application Model Perspective

Scheduling domain defines the allocation domain for a job. The scheduler which governs multiple local schedulers that are responsible to administrate the specified domain is termed as meta-scheduler. Whereas, the local scheduler is responsible to actually schedule the jobs onto resources. A task or job are general corresponding names of the threads, processes, and applications under different scheduling concepts. Conceptually, *parallel job* is comprised of threads and these threads also represents the degree of parallelism. The data requirement to process the parallel job tells about the type of dependency. If a parallel job does not require any data from any other job to be processed then it is considered as *parallel independent job* because there is no dependency on the execution of the job to any another job. On the other hand, if a parallel job requires data from another job to be processed then it is considered as *parallel dependent job* and such jobs are presented as *directed acyclic graph* (DAG) [] [CJSZ08]. Therefore, schedulers can be classified as meta-scheduler or local scheduler and scheduling application can broadly be classified as dependent or independent parallel job.

Architectural Model Perspective

In order to distinguish the characteristics of the computing resources modelled in different scheduling studies, the taxonomy from the perspective of underlying system architecture and operating environment is considered. The architectural model taxonomy consists of four sub-taxonomies: *management control, resource composition, execution support* and *scheduling system.*

- *Management control* depicts how resources are organized and modelled in a computing environment. All resources can be controlled by a *central* resource management system (RMS) or scheduler, such type of system is considered as *centralized* system. Whereas, in a system in which, resources are grouped into domains and each domain of resources is controlled and managed by its local management system, is considered as *decentralized* system.

- *Resource composition* tells the diversity of resources as *homogeneous* or *hetrogeneous.* A system whose composition consists of same resource components and configurations is termed as homogeneous, whereas heterogeneous system consists of resources having different components and configurations.

- *Execution support* classifies resource model into two categories based on the processing support provided by the underlying operating system. One of the categories is termed as *single-programming* or *space-shared* in which at most only a single job can be executed at any one time on a processor, and the other category is termed as *multiprogramming* or *time-shared* in which multiple jobs can be executed concurrently at any one time on a processor.

- If the *scheduling system* schedule the new application or job without disrupting the already made scheduling decision then it is termed as *rigid* whereas, in *flexible* scheduling system all resources are assumed to be free of load and upon arrival of new job, a new schedule is then calculated.

Resource Access Perspective

Resource access refers to the mechanism to gain access to the resources. If the resource access mechanism involves any trading mechanism between the provider and user then it is refered as market-oriented model. Under this model, the worth of the user submitted job is analyzed from user and/or system perspective, depending upon the system implementation. An alternative to this is, the system-oriented model which only focuses on the utilization or throughput or increased revenue aspect of the system indifferent of the worth of the job from the user aspect. As the market model involves trading mechanism therefore, number of economic models have been defined from computing aspect in [BSGA01] [GB11] but the renowned ones are listed below:

- In the *commodity* market model, service providers specify their service prices and users pay the amount according to the consumption of service. Pricing Schemes in a Commodity Market Model can be based on *flat fee, usage quantity, subscription* and *demand & supply.*

- In an *auction* market, users submit their bids through an auctioneer to use the resources or service. It uses market forces to negotiate a clearing price for the service. The auctioneer acts as a coordinator and sets the rules of the auction. The user who submits the highest bid wins the access to a service. The negotiation continues until the highest bid is accepted.

- In *proportional share* model, a user application gets the percentage of resource share in proportion of the submitted bid value to the other users' bids or system may be set up to divide the resource share between the jobs based on the execution time and deadline time mentioned at the time of job submission.

- In the *contract-net* market, the process initiated by user by announcing his/her requirements against which the bids are submitted by the potential service providers. The service providers submit their bids after evaluating their capabilities and the requirements advertised by the user. The user after receiving the bids then selects the most suitable service provider to form a contract.

- In the *bartering market*, users form a community in which each user from the community can be either user or service provider. The community is a pool where users share resources and services with each other to create a cooperative computing environment.

Quality of Service (QoS) Perspective

In order to guarantee the quality of service (QoS) the specification of QoS related parameters and policies have been investigated from the scheduling and resource management perspective, hence the design of scheduling can be classified into three cagtegories: *QoS Parameters, QoS Specification* and *QoS Guarantee.*

The user of the service specifies the type of service that he/she expects from the system to deliver using the *QoS parameters.* The common parameters are time, economic cost, reliability, security etc. A typical example of a time QoS parameter is the deadline by which the user submitted is expected to finish. The economic cost parameter includes budget that user willing to pay for the successfully completion and penalty parameter that tells the service provider that how much it has to pay back to the user if the service is not provided successfully.

QoS specification encompasses requirements for expected performance in order to understand the users satisfaction. A QoS specification can be Constraint-based, Optimization-based or some other type. Under constraint-based QoS specification the service provision depends on whether the delivered QoS result falls within the defined range of values or not for a particular QoS parameter. The optimization-based QoS specification expresses the user's desire to optimize a particular QoS parameter.

The *QoS Guarantee* can be of two types: rigid and flexible. In rigid QoS guarantee, the system has to check for the defined threshold of a QoS parameters and if it crosses over the threshold values then the system stop scheduling the job due to the voilation of agreed SLA. Whereas, flexible QoS guarantee ensures that the job will be completed at a certain time with minimal penalty paid to the user by the service provider, which means overrunning the deadline is allowed with flexible guarantee system.

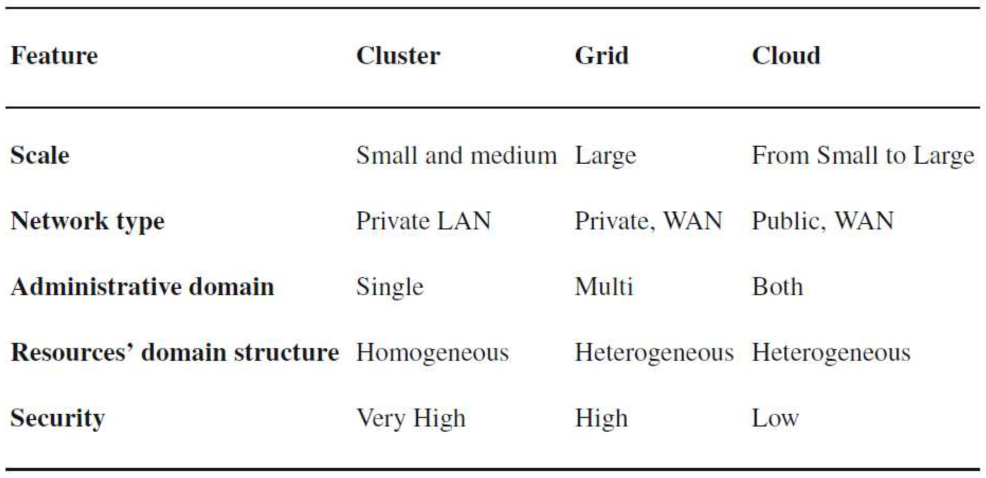
Scheduling Perspective

The full-plan-ahead or simply full-ahead refers to the planning of an application scheduling before the start of its execution and opposite of this scheme is called just-in-time or simply in-time scheduling in which the scheduling decision is made for each individual task only when the task is ready to start. There is a possibility of provision to reserve the resources in advance to execute the user application. The scheduling system may then be distinguished in terms whether it supports advance reservation [SFT00] or not. The system can further be classified whether it supports the schedule updates i.e., the assignment to a resource and the scheduled start time can be modified or not. If it can be modified then the schedule updates is of dynamic type otherwise it is considered as static.

2.3 Survey of Scheduling & Resource Management Systems

As discussed earlier, metacomputing system is distinguished from a simple collection of computers by a software layer, often called *middleware*, which transforms a collection of independent resources into a single, coherent, virtual machine [LGFH95]. It consists of a set of services that enables the system to execute jobs on number of resources seamlessly. The high performance distributed systems (HPDSs) that are primarily concerned with the development of high-performance applications, which can be executed simultaneously on multiple connected (or networked) computers or supercomputers. Today, the high performance distributed systems (HPDSs) can be categorized into three main categories [Kol12] i.e., *Cluster*, *Grid* [BFH03] and *Cloud* [WvLKT08] systems. Despite the fact that all HPDSs models are collectively based on the distributed computing paradigm but they have different characteristics [Kol12].

The modern Cluster Systems are composed of computers usually restricted to a single virtual local-area network (VLAN). These systems are designed as platforms for processing the data intensive applications, multi-level system management, and the implementation of the scalable methodologies and techniques as well as for replicated storage and backup servers [Kol12]. The main goal of the Grid Systems is to connect geographically distributed resources through wide area high speed networks (Internet). In contrast with the cluster and other conventional distributed systems, grids account for the different administrative domains with their own access policies, users privileges and requirements [Kol12]. The Cloud Systems class is the most recently developed HPDS category involving over-the-Internet provisions of both physical and virtualized scalable resources and it may be seen as a next step in the evolution of the grid environments [Kol12]. The cloud providers deliver common business applications online, while the software and data are stored on physical servers. The main features of the three mentioned categories of HPDS are shown in Figure 2.4.



Feature	**Cluster**	**Grid**	**Cloud**
Scale	Small and medium	Large	From Small to Large
Network type	Private LAN	Private, WAN	Public, WAN
Administrative domain	Single	Multi	Both
Resources' domain structure	Homogeneous	Heterogeneous	Heterogeneous
Security	Very High	High	Low

Figure 2.4: Main Features of Three Categories [Kol12]

We discuss different middleware, based on which number of solutions for the development of metacomputing infrastructure have been proposed, together with the different metacomputing infrastructures proposed in different literatures.

Globus

A metacomputing/Grid infrastructure toolkit called *Globus* [FK96] was proposed to harness the power of computing resources. It is a community-based, open-architecture, open-source set of services and software libraries that support Grid infrastructure and its applications [FKNT04]. The project started in late 1990s [Fos06] and the latest version available at the time of writing is 5 [gloa]. It is a collection of software components that enable the development of applications for high-performance distributed computing Grids. Globus provides basic mechanism of communication, resource discovery, resource scheduling, authentication, network information and data access to construct alternative infrastructures, services and applications. This paved a foundation for the provision of effectively infinite cycles and storage in the form of *Grid* [BFH03]. Grid enables users to access single computer or the aggregation of several resources as *virtual computer* through an interface seamlessly. It allows to reserve the resources either in advance or for immediate use and is able to discover resources dynamically through GARA (Globus Architecture for Reservation and Allocation) [FKL^{+}99] for end-to-end provision of high quality of service (QoS). The component Grid Resource Allocation and Management (GRAM) protocol and its 'gatekeeper' service, which provides for secure, reliable, service creation and management; the Meta Directory Service (MDS-2), which provides for information discovery through soft state registration, data modeling, and a local registry ('GRAM reporter'); and the Grid Security Infrastructure (GSI), which supports single sign on, delegation, and credential mapping [FKNT04]. Different administrative domains can be integrated together without compromising on their autonomicity. Each domain is termed as *virtual organisations (VO)* [Fos01]. A Virtual Organisation (VO) comprises a set of individuals and/or institutions having direct access to computers, software, data, and other resources for collaborative problem-solving [LHP+04]. The entity responsible to manage all VOs is called *virtual organisation management system (VOMS)* [FGG+04]. Globus can be combined with different other tools and applications to construct Grid-enabled systems [BAG00].

UNICORE

The UNICORE (UNIform Interface to COmputing REsources), another metacomputing middleware, is a European Grid Technology started in 1996 to establish a uniform access to distributed computing resources and at the time of writing it has progressed to version 6 [SBBR+10]. It has already been used for different projects requirement successfully [unia]. The architecture of the technology is three-layered with client, service and system layers. The client layer refers to the interface to users which ranges from graphical user interface to command line. The service layer is the middle one responsible for the job management and execution therefore, it is comprised of all services and components of the UNICORE service-oriented architecture. The system layer is considered as bottom layer and is responsible for individual resource management/batch system and operating system of Grid resources. There is no dedicated SLA management component, however, it provides basis to implement the whole life cycle of SLA. UniGrids is a follow-on project from GRIP (Grid Interoperability Project), aims to develop an SLA framework for resource management system and the interoperability of Globus and Unicore [unib]. The management of VOs is done through *UNICORE VO Service (UVOS)* which is a client-server system. UVOS can be used with different systems but it is primarily designed to support UNICORE [SBBR+10][unic].

Legion

Legion [GWTLT97] [CKKG99] [leg] is an object-oriented resource management system for a metacomputing environment. It is also based on the idea of connecting a range of hosts from workstations to massively parallel supercomputer in the form of Grid. Therefore, seamless access to remote resources on the Grid with security and autonomicity was intended. The modular approach of Legion allows to implement user-level scheduling policies and customize system behaviour. It also supports for the reservation of resources. Legion provides simple schedulers that can be outperformed by specialized scheduling algorithm [CKKG99]. The specific scheduling scheme can be devised to schedule an application to meet the desired level of performance. An effort of implementation of adaptive scheduling scheme based on an established methodology called *AppLeS (Application Level Scheduling)* [BWF+96] for *Magnetohydrodynamics Application* is presented in [DOB+00] for Legion system. Legion supports process migration and check-pointing in case of failure to resume the application on different available resource.

Legion can be compared with Globus in a way of resulting Grid or metacomputing infrastructure. The main difference between the two is the realization of the model.

Globus presents *sum-of-service* architecture layered over pre-existing components whereas Legion has object-oriented programming model designed and developed with a goal of *whole-cloth* [CKKG99].

gLite

Enabling Grid for E-sciencE project (EGEE) is Europes flagship Research Infrastructures Grid project for the domains like Astronomy, Biomedicine, Computational Chemistry, Earth Sciences, Financial Simulations, and High Energy Physics [LHP+04] [LGF+06] [gli]. An extended middleware is developed by combining the components developed in various related projects, in particular Condor and Globus to provide user with high level services for scheduling and running computational jobs, data manipulation and consistent security infrastructure. The resultant middleware is called *gLite* that follows the Service Oriented Architecture approach which allows for an easy interaction with other Grid Services [LHP+04]. The architecture of gLite, thematically can be divided into five service groups i.e., access services, security services, information and monitoring services, job management services and data services [LGF+06]. Security services encompass the authentication, authorization, and auditing services which enable the identification of entities (users, systems, and services), allow or deny access to services and resources. Information and monitoring services provide a mechanism to publish and consume information and to use it for monitoring purposes. The job management/execution service are mainly related to the computing element, the workload management, accounting, job provenance, and package manager services (or simply dynamic deployment of application software). The three main services that relate to data and file access are: Storage Element, File & Replica Catalog Services and Data Management. In all of the data management services,the granularity of the data is on the file level which are generic enough to be extended to other levels of granularity. The gLite middleware provides a WS-Agreement based agreement service for the possible interaction with resource reservation service.

XtremWeb

The XtremWeb [FGNC01] is an open Source platform designed to build Grid computing environment. The project started with the concept of experimental global computing platform to study the issues related to parallel architecture. The general architecture of XtremWeb can be represented by three components i.e. *Client, Coordinator* and *Workers.* The workers contact the server to get jobs. In response, the server send a set of parameter and it may also send an application if the application is not already stored in the workers. When the workers have finished to compute their job, they contact the Coordinator to send the results [xtra]. It assumes that computing resources can be spread over the Internet. Furthermore, XtremWeb can also be used to build centralized Peer-to-Peer Systems. Conceptually, it relies on a pull model in which workers pull tasks from the server whereas in general, other models rely on a push model in which the Matchmaker selects the best machine to run the job and push the job on it [xtra].

XtremWeb-CH

Originally, XtremWeb-CH (XWCH) [xtrb] [AB05] was an upgraded version of the XtremWeb. But to obtain a reliable and efficient system the software's architecture redesigned completely and new modules added to develop a middleware that can easily deploy and execute parallel and distributed applications on a public-resource computing infrastructure. It aims at building an effective Peer-To-Peer system for CPU high performance applications.

Condor

Condor [LLM88], a workstations based distributed computing environment tool was developed at University of Wisconsin in 1983. The idea behind condor is that most of the time the connected workstations remain idle and there should be a way to identify such idle workstations and schedule a job in background on those workstations and as soon as the owner of any workstation resumes its activity the background job should be stopped and migrated to somewhere else. This will not only increase the utilization of under utilized resources but also make the system flexible to discover resources and keep the resource autonomicity. The uniqueness of condor is the central coordinator which merely assigns capacity to workstations which they use to schedule their own jobs and each workstation keeps the state information of its own jobs and has the responsibility of scheduling them with the relative priority. This enables the system to recover from any failed state if the central coordinator fails. Only the allocation of new capacity to requesting user is affected in case of failure of central coordinator as local schedulers of the workstations continue to perform their work. The *check pointing* facility enables the system to save the state of executing job and allows the system to migrate the job to another workstation if the user of the workstation resumes its activity. Classified Advertisement language (ClassAd) MatchMaking tool enables users to specify which resource to allocate.

Condor in its original form was not suitable for interactive jobs and parallel jobs of heavy-weight type. The reason is, a thread of a job can discontinue its execution any time when a user resumes its activity on the workstation therefore in case of an interactive job the executing job will have to wait for re-scheduling. Also, for the parallel heavy-weight job there is no mechanism which guarantees the execution of all threads of a parallel job synchronously and on the resumption of any user activity all the related instances of the parallel job needs to be check-pointed and then wait for the reschedule until the required number of resources are available. The provision of deadline based scheduling is not there in its original form.

Condor-G [FTL$^+$02] is considered as the marriage of Condor and Globus projects [TTL05]. It combines the inter-domain resource management protocols of the Globus Toolkit and the intra-domain resource management methods of Condor which results in a powerful tool for managing a variety of parallel computations in Grid environments. The merger of the two technologies enables the system to discover and acquire appropriate resources, mapping of jobs to resources, monitoring and managing execution of the jobs and detecting and responding to the failure. Condor-G keeps the same motto of leaving the owner of resource in control regardless the pay off and is the well defined batch computing system for high-throughput computing and opportunistic computing [TTL05].

LoadLeveler

LoadLeveler [SCZL96] [ibm] is a modified and commercial version of condor developed by IBM. It is a distributed, network-based, job scheduling program for massively parallel machines (MPPs) particularly for IBM SP [SCZL96]. Furthermore, it can schedule serial and parallel jobs. It also supports failure and recovery mechanism to some extent. It improved the system performance, turnaround time and equitable resource distribution for all users.

Nimrod

Nimrod [ASGH95], inspired by condor [LLM88], is a tool for managing execution of parameter sweep applications on distributed workstations. The parametric simulation studies deal with a calculation on a range of different simulations of some program. Each simulation consist of independent search point, so it is possible to run them concurrently on separate machine as light-weight threads. It takes the cross product of the user submitted parameters to generate set of simulations for a job. After this, Nimrod manages the distribution to machines and organises the aggregation of the results. It does not assume a shared file system therefore it copies files between systems before and after the execution of a program.

Nimrod in its original form presented in [ASGH95] has the following demerits. It assumed the fixed infrastructure of the system and hence provided no automatic device discovery mechanism. User cannot specify the deadline time by which the results should be submitted back to the user. As it uses RPC to call the sub-processes, therefore it is not possible to communicate master process while its sub-processes are in progress. However, there is a possible way by which RPC can be replaced with mechanism like parallel virtual machine (PVM) [PL95].

One of the most important issues of Nimrod to overcome is its inflexibility towards resource discovery which is why Nimrod was not suitable for Grid as it cannot acquire the unlimited computing power for the parametric modelling automatically. Such shortcomings addressed by Nimrod/G [BAG00] which was implemented by leveraging existing Globus toolkit [FK96]. This enabled the Nimrod/G scheduling from local to global computational grid allowing it to discover resources, processing of jobs within deadline time and monitoring of experiments from different machines and not just from the machine using which the job was submitted. Nimrod/G supports deadline and budget based scheduling. With the recent development, Nimrod now contains tools that perform a complete parameter sweep across all possible combinations (Nimrod/G) [BAG00], or over a subset selected by experimental design techniques (Nimrod/E) [PDA$^+$08], or search using non-linear optimization algorithms (Nimrod/O) [ALPF01] or over a workflow on a Grid machine (Nimrod/K) [AEA08].

Maui

The Maui [FRJ$^+$01] scheduler is known for its advance reservation, quality of service and increased utilization support through backfilling. It assigns each job an individual priority. The initial development of Maui scheduler did not have database and was introduced by

Molokini to store the users, their accounts and the state of the whole system as a backup solution in Maui Molokini Edition (MauiME) [mau]. The advanced version of Maui is called MOAB/Maui [moa].

MOAB/Maui

MOAB/Maui is also a commercial product from Adaptive Computing Enterprises Inc. for the management of cluster resources [moa]. It is a full-featured product like Oracle Grid Engine and it also supports QoS parameter in the form of thresholds. QoS thresholds allow an administrator to set usage metric thresholds that affect the overall performance. If for interested parameter the threshold is reached, then particular event takes place to maintain the quality of service of the jobs. The threshold actions from least to most impact on a system are priority, reservation access, fire trigger failure variables, reservation creation, pre-emption, and powering nodes on [moa].

PBS

Portable Batch System (PBS) [ope] is a POSIX compliant batch queuing and workload management software system originally developed at NASAs Ames research. It can cater the power of wide range of high power computer architectures from heterogeneous clusters of loosely coupled workstations, to massively parallel supercomputers. PBS includes several built-in schedulers but the default is the FIFO scheduler whose behaviour is to maximize the CPU utilization i.e. it loops through the queued job list and starts any job for which fits in the available resources. However, this effectively prevents large jobs from ever starting. It employs *starving job* mechanism to allow large jobs to start. But still there is a requirement for an alternative as starving job mechanism may not work all the time [SAL$^+$04]. Therefore, Maui scheduler needs to be combined with PBS [pbs]. The aggressive scheduling policies of Maui optimize the resource utilization and minimize the job response time.

Libra

Libra [SAL$^+$02] [SAL$^+$04] is developed as a plug-in scheduler for PBS which supports allocation of resources based on users' quality of service (QoS) requirements. The utility of job is determined through the *budget* and *deadline* parameters associated with jobs. Jobs are submitted with estimated runtime, deadline and budget using centralized cluster management system (CMS). Once admitted into the system jobs are then executed on the dedicated node. The job is time-sliced based on the runtime and deadline ratio. It is market-based economic driven service for the management of batch jobs. It assumes that the resources are homogeneous and jobs will not be of interactive type.

Load Sharing Facility (LSF)

Load Sharing Facility (LSF) software is a product of Platform Computing Corporation [lsf98]. It is available in two different editions, *LSF Enterprise Edition (LSF EE)* and *LSF Standard Edition (LSF SE)*. LSF SE is a basic tool for batch job processing system for distributed and heterogeneous environments. It includes necessary components to ensure optimal resource sharing and load balancing. The full featured version LSF EE includes distributed production job scheduler that integrates heterogeneous servers into a virtual mainframe or virtual supercomputer. Additionally, it maintains the resources ownership and cluster autonomy in shared multiple cluster environment. It manages parallel job execution in a production networked environment.

LSF MultiCluster [Xu01] is a product of Platform Computing Corporation [lsf98]. LSF MultiCluster acts a meta-scheduler, enabling cooperation among the computing sites running under LSF clusters to share the workload globally. Therefore, it is responsible for the inter-cluster information management and the job runtime control with workload sharing. It supports several scheduling strategies such as priority based, deadline constraint based, fair-share etc.

TORQUE

TORQUE [tor] is an enhanced version of openPBS [ope]. It is solely a resource manager primarily used in batch jobs using the scheduler of openPBS. It supports parallel and interactive jobs. The central master node is responsible for the management, polling and allocation of resources to jobs. It does not support backfilling. By default it does not support advance reservations but configuration can be made for quality of service (QoS) and advance reservations. Though, TORQUE is an enhanced version of openPBS, but because of enhancements it cannot read the job database of an OpenPBS server due to which both are incompatible to each other.

Oracle Grid Engine

Oracle Grid Engine (OGE) [ora10] is commercial product for workload management in cluster environment. It is the latest version of its predecessors Sun Grid Engine (SGE) which was the enhancement of CODINE [Gen01]. It has a variety of services on top of which a complete cluster can be built. It can handle the jobs of batch, parallel and interactive nature. It offers failure and recovery of jobs along with the advance reservation, backfilling and check-pointing features.

REXEC

REXEC is a decentralized remote execution facility for the cluster developed at University of California, Berkley [CC00]. It decouples the centralized node discovery and selection mechanism. It is achieved by deploying three types of entities. One running at cluster node as host, second running at user node as client program and the third is a replicated entity which provides node discovery and selection services running separately. It allows to run sequential as well as parallel and distributed programs. REXEC is responsible to make decision for data, computation and processes communication.

Tycoon

Tycoon [LRA+04] is a market based distributed resource allocation system based on proportional share. The price is calculated on the bases of budget, resource and duration parameters of the user submitted jobs. Based on the price, the share of resources is calculated for the user submitted jobs. This results in multiplexing of the resources and is considered as virtualized resource environment. Hence, Tycoon trades the networked resources of virtualized or sliced form.

AssessGrid/openCCS

AssessGrid [ass] [DGP+08] is a European Project whose first prototype was released in fall 2007 and the final one in February, 2009. It was designed for the next generation of Grid technologies that support SLAs. The prime feature of AssessGrid is its framework for the risk assessment and management for *end-user, broker* and *resource provider*. From provider's perspective, the risk assessment feature relates the failure risk of a resource to the penalty cost for the offered SLA. This also enables the end-user to know about the risk of an SLA violation for further decision. The Negotiation Manager is embedded in a Globus Toolkit which supports SLA negotiation parameters such as *number of nodes, job runtime, deadline time, security policy, migration policy, memory size* and *fault tolerance.*

A distributed resource management system called Computer Center Software (CCS or openCCS) [RRK94] is used in AssessGrid. It is an open source distributed software package and is responsible for user authorization, accounting, serving interactive and batch jobs, and scheduling. It also allows to make a reservation on the resources for future use.

ASKALON

ASKALON [ask] is a Grid scientific application development environment as workflows. The project is developed at University of Innsbruck using Globus toolkit. Its development architecture is of service oriented type with the support of SLAs. Therefore, Grid resources e.g., CPUs, can be reserved for a specified time interval as part of SLA.

Community Scheduler Framework (CSF)

The Community Scheduler Framework (CSF) [csf] is an open source framework for implementing a grid meta-scheduler with the support of WS-Agreement specification. It's an add-on to the Globus Toolkit and was developed by Platform Computing and Jilin University, China. A community scheduler serves each VO and is responsible of matching the requirements of a grid application with the available resources without violating the conditions of VO's resource managers. Users submit their grid applications to the community scheduler, which in turn matches the resource requirements of the jobs with the underlying physical resources through interaction with local resource managers. The local resource manager becomes a service provider, and a community scheduler consumes services based on SLAs [csf03]. It includes two basic scheduling algorithms i.e. Round-robin and reservation based algorithm. Using advance reservation users can make reservation of resources for a specified time on specified resource. The latest version called CSF4 takes charge of global resource management and job scheduling. It acts as an intermediary between a user community and local resources by providing a single point of task management and global policy enforcement [XZS+06].

VIOLA

The project VIOLA (Vertically Integrated Optical Testbed for Large Applications in DFN) started in June 2004 and ended in April 2007. It is an integrated testbed for applications and advanced network services built with the goals of testing different signalling mechanisms for the development of a user-driven dynamic bandwidth allocation. The VIOLA MetaScheduling Service (MSS) [WWZ06] is utilised to co-allocate computational and network resources as well in the Grid. It is responsible for negotiation for resources' reservation and allocation with the local scheduling systems through adapters using WS-Agreement. The negotiation involves potential start time, reservation of network and reservation of computational resources.

GridWay

The GridWay Meta-scheduler enables large-scale, reliable and efficient sharing of computing resources like clusters, supercomputers, stand-alone servers. It provides a single point of access to all resources in an organization, from in-house systems to Grid infrastructures and Cloud providers [gri]. The automatic negotiation with external providers is done by the plug-in which acts as a daemon to perform and manage SLAs. It periodically monitors the internal state of GridWay and SLA situation. On the detection of scarce resource situation it automatically begins the negotiation process. In addition, it is also responsible for adding new resources to GridWay after an agreement [BMG+09]. GridWay can dynamically access Cloud resources as well as can be used on main production Grid infrastructure.

Multi-level Scheduler (MLS)

In [PBC+11], a two-level scheduler has been proposed. At the top of the hierarchy the meta-scheduler classifies the incoming jobs to balance the load between the computing sites whereas, at the second level i.e., at cluster level, the local scheduler is responsible to schedule the jobs to exploit the backfilling mechanism. It schedules the jobs that do not require a co-allocation mechanism. The heuristics are applied at both the level to calculate the priority of the jobs. At meta-scheduler level the job classification is done to balance the load between the different local scheduler which is done using three different heuristics that are based on *deadline, licenses* and *user*. At local scheduler level the priority of a job is calculated using four different heuristics proposed in [CBP+07] that are based on *aging, deadline, licenses* and *wait minimization*.

Mutual Agreement Protocol (MAP)

Balakrishnan et. all in [BTSRB08], proposed and evaluated SLA based resource negotiation for the scheduling of resources in Grid environment. Using percentage deviation coefficient, the deviation between the requested resource and available resources is calculated based on which SLAs are negotiated. It showed that using the proposed deviation scheme the resource ordering reduced the number of negotiations that reduced the negotiation time and success rate as well as increased the throughput of the system.

The summarized characteristics of all of the above discussed metacomputing solutions are presented in Table 2.1, 2.2, 2.3, 2.4 and 2.5 based on the taxonomy discussed in Section 2.2.

Table 2.1: Scheduling Domain & Application Model Perspective

	Meta-scheduler	Independent Parallel Jobs
Globus	$\surd$	-
UNICORE	$\surd$	-
gLite	$\surd$	-
Legion	$\surd$	-
XtremWeb	$\surd$	-
XtremWeb-CH	$\surd$	-
Nimrod G/K/O/E	$\surd$	Nimrod K
Condor-G	$\surd$	$\surd$
Tycoon	$\surd$	$\surd$
LSF Multicluster	$\surd$	$\surd$
AssessGrid/OpenCCS	$\surd$	$\surd$
ASKALON/SLASKALON	$\surd$	$\surd$
CSF4	$\surd$	$\surd$
VIOLA	$\surd$	$\surd$
GridWay	$\surd$	$\surd$
LoadLeveler	$\times$	$\surd$
PBS/PBS Pro	$\times$	$\surd$
Libra/LibraSLA	$\times$	$\surd$
LSF Multicluster	$\surd$	$\surd$
TORQUE	$\times$	$\surd$
OGE	$\times$	$\surd$
REXEC	$\surd$	$\surd$
MAUI/MAUI ME	$\surd$	$\surd$
MOAB/MAUI	$\surd$	$\surd$
SLA BSH	$\times$	$\surd$
MLS	$\surd$	$\times$
MAP	$\surd$	$\surd$

Table 2.2: Architectural Model Perspective

	Centralized Management Control	Homogenous Resource Composition	Time-Shared	Flexiblie Scheduling System
Globus	-	×	-	-
UNICORE	-	×	-	-
gLite	-	×	-	-
Legion	-	×	-	-
XtremWeb	×	×	-	-
XtremWeb-CH	×	×	-	-
Nimrod G/K/O/E	×	×	×	√
Condor-G	×	×	√	√
Tycoon	×	×	√	×
LSF Multicluster	×	×	×	√
AssessGrid/OpenCCS	×	×	×	√
ASKALON/SLASKALON	×	×	Space/Time	√
CSF4	√	×	√	√
VIOLA	√	×	×	√
GridWay	√	×	-	√
LoadLeveler	√	×	√	√
PBS/PBS Pro	√	×	√	√
Libra/LibraSLA	√	√	√	√
LSF Multicluster	×	×	-	-
TORQUE	√	×	-	-
OGE	√	×	×	√
REXEC	×	×	×	√
MAUI/MAUI ME	√	×	Time/Space	Policy Dependent
MOAB/MAUI	√	×	Time/Space	Policy Dependent
SLA BSH	×	×	×	√
MLS	×	×	×	√
MAP	√	×	×	×

Table 2.3: Resource Access Perspective

	Market Model	Benefit Focus
Globus	-	-
UNICORE	-	-
gLite	-	-
Legion	-	-
XtremWeb	-	-
XtremWeb-CH	-	System
Nimrod G/K/O/E	Commodity/Contract-Net	User
Condor-G	-	System
Tycoon	Auction	User
LSF Multicluster	Proportional Share	System
AssessGrid/OpenCCS	Commodity	User
ASKALON/SLASKALON	-	User
CSF4	-	System
VIOLA	-	System
GridWay	Commodity	User
LoadLeveler	-	System
PBS/PBS Pro	Proportional Share	System
Libra/LibraSLA	Proportional Share	System
LSF Multicluster	-	System
TORQUE	-	System
OGE	-	System
REXEC	Commodity/Proportional Share	System
MAUI/MAUI ME	-	Policy Dependent
MOAB/MAUI	-	Policy Dependent
SLA BSH	Commodity	User
MLS	Commodity	System-User
MAP	-	System

Table 2.4: Quality of Service (QoS) Perspective

	Attributes	Optimization based Specification	Guarantee
Globus	-	-	-
UNICORE	-	-	-
gLite	-	-	-
Legion	-	-	-
XtremWeb	-	-	-
XtremWeb-CH	-	-	-
Nimrod G/K/O/E	Time, Cost	$\surd$	Rigid
Condor-G	$\times$	-	-
Tycoon	Time, Cost	$\surd$	Rigid
LSF Multicluster	Time	$\surd$	Flexible
AssessGrid/OpenCCS	Time, Cost	$\surd$	Flexible
ASKALON/SLASKALON	Time, Cost	$\surd$	Rigid
CSF4	Time	$\surd$	Rigid
VIOLA	Time	$\surd$	Rigid
GridWay	$\times$	-	-
LoadLeveler	$\times$	-	-
PBS/PBS Pro	Time	$\surd$	Rigid
Libra/LibraSLA	Time, Cost	$\surd$	Rigid/Flexible
LSF Multicluster	Time	$\surd$	Flexible
TORQUE	$\times$	-	-
OGE	$\times$	-	-
REXEC	$\times$	-	-
MAUI/MAUI ME	Time	$\surd$	Flexible
MOAB/MAUI	Time	$\surd$	Flexible
SLA BSH	Time, Cost	$\surd$	Flexible
MLS	Time, Cost	$\surd$	Flexible
MAP	CPU	$\times$	-

Table 2.5: Scheduling Perspective

	Full-ahead Planning	Advance Reservation	Dynamic Schedule Updates
Globus	-	√	-
UNICORE	-	√	-
gLite	-	√	-
Legion	-	√	-
XtremWeb	-	√	-
XtremWeb-CH	-	√	-
Nimrod G/K/O/E	×	Nimrod K	√
Condor-G	×	×	√
Tycoon	√	√	×
LSF Multicluster	√	√	×
AssessGrid/OpenCCS	√	√	√
ASKALON/SLASKALON	√	√	√
CSF4	√	√	√
VIOLA	√	√	√
GridWay	√	√	√
LoadLeveler	√	×	√
PBS/PBS Pro	√	√	×
Libra/LibraSLA	√	√	×
LSF Multicluster	√	√	×
TORQUE	√	×	-
OGE	√	√	√
REXEC	√	×	√
MAUI/MAUI ME	√	√	Policy Dependent
MOAB/MAUI	√	√	Policy Dependent
SLA BSH	√	√	√
MLS	√	×	√
MAP	√	√	-

2.4 Summary

The review of service level agreement (SLA) and its' structure presented in this chapter. It has been discussed that jobs are defined using job service description language (JSDL) but in order to create communication between jobs, workflow language is required. Similarly, WS-Agreement is required for appropriate negotiation and agreement protocol to optimally schedule job onto resources.

Further, taxonomies defined, based on which number of discussed metacomputing solutions classified. It has been noted that in all the discussed works, different approaches used for the SLA based job management but have one feature in common that is, every job has a separate SLA. The details of the presentation, SLA terms and level of implementation could be different and this may be the reason why different system performances observed under different schemes.

It has also been mentioned that number of metacomputing solutions have been proposed to solve particular problem with the strong assumptions for the support of management of SLAs to guarantee the quality of service (QoS). Further, from the discussed metacomputing enabled middleware (such as for Grid), except for gLite, no middleware has a direct support for the management of SLAs at this level. Majority of the work related to either SLA based service provision or to SLA based scheduling, have been done in the context of Web based services [YS06]. Recent works such as [PBC+11] [BTSRB08] [YB06a], show the importance and need of the SLA based computing services. Most of the research has been done towards the modelling, negotiation and monitoring of SLAs and little work has been done towards SLA based job scheduling.

Chapter 3

Research Methodology

This chapter discusses the methodology used to carry out experiments presented throughout in this thesis. Starting the discussion with the use case model together with the computing architecture, then discusses the designed simulator together with the scheduling heuristics and simulator execution scheme. Then, a brief description of SLA-based workload generator, assumptions made to perform the experiments, methods to compute and compare the results, and limitations of the designed simulator are presented in this chapter.

3.1 Use Case Model

The use case model is the grid architecture which is discussed in [SY08] and presented in Figure 3.1, representing the fundamental interactions that may arise with the use of SLAs in the grid environment for a job submitted for execution to high-performance computing resources. The presented architecture is comprised of three main entities i.e., the Client, the Provider and the Agreement (SLA) between the client and the provider. The client may be considered as the user or broker that negotiate the usage of resources with resource provider. The resources provider or simply resource owner managing its resources with its own local scheduler, containing the information about the level of service agreed between the two parties. The SLAs may be mapped to one or more SLAs, before being negotiated. To distinguish between the two types of SLAs, the term meta-SLA and sub-SLA has been used. More than one Sub-SLAs may be derived from a meta-SLA which are then negotiated and agreed with the local resources and their schedulers. For example, for a workflow application, the user may want to set constraints for the workflow as a whole (i.e., meta-SLA) and the broker may have to translate it to specific SLAs (i.e., sub-SLAs) for individual tasks of a workflow [ZS07].

To simulate the behavior of a client the load generator which is briefly detailed in Section 3.2 is used to generate the SLA based jobs. The basic architecture of the simulating environment is depicted in Figure 3.2. The first part of the architecture belongs to the generation of workloads. To carry out the experiments, the workloads are generated first using the load generator discussed in [SY08] [YS06]. Every single generated workload consist of 1000 SLAs and each SLA is of the form of quadruple i.e., {*Earliest Start Time*

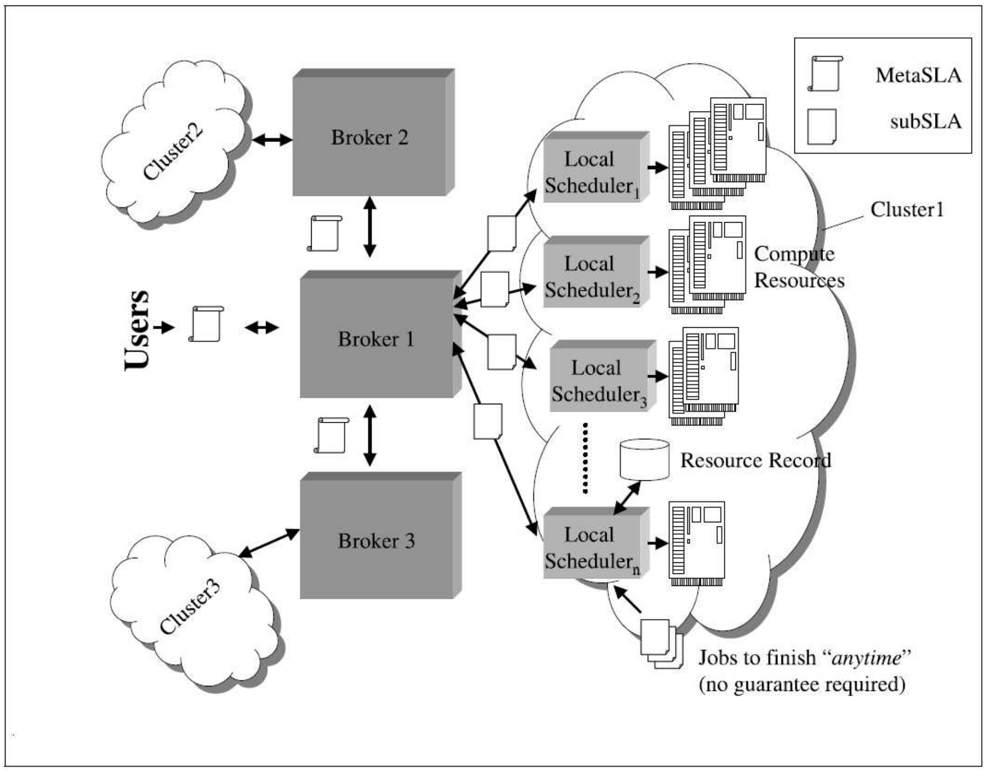


Figure 3.1: An architectural overview for SLA based job scheduling [SY08]

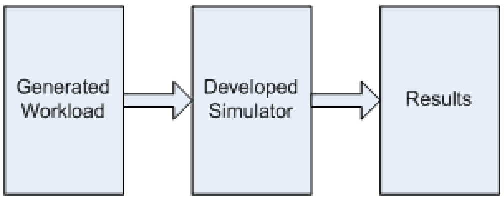


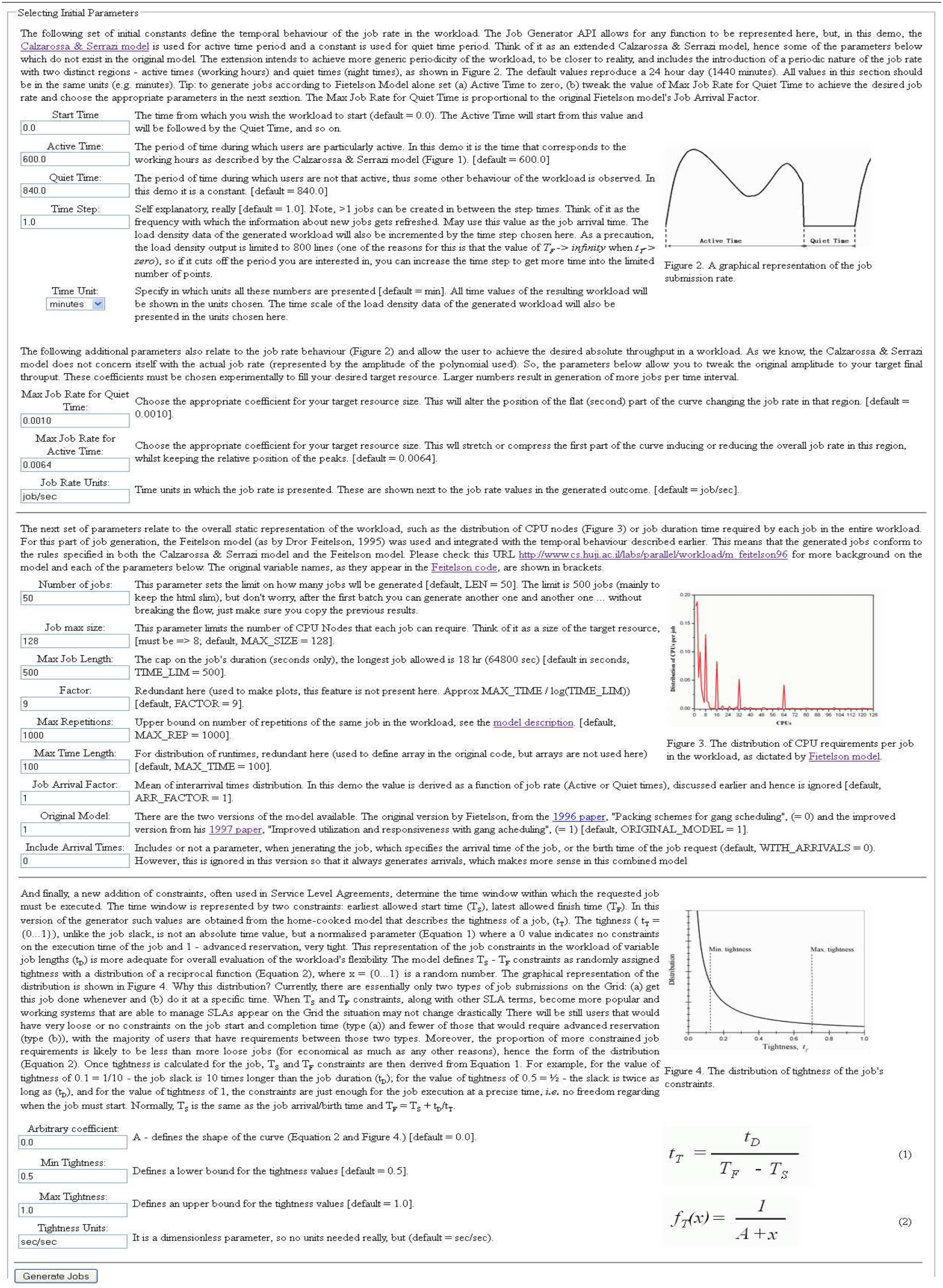
Figure 3.2: Basic Architecture

(T_s), runtime (t_D), Latest Finish Time (T_f), Number of CPUs required (N_{cpu})}. The generated workloads are then submitted to the designed simulator which then schedule the SLAs. The details of the simulator are given in Section 3.3. The results are observed on the basis of successfully served SLAs i.e., the SLAs that are served within their deadline time. The process of negotiation is not part of this study but the main goal is to address the scheduling aspect under agreements (SLAs).

3.2 User Behaviour Model

To capture the user behaviour with respect to the construction of workload for the system evaluation, two renowned approaches are used. One is based on the traces of real workload (e.g., workload archives [par]) and the other is based on the mathematical models to generate a synthetic workload (e.g., [JPF+97]). Workload issues span all methods of parallel job scheduling evaluation [FF05], therefore choice of workload generation/construction and selection to emulate user behaviour is one of the major task. Especially, with respect to more widely used paradigm [YS06] [YB06a], which uses Service Level Agreement (SLA), as a contract to specify the terms and guarantees bound to a specific job or user request, and there is no well developed method to construct SLA based workload. Therefore, to avoid workload related issues as well as to have workloads comprised of Service Level Agreements (SLAs), the workload generator discussed in [SY08] [YS06] is used throughout in this thesis to mimic the user behaviour. The load generator generates number of requests in the form of SLAs based on the set up variables, the generated requests (or SLAs) consist of quadruple {*Earliest Start Time (T_s), runtime (t_D), Latest Finish Time (T_f), Number of CPUs required (N_{cpu})*} which defines the Service Level Agreement(SLA) for a job. Number of different workloads can be generated using [SY08] [YS06] by setting up the load generator values. To generate a workload, the interface depicted in Figure 3.3 is used all the time and the variables (discuss below) are required to set up to the desired values. The variables of interest with their default values are mentioned below.

- Maximum job rate for Quiet Time: 0.0010 jobs/sec
- Maximum job rate for Active Time: 0.0064 jobs/sec
- Job Max Size: 128
- Max Job Length: 500 sec
- Maximum Repetitions: 1000
- Min Tightness: 0.5
- Max Tightness: 1



Selecting Initial Parameters

The following set of initial constants define the temporal behaviour of the job rate in the workload. The Job Generator API allows for any function to be represented here, but, in this demo, the Calzarossa & Serrazi model is used for active time period and a constant is used for quiet time period. Think of it as an extended Calzarossa & Serrazi model, hence some of the parameters below which do not exist in the original model. The extension intends to achieve more generic periodicity of the workload, to be closer to reality, and includes the introduction of a periodic nature of the job rate with two distinct regions - active times (working hours) and quiet times (night times), as shown in Figure 2. The default values reproduce a 24 hour day (1440 minutes). All values in this section should be in the same units (e.g. minutes). Tip: to generate jobs according to Fietelson Model alone set (a) Active Time to zero, (b) tweak the value of Max Job Rate for Quiet Time to achieve the desired job rate and choose the appropriate parameters in the next sextion. The Max Job Rate for Quiet Time is proportional to the original Fietelson model's Job Arrival Factor.

Start Time	0.0	The time from which you wish the workload to start (default = 0.0). The Active Time will start from this value and will be followed by the Quiet Time, and so on.
Active Time:	600.0	The period of time during which users are particularly active. In this demo it is the time that corresponds to the working hours as described by the Calzarossa & Serrazi model (Figure 1). [default = 600.0]
Quiet Time:	840.0	The period of time during which users are not that active, thus some other behaviour of the workload is observed. In this demo it is a constant. [default = 840.0]
Time Step:	1.0	Self explanatory, really [default = 1.0]. Note, >1 jobs can be created in between the step times. Think of it as the frequency with which the information about new jobs gets refreshed. May use this value as the job arrival time. The load density data of the generated workload will also be incremented by the time step chosen here. As a precaution, the load density output is limited to 800 lines (one of the reasons for this is that the value of T_F -> *infinity* when t_T-> *zero*), so if it cuts off the period you are interested in, you can increase the time step to get more time into the limited number of points.
Time Unit:	minutes	Specify in which units all these numbers are presented [default = min]. All time values of the resulting workload will be shown in the units chosen. The time scale of the load density data of the generated workload will also be presented in the units chosen here.

Figure 2. A graphical representation of the job submission rate.

The following additional parameters also relate to the job rate behaviour (Figure 2) and allow the user to achieve the desired absolute throughput in a workload. As we know, the Calzarossa & Serrazi model does not concern itself with the actual job rate (represented by the amplitude of the polynomial used). So, the parameters below allow you to tweak the original amplitude to your target final throuput. These coefficients must be chosen experimentally to fill your desired target resource. Larger numbers result in generation of more jobs per time interval.

Max Job Rate for Quiet Time:	0.0010	Choose the appropriate coefficient for your target resource size. This will alter the position of the flat (second) part of the curve changing the job rate in that region. [default = 0.0010].
Max Job Rate for Active Time:	0.0064	Choose the appropriate coefficient for your target resource size. This wll stretch or compress the first part of the curve inducing or reducing the overall job rate in this region, whilst keeping the relative position of the peaks. [default = 0.0064].
Job Rate Units:	job/sec	Time units in which the job rate is presented. These are shown next to the job rate values in the generated outcome. [default = job/sec].

The next set of parameters relate to the overall static representation of the workload, such as the distribution of CPU nodes (Figure 3) or job duration time required by each job in the entire workload. For this part of job generation, the Feitelson model (as by Dror Feitelson, 1995) was used and integrated with the temporal behaviour described earlier. This means that the generated jobs conform to the rules specified in both the Calzarossa & Serrazi model and the Feitelson model. Please check this URL http://www.cs.huji.ac.il/labs/parallel/workload/m_feitelson96 for more background on the model and each of the parameters below. The original variable names, as they appear in the Feitelson code, are shown in brackets.

Number of jobs:	50	This parameter sets the limit on how many jobs wll be generated [default, LEN = 50]. The limit is 500 jobs (mainly to keep the html slim), but don't worry, after the first batch you can generate another one and another one ... without breaking the flow, just make sure you copy the previous results.
Job max size:	128	This parameter limits the number of CPU Nodes that each job can require. Think of it as a size of the target resource, [must be => 8; default, MAX_SIZE = 128].
Max Job Length:	500	The cap on the job's duration (seconds only), the longest job allowed is 18 hr (64800 sec) [default in seconds, TIME_LIM = 500].
Factor:	9	Redundant here (used to make plots, this feature is not present here. Approx MAX_TIME / log(TIME_LIM)) [default, FACTOR = 9].
Max Repetitions:	1000	Upper bound on number of repetitions of the same job in the workload, see the model description. [default, MAX_REP = 1000].
Max Time Length:	100	For distribution of runtimes, redundant here (used to define array in the original code, but arrays are not used here) [default, MAX_TIME = 100].
Job Arrival Factor:	1	Mean of interarrival times distribution. In this demo the value is derived as a function of job rate (Active or Quiet times), discussed earlier and hence is ignored [default, ARR_FACTOR = 1].
Original Model:	1	There are the two versions of the model available. The original version by Fietelson, from the 1996 paper, "Packing schemes for gang scheduling", (= 0) and the improved version from his 1997 paper, "Improved utilization and responsiveness with gang acheduling", (= 1) [default, ORIGINAL_MODEL = 1].
Include Arrival Times:	0	Includes or not a parameter, when jenerating the job, which specifies the arrival time of the job, or the birth time of the job request (default, WITH_ARRIVALS = 0). However, this is ignored in this version so that it always generates arrivals, which makes more sense in this combined model

Figure 3. The distribution of CPU requirements per job in the workload, as dictated by Fietelson model.

And finally, a new addition of constraints, often used in Service Level Agreements, determine the time window within which the requested job must be executed. The time window is represented by two constraints: earliest allowed start time (T_S), latest allowed finish time (T_F). In this version of the generator such values are obtained from the home-cooked model that describes the tightness of a job, (t_T). The tighness (t_T = {0...1}), unlike the job slack, is not an absolute time value, but a normalised parameter (Equation 1) where a 0 value indicates no constraints on the execution time of the job and 1 - advanced reservation, very tight. This representation of the job constraints in the workload of variable job lengths (t_D) is more adequate for overall evaluation of the workload's flexibility. The model defines T_S - T_F constraints as randomly assigned tightness with a distribution of a reciprocal function (Equation 2), where x = {0...1} is a random number. The graphical representation of the distribution is shown in Figure 4. Why this distribution? Currently, there are essentially only two types of job submissions on the Grid: (a) get this job done whenever and (b) do it at a specific time. When T_S and T_F constraints, along with other SLA terms, become more popular and working systems that are able to manage SLAs appear on the Grid the situation may not change drastically. There will be still users that would have very loose or no constraints on the job start and completion time (type (a)) and fewer of those that would require advanced reservation (type (b)), with the majority of users that have requirements between those two types. Moreover, the proportion of more constrained job requirements is likely to be less than more loose jobs (for economical as much as any other reasons), hence the form of the distribution (Equation 2). Once tightness is calculated for the job, T_S and T_F constraints are then derived from Equation 1. For example, for the value of tightness of 0.1 = 1/10 - the job slack is 10 times longer than the job duration (t_D); for the value of tightness of 0.5 = ½ - the slack is twice as long as (t_D), and for the value of tightness of 1, the constraints are just enough for the job execution at a precise time, *i.e.* no freedom regarding when the job must start. Normally, T_S is the same as the job arrival/birth time and $T_F = T_S + t_D/t_T$.

Figure 4. The distribution of tightness of the job's constraints.

Arbitrary coefficient:	0.0	A - defines the shape of the curve (Equation 2 and Figure 4.) [default = 0.0].
Min Tightness:	0.5	Defines a lower bound for the tightness values [default = 0.5].
Max Tightness:	1.0	Defines an upper bound for the tightness values [default = 1.0].
Tightness Units:	sec/sec	It is a dimensionless parameter, so no units needed really, but (default = sec/sec).

$$t_T = \frac{t_D}{T_F - T_S} \quad (1)$$

$$f_T(x) = \frac{1}{A + x} \quad (2)$$

Figure 3.3: Screen Shot of Load Generator Interface [SY08] [YS06]

Arrival Rate (AR)

In all of the experiments presented in this thesis both *maximum job rate for quiet time* and *maximum job rate for active time* are treated as one simple variable called *arrival rate (AR)*. Setting arrival rate (AR) to desired values allows the generator to generate workload of SLAs with specified maximum possible value of arrival rate for the SLAs. The jobs/SLAs are generated per second.

Job Max Size

Job Max Size is maximum allowable number of CPUs that an SLA can have within a workload. It is represented by N_{cpu} i.e., the number of resources (CPUs). The generator accepts the value in a range from 8 to 128 i.e., maximum value that the generator can accept is 128 and minimum is 8. If this variable is set to 8 then it generates number of SLAs with different values ranges between 1 and 8 but not greater than 8 or less than 1.

Max Job Length

It is the maximum allowable runtime or execution time an SLA can have within a workload. The default value is 500 seconds whereas the maximum allowable value is 64,800 seconds. If this value is set to 500 seconds than it means that all the SLAs within the generated workload will have a runtime value greater than 0 and less than equal to 500.

Maximum Repetitions

It specifies how many number of times the same required level of service i.e., SLA of same characteristics can appear in the workload. The least possible value is 1 and the default is 1000. Whereas, the value of 50 is used for the generation of workloads for the experiments, which means at most one type of request may be generated not more than 50 times within a workload.

Min & Max Tightness

Tightness values deals with laxity of an SLA or the flexible time between the runtime and deadline of a job (i.e., deadline time - runtime - start Time). The tightness ranges between 0.1 and 1. If it is 1 for both minimum and maximum tightness variable, then it means no flexibility will be available in the deadlines of the generated jobs. But if the minimum tightness is set to 0.1 then the maximum flexibility or laxity for a job will be 10 times longer than the job's runtime. Hence, a job can have maximum laxity of 10 times of its runtime.

Three different workloads are generated and presented in Table 3.1, 3.2 and 3.3, as an example set. Each workload is comprised of 50 jobs/SLAs. The last four columns of the generated workload is of interest and belongs to SLA parameters. The workload presented in Table 3.1, is generated using low arrival rate and high minimum tightness values for maximum jobsize value (i.e., CPUs) equals to 8. Table 3.2 represents a workload, generated using slightly higher arrival rate and low minimum tightness values for maximum jobsize value (i.e., CPUs) equals to 8. Whereas, Table 3.3 represents a workload, generated using high arrival rate and high minimum tightness values for maximum jobsize value (i.e., CPUs) equals to 8.

Once a workload is generated by setting up above discussed variables to the desired values, it is then submitted to the developed simulator to observe the performance of the system under the submitted workload against different scheduling heuristics, discuss in the following sections.

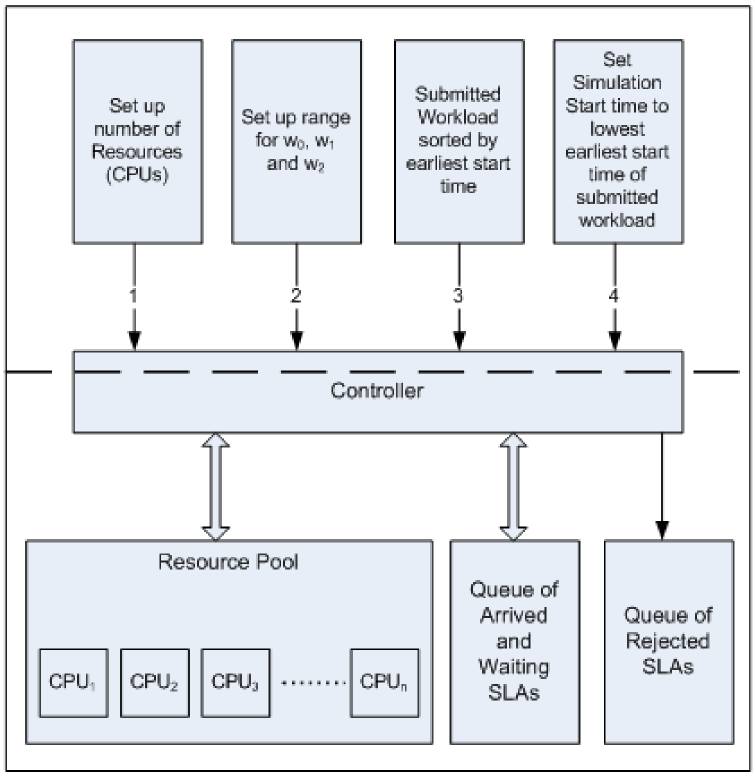

Set up number of Resources (CPUs)
Set up range for w_0, w_1 and w_2
Submitted Workload sorted by earliest start time
Set Simulation Start time to lowest earliest start time of submitted workload
1
2
3
4
Controller
Resource Pool
CPU_1
CPU_2
CPU_3
CPU_n
Queue of Arrived and Waiting SLAs
Queue of Rejected SLAs

Figure 3.4: Architecture

3.3 Simulator Architecture

A simulator which simulates the environment of number of homogeneous processing nodes (S_{cpu}) onto which the SLAs/jobs are scheduled, is developed to behave as a resource provider. The simulator architecture with its pre-requisites to initialize the simulation is presented in Figure 3.4. The architecture is divided by a dashed line into two parts. The part above the dashed line is the required *initialization phase*. Once the initialization successfully completed then the *controller* takes care of the complete simulation i.e., the *execution phase* and play the core role to manage the simulation.

The *controller* is a central part of the system. The priority and feasibility to successfully schedule the SLAs within their deadline times is checked based on which the decision to accept or reject the SLA is taken by the controller. The controller is also responsible for the incrementing the simulator clock based on which the completion of the scheduled

Table 3.1: Example Workload 1

Time	Job Rate	Tightness	CPUcount	Job Duration	Earliest Start	Latest Finish
			(N_{cpu})	(t_D)	(T_s)	(T_f)
min	job/sec	sec/sec	CPU	min	min	min
0	7.75E-06	0.9320303	4	464.25	0	498.1
0	7.75E-06	0.9320303	5	2.55	2120.467	2123.2
2120	6.00E-05	0.8826659	2	294.817	2447.133	2781.133
2447	6.00E-05	0.8352322	2	80.817	2944.35	3041.117
2944	5.06E-05	0.76966196	2	80.817	3041.133	3137.9
3041	5.78E-05	0.92041695	2	80.817	3137.917	3234.683
3138	4.34E-05	0.8207264	6	39.483	3877.8	3929.1
3878	6.00E-05	0.9497668	6	39.483	3929.117	3980.417
3929	6.00E-05	0.93243194	6	39.483	3980.433	4031.733
3980	6.00E-05	0.772276	6	39.483	4031.75	4083.05
4032	6.00E-05	0.94153845	2	12.3	4590.6	4603.967
4591	4.12E-05	0.9180315	2	12.3	4603.983	4617.35
4604	3.90E-05	0.83265877	2	12.3	4617.367	4630.733
4617	3.68E-05	0.7843037	2	12.3	4630.75	4644.117
4631	3.46E-05	0.93029875	2	12.3	4644.133	4657.5
4644	3.27E-05	0.78561026	2	12.3	4657.517	4670.883
4658	3.09E-05	0.8534094	2	0.55	5344.1	5344.767
5344	6.00E-05	0.8901935	4	5.667	5813.6	5819.567
5814	4.65E-05	0.772078	5	0.1	5926.367	5926.467
5926	5.73E-05	0.8905563	5	0.1	5926.483	5926.583
5926	5.73E-05	0.8905563	5	0.1	5926.6	5926.7
5926	5.73E-05	0.8905563	2	274.4	6064.35	6419.667
6064	3.56E-05	0.81226707	1	1.267	6148.033	6149.383
6148	2.83E-05	0.9716657	1	1.267	6149.4	6150.75
6149	2.83E-05	0.7584099	1	1.267	6150.767	6152.117
6151	2.84E-05	0.77477026	2	1.383	8637.5	8639
8638	6.00E-05	0.8834114	2	1.383	8639.017	8640.517
8639	6.00E-05	0.9142524	2	1.383	8640.533	8642.033
8641	8.69E-06	0.89182884	2	0.417	8996.767	8997.267
8997	2.91E-05	0.7720572	2	0.417	8997.283	8997.783
8997	2.91E-05	0.7720572	2	0.417	8997.8	8998.3
8997	2.91E-05	0.7720572	2	0.417	8998.317	8998.817
8998	2.91E-05	0.90242213	2	0.417	8998.833	8999.333
8998	2.91E-05	0.90242213	2	0.417	8999.35	8999.85
8999	2.90E-05	0.8102572	8	18.9	9450.417	9474.517
9450	6.00E-05	0.87998533	8	18.9	9474.533	9498.633
9475	6.00E-05	0.80020756	8	18.9	9498.65	9522.75
9499	6.00E-05	0.8118692	8	18.9	9522.767	9546.867
9523	6.00E-05	0.9616384	2	11.367	9785.8	9798.567
9786	6.00E-05	0.9616924	2	11.367	9798.583	9811.35
9799	6.00E-05	0.9232251	2	11.367	9811.367	9824.133
9811	6.00E-05	0.769499	2	11.367	9824.15	9836.917
9824	6.00E-05	0.79458606	5	47.55	10308.783	10359.9
10309	4.83E-05	0.80483305	6	300.9	10995.05	11332.933
10995	6.00E-05	0.80431026	2	14.45	11134.65	11150.883
11135	6.00E-05	0.79478765	2	14.45	11150.9	11167.133
11151	6.00E-05	0.75111866	2	14.45	11167.15	11183.383
11167	6.00E-05	0.83872163	3	38.217	11663.217	11711.867
11663	5.93E-05	0.97400004	3	38.217	11711.883	11760.533
11712	5.40E-05	0.8561358	3	38.217	11760.55	11809.2

Table 3.2: Example Workload 2

Time	Job Rate	Tightness	CPUcount	Job Duration	Earliest Start	Latest Finish
			(N_{cpu})	(t_D)	(T_s)	(T_f)
min	job/sec	sec/sec	CPU	min	min	min
0	0.001291539	0.113286056	8	97.366	0	859.483
54	0.001291539	0.113286056	8	25.866	54.4	251.817
65	0.00775122	0.10544711	5	60.883	65.433	320.733
76	0.008500839	0.14868043	2	180.2	76.416	500.083
88	0.009075466	0.4601233	3	48.65	88.116	278.7
91	0.009523761	0.16146648	1	1.65	91.716	95.3
92	0.009635987	0.23926172	1	1.65	95.316	98.9
95	0.009708942	0.18579593	1	1.65	98.916	102.5
99	0.009792028	0.48454624	1	2.9	100.266	118.233
100	0.009810365	0.18591025	2	0.683	103.833	106.683
104	0.009874411	0.19649343	2	0.683	106.7	109.55
107	0.009913084	0.21528882	1	0.933	108.083	113.1
108	0.009924268	0.19364291	2	0.683	109.566	112.417
110	0.00994416	0.5022176	2	2.266	111.6	116.283
112	0.00996084	0.9322605	1	0.933	113.116	118.133
113	0.009968011	0.24925773	1	0.933	118.15	123.167
118	0.009992701	0.63123083	1	2.9	118.25	136.217
118	0.009992701	0.63123083	8	2.266	118.25	133.8
120	0.009999837	0.67765534	7	17.716	120.95	143.6
129	0.009983838	0.17874056	3	0.716	128.616	147.267
134	0.009962195	0.87954444	7	2.266	133.816	146.017
134	0.009962195	0.87954444	2	2.566	134.266	146.183
134	0.009962195	0.87954444	1	2.9	136.233	144.2
136	0.009948269	0.5618368	3	0.716	137.283	140.933
137	0.009940567	0.22734493	2	15.766	138.666	220.083
139	0.009923736	0.37824985	2	549.566	144.583	1238.867
145	0.009862506	0.29214022	8	2.266	146.033	158.233
146	0.009850833	0.18678027	2	2.566	146.2	158.117
146	0.009850833	0.18678027	4	13.383	147.266	161.617
147	0.009838764	0.3461909	2	4.466	152.766	170.683
153	0.009758504	0.13486448	8	7.766	157.883	170.183
158	0.009682207	0.44426894	2	2.566	158.133	170.05
158	0.009682207	0.44426894	4	866.016	158.266	1530.217
158	0.009682207	0.44426894	4	129.75	161.833	353.3
162	0.00961566	0.13364859	8	7.766	170.2	182.5
170	0.009469609	0.796531	2	4.466	170.7	188.617
170	0.009469609	0.796531	8	679.883	171.016	3974.767
171	0.009450251	0.90712166	2	0.816	174.85	175.783
175	0.009370597	0.47077608	4	345.233	175.733	568.25
175	0.009370597	0.47077608	2	0.816	175.8	176.733
175	0.009370597	0.47077608	2	0.816	176.75	177.683
177	0.009329504	0.9006066	2	17.783	176.85	197.067
177	0.009329504	0.9006066	3	92.516	180.95	345.617
181	0.009244963	0.44085342	8	7.766	182.516	194.817
183	0.00920158	0.14073174	2	2.3	184.05	194.167
184	0.009179624	0.74949074	2	4.466	188.633	206.55
189	0.009067343	0.28939685	8	29.616	188.65	266.95
189	0.009067343	0.28939685	2	2.3	194.183	204.3
194	0.008951189	0.77347785	2	17.783	197.083	217.3
197	0.008879792	0.14661181	3	0.1	199.183	199.533

Table 3.3: Example Workload 3

Time	Job Rate	Tightness	CPUcount	Job Duration	Earliest Start	Latest Finish
			(N_{cpu})	(t_D)	(T_s)	(T_f)
min	job/sec	sec/sec	CPU	min	min	min
0	0.11623851	0.8538849	5	0.767	0	0.9
0	0.11623851	0.8538849	3	0.283	0.133	0.467
0	0.11623851	0.8538849	3	0.117	0.35	0.483
0	0.11623851	0.8538849	3	0.283	0.483	0.817
0	0.11623851	0.8538849	4	22.083	0.7	26.567
0	0.11623851	0.8538849	3	0.283	0.833	1.167
0	0.11623851	0.8538849	1	0.933	0.9	2
0	0.11623851	0.8538849	2	13.717	1.133	17.2
1	0.13042246	0.8376796	8	14.167	1.533	18.117
1	0.13042246	0.8376796	3	1.15	1.617	2.967
1	0.13042246	0.8376796	4	0.267	1.983	2.3
1	0.13042246	0.8376796	4	0.267	2.317	2.633
2	0.14451288	0.955227	1	0.35	2.433	2.85
2	0.14451288	0.955227	4	3.55	2.55	6.783
2	0.14451288	0.955227	4	0.267	2.65	2.967
2	0.14451288	0.955227	3	1.15	2.983	4.333
2	0.14451288	0.955227	2	0.267	3.033	3.35
3	0.1585062	0.9736739	5	0.533	3.317	3.867
3	0.1585062	0.9736739	2	652.667	3.617	686.867
3	0.1585062	0.9736739	2	0.283	3.8	4.1
3	0.1585062	0.9736739	1	333.05	3.9	352.567
3	0.1585062	0.9736739	3	1.15	4.35	5.7
4	0.172399	0.8782608	4	770.217	4.4	795.45
4	0.172399	0.8782608	4	671.633	4.467	707.583
4	0.172399	0.8782608	2	1.3	4.7	6.033
4	0.172399	0.8782608	2	987	5	1018.683
5	0.18618803	0.86838925	2	203.967	5.183	214.667
5	0.18618803	0.86838925	4	49.8	5.767	56.917
5	0.18618803	0.86838925	4	4.017	6.083	10.65
6	0.19987014	0.82200396	2	0.1	6.15	6.267
6	0.19987014	0.82200396	4	792.567	6.25	908.683
6	0.19987014	0.82200396	2	0.1	6.283	6.4
6	0.19987014	0.82200396	2	0.1	6.417	6.533
6	0.19987014	0.82200396	3	2.433	6.5	9.267
6	0.19987014	0.82200396	2	69.6	6.7	86.85
6	0.19987014	0.82200396	8	484.733	6.883	565.083
6	0.19987014	0.82200396	1	7.767	7.717	16.667
8	0.22690178	0.8133747	8	193.517	7.9	243.317
8	0.22690178	0.8133747	2	0.333	7.983	8.383
8	0.22690178	0.8133747	4	0.483	8.217	8.8
8	0.22690178	0.8133747	5	0.633	8.283	9.05
8	0.22690178	0.8133747	2	0.333	8.4	8.8
8	0.22690178	0.8133747	3	0.917	8.433	9.55
8	0.22690178	0.8133747	4	10.733	8.65	21.7
8	0.22690178	0.8133747	2	0.333	8.817	9.217
8	0.22690178	0.8133747	4	54.25	9.1	75.1
9	0.24024574	0.9060782	2	0.333	9.233	9.633
9	0.24024574	0.9060782	3	2.433	9.283	12.05
9	0.24024574	0.9060782	4	0.4	9.383	9.867
9	0.24024574	0.9060782	2	0.333	9.65	10.05

SLAs and arrival of a new SLA is checked. The scheduling schemes to evaluate the SLA priorities as well as to schedule the SLAs, are also adopted from [YS06] which performs iterations to schedule an SLA. The problem of scheduling parallel jobs with critical service levels same as SLA is considered as NP-hard problem [Joh06] and in the work [YS06], the optimized scheduling decision for the maximum performance of the system is achieved by using the integrated heuristic. It is advised that the multiple parameters from the SLA can be added together by multiplying some weight factor w for the priority calculation of the SLAs. The best integrated heuristic of the work [YS06] is given in Equation 3.1, for the performance criterion of serving the highest number of SLAs. The H is the computed priority based on the minimum value of the heuristic given in Equation 3.1. For a set of SLAs or simply workload, the prioritization is done for each combination of defined w_1 and w_2 values iteratively, until all the defined w_1 and w_2 values are exhausted. Once all the values are exhuasted, that pair of w_1 and w_2 values is picked up which results in optimized performance. The performance criterion for this heuristic approach is the number of served SLAs. The computation is required to sweep through (or to exhaust) all the defined values of w_1 and w_2 one at a time. The construction and checking of different schedules using each combination of w_1 and w_2 values incurs computation cost. This type of evaluation of schedules is called parameter sweeping and as this evaluation incurs computation but the time of computation is dependent on the defined range of w parameters, number of SLA parameters (or w) involved, number of resources within the system and the number of SLAs to consider.

$$H = min(T_f + w_1 A + w_2 t_L) \tag{3.1}$$

Where

T_f = deadline time

t_L, Laxity $= T_f - T_s - t_D$

and A = jobsize i.e., the product of runtime (t_D) and number of processors (N_{cpu})

To initialize the simulation, the controller checks for the values of number of resources, w_0, w_1 and w_2 and then construct a simulation environment comprised of the defined number of resources. The controller also sorts the workload according to the earliest start time in ascending order and then sets the simulator clock to the lowest start time of the submitted workload. Once all the variables are set for the simulating environment the scheduling heuristic is then applied based on the set up variables. The initialized simulation results in the simplified architecture of computing infrastructure which is depicted in Figure 3.5. It is limited to one site (or one scheduler) containing all the processing elements, on to which SLAs are scheduled by the local scheduler using the scheme initialized by the controller (detailed in Section 3.3.1). The number of resources contained within a system are also initialized by the controller at the initialization phase based on the desired input value, as discussed earlier.

3.3.1 Scheduling Heuristics

The heuristic given in Equation 3.1 is slightly transformed and represented in Equation 3.2 to be use by the controller. The H is the computed priority based on the criterion

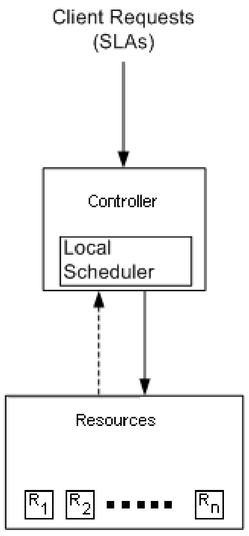


Figure 3.5: Conceptual Representation of Simulator

(CRI); *minimum* or *maximum* value of the heuristic $w_0T_f + w_1A + w_2t_L$. The criterion (CRI) is set at the initialization phase. Once set, it remains to the set criterion for a complete simulation. The jobs/SLAs are prioritized to schedule onto resources for each combination of w_0, w_1 and w_2 values iteratively, for a workload until the optimized performance is achieved, as discussed above. Again, as mentioned, the range of values for w_0, w_1 and w_2 are defined at the initialization phase of the simulator. Now, for example, if CRI is set to *minimum* and w_0 to 1 then Equation 3.2 is simply transformed into the one presented in Equation 3.1 which is the integrated heuristic of the work [YS06].

$$H = CRI(w_0T_f + w_1A + w_2t_L) \tag{3.2}$$

Where

CRI = minimization (Min) or maximization (Max) criterion
T_f = deadline time
Laxity, $t_L = T_f - T_s - t_D$
and A = jobsize i.e., the product of runtime (t_D) and number of processors (N_{cpu})

The performance of the system under different scheduling heuristics is measured in terms of number of served SLAs. Seven different scheduling heuristics can be derived from Equation 3.2 to calculate the priority of SLAs in order to schedule them onto resources. One of them is the best heuristic of the work [YS06] which is given in Equation 3.1 and is termed as *integrated* heuristic. The H is the computed priority based on the criterion (CRI) of minimum value of the heuristic combination of $w_0T_f + w_1A + w_2t_L$, using which the jobs are prioritized for the scheduling. The w_0 is set to 1 whereas w_1 & w_2 values are swept through the defined range, one at a time and that value of w_1 & w_2 is selected which results in highest number of successfully accepted SLA. If more than one set of values results in high number of acceptance of SLAs than the one with least expected finish time is selected.

Further, the following six *simple* heuristics are used that are derived from Equation 3.2 by setting up CRI, w_0, w_1 & w_2 as described below.

- $Min(T_f)$: prioritise jobs according to their minimum values of deadline times or simply earliest deadline first (EDF). To set up $Min(T_f)$, the CRI is set Min and w_0 is set to 1 whereas w_1 & w_2 are set to 0.
- $Min(t_L)$: prioritise jobs according to their minimum values of laxity or slack i.e., lowest laxity first. To set up $Min(t_L)$, the CRI is set Min and w_2 is set to 1 whereas w_0 & w_1 are set to 0.
- $Min(A)$: prioritise jobs according to their minimum job size values i.e., lowest job size first. To set up $Min(A)$, the CRI is set Min and w_1 is set to 1 whereas w_0 & w_2 are set to 0.

Then there are counter parts of above heuristics.

- $Max(T_f)$:prioritise jobs according to their maximum values of deadline times or simply earliest deadline last (EDL). To set up $Max(T_f)$, the CRI is set Max and w_0 is set to 1 whereas w_1 & w_2 are set to 0.
- $Max(t_L)$: prioritise jobs according to their maximum values of laxity or slack i.e., highest laxity first. To set up $Max(t_L)$, the CRI is set Max and w_2 is set to 1 whereas w_0 & w_1 are set to 0.
- $Max(A)$: prioritise jobs according to their maximum job size values i.e., highest job size first. To set up $Max(A)$, the CRI is set Max and w_1 is set to 1 whereas w_0 & w_2 are set to 0.

Based on the w_0, w_1, w_2 and CRI definitions, the controller schedule the arrived SLAs from a workload using the scheduling algorithm discuss next.

3.3.2 Execution Algorithm

The simulation runs in a unit time step and on each unit time, the controller checks for the arrival of an SLA and also the completion of already scheduled/executing SLAs. As the workload generator (discussed above), generates the time variables i.e., start time (T_s), finish time (T_f) and runtime (t_D), of all SLAs in minutes as shown in example workloads given in Table 3.1, 3.2 and 3.3. Therefore, there is need to scale the time variables of all SLAs before the start of simulation. For that purpose, the controller first finds the minimum runtime ($t_{D_{min}}$) value from the submitted workload. Based on the minimum runtime value, it calculates the time scaling factor (sf) for the simulating scenario using Algorithm 1. On line 1, controller finds the minimum runtime ($t_{D_{min}}$) value within the submitted workload, corresponding to this minimum value all other time based variables are then adjusted. The scaling factor (sf) is calculated first. Initially the sf is set to 60 which means all time variable are required to convert from minutes to seconds (line 2). Then, minimum runtime ($t_{D_{min}}$) is multiplied by sf and checks if the product is still less than 1 (line 3). If it is then sf is multiplied by 10 (line 4) and again loop back to line 2 to check if it is still less than 1. It performs this check and multiplies the sf by 10 until it is less than 1. Once it is greater than 1, the sf is used to convert all time

Algorithm 1 Time Scaling

```
1. Find minimum $t_D$ ($t_{D_{min}}$) from the submitted workload
2. Set scaling Factor ($sf$) to 60
3. While($t_{D_{min}}$ * $sf$ < 1)
4.         $sf = sf * 10$
5. For each SLA 'i' in a workload
6.         Multiply $T_s$, $T_f$, $t_D$ of $SLA_i$ with $sf$
7.         Truncate value of $T_s$, $T_f$, $t_D$ of $SLA_i$ after decimal
```

variables of all SLAs within the workload and truncates the values after decimal (line 6 and 7, respectively).

Once the scaling is performed on the submitted workload, the controller first sorts the workload according to the earliest start time in ascending order and then sets the simulator clock to the lowest value of earliest start time. The earliest start time of SLAs is considered as the arrival time of SLAs, which means as soon as simulator clock ticks for the value equal to the value of earliest start time of next SLA, the controller accepts the SLA and considered it as an arrived SLA and is eligible to schedule onto resources provided that its deadline and deadline of already accepted SLAs is not over run. The scheduling part of the controller is adopted from the work [YS06] and it follows the steps given below to schedule an arrived SLA together with the already accepted SLAs.

1. Upon arrival of an SLA, initialize heuristic (given in Equation 3.2) based on the set up values for CRI and the range defined for w_0, w_1 and w_2.

2. Based on the set up values for heuristic, compute the priority and prioritize all waiting and newly arrived SLAs using the computed priority.

3. Try to find N_{cpu} (computing nodes) that are available from T_s to $(T_s + t_D)$ virtual hours.

4. If not found in Step 3, try finding N_{cpu} with $T_s + \triangle t$ to $(T_s + \triangle t + t_D)$ virtual hours.

5. Repeat Step 4 until either successful finding or $(T_s + \triangle t + t_D) \leq T_f$

6. If N_{cpu} is not found for the mentioned time interval and not all sweeping range values are exhausted then, increment sweeping parameters and go to Step 2.

7. If N_{cpu} is still not found reject the SLA and restore last calculated schedule as well as last used values of the parameters w_0, w_1 and w_2.

8. Execute the scheduled SLAs until the arrival time of next SLA, on arrival of an SLA, go to Step 1.

A submitted workload is then evaluated against all of the discussed seven heuristics (i.e., both simple and integrated scheduling heuristics), one by one to observe the effect of the heuristics on the achieved performance i.e., number of served SLAs. After the evaluation, it is then checked whether performance is improved using the integrated heuristic or not in comparison to the performance obtained using simple heuristics. If the achieved performance using integrated heuristic is better than any of the simple heuristics then the workload is categorized as *reactive* (to integrated heuristic) otherwise it is considered as non-reactive (i.e., no improvement in performance with the application of integrated heuristic). The scheduling algorithm which sweeps parameters on the integrated heuristic can be considered as *exhaustive* scheme as it exhausts all defined values for w_1 and w_2 to obtained optimized schedule. All experiments discuss in this thesis are executed on windows XP running on Intel core2Duo (1.866 GHz) platform with 1GHz of RAM under Java [javb] environment using NetBeans IDE 6.9.1.

3.4 Assumptions Made

The work presented in this thesis has been carried out based on the following assumptions:

- A job is expressed in the form of an SLA which is comprised of *start Time, runtime, deadline Time,* and *number of CPUs required.*
- Jobs are of non-preemptive type i.e., once started their execution then they will run till their completion exclusively on the assigned resources.
- Apart from the example workloads discussed in this chapter, all workloads in this thesis are comprised of 1000 SLAs.
- It is assumed that the mentioned runtime of a job within an SLA is always exact.
- The start time (T_s) of an SLA/job is considered as its submission time which means SLA/job is ready to get scheduled/run as soon as it enters into the system.
- The execution environment is comprised of homogeneous processing elements and they will always be available without any disruption.
- If necessary, the meta-scheduler can carry out computations to identify the suitable parameter values for the integrated heuristic without any restriction.
- The performance metric used throughout for the evaluation is *number of served SLAs.*
- The performance of the SLA based scheduling may be measured in terms of revenue as SLA may include the monetary terms such as price/budget or penalty. However, this requires the specification of a pricing policy. Based on the pricing policy, the performance in terms of revenue can be measured, but this is out of the scope of this work. Furthermore, the metrics used in traditional batch system for the measurement of performance (e.g., makespan), are irrelevant for the evaluation of an SLA based system [YS06].

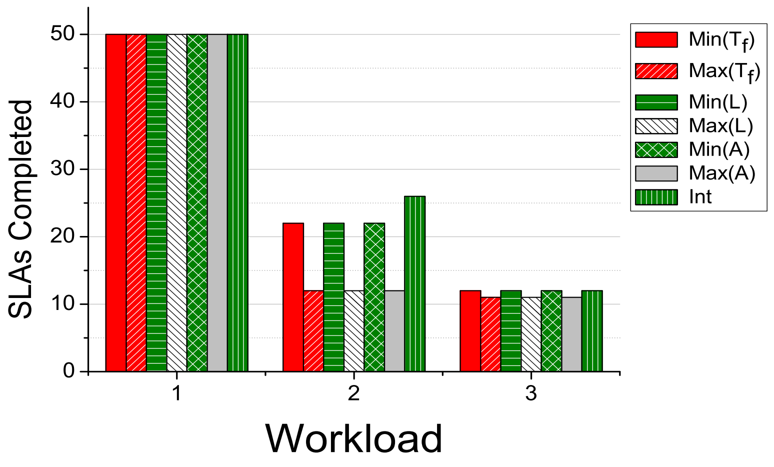
- The investigation of the thesis is focused on the scheduling aspect of SLA-bound jobs, submitted for execution to high-performance computing resources, therefore, no particular architecture for the experiments is considered, apart from the simple one in which all resources are governed by one scheduler (i.e., local scheduler).

3.5 System Evaluation Approach

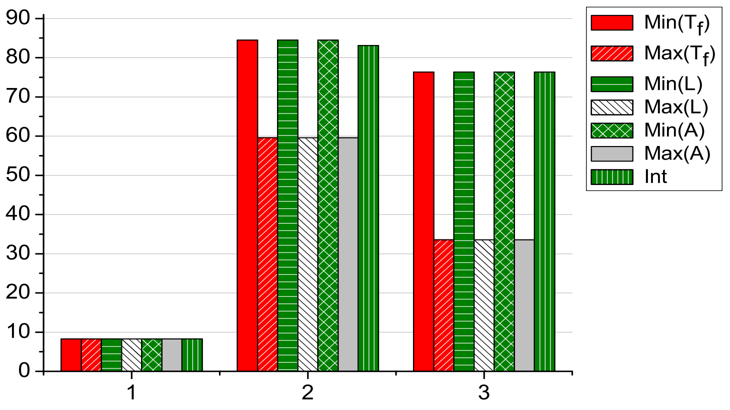
This section gives the idea of evaluation approach used throughout in this thesis. To understand it, the system is set up with 8 resources/CPUs onto which the workloads discussed in Section 3.2 and presented in Table 3.1, 3.2 and 3.3 are used to evaluate the performance of the system (discussed in Section 3.3) under simple heuristics and integrated heuristic (discussed in Section 3.3.1). The performance metric throughout in the thesis is *number of served SLAs* but again for the sake of presentation as mentioned and discussed in Section 1.5 the utilization of the system under simple and integrated heuristics for all workloads, is also observed.

For the workloads presented in Table 3.1, 3.2 and 3.3, the observed number of SLAs served and utilization of the system, against each of the simple heuristics as well as the integrated heuristic, is presented in Figure 3.6(a) and 3.6(b), respectively. Reference to Figure 3.6(a), the number of served SLAs under workload 1 are same for all used heuristics, i.e., 50 SLAs. Similarly, reference to Figure 3.6(b), the utilization observed under workload 1 is same for all used heuristics, i.e., 8.26%. The reason is obvious, if the workload 1 presented in Table 3.1 is observed closely then it is quite evident that the jobs/SLAs are arriving into the system sequentially i.e., with the gaps in the deadline time (latest finish time) of an arrived SLA and earliest start time of successive SLA. Hence, no contention of resources at any point in time even if the request is made for 8 CPUs.

Under workload 2, the number of served SLAs are different for different heuristics. For simple heuristics based on least or earliest value i.e., *Min(*)*, the number of served SLAs are 22 with the utilization value of 84.47%, as shown in Figure 3.6(a) and 3.6(b), respectively. Similarly, for simple heuristics based on latest or highest value i.e., *Max(*)*, the number of served SLAs are 12 with the utilization value of 59.56%, as shown in Figure 3.6(a) and 3.6(b), respectively. For integrated heuristics (*Int*), the number of served SLAs are 26 with the utilization value of 83.11%, as shown in Figure 3.6(a) and 3.6(b), respectively. It can be noted, the maximum number of served SLAs by any simple heuristics is 22 and by integrated heuristic it is 26 i.e., integrated heuristic served 4 more SLAs in comparison to maximum performance achieved by any simple heuristics. Further, the utilization of the system is decreased slightly from 84.47% to 83.11% by using integrated heuristic, which could be a possible trade off between number of served SLAs and utilization, as detailed in [YS06] [HSR09]. The reason for such drop in utilization under workload 2 is, the integrated heuristic favors jobs with low CPU requirement [YS06]. That is, at some point integrated heuristics might have accepted a long running job with lower number of CPU requirement and allowed other similar kind of jobs to execute on the system while denying the job(s) with higher number of CPU requirements. This increases the number of served SLAs but on the other hand decreases the utilization while leaving the system resources idle for a while. Therefore, under workload 2, integrated heuristic exploited the backfilling feature of its implicit nature [HSR09] for some low



(a) SLAs Served by Individual Heuristics



(b) Utilization Observed under Individual Heuristics

Figure 3.6: Performance Observed Against Example Workloads

CPU requirement job, when it found a possibility to delay any SLA to adjust a new one without over running the deadline of all accepted SLAs and later on denying the resources to a newly arrived job requiring high number of CPU requirement.

Again, under workload 3, the number of served SLAs are different for different heuristics. For simple heuristics based on least or earliest value i.e., *Min(*)* and integrated heuristic (*Int*), the number of served SLAs are 12 with the utilization value of 76.34%, as shown in Figure 3.6(a) and 3.6(b). Similarly, for simple heuristics based on latest or highest value i.e., *Max(*)*, the number of served SLAs are 11 with the utilization value of 33.57%, as shown in Figure 3.6(a) and 3.6(b). Under this particular workload, integrated heuristic unable to exploit the backfilling feature as requests/SLAs are arriving into the system with the high rate and with very low flexibility in their deadline time with respect to the runtime, therefore, there is no possibility at any time to delay any SLA to adjust a new one without over running the deadline of all accepted SLAs.

3.5.1 Performance Metric

The performance metric is *number of served SLAs* throughout in this thesis. Therefore, to represent the SLAs completed by all discussed heuristics for large set of workloads is difficult to present in the form shown in Figure 3.6(a). Rather, the difference between the maximum number of served SLAs by any simple heuristic and by integrated heuristic, is useful to evaluate the system. Hence, to calculate the performance difference (PD) between simple heuristics and integrated heuristic, in terms of percentage the formulae given in Equation 3.3, is derived.

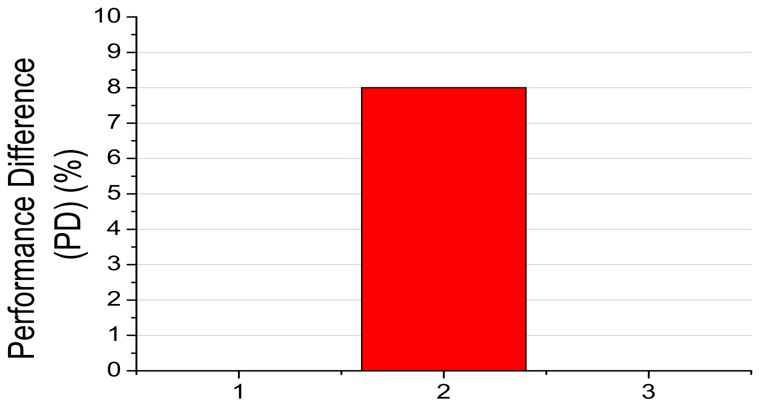
$$PD = \frac{(P_{Int} - Max\_P_S)}{Total\ Number\ of\ Submitted\ SLAs} * 100 \tag{3.3}$$

Where

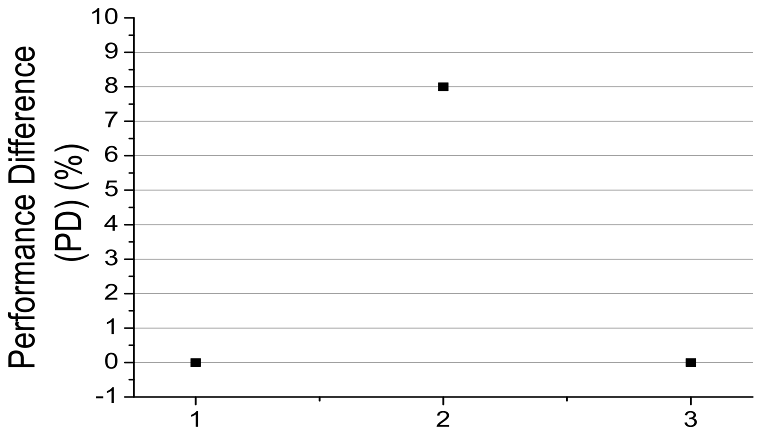
P_{Int} = Performance achieved by Integrated Heuristic
Max_P_S = Maximum Performance achieved by any Simple Heuristics

Now, if the observation given in Figure 3.6(a) is re-structured and presented in the form of performance difference (PD) then, the observation becomes clear and easy to understand for even small set of workloads and as well as for large set of workloads. The observation in terms of percentage of performance difference (PD) for the workloads presented in Table 3.1, 3.2 & 3.3, can be shown in two different formats as given in Figure 3.7.

Figure 3.7(a), represents the percentage of performance difference for all three workloads in the form of bar. Whereas, 3.7(b), represents the percentage of performance difference for all three workloads in scattered form. The cleaned version of the observation (given in Figure 3.7), clearly shows that for workload 1 and 3, there is no difference in system performance no matter which heuristic system uses to schedule the workloads. However, there is an improvement of 8% with the use of integrated heuristic for workload 2. Figure 3.7(a) and 3.7(b), both show the difference in understandable form but the scattered form (i.e., given in Figure 3.7(b)) gives easy to catch vision for the large



(a) Bar Format



(b) Scattered Format

Figure 3.7: Percentage of Performance Difference

set of workloads. Therefore, the scattered form is chosen to represent the percentage of performance difference (PD) throughout in this thesis.

3.6 Limitations of Designed Simulator

As mentioned, the investigation of the thesis is focused on the scheduling of SLA-bounded jobs without considering any particular computing architecture, hence following are the limitations of designed simulator that are required to be addressed, if anyone is required to move into any particular research direction using the designed simulator.

- Due to the nature of investigation, not any particular negotiation protocol has been implemented to avoid rejection of any particular SLA or otherwise.
- The resources constructed within the simulation are processors of homogeneous type.
- All constructed resources are tied to single scheduler i.e, only single scheduler is supported by the current implementation of the simulator.
- It is assumed that any SLA-bounded job can run on any resource without needing the setting up prior to the execution of the job i.e., if the data is required to run the job then data would be readily available to the resources.
- The simulator is designed using Java [javb] programming language and the proposed metric is calculated using an array of double type in Java therefore, the allowable range for an array is dependent on heap size of HotSpot VM [javc]. If not carefully checked the simulator will raise memory leak error [javc].
- The simulator is used for maximum of 128 resources and it should work for higher number of resources as well. The maximum limit will again be based on heap size of HotSpot VM.

3.7 Summary

The chapter discussed in detail the usage of SLA-based workload generator (discussed in [SY08] [YS06]) as well as described the load generator's input variables together with the steps to generate a desired workload. The architectural and procedural details of the designed simulator presented, that described the setting-up of different heuristics for the experiments. Seven different heuristics implemented and discussed together with the execution algorithm of the simulator. In the simulated environment resources are governed by single/central scheduler or controller which controls the complete simulation environment and responsible for ticking the simulation clock as well as for the checking of next arrival of an SLA/job. The controller is the core of the simulator. If an SLA/job arrived into the system, controller then checks the feasibility to schedule the arrived SLA/job using the preset scheduling heuristic without over running the deadline of newly arrived SLA/job as well as the deadline of already accepted SLAs/jobs. If there is no possibility to accept the newly arrived SLA/job using the preset heuristic then it is rejected and put into the rejected queue of the simulated environment.

Finally, the simulator saves the result against the preset heuristic in terms of number of served SLAs/jobs and system utilization. The comparison of the obtained performance under different heuristics is done using performance difference (PD), which is the difference between the maximum number of served SLAs by any simple heuristic and by integrated heuristic in terms of percentage against the total number of submitted SLAs/jobs.

Chapter 4

Workload Classification

As observed in [YS06] and discussed in Chapter 1 & 3, not all simple heuristics produce the maximum performance but one among them is suitable for particular workload condition. However, there is a possibility of a condition where for a certain type of workloads, no matter what heuristic system uses but the resulting performance is remain the same, even under integrated heuristic. At this condition the use of the integrated heuristic by the system, is not worthwhile. Therefore, the identification scheme for the need of parameter sweeping on the integrated heuristic is required, to avoid useless computation. It is therefore, aimed to propose and evaluate a metric in this chapter to quantify SLA-based workloads and based on the metric values, workloads are then characterize for serving with integrated heuristic or simple heuristics i.e., whether the highest performance in terms of served number of SLAs can be extracted using the integrated heuristic or not.

In order to identify whether the integrated heuristic (discussed in Section 1.3) is useful choice to serve a workload or not, following methodology is used in this chapter.

- First, workloads each consist of 1000 SLAs, are generated with low arrival rate and with small runtime values using a load generator (discussed in Section 3.2). The aim of generation of workloads is to generate a set of workloads that have either no or low difference in performance under *simple* and *exhaustive* scheduling schemes in the first instance (both discussed in Section 3.3.1).

- The generated workloads are then evaluated under simple scheme (which uses all six simple heuristics one by one for each workload) and exhaustive scheme (which uses integrated heuristic). After evaluation of workloads against the scheduling schemes, the obtained performance is recorded. The recorded performance is then used to measure the difference in performance in terms of served number of SLAs between the two schemes. The difference in performance is measured in terms of *percentage of performance difference (PD)*, as discussed in Section 3.5.

- A metric to quantify the SLA-based workloads is then proposed using which the generated workloads are then quantified.

- The obtained performance differences between the simple and exhaustive schemes for all the generated workloads and the calculated metric values against all the workloads are then compared to each other, to observe a relation between the system performance (under both scheduling schemes) and proposed metric.

- In the exercise carried out in this chapter, workload which is a set of 1000 SLAs/jobs is whole as a unit is intended to identify for the suitability of simple or integrated heuristic not any subset of it.

All simulated scenarios discuss in this chapter are executed using the architectural environment discussed in Section 3.3.

4.1 Generation & Evaluation of Workloads

The first two steps of the methodology are followed in this section i.e., generation of workloads and their evaluation. The workloads are generated using low arrival rate values with the assumption that the system performance will be the same for most of the generated workloads and then on the next set of workloads those values will be used for the generation of workloads that have difference in performance. As mentioned earlier, the performance is measured in terms of served number of SLAs and workloads are generated using load generator discussed in Section 3.2. Hence, as a starting point, the values presented in Table 4.1 are used to generate initial set of workloads. Based on the combination of values mentioned in Table 4.1, total of 24 workloads are generated i.e., for each arrival rate value given in Table 4.1, all combinations of maximum runtime and minimum tightness values are used to generate a workload. Each of the generated workload consists of 1000 SLAs and all the workloads are generated using maximum job size or maximum allowable number of CPUs for an SLA equals to 8.

Table 4.1: Values Used for the Generation of Workloads

Arrival Rate	Max. Runtime	Min. Tightness
0.00001	100	0.1
0.0001	1000	0.25
0.001		0.5
		0.75

Then, the generated 24 workloads are evaluated using simple and exhaustive schemes under an environment containing 8 CPUs (called system CPUs (S_{cpu})), as discussed in Section 3.5. For the exhaustive scheme which requires parameter sweeping on the integrated heuristic, the simulation is run from -1.0 to +1.0 with the increment of 0.01 for each parameter. As discussed in Section 3.5.1, the difference in performance of the system for the two schemes against the generated 24 workloads, is observed. As assumed, the observed performance difference for 17 workloads out of 24 is zero and 7 workloads show an improvement in the performance of greater than 1% under exhaustive scheme. All seven workloads are generated using the arrival rate of 0.0001 and 0.001. Hence, for more observations further workloads are required to generate with the next starting value of the arrival rate equals to 0.001. Therefore, the values mentioned in Table 4.2 are used to generate further workloads i.e., for each arrival rate value given in Table 4.2, all combinations of maximum runtime and minimum tightness values are used to generate a workload. Total of 36 workloads are generated using the combination of values given in Table 4.2 for higher arrival rate values with an extra higher runtime value but keeping the maximum allowable CPUs equal to 8. Again, these generated workloads are evaluated using simple and exhaustive schemes under an environment containing 8 CPUs ($S_{cpu} = 8$) and the difference in performance of the system for the two schemes against the generated 36 workloads, is observed.

Table 4.2: Values Used for the Generation of Workloads: Effect of Arrival Rate

Arrival Rate	Max. Runtime	Min. Tightness
0.001	100	0.1
0.01	1000	0.25
0.1	10000	0.5
		0.75

The observed performance difference for all 60 workloads that are generated using the values given in Table 4.1 & 4.2 is depicted in Figure 4.1 against the generated sequence of the workloads. It is observed that out of 60 workloads, 18 workloads have no difference in performance and 42 workloads have difference in performance. Workloads with no difference in performance are termed as *non-reactive (NR)* workloads i.e., no improvement in performance can be made using exhaustive scheme and the workloads which show difference in performance are termed as *reactive (R)* workloads. Hence, out of 60 generated workloads, 18 are non-reactive and 42 are reactive. Further observation shows that out of 42 reactive workloads, 35 have difference in performance of 1% or more i.e., exhaustive scheme performed better than simple scheme for 35 workloads.

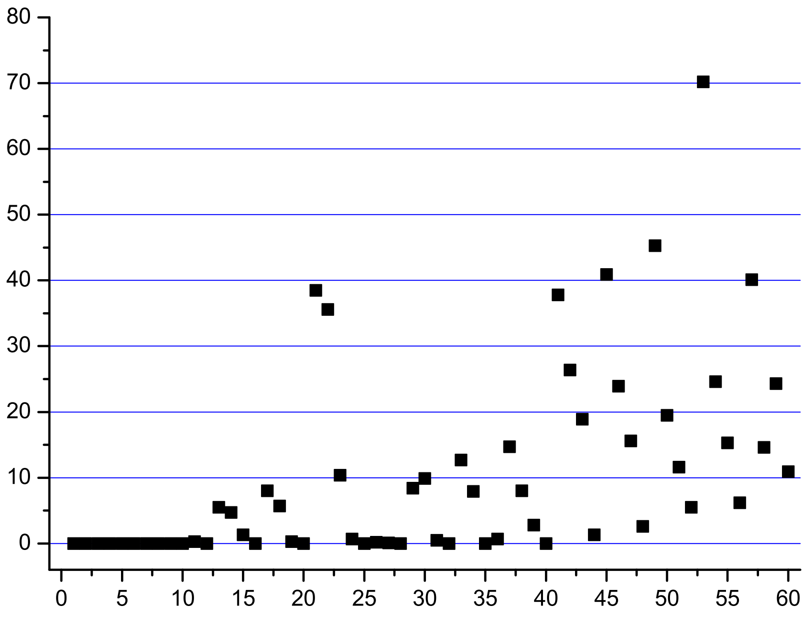


Figure 4.1: Performance Difference between Simple and Exhaustive Schemes

After obtaining the performance difference against the generated workloads, now it is desired to quantify the workloads against the metric proposed in the next section.

4.2 Quantification Metric

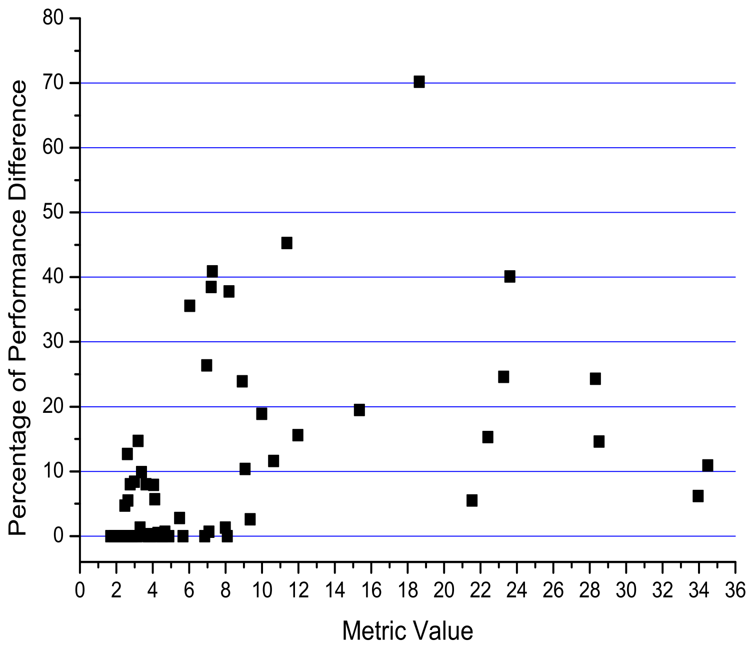
A metric is proposed to quantify the workloads. The proposed metric is presented in Equation 4.1 and the scheme to calculate the metric is given in Algorithm 2. The proposed metric is based on the idea of resource utilization (RU) and it computes an SLA demand on the total number of system CPUs (S_{cpu}). The calculation is based on the values of runtime (t_D), difference of deadline time & start time ($T_f - T_s$), CPUs requirement (N_{cpu}) by an SLA and total system resources or CPUs (S_{cpu}). As the metric is calculated for each SLA of a workload therefore, it simply calculates the possible resource utilization (RU) of the system if the SLA is admitted into the system. The RU is calculated for each time unit for all SLAs of a workload as given in Algorithm 2. This tells the amount of resource required for a unit of time by all of the SLAs of workload.

$$RU = \frac{t_D}{T_f - T_s} * \frac{N_{cpu}}{S_{cpu}} \tag{4.1}$$

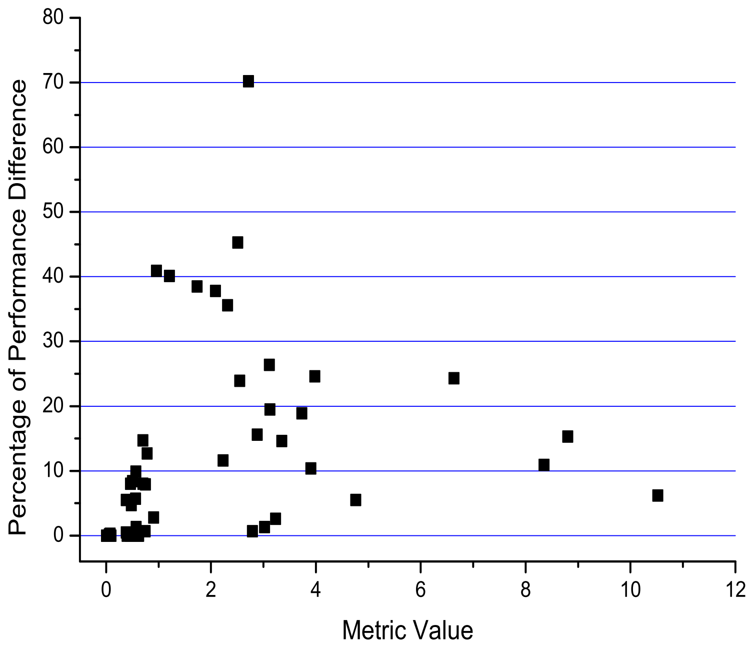
In Algorithm 2, the utilization or posed resource utilization (RU) by an SLA is computed on each unit time from the start time to the deadline time mentioned in the SLA. If the arrival time or start time of an SLA is in decimal then it is lower down to the nearest integer value ($\lfloor T_s \rfloor$). For example, if an SLA has a start time (T_s) of 100.15 units then its utilization is calculated from $t = 100$ to the mentioned deadline time of the SLA with unit time increment. Against the calculated value of the $RU[t]$ on each t time unit, the maximum and average of $RU[t]$ are computed. This simple algorithm calculates the metric RU in order to quantify the submitted workload.

Algorithm 2 Metric Calculation Algorithm - I

```
FOR a workload, sort all SLAs with their start time ($T_s$) in ascending order
Find minimum start time (Min $T_s$) and maximum deadline time (Max $T_f$) from the submitted workload
Set $RU[t] = 0$, for $t$ = Min $T_s$ to Max $T_f$
FOR each SLA {
    for $t = \lfloor T_s \rfloor$ (start time) to $T_f$ (deadline time) with unit time increment in $t$ {
        $RU[t] = RU[t] + \frac{t_D}{T_f - T_s} * \frac{N_{cpu}}{S_{cpu}}$
    }
}
Calculate Max. RU and Average RU.
```



(a) Against Max. *RU*



(b) Against Avg. *RU*

Figure 4.2: Performance Difference (PD) Against Metric Values For 60 Workloads

4.3 Quantification & Observations

Based on the Algorithm 2, the maximum metric value ($max(RU)$) and average metric value ($avg(RU)$) are calculated and recorded. Then, the performance differences (PDs) of the generated 60 workloads (observed in Section 4.1) are mapped against maximum metric value ($max(RU)$) and average metric value ($avg(RU)$) which is shown in Figure 4.2(a) and 4.2(b) , respectively. From Figure 4.2(a) , it can be observed that maximum value of the metric is not resulting in better identification scheme as it contains number of workloads with performance difference less than 1% for a wide range of values. In contrast to it, the average metric values show better results as shown in Figure 4.2(b). In Figure 4.2(b), for a small range of average RU ($avg(RU)$) values, few workloads with performance difference (PD) greater than 1% workloads are observed together with no performance difference. The range of $avg(RU)$ values for which workloads with PD less than or equal to 1% are observed, is considered as disputed range. To find the disputed range for $avg(RU)$ values the steps given below are followed:

1. Sort the workloads together with their percentage of performance difference, in descending order based on the $avg(RU)$ values.
2. Search for the first occurrence of no performance difference. When found record the value of the metric.
3. Search for the last occurrence of performance difference greater than 0 and record the value of the metric.

After following the above steps, the disputed range for the $avg(RU)$ is observed to be between 0.621 and 0.06 i.e., $0.06 \leq avg(RU) < 0.621$. The $avg(RU)$ greater than or equal to 0.621 means the workload is reactive (or PD is greater than 0) and less than 0.06 means the workload is non-reactive (or PD is equal to 0). But for the disputed range, both reactive and non-reactive workloads are observed. 19 workloads have been observed within the disputed range and, 7 out of 19 are non-reactive. Further, observations led to two reasons, due to which the non-reactive workloads have high value of metric $avg(RU)$:

4.3.1 Effect of Tightness

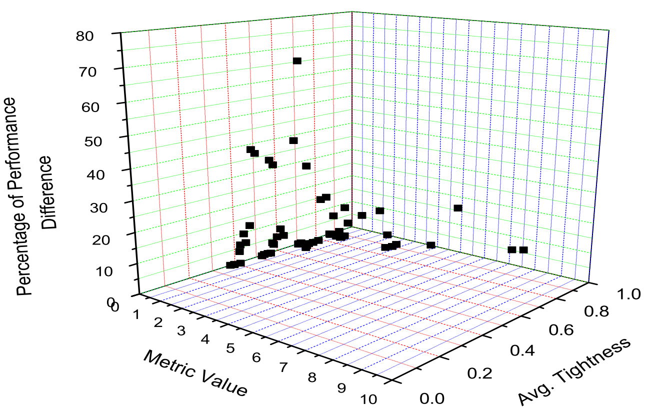
It is observed that the workloads with certain average tightness [YS06] values have specific characteristics. Whereas, the tightness (t_T) is defined as a measure of flexibility in deadline (T_f) provided within an SLA against its runtime (t_D) which is given in Equation 4.2. The value of tightness is always greater than 0 and less than or equal to 1. The tightness value close to 1 means less flexibility and 1 means no flexibility. The value of tightness shows the percentage of a runtime (t_D) span over the given time length (i.e., $T_f - T_s$). Under disputed range, it has been observed that 5 out of 7 non-reactive workloads have $avg(t_T)$ greater than 0.86 and only two non-reactive workloads have $avg(t_T)$ less than 0.86 (i.e., 0.69 and 0.41). The pattern of rejection for the workloads that have $avg(t_T)$ greater than 0.86, is somewhat same against all simple heuristics. This is due to the simultaneous arrival of number of SLAs with the $avg(t_T)$ greater than 0.86 i.e., with low flexibility. Hence, resulting in high utilization demand or high metric value but in actual, not all SLAs are served successfully. To explain this phenomenon, a snapshot of the continuous rejection of SLAs/jobs is presented in Table 4.3. It can be noted that there is a big difference between the deadline time of a job with ID 597 and the start time of a job with ID 598. That means, the system is idle when the job with ID 598 is arrived into the system. The system has maximum of 8 number of CPUs therefore, no matter what heuristics system uses, because of the arrangement of the jobs, job 598 will always get accepted, as resources will be free on its arrival therefore, system will assign the resources to job 598 and starts its execution at the same time as it arrives into the system i.e., at 416371.2 units. Hence, the job will execute on one CPU and finish its execution at 416462.4 unit. Further, it can be observed that jobs arriving into the system after the start of the job with ID 598, requiring 8 CPUs each and have deadline time less than the expected finish time of the job with ID 598, which has already acquired one CPU for it's execution. Therefore, jobs having ID from 599 to 606 will all get rejected by all the applied heuristics and resulting in the same type of performance. But calculating the utilization demand for this particular type of workload results in high *RU* values. It is understandable if the average tightness is close to 1 than SLAs have not much flexibility in their deadline times in comparison to their runtime therefore, all heuristics will tend to behave like first-come-first-serve (FCFS) scheme and hence no difference or minor difference in the performance will be observed.

$$t_T = \frac{t_D}{T_f - T_s} \tag{4.2}$$

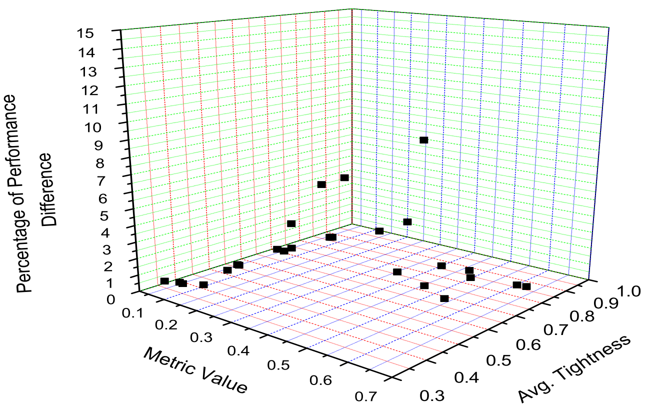
Table 4.3: Snapshot of a Workload - For Average Tightness

Job ID	Start Time (T_s)	Deadline Time (T_f)	N_{cpu}	Runtime (t_D)	Avg. Tightness $avg(t_T)$
597	413340.27	413342.82	2	2.483	0.974
598	416371.2	416464.63	1	91.2	0.976
599	416442.87	416444.05	8	1.15	0.972
600	416444.07	416445.25	8	1.15	0.972
601	416445.27	416446.45	8	1.15	0.972
602	416446.47	416447.65	8	1.15	0.972
603	416447.67	416448.85	8	1.15	0.972
604	416448.87	416450.05	8	1.15	0.972
605	416450.07	416451.25	8	1.15	0.972
606	416451.27	416452.45	8	1.15	0.972

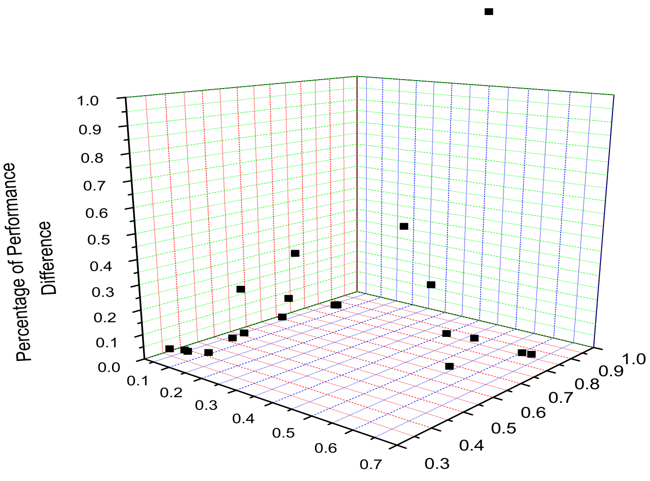
Further, the observation against the 60 workloads obtained for $avg(RU)$, $avg(t_T)$ and PD, is shown in Figure 4.3. Total of 7 non-reactive workloads have been observed within the disputed range and out of 7, 5 workloads have $avg(t_T)$ greater than or equal to 0.86 within the disputed range. Only two workloads are found within the disputed range having $avg(t_T)$ less than to 0.86. Within the disputed range, 12 workloads are observed to be reactive. Out of these 12 workloads, 7 workloads have PD of greater than 1% and all 7 workloads have $avg(t_T)$ less than 0.86. The performance difference (PD) greater than 1% is observed at the metric value greater than or equal to 0.38. The complete observation for $avg(RU)$, $avg(t_T)$ and PD, is shown in Figure 4.3(a). The observation



(a) All Workloads and Values



(b) Against the Disputed Range



(c) Against Disputed Range with PD $\leq$ 1%

Figure 4.3: Observed Avg. of Squared RU, Avg. Tightness and Performance Difference

for the values of disputed range is given in Figure 4.3(b) which is further refined to the performance difference less than or equal to 1% in Figure 4.3(c). From Figure 4.3(c), out of 7 non-reactive workloads, 5 can be seen for the average tightness greater than or equal to 0.86 together with the high metric value greater than 0.38, one non-reactive workloads can be seen for the metric value equal to 0.09 with average tightness of 0.41, and another one with metric value of 0.53 with average tightness of 0.68.

Table 4.4: Generation of Workloads: Effect of Tightness

Arrival Rate	Max. Runtime	Min. Tightness
0.001	1000	0.85
0.01		0.9
0.1		0.95
		0.99

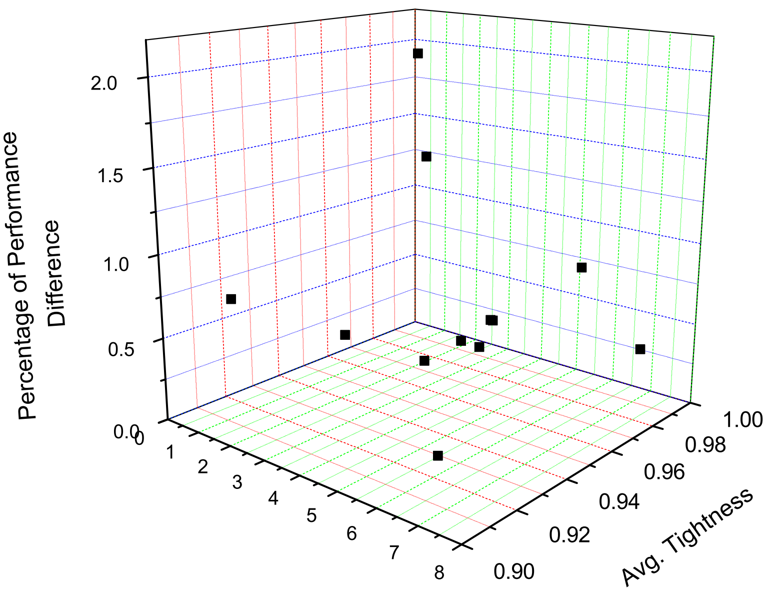


Figure 4.4: Observation For Increased Tightness

In order to further investigate a possible situation in which either PD is zero or less than or equal to 1% under higher metric value with average tightness greater than or equal to 0.85, further 12 workloads are generated using the combination of values given in Table 4.4 i.e., again, for each arrival rate value given in Table 4.4, all combinations of maximum runtime and minimum tightness values are used to generate a workload. Again, each of the generated workload comprised of 1000 SLAs. The minimum tightness is kept greater than or equal to 0.85. The workloads are then evaluated against the simple and exhaustive schemes as done previously. Then, the metric $avg(RU)$ is computed using

Algorithm 2 and average tightness is calculated against each of the generated workload. The observed metric values, average tightness and performance difference is shown in Figure 4.4. It can be seen that, except for two workloads the performance difference is less than 1% even though the metric values are higher than 1. This is due to the average tightness value which is greater than or equal to 0.93. The observed two workloads with the PD greater than 1% have metric values greater than 3.4. The high metric values shows the number of SLAs arriving into the system therefore, together with the average tightness it is the measure of higher number of SLAs arriving with either low or no flexibility in their deadline time. Again, it is observed that if the average tightness is close to 1 than SLAs have not much flexibility in their deadline times in comparison to their runtime, therefore, all heuristics tend to behave like first-come-first-serve (FCFS) scheme and hence no difference or minor difference in the performance is to be observed. Therefore, the resolution of observed disputed range of $avg(RU)$ is done using average tightness values i.e., if the $avg(RU)$ is less than 3.4 and greater than 1, then it is required to check for the average tightness ($avg(t_T)$) and in this case if $avg(t_T)$ is greater than equal to 0.93 then, the workload is either non-reactive or reactive with PD less than 1%.

Table 4.5: Snapshot of a Workload - For RU Value

Job ID	Start Time (T_s)	Deadline Time (T_f)	N_{cpu}	Runtime (t_D)	RU
64	6209.3	6209.367	2	0.0333	0.124874
65	6209.383	6209.45	2	0.0333	0.124874
66	6209.467	6209.533	2	0.0333	0.124876
67	6209.55	6209.617	2	0.0333	0.124874
68	6209.633	6209.7	2	0.0333	0.124874
69	6209.717	6209.783	2	0.0333	0.124876
70	6209.8	6209.867	2	0.0333	0.124874
71	6209.883	6209.95	2	0.0333	0.124874
72	6209.967	6210.033	2	0.0333	0.124876
				$RU[6209]=$	1.123872

4.3.2 Effect of Smaller Runtime Values

As the $RU[t]$ is calculated on integer intervals of time units with the increment of 1 therefore, it is observed that number of SLAs with runtime (t_D) value less than 1 unit time with the difference between the deadline and start time (i.e., $T_f - T_s$) less than 1 unit time, are the reason of increasing values for the metric even the load is of non-reactive type. For example, a snapshot of one of the workloads for low runtime values is presented in Table 4.5. It can be observed that for all the mentioned SLAs the RU will be calculated at time unit 6209 i.e., $RU[6209]$. Further, if observe closely the T_s and T_f of all SLAs, then, it can be concluded easily that all SLAs can be served on only 2 CPUs as they are consecutive and non overlapped. But having small t_D and $T_f - T_s$ values, the calculated value of RU for each SLA is worth noting as given in last column of Table 4.5. After adding up all values for the utilization value at 6209 i.e., $RU[6209]$, becomes a significant value of 1.123875 given in last row of the same table. Although, the given SLAs are non-overlapped, have average tightness less than 0.86 and do not require any sweeping of parameters to schedule but the low values of t_D and $T_f - T_s$ resulting in high utilization value because of the calculating nature of metric. However, it is also observed that for some workloads, such small runtime values do not effect the metric values, because of the their range of time units i.e., the start time to maximum deadline time of SLAs. As the metric is calculated and averaged against the range of the time units therefore, the high value range of time units at which the metric is calculated reduced the effect of the such small runtime noise values. But to cope with this situation, it is proposed to ignore SLAs having runtime value (t_D) less than 0.5.

Based on the observed values for the metric (M) and average tightness (or flexibility), a decision chart is constructed and is given in Figure 4.5. Three levels can be seen in the chart. At first level, the value of metric is checked and based on the metric value, the corresponding threshold value of average tightness is checked at the second level. If the average tightness is less than the threshold value under the corresponding metric value then the integrated heuristic is decided to use otherwise use of any simple heuristic is decided, at the third level or decision level. The observed values for metric and flexibility may vary for the different number of computing resources therefore, the classification table is constructed from the observed values for 8 CPU system. In Table 4.6, the range of average tightness values are classified as *varying, less tight, tight* and *very tight.* Any value of average tightness below 0.86 is considered as *varying*, value greater than 0.86 and less than 0.93 is considered as *less tight*, value greater than 0.93 and less than 0.99 is considered as *tight* and average tightness value greater than or equal to 0.99 is considered as *very tight*, as mentioned in Table 4.6.

Table 4.6: Flexibility Classification

Flexibility variation	Average Tightness ($avg(t_T)$)
Varying	$avg(t_T) < 0.86$
Less Tight	$0.86 \leq avg(t_T) < 0.93$
Tight	$0.93 \leq avg(t_T) < 0.99$
Very Tight	$avg(t_T) \geq 0.99$

Similarly, the metric values for *average RU* are grouped and presented in Table 4.7.

Table 4.7: Demand Classification

Utilization Demand Class	Metric Value (M)
Very Low	M < 0.38
Low	0.38 ≤ M < 0.62
Moderate	0.62 ≤ M < 3.4
High	M > 3.4

Based on the defined classes for utilization demand and flexibility, a workload classification chart is constructed to identify the workload as reactive (R) and non-reactive

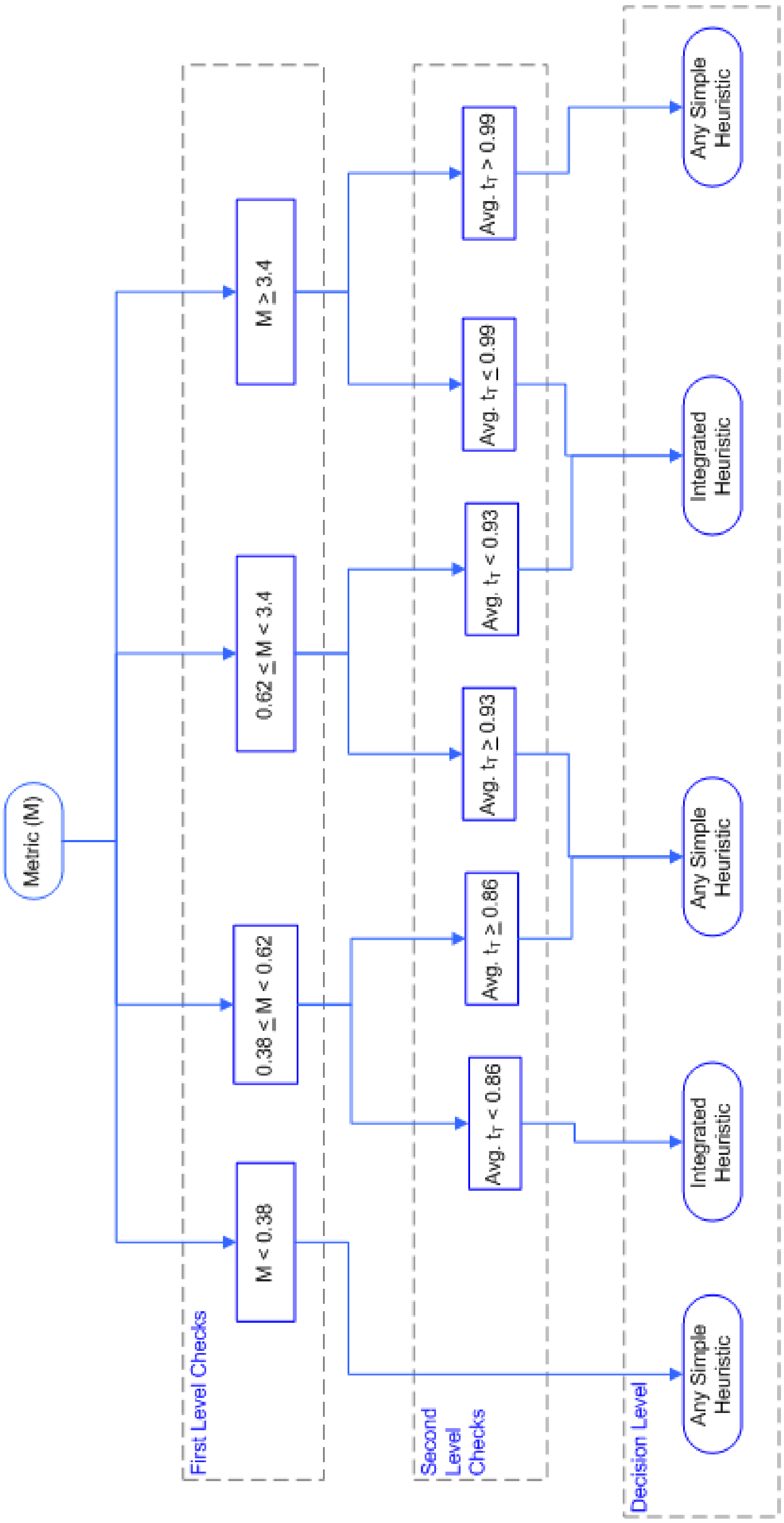


Figure 4.5: Observed Decision Tree

(NR), which is presented in Table 4.8.

Table 4.8: Workload Classification

Utilization Demand	Flexibility			
(Metric Value)	Varying	Less Tight	Tight	Very Tight
Very Low	NR	NR	NR	NR
Low	R	NR	NR	NR
Moderate	R	R	NR	NR
High	R	R	R	NR
NR: Non-Reactive				
R: Reactive				

Now, based on the defined classes for the utilization demand and flexibility presented in Table 4.7, 4.6 and 4.8, the metric calculation scheme is re-constructed and is presented in Algorithm 3 to work as a workload characterization scheme. It calculates the *RU* against the SLAs that have runtime (t_D) greater than or equal to 0.5. Based on the calculated demand utilization and flexibility, the scheme suggests whether the workload is suitable for Exhaustive Scheme/Integrated heuristic or not.

Algorithm 3 Workload Characterization Scheme

```
1.  Set Tightness = 0
2.  FOR each SLA 'i' in workload {
3.        IF $t_{D_i} \geq 0.5$ {
4.              FOR $t = \lfloor T_{s_i} \rfloor$ (start time) to $T_{f_i}$ (deadline time) with unit time increment in t {
5.                    $RU[t] = RU[t] + \frac{t_{D_i}}{T_{f_i} - T_{s_i}} * \frac{N_{cpu_i}}{S_{cpu}}$
6.              }
7.        }
8.        $Tightness = Tightness + \frac{t_{D_i}}{T_{f_i} - T_{s_i}}$
9.  }
10. Calculate avg. demand utilization and flexibility.
11.
12. IF Metric value is High AND flexibility is not Very Tight
13.       Suggest Workload as Reactive.
14. ELSE IF Metric value is Moderate AND flexibility is not Tight & Very Tight
15.       Suggest Workload as Reactive.
16. ELSE IF Metric value is Low AND flexibility is Varying
17.       Suggest Workload as Reactive.
18. ELSE
19.       Suggest Workload as Non-Reactive.
```

4.4 Workload Characterization Scheme

The workload characterization scheme given in Algorithm 3 is constructed from the observations discussed earlier. The *Tightness* variable is set to 0 on Line 1. Line 2 to 10 are same as given in Algorithm 2 to calculate the metric value or demand utilization. Except for Line 3 where it restricts the SLA from being considered if the runtime of an SLA is less than 0.5 (as discussed in Section 4.3.2). On Line 8, the tightness value is calculated against each SLA and is added into the previously obtained value to calculate the average of the tightness values at Line 10. From Line 12 onwards, the characterization scheme starts to check for the effectiveness of integrated heuristic on the submitted workload using the classification scheme based on Table 4.7, 4.6 and 4.8. The reactive workload suggestion means the performance can be improved by applying integrated heuristic on the submitted workload and non-reactive workload means no major performance can be obtianed using the integrated heuristic.

The scheme is first evaluated against the discussed 60 workloads, whose performance differences have already been measured, in order to confirm the behaviour. It is observed that out of 60, only 5 workloads have been characterized wrongly. The remaining 55 workloads characterized correctly as reactive or non-reactive. Out of 5 incorrectly characterized workloads, 1 workload is suggested for exhaustive scheme whereas the workload originally is of non-reactive type and 4 workloads are suggested as non-reactive whereas they are of reactive type with performance difference (PD) less than or equal to 0.7%. If the 4 workloads, that are suggested as non-reactive (but in fact are reactive with PD less than 0.7%) are considered as good suggestion in terms of computing cost to obtaine such low performance difference than, only 1 out of 60 workloads is completely incorrect i.e., 1.67% of incorrect suggestion, considering the fact that the scheme avoided exhaustive scheme for 20 workloads, which saves lot computation to evaluate the integrated heuristics.

4.5 Evaluation

In this section, the proposed workload characterization scheme is evaluated against the two different sets of workloads that are generated using two different methods. One of the generation methods is *systematic* and the other is *random*. In both of the generation methods, all workloads are generated for the maximum allowable CPU equal to 8. The generated workloads are first evaluated against simple and exhaustive schemes to check for the performance difference, as explained in Section 4.1. Then, the workloads are submitted to the proposed *workload characterization scheme* to characterize as reactive or non-reactive workloads. The reactive characterization of a workload suggests that the workload should be served with the integrated heuristic (or exhaustive scheme) for optimized performance. Whereas, non-reactive refers to the use of any simple heuristic to serve that workload. In order to validate the characterizations or suggestions, the actual obtained performance difference is used to classify the workloads as reactive and non-reactive which is then compared with the characterization done by the proposed scheme for the two generated sets of workloads. Under *systematic* method, total of 320 workloads are generated using pre-defined set of values of load generator parameters. Briefly, the systematic scheme generates workloads for each arrival rate value from 0.9 to 0.0001 and it sets different minimum tightness value with different combination of the maximum job length value (or maximum runtime). The maximum job size or maximum allowable number of CPUs that an SLA can have within a workload, is set to 8 for all workloads. The allowable values for minimum tightness are 0.1, 0.25, 0.5 and 0.75. The maximum job length is started from default value of 500 seconds and increased upto 32000 seconds by multiplying with 4 each time with the previously set value and if it is reached to the value greater than 32000 it is then reset back to 500 seconds for the next round of generation. It follows the generation scheme until the arrival rate value is greater than equal to 0.0001. Hence, for each value of arrival rate with the combination of maximum runtime and minimum tightness it generates 320 workloads each comprising of 1,000 SLAs. The combinations of values used to generate the workloads are also given in Table 4.9. In *random* generation method, arbitrary values for the combination of load generator parameters (i.e., for arrival rate, maximum job length, minimum tightness, repetition) picked up while keeping the maximum allowable CPU equal to 8. Total of 100 workloads are generated for the random set. The statistical characteristics of each of the workloads of systematic and random set can be found in Appendix A and B, respectively.

Table 4.9: Systematic Load Generation Scheme Values

Arrival Rate	Max. Job Length or Max. Runtime	Min. Tightness
0.9		
0.7		
0.5	500	0.1
0.3		
0.1		
0.09		
0.07		
0.05	2000	0.25
0.03		
0.01		
0.009		
0.007		
0.005	8000	0.5
0.003		
0.001		
0.0009		
0.0007		
0.0005	32000	0.75
0.0003		
0.0001		
Total = 20 x	4 x	4 x = 320

After the generation of workloads for the two mentioned sets, the workloads are then evaluated against the Simple and Exhaustive Schemes. The average time per workload taken by the exhaustive scheme to evaluate the workloads of systematic and random sets, is found to be 1162.046 minutes and 1062.376 minutes, respectively. The observed percentage of performance difference (PD) for each workload of systematic and random sets is shown in Figure 4.6(a) and 4.6(b), respectively. The observed system utilization under each workload of systematic set is splitted into four graphs and presented in Figure 4.8 and 4.9, whereas the utilization of the system under each workload of random set is presented in Figure 4.10. The overall utilization statistics in the form of average and standard deviation are also presented in Table 4.10, which shows a slight difference in terms of utilization of the system under both of the scheduling schemes.

Table 4.10: Utilization Statistics

Utilization	Systematic Set		Random Set	
	Avg.	Std. Dev.	Avg.	Std. Dev.
Simple Scheme	70.29	±26.14	69.4	±26.97
Exhaustive Scheme	71.77	±25.98	71.34	±26.91

Table 4.11: PD Threshold Per Workload

Threshold	Range
T1	PD = 0%
T2	0% < PD ≤ 10%
T3	10% < PD ≤ 20%
T4	20% < PD ≤ 30%
T5	30% < PD ≤ 40%
T6	40% < PD ≤ 55%

Table 4.12: Overall Average PD of Reactive Workloads

PD Stats	Average
Systematic Set	11.28
Random Set	10.49

The observations given in Figure 4.6 is summarized according to the different PD threshold values and presented in Figure 4.7. Figure 4.7(a) shows the summary of the observed PD against the systematic workloads and Figure 4.7(b) shows the summary of the observed PD against the random workloads that are extracted from the observations shown in Figure 4.6(a) and 4.6(b), respectively. The performance threshold range T1, T2, T3, T4, T5 and T6 are mentioned in Table 4.11. The threshold T1 is defined for the workloads with PD equal to 0%. T2 is defined for the range of PD between 0 and 10% and similarly T3, T4 and T5 are defined for the consecutive 10% range of values for PD. But T6 is defined for 40% to 55% of PD. Now, from Figure 4.7(a), it can be seen that 15.93% of workloads have PD equals to 0% (i.e., T1 = 0%) in systematic workload set. These are the non-reactive workloads that have no difference in performance with any of the scheduling heuristics and are represented under threshold value T1. For T2, which is the representation of PD greater than 0% and less than or equal to 10%, 47.5% of workloads within systematic are observed i.e., 47.5% of workloads within systematic set have PD greater than 0% and less than or equal to 10%. Similarly, for T3, T4 and T5 20.93%, 8.125% and 5.625% of workloads observed for the defined range of PD threshold values, respectively. T6 represents the range of PD greater than 40% and less than or equal to 55%, for which 1.875% of workloads within systematic are observed, as shown in Figure 4.7(a). Similarly, for the random workload set, the observation is presented in Figure 4.7(b) i.e., 18%, 57%, 14%, 6%, 2% and 3% of random workloads observed for the PD threshold range T1, T2, T3, T4, T5 and T6, respectively.

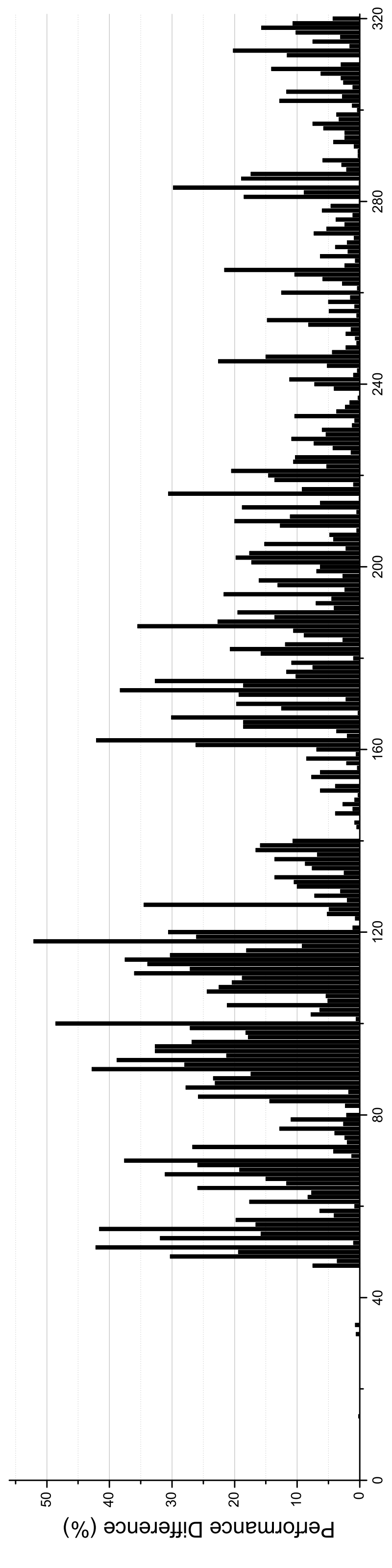
The average PD is calculated for reactive workloads of systematic and random set, to show the overall observations of PD under simple and exhaustive schemes. It is observed that on the average exhaustive scheme served 10% or more SLAs in comparison to any of the simple heuristics, as detailed in Table 4.12.

Based on the observed performance difference, the percentage of reactive and non-reactive workloads for each set is depicted in Figure 4.11(a). In the systematic set, there are total of 84.06% (i.e., 269 out of 320 workloads) of reactive and 15.94% (i.e., 51 out of 320 workloads) of non-reactive workloads observed whereas, in the random set, 82% (i.e., 82 out of 100 workloads) of reactive and 18% (i.e., 18 out of 100) of non-reactive workloads are observed, as depicted in Figure 4.11(a). In Table 4.13, out of six simple heuristics, the heuristic that served the highest number of SLAs for a percentage of reactive workloads of both systematic and random sets, is detailed. The percentages are calculated against the total number of reactive workloads i.e., 269 and 82 in systematic and random sets, respectively. From Table 4.13, the heuristic *earliest deadline first* ($Min(T_f)$) extracted highest performance for 42.75% and 43.9% of reactive workloads of systematic and random sets, respectively. *Least laxity first* ($Min(t_L)$) extracted highest performance for 24.54% and 24.39% of reactive workloads of systematic and random sets, respectively. Similarly, *minimum jobsize first* ($Min(A)$) extracted highest performance for 16.36% and 10.98% of reactive workloads of systematic and random sets, respectively. The highest or maximum criterion based simple heuristics jointly extracted highest performance for 16.36% and 20.73% of reactive workloads of systematic and random sets, respectively.

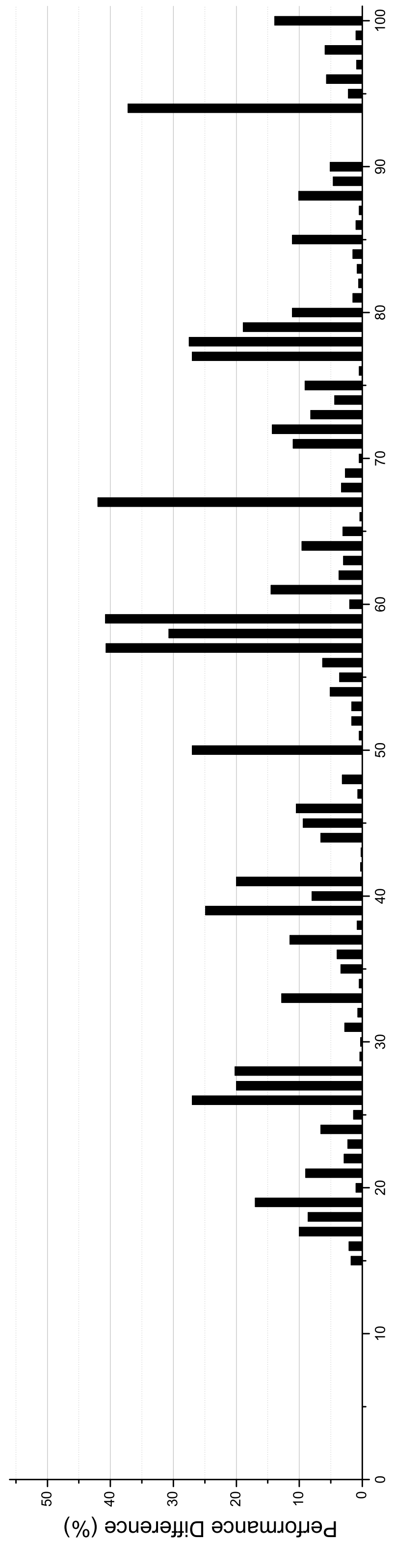
Table 4.13: Served Percentages of Reactive Workloads

Simple Heuristic	Systematic Set	Random Set
$Min(T_f)$	42.75%	43.9%
$Min(t_L)$	24.54%	24.39%
$Min(A)$	16.36%	10.98%
$Max(*)$	16.36%	20.73%

The workloads of the two sets are then submitted to the proposed characterization scheme to evaluate the suggestion against the actual observations. The observation of suggestion by the characterization scheme for the two sets is given in Figure 4.11(b). The percentage of correct identification for systematic and random set is found to be 96.56% and 97%, respectively. The percentage of incorrect identification for systematic and random set is found to be 3.44% (i.e., 11 workloads) and 3% (i.e., 3 workloads), respectively.

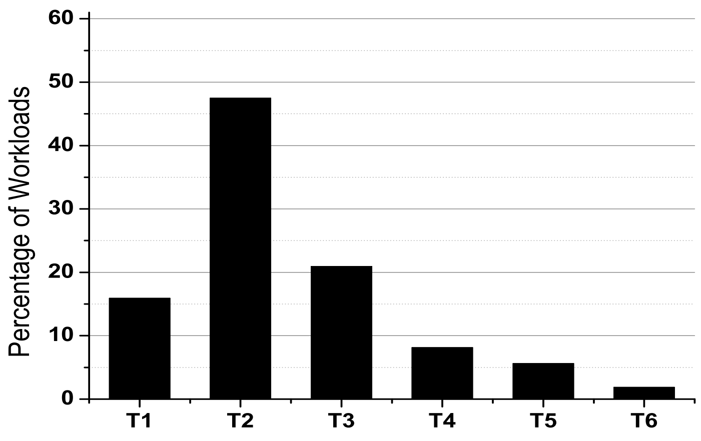


(a) Systematic Workloads

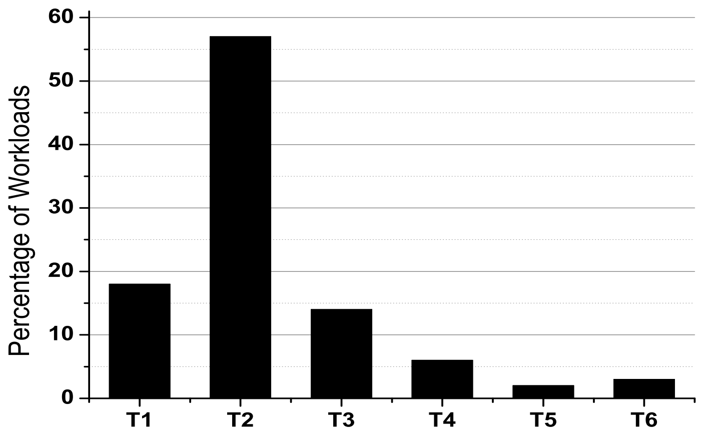


(b) Random Workloads

Figure 4.6: Observed Performance Difference Per Workload

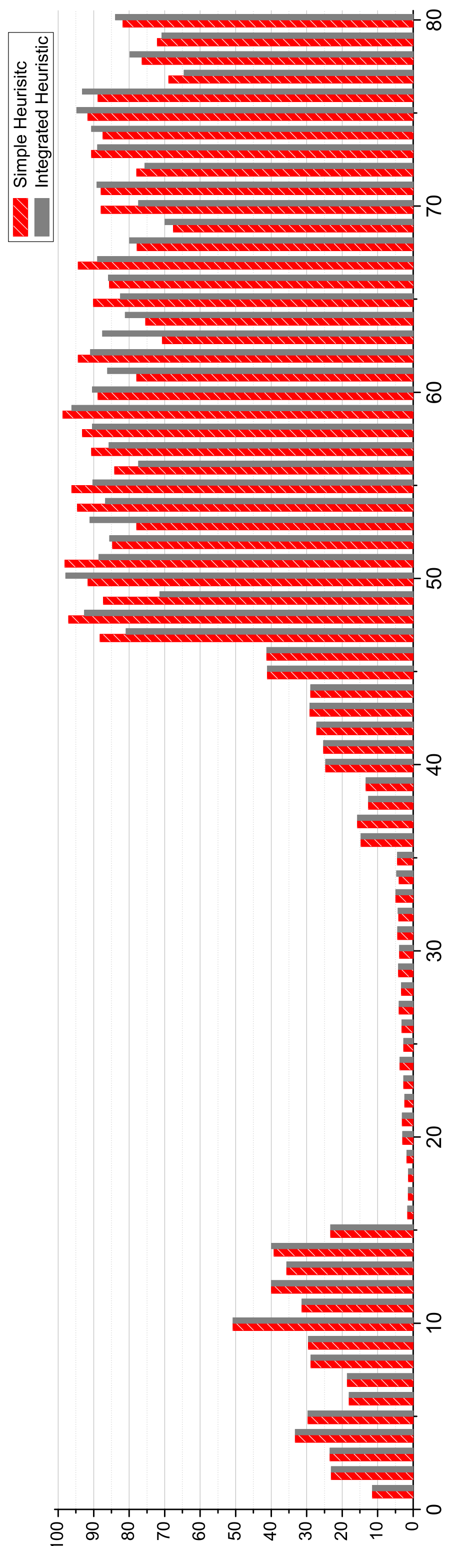


(a) Systematic Workloads

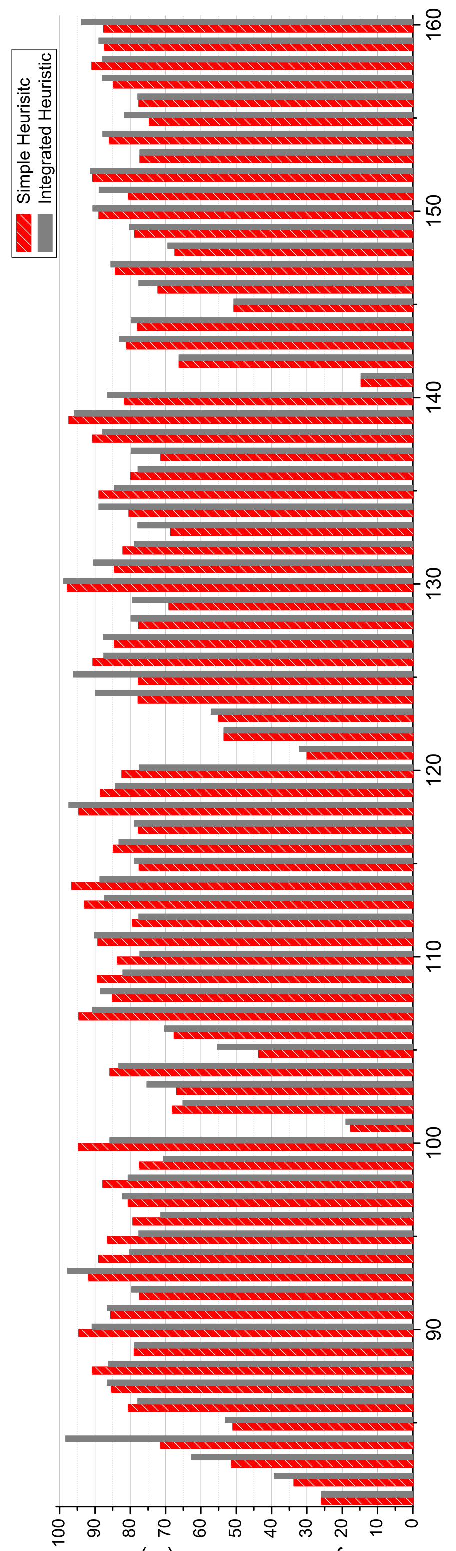


(b) Random Workloads

Figure 4.7: Percentage of Workloads Against Different PD Thresholds

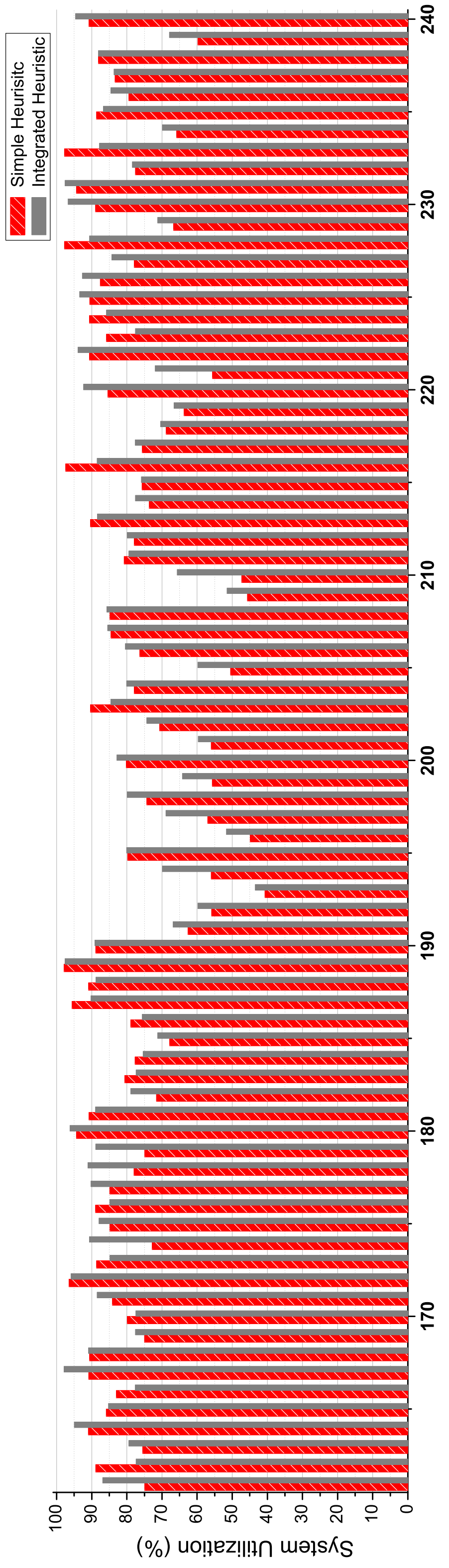


(a) Systematic Workloads (1-80)

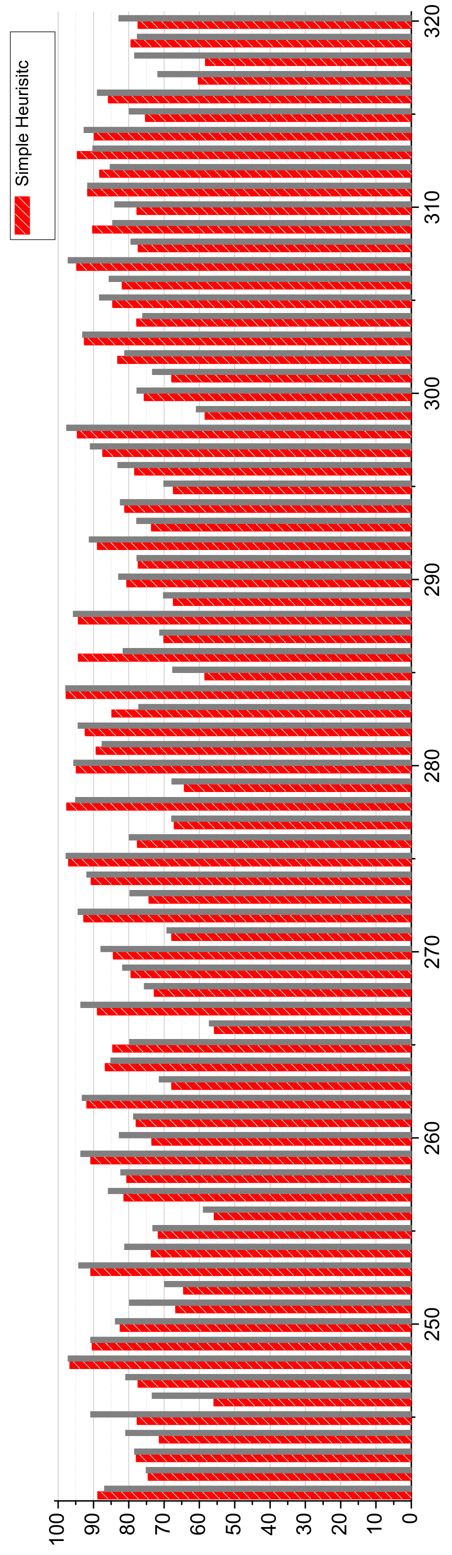


(b) Systematic Workloads (81-160)

Figure 4.8: Observed System Utilization Per Workload

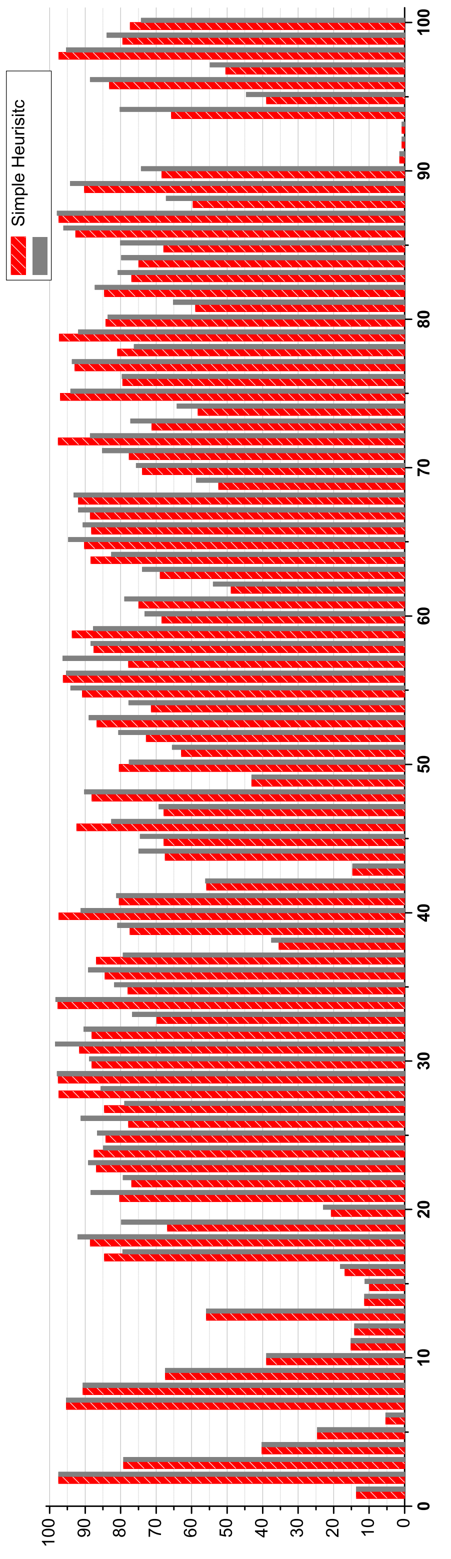


(a) Systematic Workloads (161-240)



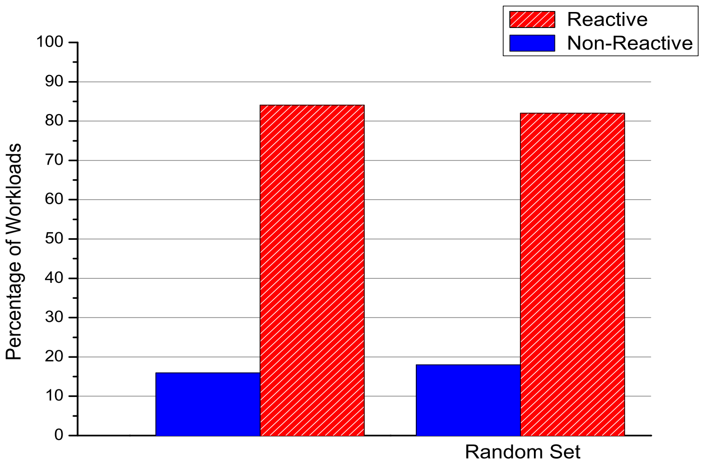
(b) Systematic Workloads (240-320)

Figure 4.9: Observed System Utilization Per Workload

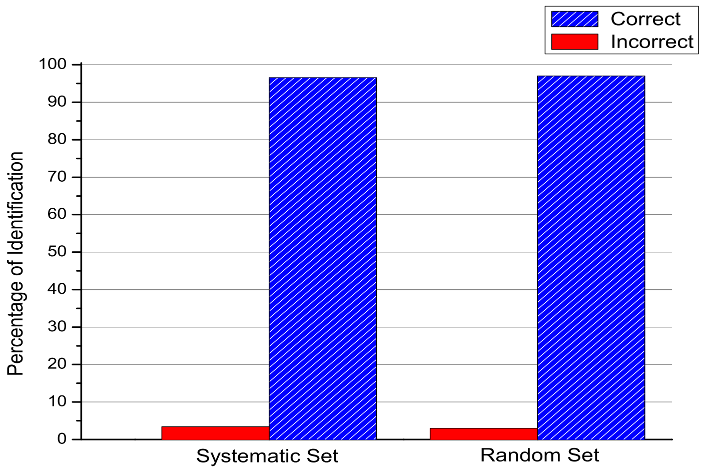


(a) Random Workloads

Figure 4.10: Observed System Utilization Per Workload



(a)



(b)

Figure 4.11: Evaluation Against Two Sets of Workloads

Further, it is noticed that out of 11 incorrect identification for systematic set, 4 workloads are identified as reactive whereas they are of non-reactive type. Remaining 7 workloads are identified as non-reactive whereas they are of reactive type. The PD of 7 incorrectly identified workloads are 0.3%, 0.6%, 0.6%, 0.7%, 0.9%, 1.7% and 2.1%. For random workloads set, 3 workloads are identified as non-reactive whereas they are observed to be reactive with PD of 0.3%, 0.7% and 3.2%. If the observations of incorrect identification for both systematic and random sets of workloads are scaled on threshold values defined in Table 4.11, then it can be seen that the erroneous observations made for the range T2 i.e., $0\% < \mathrm{PD} \leq 10\%$.

Although, the wrong or erroneous identifications are observed but if compared with the correct identification percentage which is around 97% for the two different sets of workloads then, having the scheme in place can save greater amount of computation time.

4.6 Summary

As discussed and evaluated in [YS06] that not a single heuristic from the simple ones (discussed in Section 3.3.1), is able to produce optimal performance in different scenarios which is why the maximum performance in scheduling of SLAs is achieved by integrated heuristics that adds multiple parameters from SLA. Each of the parameter is then multiplied by some weight factor and based on the values of weight factors the priority to schedule the SLA is calculated. The suitable value of weight factors is determined through parameter sweeping and this computation requires number of values to sweep through and therefore, this evaluation results in long computation time which is dependent on the range of weight factors or parameters, number of SLA parameters involved, number of resources within the system and number of SLAs to consider. The scheme becomes very expensive if no value of the weight factors results in improvement in performance. Therefore, to avoid the unnecessary computation for non-reactive workloads, a metric based identification scheme called *workload characterization scheme*, proposed and evaluated to quantify the SLA based workloads using proposed RU metric. Based on the quantified values for the average of RU metric and average tightness, the identification is made. The metric RU is calculated on each time unit to check the demand of utilization by the SLAs on the system resources.

The effect of average tightness on the performance difference (PD) also observed. The high metric values shows the number of SLAs arriving into the system but average tightness measures the flexibility provided/available within deadline times of submitted SLAs. It has been observed that if the average tightness is close to 1 than SLAs have not much flexibility in their deadline times in comparison to their runtime, therefore, all heuristics tend to behave like first-come-first-serve (FCFS) scheme and hence no difference or minor difference in the performance, observed.

The proposed scheme evaluated against two different sets of workloads i.e., systematic and random. The evaluation of the proposed workload quantification schemem showed that the percentage of correct identification is around 97% for both the sets. Only few reactive workloads with small performance difference, characterized as non-reactive for both the sets. Only for systematic set, the proposed scheme characterized three non-reactive workloads as reactive and if this is compared with the percentage of correct identification and saving of the computation time then, the erroneous identification can be ignored. Also, the proposed scheme completely avoids the need of simulating the computing scenario to find the usefulness of the integrated heuristic.

Chapter 5

Adaptive Scheduling Scheme

In Chapter 4, the possibility to characterize the complete workload to schedule on the available resources using metric ($avg(RU)$) and average tightness, proposed and evaluated. The exercise carried out in Chapter 4 helped to identify the suitability of simple or integrated heuristics for a given workload. In that exercise the performance of the system, under all workloads against all heuristics, obtained separately in order to know which workload is reactive (R) and which one is non-reactive (NR). Then, the proposed characterization scheme executed to classify the complete workload (i.e., workload containing 1000 SLAs) to be reactive (R) or non-reactive (NR). The obtained results from the characterization scheme compared with the actual obtained results. At this point the characterization scheme works separately and considers all 1000 SLAs of a workload. Whereas, in this chapter the characterization scheme is intended to be integrated into scheduling scheme to adapt scheduling heuristic with respect to the number/characteristics of SLAs within the system not within the submitted workload i.e., the size of the workload within the system may vary depending on the characteristics of the submitted workload set and the state of the simulation. The resulting scheduling scheme is called *adaptive scheduling scheme*, which upon arrival of a new SLA decides whether to use integrated heuristics or a simple heuristic for waiting and newly arrived SLAs to obtain maximum performance i.e., maximum number of served SLAs.

The adaptive scheduling scheme conditionally uses exhaustive scheduling scheme (which evaluates integrated heuristic) to calculate the schedule for the SLAs otherwise it will use the simple scheduling scheme (i.e., which evaluates simple heuristic). The adaptive scheduling scheme consists of two parts to decide upon the suitability of the exhaustive scheme on the arrived SLAs. Briefly, the **first part** deals with the *quantification of the SLAs* that are waiting to get schedule with the newly arrived SLA which is given in Section 5.2 using RU metric. The **second part** of the adaptive scheme is the decision making process based on the computed values of the average of RU ($avg(RU)$) and the average tightness values ($avg(t_T)$) which is discussed in Section 5.1. Finally, the algorithm suggests whether to use exhaustive scheme or not. The adaptive scheduling scheme follows the steps given below:

1. At initialization phase, set scheduling heuristic to earliest deadline first.

2. Upon arrival of an SLA, save the current parameters values (i.e., w_1 & w_2), if any.

3. Quantify the waiting and arrived SLAs using Algorithm 5 to compute the average of $RU[t]$ (i.e., $avg(RU)$).
4. Compute the average tightness ($avg(t_T)$) of waiting and arrived SLAs, based on the current time.
5. Check for the threshold range of $avg(RU)$ and $avg(t_T)$ using Algorithm 4 (discusses in Section 5.1), to decide which scheme to use to schedule the SLAs i.e., exhaustive or simple scheme.
6. If suggested for exhaustive scheme then try to schedule the SLAs with exhaustive scheme (as discussed in Section 3.3).
7. Else try to schedule the SLAs using simple scheme (as discussed in Section 3.3) using either the last used integrated heuristic or earliest deadline first heuristic.
8. If not scheduled at all then reject the SLA.
9. Execute the scheduled SLAs until the finish time of scheduled SLAs or the arrival time of the next SLA and, on arrival of an SLA go to Step 2. If no more SLA is available through the submitted workload then execute all scheduled SLAs till their completion.

Again, for the ease, the simplified single scheduler architecture discussed in Chapter 3 is used to evaluate the adaptive scheduling scheme. The adaptive scheme uses exhaustive scheme and simple scheme as mentioned. In Chapter 4, conditions at which the integrated heuristic shows difference in performance identified using the defined metric and average tightness values. Also, the conditions at which the performance of all simple heuristics and integrated heuristic is same, are identified. For conditions where all heuristics exhibits the same performance, the system can use any of the simple heuristics without the loss of performance. Based on this observation, the adaptive scheme utilizes simple scheme with a little modification, which is, either it tries to schedule the SLAs using earliest deadline first heuristic or with the integrated heuristic using the already computed weights (i.e., the weights computed in earlier round of schedule using exhaustive scheme). The reason of using earliest deadline first as a baseline heuristic is the statistics obtained in Chapter 4 Table 4.13, which shows that *earliest deadline first* ($Min(T_f)$) extracted highest performance for 42.75% and 43.9% of reactive workloads of systematic and random sets, respectively. Hence, no other simple heuristics is used by the simple scheme in adaptive mode.

5.1 Workload Classification Scheme

Based on the observations made in Section 4.2, the demand utilization metric (M $=$ $avg(RU)$) and flexibility ($avg(t_T)$) values are checked for the candidate SLAs, to decide the use of exhaustive scheme or simple scheme, as discussed in Section 4.4. This observation is reproduced and given in Algorithm 4 and the levels of checks to reach to the decision, are presented in Figure 5.1 which is similar to the scheme presented in Figure 4.5.

Again, the three levels are used. At first level, the value of metric is checked and based on the metric value, the corresponding threshold value of average tightness is checked next. If the average tightness is less than the threshold value under the corresponding metric value then the exhaustive scheme is decided to use otherwise simple heuristic either with least laxity heuristic or with previously calculated integrated heuristic, is decided to use at the third level or decision level of the presented decision tree. The classification table constructed from the observation (for 8 CPU system) made in Chapter 4 are also reproduced here. In Table 5.1, the range of average tightness values are presented that are classified as *varying, less tight, tight* and *very tight.* Any value of average tightness below 0.86 is considered as *varying*, value greater than 0.86 and less than 0.93 is considered as *less tight*, value greater than 0.93 and less than 0.99 is considered as *tight* and average tightness value greater than or equal to 0.99 is considered as *very tight*, as mentioned in Table 5.1. Similarly, the metric values (for *average RU*) are reproduced in Table 5.2 and the workload classification chart is presented in Table 5.3.

Algorithm 4 Suggestion based on Thresholds

```
1. IF Metric value is High AND flexibility is not Very Tight
2.         Use Exhaustive Scheme.
3. ELSE IF Metric value is Moderate AND flexibility is not Tight & Very Tight
4.         Use Exhaustive Scheme.
5. ELSE IF Metric value is Low AND flexibility is Varying
6.         Use Exhaustive Scheme.
7. ELSE
8.         Use Simple Scheme.
```

Table 5.1: Flexibility Classification

Flexibility variation	Average Tightness ($avg(t_T)$)
Varying	$avg(t_T) < 0.86$
Less Tight	$0.86 \leq avg(t_T) < 0.93$
Tight	$0.93 \leq avg(t_T) < 0.99$
Very Tight	$avg(t_T) \geq 0.99$

5.2 Quantification of SLAs

As mentioned above, in adaptive mode the quantification is performed on an arrival of an SLA into the system against the number of system CPUs. Based on that quantification the decision to use exhaustive scheme or simple scheme is made for the SLAs waiting to get scheduled. The steps to perform quantification are given in Algorithm 5 which is dervied from the Algorithm 3 discussed in Section 4.4. Algorithm 5 checks for the

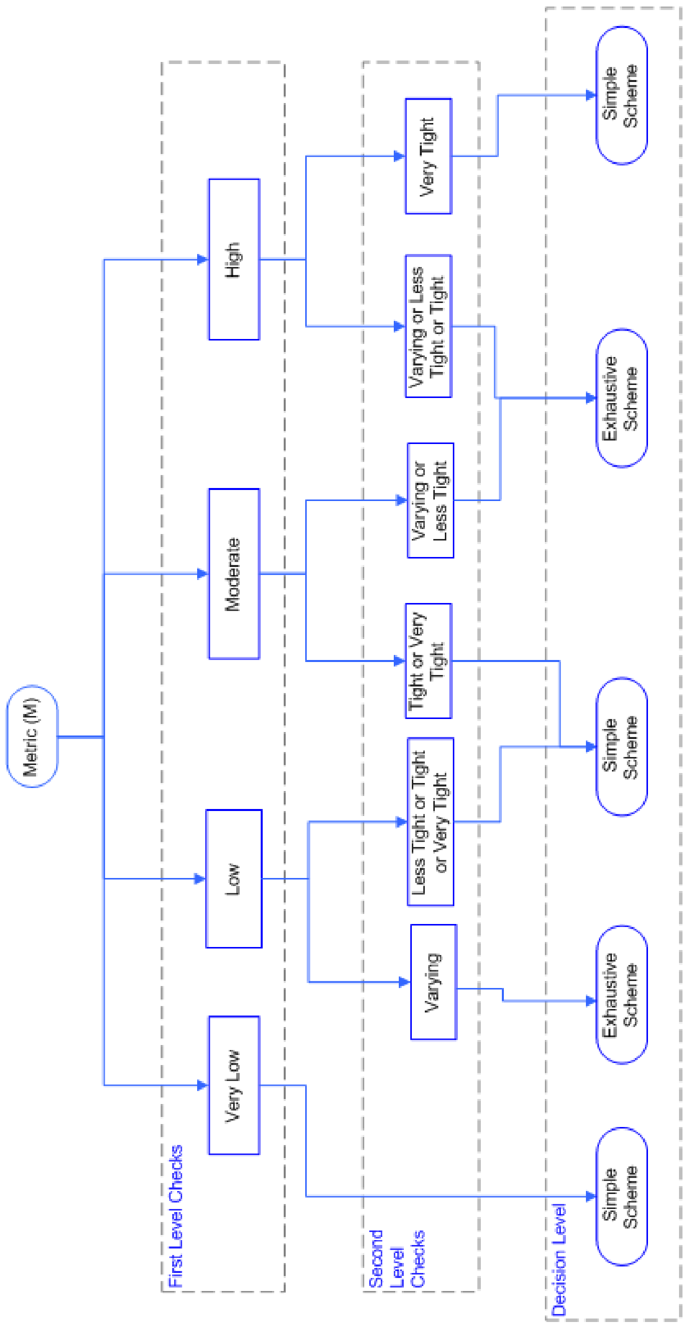


Figure 5.1: Decision Tree

Table 5.2: Demand Classification

Utilization Demand Class	Metric Value (M)
Very Low	M < 0.38
Low	0.38 ≤ M < 0.62
Moderate	0.62 ≤ M < 3.4
High	M > 3.4

Table 5.3: Scheduling Scheme Selection

Utilization Demand	Flexibility			
(Metric Value)	Varying	Less Tight	Tight	Very Tight
High	Exhaustive	Exhaustive	Exhaustive	Simple
Moderate	Exhaustive	Exhaustive	Simple	Simple
Low	Exhaustive	Simple	Simple	Simple
Very Low	Simple	Simple	Simple	Simple

Algorithm 5 Quantification of the SLAs

```
1. Find maximum deadline time ($T_{f_{max}}$) of all waiting and newly arrived SLAs
2. Set start time ($T_s$) of all waiting SLAs to the current system time ($T_c$).
3. Allocate range to RU array i.e., RU[ $\lceil T_{f_{max}} - T_c \rceil$ ]
4. Set $RU[t] = 0$, for t = 0 to $(\lceil T_{f_{max}} - T_c \rceil) - 1$.
5. FOR each waiting and newly arrived SLA 'i' {
6.         FOR t = $\lfloor T_{s_i} \rfloor$ to $T_{f_i}$ with unit time increment in t {
7.                 $RU[t - T_c] = RU[t - T_c] + \frac{t_{D_i}}{T_{f_i} - T_{s_i}} * \frac{N_{cpu_i}}{S_{cpu}}$
8.         }
9. $t_T = t_T + \frac{t_{D_i}}{T_{f_i} - T_{s_i}}$
10. }
11. Calculate Average Demand Utilization ($avg(RU)$) and flexibility ($avg(t_T)$)
```

range of the time to consider which is based on the current system time (i.e., the time at which the system performing the scheduling check) and the maximum deadline time of the under consideration SLAs (Line 1). Set the start time of all candidate SLAs to the current system time (Line 2). At line 3, it allocates the required amount of memory to hold the range of RU values in an array form. The range of RU array is calculated using the difference of maximum deadline time ($T_{f_{max}}$) of the candidate SLAs and the current system time (T_c) at which the new SLA is arrived into the system. To avoid array out of bound exception [java] which results when the array index goes beyond the range value, ceiling function is applied on the difference of $T_{f_{max}}$ and T_c to round up to the next integer value i.e., $\lceil T_{f_{max}} - T_c \rceil$. Then, $RU[t]$ is set to zero, for the values of t from index zero to the maximum range value i.e.,$(\lceil T_{f_{max}} - T_c \rceil) - 1$ (Line 4). The quantification of $RU[t]$ and tightness is calculated against each SLA 'i' in Line 5 through 10, as discussed in Section 4.3. After the quantification of the SLAs on each time unit for the demand resource utilization ($RU[t]$), the average of $RU[t]$ is calculated against the considered time interval i.e., $T_{f_{max}} - T_c$ together with average tightness values (i.e., flexibility) which is calculated against the considered number of SLAs (Line 11).

5.3 Performance Measurement

To observe and evaluate the proposed adaptive scheme, the simulator discussed in Section 3, is extended for the adaptive scheme. It follows the steps discussed above to calculate the metric and average tightness values upon arrival of an SLA and based on those values a decision is made to select exhaustive or simple scheme to schedule the SLAs.

5.3.1 Workloads

The evaluation of the scheme is done using the two sets of workloads i.e., *systematic* and *random*, discussed in Section 4.5 of Chapter 4. The two sets of workloads consist of 320 and 100 workloads, respectively and each workload comprised of 1000 SLAs. In the previous chapter, each set of the workloads divided into two categories i.e., non-reactive and reactive. No performance difference (PD) (or zero performance difference) observed for the non-reactive workloads, no matter what heuristic system uses to schedule the SLAs of non-reactive workloads. Whereas, reactive workloads have performance difference (PD) greater than zero between simple heuristics and integrated heuristics. Similarly, in this chapter the performance is evaluated for each set of workloads with respect to the categories defined in the previous chapter i.e., non-reactive and reactive to check how many times simple heuristic/scheme is suggested or used for reactive and non-reactive workloads.

5.3.2 Performance Metrics

The performance of the adaptive schemes is measured and evaluated against the performance obtained using the exhaustive scheme as performance difference (PD) (explained in Section 3.5). The difference in performance (PD) between the adaptive and exhaustive scheme is calculated by subtracting the performance obtained using the adaptive scheme from the performance obtained using the exhaustive scheme. The performance difference (PD) for a workload is calculated using Equation 5.1. As the two sets of workloads divided earlier into non-reactive and reactive therefore, the adaptive scheme is evaluated against exhaustive scheme for the two sets of workloads (i.e., systematic and random) with respect to the already categorized workloads i.e., non-reactive and reactive, separately.

$$PD = \frac{(P_{Int} - P_A)}{Total\ Number\ of\ Submitted\ SLAs} * 100 \tag{5.1}$$

Where

P_{Int} = Served Number of SLAs by Exhaustive (or Integrated) Scheme
P_A = Served Number of SLAs by Adaptive Scheme

Further, the number of times the exhaustive scheme is suggested by the adaptive scheme, in comparison to the pure exhaustive scheme to schedule the SLAs, is also observed against each workload of systematic and random sets. The percentage of times the simple scheme is suggested by adaptive scheme for a workload is computed using Equation 5.2.

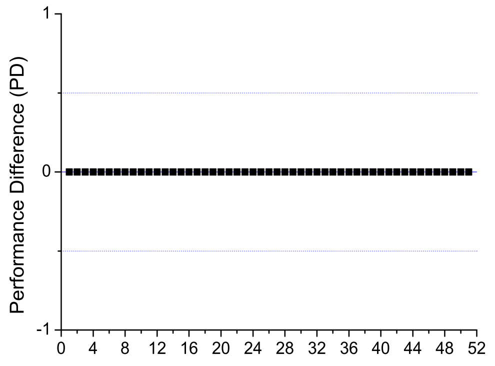
$$S_S = \frac{S_{E_{Int}} - S_{E_A}}{Total\ Number\ of\ Submitted\ SLAs} * 100 \tag{5.2}$$

Where

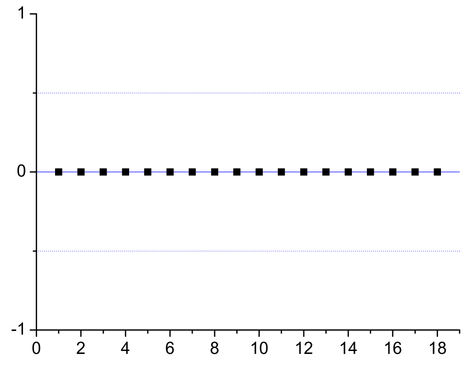
Total number of submitted SLAs (i.e., 1000 in each workload)
$S_{E_{Int}}$: Number of times integrated heuristic used by exhaustive scheme for each workload (which is 1000 for each workload).
S_{E_A}: Number of times exhaustive scheme (or integrated heuristic) suggested by Adaptive scheme
S_S: Reduction in suggestions of exhaustive scheme or simply the number of times simple scheme is used by adaptive scheme for a given workload

5.3.3 Observation and Evaluation

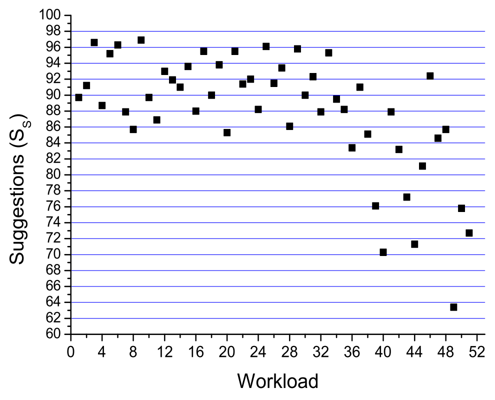
The observations for non-reactive and reactive workloads for the two sets (i.e., systematic and random) are presented in Figure 5.2 and 5.3, respectively. The performance difference (PD) for non-reactive workloads will be zero as the non-reactive workloads are not effected by any heuristic which can be seen in Figure 5.2(a) and 5.2(b) for systematic and random set, respectively. But the interesting observation is made for the percentage of suggestions for the simple scheme by the adaptive scheme for the non-reactive workloads of systematic and random set, which is given in Figure 5.2(c) and 5.2(d), respectively. In Figure 5.2(c), the percentage of suggestions is plotted against the 51 non-reactive workloads of systematic set. It can be seen that except for one workload, all workloads have been served at least 70.3% of times with simple scheme and if the percentage of suggestions is further increased to 81.1% than out of 51, 44 workloads have been served 81.1% or more times with simple scheme. The minimum percentage of suggestions for the simple scheme is observed to be 63.4% for the non-reactive workloads of systematic set. The observation for random set is depicted in Figure 5.2(d), it can be seen that the minimum percentage of suggestion is 72.3% for 18 non-reactive workloads of random set. Except for 4 workloads, the percentage of suggestion is greater than or equal to 81%. The average percentage of suggestions per workload for the non-reactive workloads of systematic and random sets is observed to be 87.87% and 86.26%, respectively.



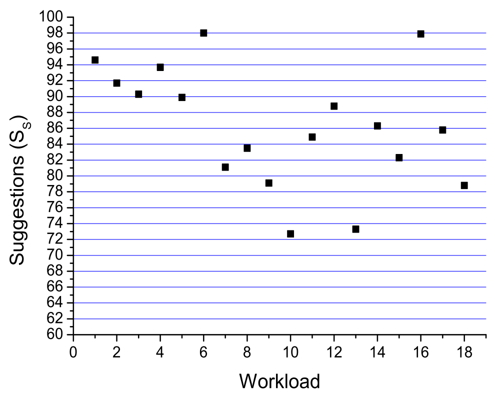
(a) PD under Systematic Set



(b) PD under Random Set



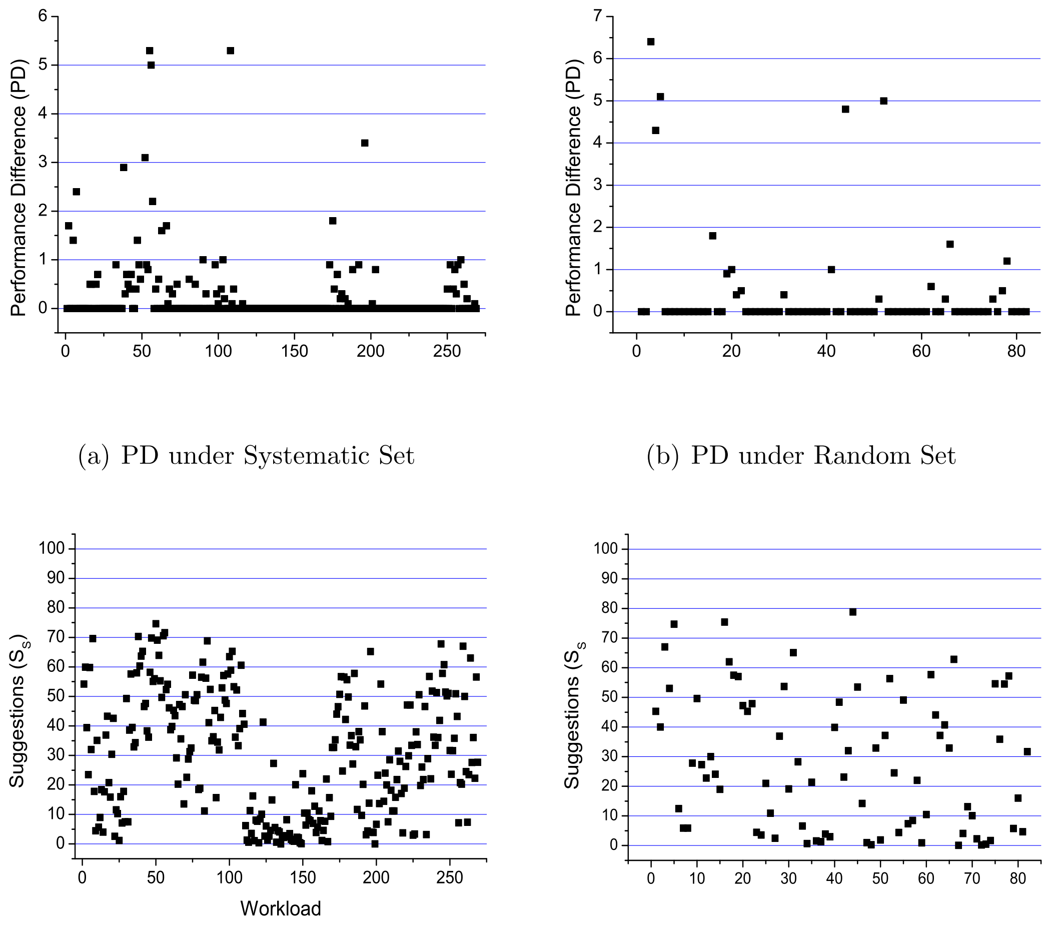
(c) Suggestions under Systematic Set



(d) Suggestions under Random Set

Figure 5.2: Evaluation of Adaptive Scheme For Non-Reactive Workloads

Similarly, the observations are made for the reactive workloads of the systematic and random sets. The performance difference and the percentage of suggestions for systematic set is presented in Figure 5.3(a) and 5.3(c), respectively. It has been observed that the maximum of 67.8% of suggestions are made for the simple scheme with performance difference equal to zero and 74.6% of suggestions are made with the performance difference of 7.2%. Further, for the random set, it has been observed that the maximum of 62% of the suggestions are made for the simple scheme with performance difference equal to zero and 67% of the suggestions are made with the performance difference of 6.4%. Overall, the average percentage of suggestions for reactive workloads of systematic and random



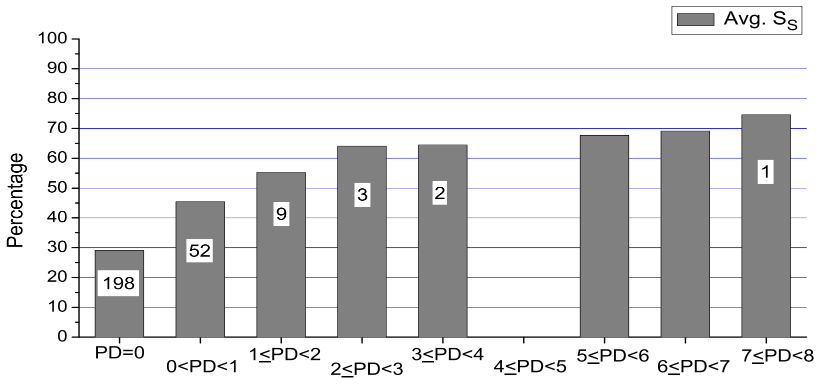
(a) PD under Systematic Set

(b) PD under Random Set

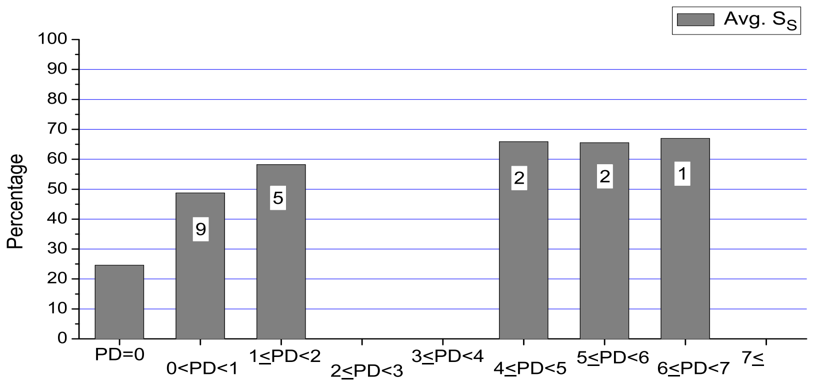
(c) Suggestions under Systematic Set

(d) Suggestions under Random Set

Figure 5.3: Evaluation of Adaptive Scheme For Reactive Workloads



(a) Suggestions under Systematic Set



(b) Suggestions under Random Set

Figure 5.4: Avg. Percentage of Suggestions against PD Thresholds For Reactive Workloads

set, is observed to be 34.48% and 31.82%, respectively.

The clear observation of average percentage of suggestions per workload, number of workloads against which the average is taken under different PD thresholds for the systematic and random set, is presented in Figure 5.4(a) and 5.4(b), respectively. From Figure 5.4(a), for the reactive workloads of systematic set, the average percentage of suggestions per workload with PD equal to zero is 29.06%. The average is calculated against 198 workloads as shown. Similarly, for the PD greater than zero & less than 1%, PD greater than equal to 1% & less than 2%, PD greater than equal to 2% & less than 3%, PD greater than equal to 3% & less than 4%, PD greater than equal to 5% & less than 6%, PD greater than equal to 6% & less than 7% and PD greater than equal to 7% & less than 8% are observed to be 45.37%, 55.15%, 64.06%, 64.5%, 67.6%, 69.11% and 74.61%, respectively that are taken against number of workloads 52, 9, 3, 2, 3, 1 and 1, respectively. No workload fall within the threshold range of PD greater than equal to 4% & less than 5%. Similarly, the observation for the reactive workloads of random set for average percentage of suggestions per workload and the number of workloads against which the average is taken for different PD thresholds, is shown in Figure 5.4(b). The average number of suggestions for simple scheme per workload with PD equal to zero is 24.6%. The average is calculated against 63 workloads as shown. Further, for the PD greater than zero & less than 1%, PD greater than equal to 1% & less than 2%, PD greater than equal to 4% & less than 5%, PD greater than equal to 5% & less than 6% and PD greater than equal to 6% & less than 7%, are observed to be 48.73%, 58.2%, 65.91%, 65.52% and 67%, respectively that are taken against number of workloads 9, 5, 2, 2 and 1, respectively. No workload fall within the threshold range of PD greater than equal to 2% & less than 3%, PD greater than equal to 3% & less than 4% and PD greater than equal to 7% & less than 8%.

5.3.4 Reason of Erroneous Suggestions

As mentioned above, in adaptive mode the quantification is performed on an arrival of an SLA into the system against the number of CPUs. Based on that quantification the decision to use exhaustive scheme or simple scheme is made by the adaptive scheme for the SLAs waiting to get scheduled. Non-reactive workloads should have minimal or ideally no suggestion for exhaustive scheme as they are the types of workloads that can be served by any heuristic without any performance difference. But as the quantification scheme does not act as an admission control and it quantifies the small subset of SLAs that are arrived into the system without considering executing SLAs therefore, it may be possible that the newly arrived SLAs unable to get scheduled but resulting in exhaustive scheme suggestion by the proposed adaptive scheme. For example, consider the jobs/SLAs given in Table 5.4 which is the modified example given in Table 1.1 discussed in Chapter 1. Assume that the job/SLA *E* is arrived into the system at time t=0 and system consists of 8 CPUs which at t=0 are all free. At this time system will accept the SLA *E* and schedule it to execute. Then, at time unit equals to 1 four new SLAs i.e., *A, B, C* and *D* arrive into the system. As quantification scheme does not have the knowledge of executing SLA *E* therefore, adaptive scheme based on the quantification values will suggest exhaustive scheme for the SLAs however, it can be seen that while SLA *E* is executing, none of the newly arrived

SLAs is able to execute successfully within their specified deadline. Hence, the conclusion may be derived from this observation is that consideration of executing SLAs within the quantification may reduce the false suggestions of using exhaustive scheme.

Table 5.4: Example 1 for Wrong Suggestions

SLA	Start Time (T_s)	Runtime (t_D)	No. of CPUs (N_{cpu})	Deadline Time (T_f)
E	0	6	8	3
A	1	1	8	6
B	1	2	5	5
C	1	8	2	10
D	1	2	3	6

Similarly, for reactive workloads, the scheme suggested wrongly for condition where already accepted SLAs require exhaustive scheme but as there is no room for the newly arrived SLA may be because of the already executing SLAs (as discussed above) or may be because of the already accepted SLAs. For example, consider the SLAs given in Table 5.5 which is again the modification of example Table 1.1 of Chapter 1. Again, assume that the job/SLA *E* is arrived into the system at time t=0 and system consists of 8 CPUs which at t=0 are all free. At this time system will accept the SLA *E* and schedule it to execute. Then, at time equals to 1 unit four new SLAs i.e., *A, B, C* and *D* arrive into the system. The adaptive scheme based on the quantification values will suggest exhaustive scheme for the SLAs to schedule and execute them successfully within their specified deadline as SLA *E* is finishing its execution at time unit 1.1. Further, at time 1.1 unit another SLA *F* is arrived into the system which is not going to schedule successfully in the presence of already accepted SLAs i.e., *A, B, C* and *D*. But to retrieve the best performance that is scheduling of four SLAs out of five adaptive scheme will again suggest for the exhaustive scheme as no simple heuristics will be able to schedule successfully the already accepted four SLAs which is right decision in absence of admission control related to the heuristics considered.

Table 5.5: Example 2 for Wrong Suggestions

SLA	Start Time (T_s)	Runtime (t_D)	No. of CPUs (N_{cpu})	Deadline Time (T_f)
E	0	1.1	8	3
A	1	1	8	6
B	1	2	5	5
C	1	8	2	10
D	1	2	3	6
F	1.1	6	8	10

Algorithm 6 Modified Quantification Scheme

```
1. Find maximum deadline time ($T_{f_{max}}$) of all waiting and newly arrived SLAs
2. Set start time ($T_s$) of all waiting SLAs to the current system time ($T_c$).
3. Allocate range to RU array i.e., RU[ $\lceil T_{f_{max}} - T_c \rceil$ ]
4. Set $RU[t] = 0$, for t = 0 to $(\lceil T_{f_{max}} - T_c \rceil) - 1$.
5. FOR each waiting and newly arrived SLA 'i' {
6.     Set $BusyCPU_{max}$ = 0
7.     FOR all executing SLAs 'e' {
8.         IF ($deadline_i - runtime_i \leq finishedTime_e$)
9.             $BusyCPU_{max}$ = $BusyCPU_{max}$ + $N_{CPU_e}$
10.    }
11.    IF ($S_{cpu} - BusyCPU_{max} \geq N_{cpu_i}$) {
12.        FOR t = $\lfloor T_{s_i} \rfloor$ to $T_{f_i}$ with unit time increment in t {
13.            $RU[t - T_c] = RU[t - T_c] + \frac{t_{D_i}}{T_{f_i} - T_{s_i}} * \frac{N_{cpu_i}}{S_{cpu}}$
14.        }
15.    }
16.    ELSE {
17.        Reject Newly Arrived SLA
18.        Use Last Calculated Schedule
19.        Exit
20.    }
21. $t_T = t_T + \frac{t_{D_i}}{T_{f_i} - T_{s_i}}$
22. }
23. Calculate Average Demand Utilization (avg(RU)) and flexibility (avg($t_T$))
```

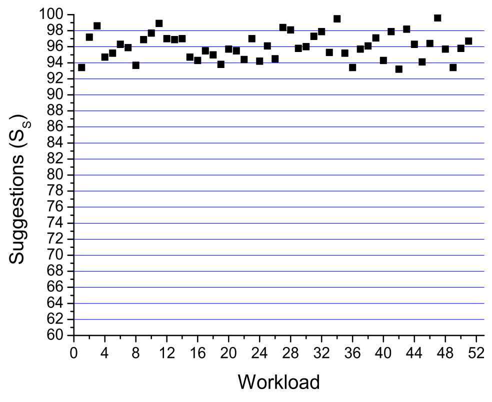
5.3.5 Modified Quantification Scheme

Based on the observation for non-reactive workloads, the quantification scheme is modified to consider the executing SLAs in order to get appropriate suggestions by adaptive scheme.The algorithm given in Algorithm 5 is modified and presented in Algorithm 6. Line 1 through 5 are same as discussed in Section 5.2 but from Line 6 to 10, the algorithm before performing the quantification, checks for the number of CPUs busy in executing the SLAs till the maximum possible start time of the candidate SLA 'i' i.e., $deadline_i - runtime_i$. If the finishing time of an executing SLA 'e' is greater than or equal to the maximum possible start time of SLA 'i' then, the number of CPUs the executing SLA is using are added into the variable $BusyCPU_{max}$ (Line 8 - 10). Once all executing SLAs are checked against the candidate SLA 'i' for the number of busy CPUs then, the quantification of $RU[t]$ is performed against the SLA 'i', provided that the difference between the total system CPUs (S_{cpu}) and total busy CPUs ($BusyCPU_{max}$) is greater than or equal to number of required processors (N_{cpu_i}) (Line 11 - 14). Otherwise, the newly arrived SLA is rejected with restoration of last calculated schedule and it exits without performing any further steps (Line 16 - 20). If not exited, then the tightness (or flexibility) is calculated (on Line 21) for each SLA 'i' and average demand utilization together with flexibility is calculated (on Line 23) based on which the adaptive scheme decides which scheduling scheme to use.

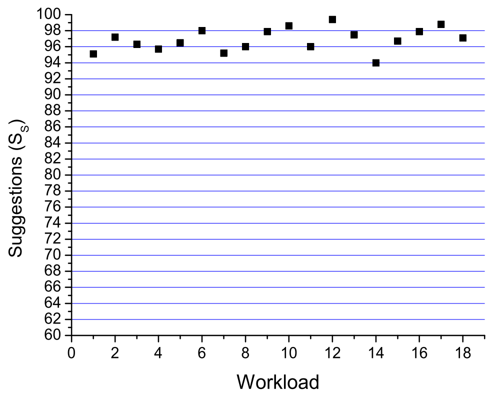
5.3.6 Evaluation of Modified Scheme

The observations for non-reactive and reactive workloads for the two sets (i.e., systematic and random) under modified scheme, are presented in Figure 5.5 and 5.6, respectively. The performance difference (PD) for non-reactive workloads will be zero as the non-reactive workloads are not effected by any heuristic which can be seen in Figure 5.2(a) and 5.2(b) for systematic and random set, respectively. But the interesting observation is made for the percentage of suggestions by adaptive scheme for simple scheme for the non-reactive workloads of systematic and random set, which is given in Figure 5.5(a) and 5.5(b), respectively. Figure 5.5(a), the percentage of suggestions for simple scheme for non-reactive workloads of systematic set, shows high number of suggestions under modified scheme if compared with Figure 5.2(c). From Figure 5.5(a), the highest observed percentage of suggestion is 99.6% and the least is 93.2%. The overall average percentage of suggestions per workload against the 51 non-reactive workloads of systematic set is observed to be 96.02%. Similarly for random set, the modified scheme suggested increased number of times for simple scheme as shown in Figure 5.5(b) in comparison to the observation shown in Figure 5.2(d). Figure 5.5(b) shows that the highest observed percentage of suggestion is 99.4% and the least is 94%. The overall average percentage of suggestions per workload against the 18 non-reactive workloads of random set is observed to be 96.88%.

Further, the observations are made for the reactive workloads of the systematic and random sets. The performance difference obtained under modified scheme between adaptive and exhaustive schemes, remains the same as shown above in Figure 5.3(a) amd 5.3(b). But modified scheme shows an improvement in terms of slight increased in number of simple scheme suggestions for both systematic and random set as shown in Figure 5.6(a) and 5.6(b), respectively. The overall average percentage of suggestions for reactive



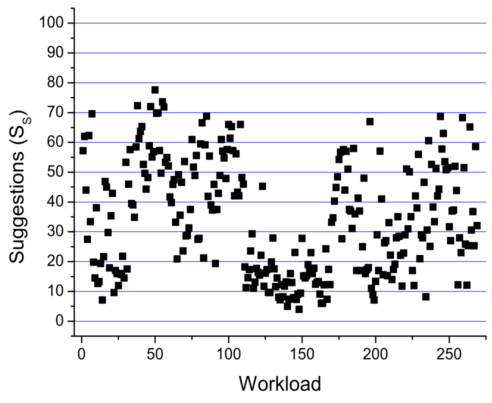
(a) Suggestions under Systematic Set



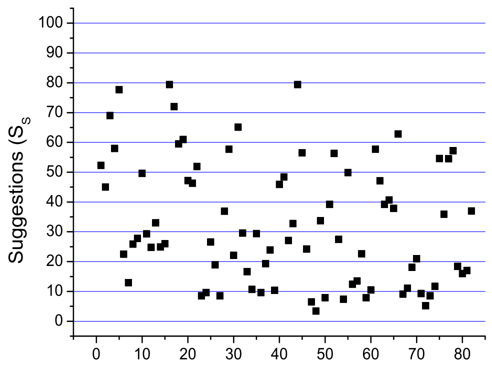
(b) Suggestions under Random Set

Figure 5.5: Evaluation of Adaptive Scheme For Non-Reactive Workloads

workloads of systematic and random set increased from 34.48% and 31.82% to 40.06% and 36.76%, respectively.



(a) Suggestions under Systematic Set



(b) Suggestions under Random Set

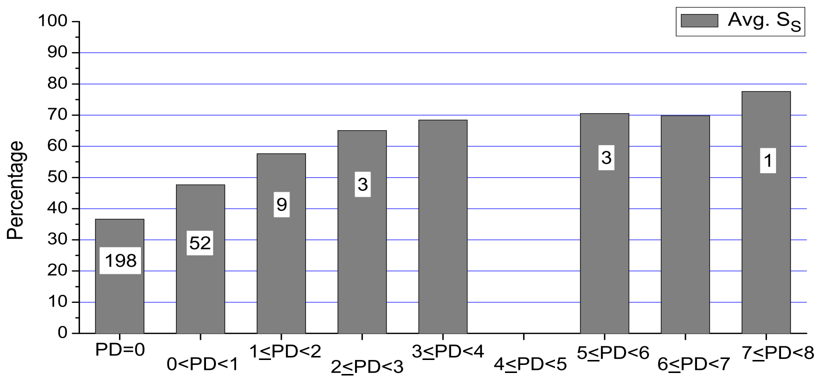
Figure 5.6: Evaluation of Adaptive Scheme For Reactive Workloads

Again, for clear observation, the average percentage of suggestions per workload, number of workloads against which the average is taken under different PD thresholds for the systematic and random set, is combined and presented in Figure 5.7(a) and 5.7(b), respectively. If the observations given in Figure 5.4(a), for the reactive workloads of systematic set without modified scheme is compared with the observation obtained under modified scheme which is presented in Figure 5.7(a) then, the increase in number of suggestions for each PD threshold is prominent. The average percentage of suggestions per workload with PD equal to zero without modification is 29.06% and under modified scheme it is observed to be 36.6%. The average is again calculated against 198 workloads as shown. Similarly, for the PD greater than zero & less than 1%, PD greater than equal to 1% & less than 2%, PD greater than equal to 2% & less than 3%, PD greater than equal to 3% & less than 4%, PD greater than equal to 5% & less than 6%, PD greater than equal to 6% & less than 7% and PD greater than equal to 7% & less than 8% are increased from 45.37%, 55.15%, 64.06%, 64.5%, 67.6%, 69.11% and 74.61%, respectively to 47.65%, 57.64%, 65.06%, 68.45%, 70.5%, 69.80% and 77.6%, respectively that are taken against number of workloads 52, 9, 3, 2, 3, 1 and 1, respectively. As mentioned earlier, no workload fall within the threshold range of PD greater than equal to 4% & less than 5%. Similarly, the observation under modified scheme for the reactive workloads of random set for average percentage of suggestions per workload and number of workloads against which the average is taken for different PD thresholds, is shown in Figure 5.7(b). Again, if this observation is compared with observation obtained without modified scheme given in Figure 5.4(b) is compared with Figure 5.7(b) then, slight increased in number of suggestions can be seen. The average number of suggestions for simple scheme per workload without modification with PD equal to zero observed around 24.6% and with modified scheme it is observed to be 30.5%. This average is calculated against 63 workloads as shown. Further, the suggestions observed increased for the PD greater than zero & less than 1%, PD greater than equal to 1% & less than 2%, PD greater than equal to 4% & less than 5%, PD greater than equal to 5% & less than 6% and PD greater than equal to 6% & less than 7%, from 48.73%, 58.2%, 65.91%, 65.52% and 67%, respectively to 50.84%, 59%, 68.70%, 67% and 69%, respectively under modified scheme. The averages are taken against number of workloads 9, 5, 2, 2 and 1, respectively. No workload fall within the threshold range of PD greater than equal to 2% & less than 3%, PD greater than equal to 3% & less than 4% and PD greater than equal to 7% & less than 8%.

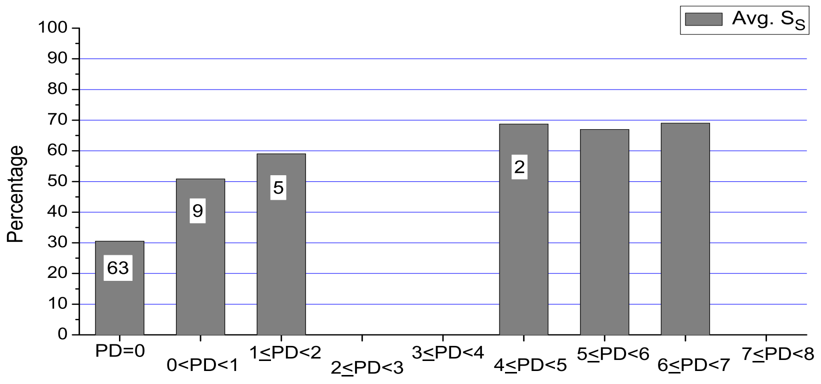
Table 5.6: *PD* Stats for Reactive Workloads

Reactive Workloads of	Average *PD*	Std. Dev. *PD*
Systematic Set	0.91%	0.302%
Random Set	1.26%	0.44%

Briefly, the average *PD* between exhaustive and adaptive schemes is observed to be 0.91% together with deviation of 0.30% for the reactive workloads of systematic set. Similarly, the average *PD* between exhaustive and adaptive schemes is observed to be 1.26% together with deviation of 0.44% for the reactive workloads of random set, as detailed in Table 5.6. Further, for non-reactive workloads, the adaptive scheme with modification on the average suggested the use of simple scheme for at least 96% of times per workload i.e., the reduction of 96% in computation cost by avoiding exhaustive scheme. For reactive workloads, it suggested simple scheme on the average 40.6% and 36.76% for systematic and random set, respectively. Overall, the maximum performance difference of 6.4% observed. Further, as observed and mentioned in Section 4.5 that the average time per workload taken by the exhaustive scheme to evaluate the workloads of systematic and random sets, is found to be 1162.046 minutes and 1062.376 minutes, respectively. The observed average time taken by the adaptive scheme with the modification to serve the systematic and random sets is, around 410.667 minutes and 388.617 minutes, respectively. If these simulation timings are compared with the simulation timings of the exhaustive scheme than it can be observed that adaptive scheme reduced the times approximately by 64% for both the sets.



(a) Suggestions under Systematic Set



(b) Suggestions under Random Set

Figure 5.7: Avg. Percentage of Suggestions against PD Thresholds For Reactive Workloads

5.4 Summary

In this chapter, the adaptive quantification scheme to schedule the incoming SLAs, discussed and evaluated. It uses the SLA quantification scheme and workload classification scheme investigated and observed in Chapter 4. The difference from the last chapter work is, adaptive scheme does the quantification based on the current load and the state of the system. On an event, the quantification of $RU[t]$ is done against the number of resources if a newly arrived SLA can be scheduled without overrunning its deadline in comparison to executing SLAs. The quantification is performed when an SLA is arrived into the system. Once the quantification is done then the metric is calculated together with the average tightness ($avg(t_T)$) of all the candidate SLAs. Then, the threshold range values are checked against to finally suggest the suitability of the scheduling of the SLAs using either simple or exhaustive scheme.

The adaptive scheme has been evaluated against the exhaustive scheme for the performance difference (PD) together with the percentage of the times the adaptive scheme suggested the use of simple scheme. The evaluation has been done for the two types of workloads i.e., non-reactive and reactive of two sets (i.e., systematic and random), separately, in order to check the reduction in computation cost for the two types of workloads as well as performance difference, clearly. As the quantification scheme does not act as an admission control system and do not treat the executing SLAs at the time of quantification therefore, it resulted in some erroneous suggestions for exhaustive scheme. To avoid such erroneous suggestions, a condition where none of the newly arrived SLAs are able to execute successfully within their specified deadline due to the executing SLAs the modification made to the adaptive scheme to reduce the false suggestions.

It has been observed that on an average the percentage of suggestion per workload for simple scheme for non-reactive workloads of systematic and random sets, observed to be 96.02% and 96.88%, respectively. Further, for the reactive workloads, the overall average percentage of suggestion for simple scheme for systematic and random sets, observed to be 40.6% and 36.76%, respectively.

The simulation running times have also been observed for the adaptive scheme and exhaustive scheme and it has been found that the adaptive scheme reduced the time of the simulation on an average by 64%.

Chapter 6

Conclusion & Future Work

This chapter summarizes the thesis in Section 6.1, points out its limitations in Section 6.2 and finally lays out the directions for the future work in Section 6.3.

6.1 Summary of the Thesis

The ultimate aim of this thesis is to investigate and build a light weight solution to characterize the SLA based workload of parallel-independent-jobs, using the calculated values of proposed metric for the optimum performance of the system. The performance criterion is the number of jobs/SLAs served for whole of the work of this thesis. It has been discussed in the work [YS06] that the optimum performance of a system against a workload can be achieved by calculating the priority of the jobs within a workload using the combination of different *service level agreement (SLA)* parameters and associating each parameter with a specific weight to form an integrated heuristic. Those weights are then calculated by sweeping the parameter/weight values in order to obtain the best performance of the system under the workload. This scheme is called an exhaustive scheme. The exhaustive scheme observed to be computationally expensive and without simulation there is no way to predict the weights to yield the best results directly [HSR09]. Therefore, there was a need of identification of the suitability of the exhaustive scheme for the arrived SLAs. This need has been addressed through the proposed metric based classification scheme which then integrated into a scheduling scheme to construct an adaptive scheduling scheme. The idea is, embedding of such characterization scheme into a meta-scheduler or to a local scheduler of the site which allows them to evaluate the arriving SLAs and check whether the exhaustive scheme will be a fruitful choice or not, without extensive computation. The computation time taken by the exhaustive scheme and adaptive scheme has also been observed and found that adaptive Scheme reduced the computation time by three folds. The major observations and results obtained in this thesis include:

- In Chapter 2, a review of different scheduler and resource manager presented and classified based on the discussed taxonomy. It also discussed the Service Level Agreement (SLA) life-cycle and it's structure to give an overview of SLA support, requirements and complexities with respect to the different resource managers.

- The number of served SLAs are observed under simple heuristics (or simply simple scheme) and integrated heuristic (or simply exhaustive scheme). The observation is then mapped against the average of RU ($avg(RU)$) and average tightness $avg(t_T)$ to identify the effectiveness of exhaustive scheme in Chapter 4. It has been observed that for the different range of $avg(RU)$ values, the average tightness ($avg(t_T)$) value is also required to check separately for the SLAs within a workload. Hence, different range of values for $avg(RU)$ and $avg(t_T)$ are observed to figure when to use exhaustive scheme to serve maximum number of SLAs. Therefore, for the range of $avg(RU)$ values for the SLAs within a workload, greater than equal to 3.4, the $avg(t_T)$ must be less than or equal to 0.99 in order to be evaluated with the exhaustive scheme. Similarly, for the range of $avg(RU)$ values between 0.38 and 0.62, the $avg(t_T)$ must be less than or equal to 0.93 in order to be evaluated with the exhaustive scheme. For the range of $avg(RU)$ values for the arrived SLAs between 0.38 and 0.62, the $avg(t_T)$ must be less than or equal to 0.86 in order to evaluate the SLAs with the exhaustive scheme. Outside these ranges, the workload/SLAs is suggested as non-reactive or to be evaluated with the simple scheme. Therefore, two classes of workloads created based on the served number of SLAs under simple and exhaustive schemes. If the number of served SLAs for a workload is same under both the schemes then the workload considered as non-reactive workload otherwise the workload considered as reactive. As the performance observed in terms of number of served SLAs hence, the difference in performance obtained under both the schemes for a workload, has been considered as performance measurement metric called performance difference (PD). Based on the classification, the proposed scheme evaluated against different sets of workloads and it showed that the percentage of correct identification is around 97% using the proposed scheme. Also, the proposed scheme completely avoids the need of simulating the computing scenario to find the usefulness of the integrated heuristic for a set of SLAs.

- Based on the observed results of Chapter 4, a novel approach called *adaptive scheduling of SLAs*, is proposed and evaluated in Chapter 5. It quantifies the arrived SLA together with the existing workload of SLAs in order to check whether to use exhaustive scheme or simple scheme. It has been observed that on an average the percentage of suggestion per workload for simple scheme for non-reactive workloads has been 96%. Further, for the reactive workloads, the overall average percentage of suggestion for simple scheme is around 36%. The simulation running times have also been observed for the adaptive scheme and exhaustive scheme and it has been found that the adaptive scheme reduced the time of the simulation on an average by 64% with the maximum performance difference between adaptive scheduling and exhaustive scheme of 6.4%.

Overall, The proposed identification approach has avoided the requirement of simulating the complete computing scenario in order to find whether the parameter sweeping is fruitful or not.

6.2 Limitations

Although, the efficiency and accuracy of the proposed scheme has been evaluated extensively under simulated environment but there are some limitations either due to the assumptions made or because of the nature of the problem. It is believed that the schemes can be further improved if some limitations in the following aspects can be addressed.

- The main focus of the work is on the scheduling aspect of the SLA based parallel jobs and therefore considered and evaluated extensively under an environment consists of 8 CPUs.
- The proposed scheme considered only the scheduling of SLAs onto a single site and has not considered the effect of *spanning of an SLA* on more than one site.
- It is assumed that the mentioned runtime of a job within an SLA is always exact therefore, the risk of breaking up of a calculated schedule for a workload because of the *inexact runtime*, is not considered.
- The evaluation has been carried out in the simulation environment developed with assumption that the system will have *homogeneous processing elements*. Also, it has been assumed that the resources will always be *available* without any disruption. Although, the quantification scheme may adjust itself on the available number of resources but the evaluation with respect to the dynamic environment is an extensive process.
- The only performance metric used during the complete research is *number of served SLAs*. Therefore, the best heuristics $H = min(T_f + w_1A + w_2t_L)$ of the work [YS06] for the parameter sweep scheme has been taken into an account and not considered the other best heuristics such as for the revenue aspect.

6.3 Future Work

Based on the limitations recognized in Section 6.2, the work in this thesis can be extended in the following directions:

- Although, only 8 CPU environment considered throughout in the research but it could easily be extended for number of CPUs. As mentioned in Section 4.5 (of Chapter 4), on an average the simulation time to evaluate a workload observed to be around 1100 minutes which means extensive evaluation cost will be higher in terms of simulation time.
- In order to observe the effect of *spanning of an SLA* on more than one site, the integrated heuristic may need to be modified to reflect delay/communication cost as well as the proposed metric.
- The risk of breaking up of a calculated schedule due to inexact runtime estimates within an SLA is not considered and hence, it can be observed against all the schemes which allows to extend the work further towards "how to" incorporate that risk into the calculation of proposed metric (RU).

- The effect of non-homogeneous processing elements and unavailability of the resource(s) can be observed for the scheduling heuristics and as well as for the proposed identification schemes. Again, the integrated heuristic and proposed metric may need to be modified to reflect processing capability of resources.

- The proposed metric RU may further be explored to identify the suitable simple scheduling heuristics (i.e., earliest deadline first, least laxity first & minimum jobsize first and their counter parts latest deadline first, highest laxity first & highest jobsize first heuristics) for incoming workload, in case of not using the integrated heuristic at all.

- A knowledge base may be created to keep the parameter values (i.e., w_1 and w_2) of integrated heuristic found against certain characteristics of SLA based workload and characteristics may then be computed using the proposed metric (RU). This will further reduce the parameter sweep computation by matching the stored characteristics with the incoming workload characteristics and if match is found then the stored values of parameters may then be used instead of evaluating integrated heuristic from scratch. The investigation may be done in order to find how good the metric is to compute the characteristics of a workload for parameter values.

- The criterion of performance measurement may be extended to the utilization of the system and/or to the economical aspect such as revenue extraction from the workload. Based on the performance criterion, further heuristics from the work [YS06] may be evaluated.

- The proposed scheme may be used to transform a workload into number of non-reactive sub-workloads using the metric (RU) values, so that any simple scheduling heuristic can be applied on each sub-workloads. This could be used under multi-site environment where workload is divided into different sub-workloads and then each sub-workload may then be distributed between the associated computing sites.

Bibliography

[AaRW+02] R. J. Al-ali, O. F. Rana, D. W. Walker, S. Jha, and S. Sohail. G-QoSM: Grid service discovery using QoS properties. *Computing and Informatics Journal, Special Issue on Grid Computing*, 21(4):pages 363–382, 2002.

[AB05] N. Abdennadher and R. Boesch. Towards a Peer-To-Peer Platform for High Performance Computing. In *Proceedings of the Eighth International Conference on High-Performance Computing in Asia-Pacific Region*, HPCASIA '05, pages 354–361. IEEE Computer Society, 2005.

[ACD+04] A. Andrieux, K. Czajkowski, A. Dan, K. Keahey, H. Ludwig, J. Pruyne, J. Rofrano, S. Tuecke, and M. Xu. *Web Services Agreement Specification (WS-Agreement)*, pages 1–47. Number GFD.107. The Global Grid Forum (GGF), 2004.

[AD09] D. Armstrong and K. Djemame. Towards Quality of Service in the Cloud. In *Proceedings of the 25th UK Performance Engineering Workshop, Leeds, UK*, pages 226–240, 2009.

[AEA08] D. Abramson, C. Enticott, and I. Altinas. Nimrod/K: towards massively parallel dynamic grid workflows. In *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, SC '08, pages 24:1–24:11, Piscataway, NJ, USA, 2008. IEEE Press.

[AEBCDF+] A. Anjomshoaa (EPCC), F. Brisard (CA), M. Drescher (Fujitsu), D. (UoM) Fellows, A. Ly (CA), S. McGough (LeSC), D. Pulsipher (Ovoca LLC), and A. Savva (Fujitsu). GFD-R.056: Job submission description language (JSDL) specification. http://www.ggf.org/documents/GFD.56.pdf [20-Dec-2009].

[AFAS11] A. Al Falasi and M. Adel Serhani. A framework for SLA-based cloud services verification and composition. In *Proceedings of the 7th International*

Conference on Innovations in Information Technology (IIT), pages 287–292. IEEE, April 2011.

[AHR01] D. Adcock, A. Halborg, and C. Ross. *Marketing Principles & Practice (4th edition)*. Financial Times / Pearson Education, 2001.

[AJY10] R. Albodour, A. James, and N. Yaacob. An extension of GridSim for quality of service. In *Proceedings of the 14th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*, pages 361–366, April 2010.

[ALPF01] D. Abramson, A. Lewis, T. Peachey, and C. Fletcher. An automatic design optimization tool and its application to computational fluid dynamics. In *Proceedings of the 2001 ACM/IEEE conference on Supercomputing (CDROM)*, Supercomputing '01, pages 25–31. ACM, 2001.

[ASGH95] D. Abramson, R. Sosic, J. Giddy, and B. Hall. Nimrod: a tool for performing parametrised simulations using distributed workstations. In *Proceedings of the Fourth IEEE International Symposium on High Performance Distributed Computing*, pages 112–121. IEEE Computer Society, 1995.

[ask] Askalon programming environment for grid computing. http://www.dps.uibk.ac.at/projects/askalon/ [10-AUG-2008].

[ass] AssessGrid: project summary. http://www.assessgrid.eu/index.php?id=316 [1-NOV-2008].

[BAG00] R. Buyya, D. Abramson, and J. Giddy. Nimrod/G: An architecture for a resource management and scheduling system in a global computational grid. In *Proceedings of The Fourth International Conference/Exhibition on High Performance Computing in the Asia-Pacific Region*, volume 1, pages 283–289. IEEE Computer Society, 2000.

[BCC$^+$04] D. Box, E. Christensen, F. Curbera, D. Ferguson, J. Frey, M. Hadley, C. Kaler, D. Langworthy, F. Leymann, B. Lovering, et al. Web services addressing (WS-Addressing). W3C Member Submission (http://www.w3.org/Submission/ws-addressing/), 2004.

[BCT+05] B. Benatallah, F. Casati, P. Traverso, M. Aiello, G. Frankova, and D. Malfatti. What's in an Agreement? An Analysis and an Extension of WS-Agreement. In *Service-Oriented Computing - ICSOC*, volume 3826 of *Lecture Notes in Computer Science*, pages 424–436. Springer Berlin / Heidelberg, 2005.

[BFH03] F. Berman, G. C. Fox, and A. J. G. Hey, editors. *Grid Computing - Making the Global Infrastructure a Reality.* Wiley and Sons, January 2003.

[BMG+09] M.G. Bugeiro, J.C. Mourino, A. Gomez, C. Vaquez, E. Huedo, I.M. Llorente, and D.A. Rodriguez-Silva. Integration of SLAs with GridWay in BEinEIMRT project. In *Proceedings of Third Iberian Grid Infrastructure Conference*. Netbiblo, May 2009.

[BSGA01] R. Buyya, H. Stockinger, J. Giddy, and D. Abramson. Economic models for management of resources in grid computing. *Concurrency and computation: practice and experience*, 14(13-15):1507–1542, 2001.

[BTSRB08] P. Balakrishnan, S. Thamarai Selvi, and G. Rajesh Britto. Service Level Agreement Based Grid Scheduling. In *IEEE International Conference on Web Services (ICWS)*, pages 203–210. IEEE, 2008.

[BWF+96] F. D. Berman, R. Wolski, S. Figueira, J. Schopf, and G. Shao. Application-level scheduling on distributed heterogeneous networks. In *Proceedings of ACM/IEEE conference on Supercomputing (CDROM)*, Supercomputing '96, pages 39–66. IEEE Computer Society, 1996.

[CBP+07] G. Capannini, R. Baraglia, Diego Puppin, L. Ricci, and Marco Pasquali. A job scheduling framework for large computing farms. In *Supercomputing, 2007. SC '07. Proceedings of the 2007 ACM/IEEE Conference on*, pages 1–10, 2007.

[CC00] B. Chun and D. Culler. REXEC: A decentralized, secure remote execution environment for clusters. In Babak Falsafi and Mario Lauria, editors, *Network-Based Parallel Computing. Communication, Architecture, and Applications*, volume 1797 of *Lecture Notes in Computer Science*, pages 1–14. Springer Berlin / Heidelberg, 2000.

[CCMW00] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web services description language (WSDL) 1.0. http://xml.coverpages.org/wsdl20000929.html, September 2000.

[CFF$^+$04a] K. Czajkowski, D. Ferguson, I. Foster, J. Frey, S. Graham, T. Maguire, D. Snelling, and S. Tuecke. From open grid services infrastructure to ws-resource framework: Refactoring & evolution. Globus Alliance (http://www.globus.org/), 2004.

[CFF$^+$04b] K. Czajkowski, D.F. Ferguson, I. Foster, J. Frey, S. Graham, I. Sedukhin, D. Snelling, S. Tuecke, and W. Vambenepe. The WS-Resource Framework, Version 1.0. *Initial draft release from Globus Alliance (https://www.globus.org/wsrf/specs/ws-wsrf.pdf)*, Vol.3(No.05), 2004.

[CJSZ08] L. Canon, E. Jeannot, R. Sakellariou, and W. Zheng. Comparative Evaluation Of The Robustness Of DAG Scheduling Heuristics. In S. Gorlatch, P. Fragopoulou, and T. Priol, editors, *Grid Computing*, pages 73–84. Springer US, 2008.

[CKKG99] S. Chapin, D. Katramatos, J. Karpovich, and A. Grimshaw. The Legion Resource Management System. In D. Feitelson and L. Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, volume 1659 of *Lecture Notes in Computer Science*, pages 162–178. Springer Berlin / Heidelberg, 1999.

[CL06] D. Chappell and L. Liu. Web services brokered notification 1.3 (WS-BrokeredNotification). *OASIS Standard wsn-ws_topics-1.3-spec-os*, 1, 2006.

[CMRW07] R. Chinnici, J.J. Moreau, A. Ryman, and S. Weerawarana. Web services description language (WSDL) version 2.0 part 1: Core language. *W3C Recommendation*, 26, 2007.

[csf] Community Scheduler Framework (CSF). http://www.globus.org/grid_software /computation/csf.php [20-DEC-2009].

[csf03] Open source metascheduling for virtual organizations with the Community Scheduler Framework (CSF)-(White Paper). http://www.cs.virginia.edu/g̃rimshaw/CS851-2004/Platform/CSF_architecture.pdf [20-DEC-2009], 2003.

[DGP$^+$08] K. Djemame, I. Gourlay, J. Padgett, G. Birkenheuer, M. Hovestadt, K. Voss, and O. Kao. Using WS-Agreement for Risk Management in the Grid. In *First WS-Agreement Workshop (OGF18)*. Open Grid Forum (OGF), 2008.

[DOB$^+$00] H. Dail, G. Obertelli, F. Berman, R. Wolski, and A. Grimshaw. Application-aware scheduling of a magnetohydrodynamics application in the Legion metasystem. In *Proceedings of 9th Heterogeneous Computing Workshop, (HCW '00)*, pages 216–228. IEEE, 2000.

[FCF$^+$05] I. Foster, K. Czajkowski, D.E. Ferguson, J. Frey, S. Graham, T. Maguire, D. Snelling, and S. Tuecke. Modeling and managing state in distributed systems: The role of OGSI and WSRF. *Proceedings of the IEEE*, 93(3):604–612, March 2005.

[FF05] E. Frachtenberg and D. Feitelson. Pitfalls in Parallel Job Scheduling Evaluation. In D. Feitelson, E. Frachtenberg, L. Rudolph, and U. Schwiegelshohn, editors, *Job Scheduling Strategies for Parallel Processing*, volume 3834 of *Lecture Notes in Computer Science*, pages 257–282. Springer Berlin / Heidelberg, 2005.

[FGG$^+$04] I. Foster, J. Gieraltowski, S. Gose, N. Maltsev, E. May, A. Rodriguez, D. Sulakhe, A. Vaniachine, J. Shank, S. Youssef, D. Adams, R. Baker, W. Deng, J. Smith, D. Yu, I. Legrand, S. Singh, C. Steenberg, Y. Xia, A. Afaq, E. Berman, J. Annis, L.A.T. Bauerdick, M. Ernst, I. Fisk, L. Giacchetti, G. Graham, A. Heavey, J. Kaiser, N. Kuropatkin, R. Pordes, V. Sekhri, J. Weigand, Y. Wu, K. Baker, L. Sorrillo, J. Huth, M. Allen, L. Grundhoefer, J. Hicks, F. Luehring, S. Peck, R. Quick, S. Simms, G. Fekete, J. VandenBerg, K. Cho, K. Kwon, D. Son, H. Park, S. Canon, K. Jackson, D.E. Konerding, J. Lee, D. Olson, I. Sakrejda, B. Tierney, M. Green, R. Miller, J. Letts, T. Martin, D. Bury, C. Dumitrescu, D. Engh, R. Gardner, M. Mambelli, Y. Smirnov, J. Voeckler, M. Wilde, Y. Zhao, X. Zhao, P. Avery, R. Cavanaugh, B. Kim, C. Prescott, J. Rodriguez, A. Zahn, S. McKee, C. Jordan, J. Prewett, T. Thomas, H. Severini, B. Clifford, E. Deelman, L. Flon, C. Kesselman, G. Mehta, N. Olomu, K. Vahi, K. De, P. McGuigan, M. Sosebee, D. Bradley, P. Couvares, A. De Smet, C. Kireyev, E. Paulson, A. Roy, S. Koranda, B. Moe, B. Brown, and P. Sheldon. The Grid2003 production grid: principles and practice. In *Proceedings of 13th IEEE International Symposium on High performance Distributed Computing, 2004.*, pages 236–245. IEEE Computer Society, 2004.

[FGNC01] G. Fedak, C. Germain, V. Neri, and F. Cappello. XtremWeb: a generic global computing system. In *Proceedings of First IEEE/ACM International Symposium on Cluster Computing and the Grid, 2001.*, pages 582–587, 2001.

[FK96] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *International Journal of Supercomputer Applications*, 11:115–128, 1996.

[FKL$^+$99] I. Foster, C. Kesselman, C. Lee, B. Lindell, K. Nahrstedt, and A. Roy. A distributed resource management architecture that supports advance reservations and co-allocation. In *Proceedings of Seventh International Workshop on Quality of Service, 1999. (IWQoS '99)*, pages 27–36. IEEE, 1999.

[FKNT04] I. Foster, C. Kesselman, J. Nick, and S. Tuecke. The physiology of the grid: An open grid services architecture for distributed systems integration. *Globus Project Online at: http://www.globus.org/research/papers/ogsa.pdf*, 2004.

[Fos01] I. Foster. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. In *Euro-Par 2001 Parallel Processing*, volume 15 of *Lecture Notes in Computer Science*, pages 200–222. Springer Berlin / Heidelberg, 2001.

[Fos06] I. Foster. Globus Toolkit Version 4: Software for Service-Oriented Systems. *Journal of Computer Science and Technology*, 21:513–520, 2006.

[FRJ$^+$01] D. G. Feitelson, L. Rudolph, D. Jackson, Q. Snell, and M. Clement. Core Algorithms of the Maui Scheduler. In *Job Scheduling Strategies for Parallel Processing*, volume 2221 of *Lecture Notes in Computer Science*, pages 87–102. Springer Berlin / Heidelberg, 2001.

[FTL$^+$02] J. Frey, T. Tannenbaum, M. Livny, I. Foster, and S. Tuecke. Condor-G: A Computation Management Agent for Multi-Institutional Grids. In *Cluster Computing*, volume 5, pages 237–246. Springer Netherlands, 2002.

[GB11] S. K. Garg and R. Buyya. *Market-Oriented Resource Management and Scheduling: A Taxonomy and Survey*, pages 277–305. John Wiley & Sons, Ltd, 2011.

[Gen01] W. Gentzsch. Sun Grid Engine: towards creating a compute power grid. In *Proceedings of First IEEE/ACM International Symposium on Cluster Computing and the Grid.*, pages 35–36. IEEE, 2001.

[GHM06] S. Graham, D. Hull, and B. Murray. Web services base notification (WS-BaseNotification). *OASIS standard Publish Online at: http://docs.oasis-open.org/wsn/wsn-ws_base_notification-1.3-spec-pr-02.pdf*, 2006.

[gli] gLite - introduction. http://glite.cern.ch/introduction [6-FEB-2013].

[gloa] Globus toolkit version 5 (GT5). http://www.globus.org/toolkit/about.html [20-DEC-2009].

[glob] The WS-Resource framework. http://www.globus.org/wsrf/ [22-JAN-2013].

[GNC$^+$04] S. Graham, P. Niblett, D. Chappell, A. Lewis, N. Nagaratnam, J. Parikh, S. Patil, S. Samdarshi, I. Sedukhin, D. Snelling, et al. Web services topics (WS-Topics 1.2). OASIS Standard Publish Online at: http://docs.oasis-open.org/wsn/2004/06/wsn-WS-Topics-1.2-draft-01.pdf, 2004.

[GP99] J. Gehring and T. Preiss. Scheduling a metacomputer with uncooperative sub-schedulers. In D. Feitelson and L. Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, volume 1659 of *Lecture Notes in Computer Science*, pages 179–201. Springer Berlin / Heidelberg, 1999.

[gri] GridWay. Online at: http://www.gridway.org/doku.php?id=start [12-DEC-2009].

[GT04] S. Graham and J. Treadwell. Web services resource properties 1.2 (WS-ResourceProperties). *OASIS Standard - Draft Publish Online at: http://docs.oasis-open.org/wsrf/wsrf-ws_resource_properties-1.2-spec-pr-02.pdf*, 2004.

[GWTLT97] A. S. Grimshaw, W. A. Wulf, and CORPORATE The Legion Team. The Legion vision of a worldwide virtual computer. *Communication of the ACM*, 40:39–45, January 1997.

[HML10] E. Huedo, R.S. Montero, and I.M. Llorente. Grid Architecture from a Metascheduling Perspective. *Computer, IEEE*, 43(7):51–56, 2010.

[HQK$^+$06] P. Hasselmeyer, C. Qu, B. Koller, L. Schubert, and P. Wieder. Towards Autonomous Brokered SLA Negotiation. *Exploiting the Knowledge Economy - Issues Vol(7), Applications, Case Studies (Proc. of the eChallenges e-2006 Conference), IOS Press*, 2006.

[HSR09] J. Heilgeist, T. Soddemann, and H. Richter. Distributed decision making for metaschedulers. Technical report, Institut fur Informatik, Clausthal, Fraunhofer Publica [http://publica.fraunhofer.de/oai.har] (Germany), 2009.

[ibm] IBM LoadLeveler - SP Parallel Programming Workshop. http://www.itservices.hku.hk/sp2/workshop/html/loadleveler/LoadLeveler.html [10-JAN-2010].

[java] Class ArrayIndexOutOfBoundsException - Java^{TM} Platform Standard Ed. 6. http://docs.oracle.com/javase/6/docs/api/java/lang/ArrayIndexOutOfBo_ undsException.html [22-JAN-2013].

[javb] Oracle Java^{TM} Platform Standard Ed. 6. http://www.oracle.com/technetwork/java/javase/overview/index.html [22-JAN-2013].

[javc] Oracle Troubleshooting Guide for HotSpot VM. http://docs.oracle.com/javase/7/docs/webnotes/tsg/TSG-VM/html/memleaks.html#gbyvi [22-JAN-2013].

[Joh06] B. Johannes. Scheduling parallel jobs to minimize the makespan. *Journal of Scheduling*, 9:433–452, 2006.

[JPF$^+$97] J. Jann, P. Pattnaik, H. Franke, F. Wang, J. Skovira, and J. Riordan. Modeling of workload in mpps. In *Job Scheduling Strategies for Parallel Processing*, pages 95–116. Springer, 1997.

[KBM02] K. Krauter, R. Buyya, and M. Maheswaran. A taxonomy and survey of grid resource management systems for distributed computing. *International Journal of Software: Practice and Experience (SPE)*, 32(2):135–164, May 2002.

[KL03] A. Keller and H. Ludwig. The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services. *Journal of Network and Systems Management*, 11:57–81, 2003.

[Kol12] J. Kolodziej. Scheduling problems in hierarchical grid environment. In *Evolutionary Hierarchical Multi-Criteria Metaheuristics for Scheduling in Large-Scale Grid Systems*, volume 419 of *Studies in Computational Intelligence*, pages 3–18. Springer Berlin Heidelberg, 2012.

[Kub11] R. Kubert. Service Level Agreements for Job Control in Grid and High-Performance Computing. In N. P. Preve, editor, *Grid Computing*, Computer Communications and Networks, pages 205–221. Springer London, 2011.

[leg] Legion: A worldwide virtual computer. http://legion.virginia.edu/ [10-FEB-2009].

[LGF^{+}06] E. Laure, C. Gr, S. Fisher, A. Frohner, P. Kunszt, A. Krenek, O. Mulmo, F. Pacini, F. Prelz, J. White, M. Barroso, P. Buncic, R. Byrom, L. Cornwall, M. Craig, A. Di Meglio, A. Djaoui, F. Giacomini, J. Hahkala, F. Hemmer, S. Hicks, A. Edlund, A. Maraschini, R. Middleton, M. Sgaravatto, M. Steenbakkers, J. Walk, and A. Wilson. Programming the Grid with gLite. *Computational Methods in Science and Technology.*, 12(1):33–45, 2006.

[LGFH95] G. Lindahl, A. Grimshaw, A. Ferrari, and K. Holcomb. Metacomputing - What's in it for me? (White paper). *Legion Technical Papers [http://legion. virginia. edu/papers. html]*, 1995.

[LHP^{+}04] E. Laure, F. Hemmer, F. Prelz, S. Beco, S. Fisher, M. Livny, L. Guy, M. Barroso, P. Buncic, Peter Z. Kunszt, A. Di Meglio, A. Aimar, A. Edlund, D. Groep, F. Pacini, M. Sgaravatto, and O. Mulmo. Middleware for the next generation grid infrastructure. *Computing in High Energy and Nuclear Physics, Interlaken, CERN - Switzerland.*, pages 4–7, 2004.

[Liu05] C. K. Liu. First look at WSDL 2.0. Publish Online at: http://www.sdn.sap.com/irj/scn/go/portal/prtroot/docs/library/uuid/74b-ae690-0201-0010-71a5-9da49f4a53e2?QuickLink=index&0overridelayout=true&05003637731092 [19-JAN-2013], 2005.

[LKA^{+}08] A. Litke, K. Konstanteli, V. Andronikou, S. Chatzis, and T. Varvarigou. Managing service level agreement contracts in OGSA-based Grids. *Future Generation Computer Systems*, 24(4):245 – 258, 2008.

[LLM88] M.J. Litzkow, M. Livny, and M.W. Mutka. Condor - A hunter of idle workstations. In *Proceedings of 8th International Conference on Distributed Computing Systems, 1988.*, pages 104–111, 1988.

[LM06] L. Liu and S. Meder. Web services base faults 1.2 (WS-BaseFaults). *OASIS Standard - Publish Online at: http://docs.oasis-open.org/wsrf/wsrf-ws_base_faults-1.2-spec-os.pdf*, 2006.

[LRA^{+}04] K. Lai, L. Rasmusson, E. Adar, L. Zhang, and B. A. Huberman. Tycoon: an Implementation of a Distributed, Market-based Resource Allocation System. *Multiagent and Grid Systems*, 1:169–182, January 2004.

[lsf98] LSF Programmer's Guide, Fourth Edition August 1998. http://ls11-www.informatik.uni-dortmund.de/people/hermes/manuals/LSF/program.pdf [10-JAN-2009], 1998.

[mau] SOURCEFORGE: Maui Scheduler Open Cluster Scheduler and Resource Manager. http://mauischeduler.sourceforge.net/ [10-JAN-2009].

[moa] Adaptive Computing - Technical documentation for component of the Moab product family. http://www.adaptivecomputing.com/resources/docs/ [10-JAN-2009].

[MS04] T. Maguire and D. Snelling. Web services service group 1.2 (WS-ServiceGroup). *OASIS Standard - Publish Online at: http://docs.oasis-open.org/wsrf/wsrf-ws_service_group-1.2-spec-os.pdf*, 2004.

[MSK+04] J. Maclaren, R. Sakellariou, K. T. Krishnakumar, J. Garibaldi, and D. Ouelhadj. Towards service level agreement based scheduling on the grid. In *Workshop on Planning and Scheduling for Web and Grid Services (in conjunction with the 14th International Conference on Automated Planning and Scheduling - ICAPS-04)*, pages 100–102, 2004.

[oas] OASIS | advancing open standards for the information society. https://www.oasis-open.org/ [22-JAN-2013].

[ope] OpenPBS public home. http://www.mcs.anl.gov/research/projects/openpbs/ [22-JAN-2013].

[ora10] Beginner's Guide to Oracle Grid Engine 6.2 - An Oracle White Paper. Online at: http://www.oracle.com/technetwork/oem/host-server-mgmt/twp-gridengine-beginner-167116.pdf [22-JAN-2013], 2010.

[par] Parallel workloads archive: Logs. http://www.cs.huji.ac.il/labs/parallel/workload/logs.html [12-DEC-2008].

[PBC+11] M. Pasquali, R. Baraglia, G. Capannini, L. Ricci, and D. Laforenza. A multi-level scheduler for batch jobs on grids. In *The Journal of Supercomputing*, volume 57, pages 81–98. Springer Netherlands, 2011.

[PBM08] M. Parkin, R. M. Badia, and J. Martrat. A Comparison of SLA Use in Six of the European Commissions FP6 Projects. Technical Report Number TR-0129, April 2008.

[pbs] The portable batch scheduler and the maui scheduler on linux clusters. http://www.usenix.org/publications/library/proceedings/als00/2000papers/papers/full_papers/bode/bode_html/ [22-JAN-2013].

[PDA$^+$08] T. C. Peachey, N. T. Diamond, D. A. Abramson, W. Sudholt, A. Michailova, and S. Amirriazi. Fractional factorial design for parameter sweep experiments using Nimrod/E. *Scientific Programming*, 16:217–230, April 2008.

[PL95] J. Pruyne and M. Livny. Providing Resource Management Services to Parallel Applications. In *Proceedings of the Second Workshop on Environments and Tools for Parallel Scientific Computing*, pages 152–161, 1995.

[RRK94] F. Ramme, T. Röke, and K. Kremer. A distributed computing center software for the efficient use of parallel computer systems. In *Proceedings of the nternational Conference and Exhibition on High-Performance Computing and Networking Volume II: Networking and Tools*, HPCN Europe 1994, pages 129–136. Springer-Verlag, 1994.

[SAL$^+$02] J. Sherwani, N. Ali, N. Lotia, Z. Hayat, and R. Buyya. Libra: An Economy driven Job Scheduling System for Clusters. In *6th International Conference on High Performance Computing in Asia-Pacific Region HPC Asia, 2002*, pages 81–84. arXiv preprint cs/0207077, 2002.

[SAL$^+$04] J. Sherwani, N. Ali, N. Lotia, Z. Hayat, and R. Buyya. Libra: a computational economy-based job scheduling system for clusters. *Software: Practice and Experience*, 34:573–590, May 2004.

[SB06] L. Srinivasan and T. Banks. Web services resource lifetime 1.2 (WS-ResourceLifetime). *OASIS Standard - Online at: http://docs.oasis-open.org/wsrf/wsrf-ws_resource_lifetime-1.2-spec-os.pdf*, 2006.

[SBBR$^+$10] A. Streit, P. Bala, A. Beck-Ratzka, K. Benedyczak, S. Bergmann, R. Breu, J. Daivandy, B. Demuth, A. Eifer, A. Giesler, B. Hagemeier, S. Holl, V. Huber, N. Lamla, D. Mallmann, A. Memon, M. Memon, M. Rambadt, M. Riedel, M. Romberg, B. Schuller, T. Schlauch, A. Schreiber, T. Soddemann, and W. Ziegler. UNICORE 6 — Recent and Future Advancements. *Annals of Telecommunications*, 65(11):757–762–762, December 2010.

[SC92] L. Smarr and C. E. Catlett. Metacomputing. *Communication of ACM*, 35:44–52, June 1992.

[sch] Oxford dictionaries online. http://oxforddictionaries.com/ [13-NOV-2007].

[SCZL96] J. Skovira, W. Chan, H. Zhou, and D. Lifka. The EASY LoadLeveler API project. In D. Feitelson and L. Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, volume 1162 of *Lecture Notes in Computer Science*, pages 41–47. Springer Berlin / Heidelberg, 1996.

[SFT00] W. Smith, I. Foster, and V. Taylor. Scheduling with advanced reservations. In *Proceedings of 14th International Parallel and Distributed Processing Symposium, 2000 (IPDPS 2000).*, 2000.

[Sko03] A. Skonnard. Understanding WSDL. http://msdn.microsoft.com/en-us/library/ms996486.aspx, October 2003.

[SMS$^+$02] A. Sahai, V. Machiraju, M. Sayal, A. P. A. van Moorsel, and F. Casati. Automated sla monitoring for web services. In *Proceedings of the 13th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management: Management Technologies for E-Commerce and E-Business Applications*, DSOM '02, pages 28–41. Springer-Verlag, 2002.

[SY08] R. Sakellariou and V. Yarmolenko. Job Scheduling on the Grid: Towards SLA-Based Scheduling. In *L. Grandinetti (editor) High Performance Computing and Grids in Action, Volume 16 in the Advances in Parallel Computing series*, pages 207–222. IOS Press, 2008.

[tor] TORQUE resource manager - TORQUE admin manual. http://www.adaptivecomputing.com/resources/docs/torque/ [22-JAN-2013].

[TP11] A. K. Tripathy and M. R. Patra. Modeling and monitoring SLA for service based systems. In *Proceedings of the 2011 International Conference on Intelligent Semantic Web-Services and Applications*, volume 10 of *ISWSA '11*, pages 10:1–10:6. ACM, 2011.

[TTL05] D. Thain, T. Tannenbaum, and M. Livny. Distributed computing in practice: The Condor Experience. *Concurrency and Computation: Practice and Experience*, 17(2-4):323–356, 2005.

[UHHS11] I. Ul Haq, A. A. Huqqani, and E. Schikuta. Hierarchical aggregation of service level agreements. *Data & Knowledge Engineering*, 70(5):435 – 447, 2011. Business Process Management 2009.

[unia] UNICORE - community - projects. http://www.unicore.eu/community/projects/ [22-JAN-2013].

[unib] UniGridS. http://www.unigrids.org/ [22-JAN-2013].

[unic] UVOS - UNICORE virtual organisations system. http://uvos.chemomentum.org/ [22-JAN-2013].

[WGB11] L. Wu, S.K. Garg, and R. Buyya. SLA-Based resource allocation for software as a service provider (SaaS) in cloud computing environments. In *11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, (CCGrid) '11*, pages 195–204, 2011.

[WvLKT08] L. Wang, G. von Laszewski, M. Kunze, and J. Tao. Cloud computing: A Perspective study. R.I.T Digital Media Library; https://ritdml.rit.edu/handle/1850/7821 [1-MAR-2009], 2008.

[WWZ06] O. Waldrich, P. Wieder, and W. Ziegler. A Meta-scheduling Service for Co-allocating Arbitrary Types of Resources. In R. Wyrzykowski, J. Dongarra, N. Meyer, and J. Wasniewski, editors, *Parallel Processing and Applied Mathematics*, volume 3911 of *Lecture Notes in Computer Science*, pages 782–791. Springer Berlin / Heidelberg, 2006.

[xtra] XtremWeb : the open source platform for desktop grids. http://www.xtremweb.net/ [22-JAN-2013].

[xtrb] XtremWeb-CH. http://www.xtremwebch.net/index.html [22-JAN-2013].

[Xu01] M.Q. Xu. Effective metacomputing using LSF Multicluster. In *Proceedings of First IEEE/ACM International Symposium on Cluster Computing and the Grid, 2001.*, pages 100–105, 2001.

[XZS$^+$06] W. Xiaohui, D. Zhaohui, Y. Shutao, H. Chang, and L. Huizhen. CSF4: A WSRF Compliant Meta-Scheduler. *In The 2006 World Congress in Computer Science, Computer Engineering, and Applied Computing, GCA*, 6:61–67, 2006.

[YB06a] C. S. Yeo and R. Buyya. Managing risk of inaccurate runtime estimates for deadline constrained job admission control in clusters. In *Proceedings of International Conference on Parallel Processing, 2006 (ICPP 2006).*, pages 451 –458, August 2006.

[YB06b] C. S. Yeo and R. Buyya. A taxonomy of market-based resource management systems for utility-driven cluster computing. *Software: Practice and Experience*, 36(13):1381–1419, 2006.

[YS06] V. Yarmolenko and R. Sakellariou. An evaluation of heuristics for SLA based parallel job scheduling. In *Proceedings of 20th International Parallel and Distributed Processing Symposium, 2006 (IPDPS 2006).*, page 8 pp. IEEE Computer Society, 2006.

[Zhe10] W. Zheng. *Explorations in Grid Workflow Scheduling.* PhD in Computer Science, School of Computer Science, The University of Manchester, UK, 2010.

[ZS07] H. Zhao and R. Sakellariou. Advance reservation policies for workflows. In *Job Scheduling Strategies for Parallel Processing*, pages 47–67. Springer, 2007.

Appendix A

Characteristics of Systematic Workloads Set

Work-load	Avg. N_{CPU}	Std Dev N_{CPU}	Avg. runtime (t_D) *min.*	Std Dev runtime (t_D) *min.*	Avg. laxity (t_L) *min.*	Std Dev laxity (t_L) *min.*	Avg. Jobsize (A)	Std Dev Jobsize (A)	Avg. Arrival Rate Per Sec	Std Dev Arrival Rate Per Sec
1	3.561	2.039	34.747	78.289	5.8951	14.739	137.466	310.343	0.00634	0.00107
2	3.183	2.154	80.571	176.57	24.871	75.08	273.437	674.397	0.00662	6.39E-04
3	3.315	2.035	72.134	162.35	10.444	30.717	311.577	890.991	0.02201	0.20762
4	3.375	2.007	80.756	140.46	18.04	33.424	306.362	562.506	0.00683	0.00129
5	3.383	1.921	108.5	197.49	19.283	44.882	430.236	875.028	0.02443	0.43113
6	3.293	2.062	62.035	163.16	9.0575	26.483	239.526	724.723	0.00754	0.00158
7	3.388	2.122	59.857	158.94	196.2	706.62	237.991	782.009	0.00806	0.04359
8	3.369	2.214	68.053	154.68	72.294	212.33	296.134	791.014	0.01108	0.00718
9	3.579	2.251	105.07	231.29	40.189	117.14	436.798	1112.84	0.00536	0.00205
10	2.951	1.995	118.24	281.2	13.283	41.426	555.675	1654.57	0.03159	0.26147
11	3.233	1.937	115.31	256.63	63.279	159.34	436.111	1121.73	0.00577	7.62E-04
12	3.325	2.161	132.14	243.94	25.95	60.156	466.326	925.01	0.00691	8.33E-04
13	3.444	2.154	103.12	217.36	122.56	306.45	453.765	1078.28	0.00578	5.40E-04
14	3.427	2.163	125.72	264.23	60.852	155.99	541.088	1415.27	0.00518	8.94E-04
15	3.104	2.101	90.209	223.93	15.074	46.008	333.374	929.825	0.00513	7.16E-04
16	3.385	2.14	46.507	99.636	58.606	169.08	199.892	505.742	0.00447	0.02548
17	3.432	2.199	41.95	96.869	19.059	55.642	182.122	508.884	0.00282	0.00622
18	3.39	2.141	41.773	94.587	107.82	328.29	168.114	447.808	0.00458	0.02411
19	3.592	2.3	54.204	109.39	8.0037	19.008	221.75	531.062	0.00329	0.00726
20	3.379	2.175	89.336	195.91	108.08	302.72	377.719	996.511	0.00285	0.00467
21	3.609	2.259	91.441	198.62	41.331	109.28	404.082	1054.37	0.00442	0.03745
22	3.392	2.198	77.224	173.52	206.66	596.18	312.436	860.246	0.00358	0.00875
23	3.393	2.186	86.536	181.44	14.499	36.916	362.828	923.966	0.00287	0.00537
24	3.414	2.215	115.35	267.26	129.75	380.96	484.998	1326.29	0.00565	0.04954

25	3.412	2.166	90.447	215.11	44.712	134.71	368.249	1044.17	0.00352	0.0101
26	3.429	2.178	107.28	243.7	315.24	945.16	433.705	1151.41	0.00457	0.01909
27	3.394	2.171	117.36	257.37	19.009	51.122	512.826	1394.55	0.00316	0.00686
28	3.414	2.045	116.02	270.59	117.43	358.87	435.485	1147.72	0.00349	0.0094
29	3.321	2.155	143.21	312.07	68.734	193	566.401	1477.4	0.00348	0.00988
30	3.412	2.183	122	284.4	345.9	1063.8	480.74	1305.82	0.00448	0.01308
31	3.59	2.288	133.57	308.18	21.92	63.353	551.194	1424.9	0.00465	0.02325
32	3.482	2.25	133.47	339.11	173.15	575.29	547.358	1560.29	0.00365	0.00825
33	3.434	2.162	150.7	366.4	63.734	181.46	620.37	1749.61	0.00383	0.00864
34	3.459	2.248	143.44	330.38	441.35	1486.6	627.856	1642.12	0.00587	0.05029
35	3.351	2.089	134.28	320.02	21.323	59.423	540.154	1542.41	0.0035	0.00819
36	3.471	2.265	44.61	99.306	45.996	134.47	185.452	504.413	0.04467	0.24238
37	3.48	2.248	47.837	101.17	19.673	50.43	203.326	513.231	0.02744	0.04243
38	3.384	2.165	43.297	96.153	119.63	372.85	171.951	452.723	0.04281	0.24728
39	3.372	2.114	45.432	98.797	6.8788	17.473	172.631	431.985	0.03468	0.18507
40	3.429	2.194	79.401	182.32	87.447	250.58	311.884	829.715	0.03294	0.10765
41	3.299	2.187	86.482	189.12	37.35	103.68	359.027	928.078	0.0368	0.31874
42	3.534	2.192	89.774	197.27	282.36	876.83	375.163	966.299	0.02833	0.07118
43	3.515	2.237	87.259	198.75	14.319	38.672	395.238	1080.55	0.02769	0.04855
44	3.358	2.111	104.9	241.19	16.871	45.78	385.888	987.113	0.04676	0.54635
45	3.488	2.147	133.96	306.37	21.568	57.188	532.481	1435.41	0.03507	0.09593
46	3.483	2.189	143.1	318.07	23.751	61.945	583.734	1617.77	0.0284	0.08181
47	3.017	2.178	60.618	196.97	9.1865	30.156	249.674	971.413	0.40964	0.22646
48	2.878	2.072	63.229	228.29	9.672	38.117	253.6	963.242	0.33446	0.08152
49	3.114	1.998	29.809	82.738	74.574	310.76	110.244	358.473	0.51944	0.13868
50	3.402	2.235	51.722	191.11	158.88	958.58	210.183	950.112	0.40916	0.06323
51	3.274	2.203	63.813	215.16	156.28	673.76	252.603	1122.46	0.39606	0.07388
52	3.45	2.224	76.809	238.26	158.34	676.53	335.506	1320.59	0.37819	0.08702
53	3.312	2.173	21.125	63.761	24.88	93.98	81.8285	294.686	0.53379	0.06238
54	3.222	2.102	69.151	219.62	64.192	283.04	249.157	905.101	0.41222	0.08509
55	3.527	2.32	64.761	224.13	68.894	287.46	267.076	1023.55	0.59935	0.52476
56	3.446	2.071	66.307	232.72	79.167	321.34	313.282	1297.46	0.46378	0.08942
57	3.214	2.19	25.305	72.156	11.386	39.939	105.182	326.257	0.52328	0.62074
58	3.267	1.965	55.893	188.36	22.957	84.282	242.066	929.725	0.55546	0.09182
59	3.006	1.817	51.161	195.03	24.678	123.15	220.332	1089.92	0.52094	0.2674
60	2.953	1.926	69.628	250.85	23.075	87.816	271.187	1112.89	0.67232	0.28757
61	3.634	2.396	35.418	87.345	5.7985	16.789	145.626	430.85	0.61456	0.05828
62	3.299	2.164	59.504	197.46	9.9215	35.241	243.701	1010.3	0.41215	0.13348
63	3.262	2.195	74.512	266.29	12.939	54.983	312.564	1205.82	0.54058	1.44674
64	3.488	2.286	68.185	239.63	11.021	52.652	287.962	1154.8	0.42044	0.10336
65	2.978	2.038	27.886	74.19	78.619	292.92	122.108	385.722	0.51338	0.07065
66	3.189	2.105	58.558	219.58	132	605.71	225.918	1123.1	0.5511	0.23935
67	3.233	2.102	56.879	203.05	139.38	757.93	208.922	819.404	0.46108	0.10608
68	3.431	2.174	74.742	277.67	218.85	1075.4	330.642	1477.73	0.42596	0.11961

69	3.484	2.107	21.185	64.017	23.22	88.387	84.9554	300.156	0.58316	0.11746
70	3.313	2.042	63.172	196.92	83.677	328.42	234.567	839.669	0.52008	0.07218
71	2.786	1.833	67.506	259.76	72.524	336.67	267.571	1291.25	0.72706	0.16489
72	3.042	1.995	70.698	262.33	77.714	331.02	291.573	1260.26	0.5696	0.09394
73	3.184	2.112	30.766	79.07	12.806	39.594	123.318	369.229	0.52589	0.09194
74	3.13	2.089	66.64	207.67	28.466	106.64	268.609	899.262	0.47037	0.13215
75	3.028	2.158	69.335	260.48	31.268	142.1	310.836	1635.2	0.43486	0.1327
76	3.28	2.016	69.886	247.8	30.707	136.01	249.719	1000.53	0.70714	0.12309
77	3.447	2.196	29.583	80.77	4.5842	14.02	126.66	420.804	0.62602	0.12013
78	3.441	1.928	60.22	185.96	10.005	35.666	246.99	814.427	0.44774	0.09264
79	3.446	2.117	76.334	261.06	9.7545	38.927	311.039	1320.18	0.66358	0.07554
80	3.328	1.923	73.189	260.64	10.255	41.787	310.692	1305.68	0.50597	0.10681
81	3.268	2.116	45.374	97.6	116.26	336.17	157.229	404.543	0.01596	0.00456
82	2.995	2.037	74.488	224.27	207.41	839.32	247.036	893.046	0.01196	0.00254
83	3.171	1.926	135.96	321.2	439.99	1326.9	535.444	1732.89	0.00868	0.00116
84	3.373	2.069	157.67	394.01	452.36	1314.1	644.495	1934.98	0.01308	0.00791
85	2.823	1.66	22.665	67.043	53.536	230.48	88.2141	290.818	0.03258	0.00921
86	3.411	2.427	92.556	235.17	226.04	715.21	367.612	1193.58	0.02891	0.00603
87	3.293	2.082	125.09	336.26	331.58	1224.3	489.559	1472.62	0.02314	0.00303
88	3.486	1.935	113.83	313.53	269.49	1043.4	436.039	1298.5	0.03033	0.00578
89	3.517	2.102	40.479	92.778	126.7	367.03	163.341	439.221	0.05578	0.05033
90	3.277	2.183	115.88	287.49	324.23	1001.5	563.238	1732.18	0.04336	0.01364
91	3.09	1.818	81.449	260.49	152.6	554.09	261.8	883.154	0.08582	0.02077
92	3.058	2.033	104.29	276.56	312.5	1086.7	405.889	1233.25	0.04015	0.00636
93	3.307	2.01	34.939	74.112	103.82	326.38	139.76	371.307	0.06934	0.00977
94	3.579	2.41	101.4	242.43	251.69	975.04	379.563	1054.72	0.05061	0.01084
95	3.301	2.11	89.632	285.98	226.88	949.77	333.45	1123.69	0.05436	0.15126
96	3.317	2.039	116.97	318.09	216.49	783.24	475.608	1728.06	0.07111	0.01144
97	3.472	2.257	22.785	65.509	60.058	176.66	94.8775	284.515	0.10118	0.06097
98	2.962	1.022	17.379	107.58	38.619	297.54	65.6378	404.07	0.19415	0.07751
99	3.152	2.332	79.504	277.74	193.39	747.32	386.498	1621.02	0.07413	0.05932
100	3.27	2.081	99.256	273.95	317.95	1092	411.124	1280.98	0.07304	0.01
101	3.55	1.985	26.49	72.209	36.122	114.51	114.814	328.28	0.01672	0.09457
102	3.377	2.007	129.16	303.84	136.53	383.33	600.069	1827.86	0.00808	0.00142
103	3.052	1.985	139.51	408.47	163.36	642.95	592.215	1970.66	0.00836	0.00205
104	3.66	2.109	204.22	518.36	232.72	826.99	975.412	2982.46	0.03049	0.19643
105	3.014	1.946	41.205	84.813	45.299	122.58	134.096	336.741	0.04554	0.14649
106	3.083	2.029	81.829	227.84	93.397	331.88	331.807	1194.97	0.02951	0.00585
107	3.274	2.017	103.78	283.24	117.55	425.46	483.408	1696.93	0.03369	0.00613
108	3.517	2.045	150.3	415.23	144.7	404.4	532.473	1341.36	0.02884	0.00205
109	3.26	2.27	28.057	84.452	25.641	91.437	114.872	395.315	0.0977	0.01214
110	3.265	2.174	69.632	221.49	75.389	333.28	285.367	1047.36	0.05904	0.0101
111	3.404	2.141	118.16	307.37	141.09	443.9	532.081	1690.07	0.03683	0.00722
112	3.448	2.275	119.51	323.27	93.422	305.42	450.387	1358.09	0.04923	0.00886

113	3.581	2.177	40.929	89.75	46.572	140.43	180.605	472.508	0.10227	0.04499
114	3.444	2.227	95.985	247.4	94.109	358.7	418.306	1236.84	0.05517	0.0123
115	3.19	2.037	140.48	394.18	146.88	430.7	575.252	2061.09	0.04954	0.00789
116	3.162	2.081	134.23	297.59	137.08	380.21	562.714	1503.88	0.05128	0.00904
117	3.242	2.059	21.91	55.876	19.569	58.222	77.9638	250.62	0.10598	0.03406
118	3.367	2.065	74.06	212.16	87.728	340.54	311.457	987.527	0.0865	0.08035
119	3.07	1.857	90.099	310.06	99.376	440.36	393.562	1714.57	0.06312	0.01777
120	3.527	2.24	112.85	327.8	92.497	328.56	485.034	1589.44	0.09867	0.01784
121	3.496	2.303	53.176	106.84	26.433	72.474	251.902	613.244	0.01127	0.0018
122	3.379	2.204	100.15	251.83	45.446	140.25	424.583	1267.18	0.01304	0.06894
123	3.074	1.976	134.97	334.65	67.537	209.42	478.609	1252.26	0.01466	0.1188
124	3.24	2.079	132.62	321.86	60.281	165.23	575.474	1467.41	0.02174	0.00912
125	3.524	2.316	46.205	92.411	20.472	52.801	212.168	554.237	0.03396	0.00583
126	1.997	1.784	33.833	146.89	12.463	76.906	113.664	529.978	0.06298	0.02241
127	2.542	1.85	56.844	218.98	17.701	70.797	218.938	1000.41	0.05028	0.03659
128	3.355	2.345	127.11	362.29	59.797	225.06	701.243	2291.87	0.0208	0.00619
129	3.521	2.198	22.819	58.189	9.3354	26.631	90.9463	275.967	0.07472	0.01332
130	3.259	2.103	131.48	284.21	53.749	138.96	486.182	1299.04	0.04688	0.00922
131	3.391	2.365	120.88	356	47.092	167.19	471.378	1783.34	0.04107	0.16203
132	3.124	1.998	143.95	369.66	67.857	201.1	632.355	2289.36	0.04471	0.134
133	3.288	1.829	38.506	90.859	13.715	38.65	142.209	387.125	0.08957	0.78704
134	3.227	2.103	97.794	250.06	37.474	108.91	394.86	1276.3	0.06269	0.00501
135	3.668	2.254	85.666	230.05	35.045	108.16	338.152	1013.91	0.0858	0.02526
136	2.956	2.089	115.49	331.43	58.764	216.86	349.311	1380.11	0.06273	0.01469
137	3.444	2.014	30.509	70.284	10.601	36.778	126.783	316.419	0.10301	0.02511
138	3.483	2.212	93.58	250.84	40.98	133.47	389.225	1217.27	0.06972	0.01748
139	3.312	2.145	111.24	304.17	41.822	124.08	464.471	1638.78	0.06347	0.01178
140	3.374	2.193	106.93	292.71	45.915	152.46	434.57	1459.13	0.09453	0.02084
141	3.406	2.104	36.493	88.099	6.5379	18.94	147.691	419.874	0.00768	8.86E-04
142	3.395	2.008	144.35	318.24	21.632	42.662	490.909	1064.16	0.00896	0.00184
143	3.446	2.23	139.33	370.19	24.598	76.935	676.418	2124.97	0.0104	0.01208
144	3.399	1.99	116.89	356.75	21.246	81.808	453.578	1302.73	0.01306	0.00244
145	3.565	1.991	22.042	60.944	3.6215	11.049	82.7202	281.228	0.10635	0.46902
146	3.467	2.212	101.6	281.4	14.871	42.359	445.355	1325.52	0.03012	0.00657
147	3.836	2.367	135.86	354.39	21.333	72.107	596.112	2026.36	0.02491	0.00438
148	3.198	1.945	129.68	321.38	22.033	66.968	475.444	1350.21	0.03456	0.00586
149	3.215	2.047	45.301	104.31	6.6735	19.883	181.376	508.496	0.0832	0.26522
150	3.048	2.142	97.887	243.17	10.274	30.122	411.063	1285.19	0.05997	0.28037
151	3.507	2.125	163.02	394.36	25.675	78.894	688.2	2142.5	0.03169	0.00777
152	3.159	2.223	116.08	316.55	15.04	52.801	480.309	1453.66	0.0574	0.00842
153	3.151	1.816	21.925	63.319	2.7692	8.9581	81.1215	272.195	0.06401	0.00823
154	3.243	2.247	67.685	205.47	10.82	39.945	278.13	952.137	0.15499	0.06569
155	3.02	1.974	96.424	271.6	16.063	52.926	431.455	1360.13	0.07601	0.0193
156	3.059	2.027	111.66	338.79	15.515	51.916	438.522	1750.75	0.06301	0.00631

157	3.612	1.82	43.16	94.04	5.428	14.372	172.875	414.744	0.09999	0.02662
158	3.286	2.188	97.909	253.4	14.7	48.564	427.236	1225.28	0.07645	0.06138
159	3.299	2.164	119.92	336.06	21.255	69.906	528.945	2019.19	0.06154	0.0164
160	3.614	2.331	130.6	324.95	24.88	75.842	567.618	1740.96	0.06757	0.01245
161	3.304	2.104	22.83	66.386	58.621	261.26	99.8928	340.604	0.62814	0.08913
162	3.413	2.172	61.447	194.38	190.93	935.25	260.953	952.6	0.59415	0.12135
163	3.686	2.399	59.702	247.03	137.59	700.65	220.64	960.609	0.66916	0.12125
164	3.258	2.082	80.211	275.45	208.69	841.68	329.574	1489.52	0.512	1.25262
165	3.187	2.14	21.782	64.97	57.615	230.48	95.9565	346.973	1.3643	0.3588
166	3.154	1.987	66.496	219.29	214.64	980.99	290.996	1130.85	1.56593	0.39336
167	3.411	2.197	68.26	254.39	213.38	1098.1	264.625	1073.82	1.50506	0.31983
168	3.055	1.891	63.574	248.78	190.65	934.4	242.595	1031.52	1.50092	0.39901
169	3.316	2.105	23.874	71.196	68.784	283.09	95.1364	345.16	1.81967	0.50614
170	3.012	2.116	58.847	200.24	189.92	904.22	235.946	949.418	1.67603	0.49997
171	3.17	1.945	69.911	283.28	163.27	867.3	304.809	1530.29	2.25788	0.49405
172	3.165	2.018	77.927	311.16	173.16	912.49	315.046	1358.68	2.44844	0.69183
173	3.188	2.174	23.414	73.747	77.503	352.06	90.2456	326.411	2.92151	0.8317
174	3.239	2.036	60.695	221	185.77	870.12	218.846	884.13	2.801	0.70117
175	3.237	2.074	76.114	302.44	208.98	1002.6	344.282	1766.75	2.22982	0.70837
176	3.382	2.169	75.325	249.3	219.22	1024	280.559	1155.08	2.3861	0.65999
177	2.934	1.879	23.559	78.54	51.609	229.54	94.1625	375.649	3.39122	1.125
178	3.394	2.058	55.329	206.05	131.65	658.53	226.391	961.25	3.54338	1.09333
179	3.117	2.142	66.006	288.16	276.04	1558.7	299.66	1546.07	2.83637	0.97836
180	3.23	2.083	70.77	262.55	229.89	1110.9	319.148	1481.95	3.35516	0.99178
181	3.174	2.141	29.619	77.605	37.02	113.7	114.763	375.231	0.55464	0.08404
182	3.884	2.413	56.352	184.64	68.076	267.48	212.252	674.955	0.55697	0.11464
183	3.32	2.218	57.862	208.9	63.945	317.16	250.051	1088.94	0.6017	0.10382
184	3.779	2.402	43.059	199.98	51.757	332.74	139.547	649.979	0.81446	0.19365
185	3.455	2.18	24.031	70.448	26.184	95.806	101.867	343.897	1.57711	0.49979
186	3.099	2.104	70.781	247.31	77.583	332.82	297.378	1292.24	1.04098	0.26151
187	2.702	1.258	24.315	140.19	30.746	202.97	104.097	757.135	2.75891	1.06887
188	3.425	2.118	57.259	264.83	71.897	438.66	227.097	1102.21	1.85237	0.49941
189	3.088	1.992	17.757	61.735	21.142	91.728	68.4207	268.666	2.50311	0.78744
190	3.268	1.955	55.183	195.74	59.049	273.76	236.357	908.546	2.89508	0.83039
191	3.309	2.109	77.322	293.09	84.195	374.2	345.741	1604.91	2.03158	0.48741
192	3.433	2.171	67.271	244.79	76.549	340.19	284.953	1269.97	2.5658	1.26075
193	3.47	2.171	22.213	69.47	20.28	87.249	89.0172	333.724	3.33853	0.96857
194	3.142	1.945	62.944	233.97	85.054	386.14	224.8	877.382	3.22364	0.99752
195	3.415	2.032	52.734	210.44	64.017	297.77	227.959	1042.46	3.12512	0.84737
196	3.442	2.192	69.595	251.11	69.614	298.81	269.258	1043.54	3.51604	1.15847
197	3.269	2.116	20.158	67.916	25.087	111.8	86.5754	350.591	3.80422	1.16166
198	3.132	1.977	56.597	212.49	75.996	358.2	219.304	995.197	3.38235	1.15145
199	3.192	2.127	61.155	238.17	64.671	305.2	229.323	970.07	3.55919	1.0959
200	3.263	2.131	71.344	271.42	74.851	409.43	299.25	1275.19	3.49741	1.11075

201	3.623	2.32	30.182	78.509	13.078	41.691	123.895	369.881	0.74202	0.23015
202	3.276	2.103	53.112	183.82	22.132	86.021	203.047	760.154	0.46637	0.17289
203	3.279	2.064	70.254	249.76	29.543	115.97	249.911	1016	0.60829	0.06291
204	3.015	1.998	57.273	236.5	26.953	134.83	249.343	1226.44	0.65869	0.11213
205	3.193	1.951	23.335	66.352	10.236	38.07	89.491	283.067	1.94183	0.52259
206	3.267	1.951	57.045	196.5	25.583	106.86	230.519	939.623	1.64555	0.31071
207	3.311	2.15	64.379	256.54	22.802	106.37	235.52	1035.39	2.08416	0.45842
208	3.214	2.106	53.183	206.41	25.353	122.05	205.314	990.552	1.67656	0.42848
209	3.155	2.162	20.638	65.519	9.0212	35.866	78.7072	281.376	2.94195	0.8151
210	3.313	2.027	55.636	191.54	25.494	97.062	255.025	1068.27	2.55139	0.56527
211	3.338	2.208	63.52	283.5	26.674	156.58	261.258	1390.4	2.73001	0.65177
212	3.078	2.008	58.602	214.19	24.688	112.34	252.734	1172.28	2.489	0.73728
213	3.124	2.142	17.403	63.68	7.2945	31.985	63.52	275.948	3.80534	1.19608
214	3.339	2.173	45.418	160.81	20.375	88.243	184.018	713.084	3.20892	1.13441
215	3.03	1.902	57.55	213.92	24.33	107.61	233.849	985.911	3.24237	1.0216
216	3.175	2.096	64.002	250.4	30.569	144.83	263.586	1079.51	3.09305	0.90039
217	3.15	2.001	20.751	66.209	10.209	36.211	70.9773	247.089	4.30024	0.76308
218	2.896	1.993	45.751	195.72	24.765	116.95	179.768	998.487	4.28901	1.34804
219	3.199	2.139	55.605	212.96	24.255	117.14	238.575	1131.6	3.49021	1.13938
220	3.19	2.028	69.618	291.77	21.088	122.39	297.598	1431.11	4.4838	1.29556
221	3.107	2.13	31.452	87.679	5.1579	17.671	135.593	447.822	0.60228	0.11067
222	3.687	2.324	60.487	201.26	9.3588	40.514	218.033	821.769	0.5603	0.13987
223	3.531	2.177	56.518	210.83	8.5421	38.887	210.707	791.666	0.68889	0.10298
224	3.35	2.159	80.436	240.28	13.856	53.848	327.258	1200.91	0.54071	0.07765
225	3.13	1.983	25.066	64.312	4.0922	12.794	96.2039	301.65	1.85011	0.47544
226	2.757	1.752	60.462	222.69	9.004	38.539	242.195	1010.9	2.58173	0.83239
227	3.243	2.064	60.295	213.23	7.6417	32.933	233.031	898.893	2.07034	0.51431
228	3.277	2.015	56.396	266.8	8.1779	50.544	233.537	1304.84	2.32291	0.55836
229	3.024	2.224	24.79	74.346	3.8269	13.251	102.719	363.135	2.73594	0.7291
230	2.817	1.843	47.9	178.07	8.152	36.553	175.571	795.857	2.5931	0.74544
231	3.427	2.213	65.573	260.43	9.38	43.528	272.154	1185.93	2.72896	0.59344
232	3.219	2.12	63.985	223.71	8.5683	33.409	258.656	1158.59	2.0336	0.64828
233	2.776	1.827	18.708	56.839	3.6171	12.84	69.5718	258.91	3.53071	0.80803
234	3.085	1.917	62.527	214.85	10.101	41.17	236.011	923.41	3.70109	0.80431
235	3.312	2.052	64.735	255.9	8.9496	34.685	251.51	1257.58	3.68871	1.28663
236	3.542	2.28	66.06	254.77	10.925	46.885	263.66	1115.85	3.30093	0.93864
237	3.278	2.184	23.007	69.544	3.0181	11.418	96.1374	343.66	3.95852	1.31808
238	3.188	1.969	71.978	237.32	12.614	48.723	300.099	1164.98	3.36127	1.03542
239	3.702	2.295	56.557	232.96	9.2455	44.937	252.735	1314.68	4.44045	1.00729
240	3.189	2.15	65.24	244.51	10.006	42.962	258.579	1091.81	3.77799	1.13909
241	3.089	2.115	20.647	70.129	63.279	288.9	79.0412	319.236	3.05962	1.12703
242	3.02	1.928	65.747	211.45	199.12	821.26	264.19	972.546	3.12406	1.00855
243	3.273	2.077	61.113	245.39	199.07	1216	227.405	1000.57	3.81766	1.0643
244	3.24	2.181	92.466	299.26	204.78	833.72	394.124	1446.31	3.44326	1.16801

245	3.386	2.11	24.438	72.653	83.424	323.93	103.803	365.749	6.14185	2.13952
246	3.196	2.153	56.957	193.23	188.75	838.9	202.264	794.26	7.00114	1.95483
247	3.221	2.134	92.388	356.66	276.96	1413	384.777	1695.15	5.47007	1.99058
248	3.173	2.113	81.077	270.33	178.22	774	368.976	1496.52	8.38566	1.5959
249	3.11	2.112	24.427	75.739	91.367	375.01	96.7997	359.652	8.56165	3.07508
250	2.92	1.902	56.771	202.95	213.99	1085.5	212.258	902.612	11.215	2.63903
251	3.019	1.994	70.372	287.72	197.95	1070.5	279.34	1466.01	8.76421	3.09141
252	3.236	2.137	92.782	310.17	223.9	986.57	391.39	1639.66	6.74599	2.8701
253	3.503	2.273	26.489	74.951	69.227	267.04	113.841	382.674	9.48089	3.42046
254	3.107	2.002	58.731	216.46	140.08	646.56	245.963	1013.08	10.3832	3.51302
255	3.107	2	95.669	341.29	268.53	1245.9	397.522	1749.38	8.82765	3.24447
256	3.154	1.968	101.16	343.57	252.35	1203.7	437.55	1821.77	10.6272	2.89092
257	3.439	2.256	29.996	84.145	87.661	318.43	128.349	420.549	9.2627	3.72175
258	3.14	2.084	78.071	254.11	178.19	760.5	263.516	929.173	10.2834	3.46584
259	3.362	2.117	99.267	345.57	292.25	1448.6	398.347	1664.37	12.3352	3.67681
260	3.266	2.177	93.853	339.33	263.15	1248.4	371.361	1545.61	12.7684	4.58703
261	3.171	2.199	23.669	70.574	31.822	128.82	92.3617	325.735	3.61918	1.25136
262	3.045	2.085	48.916	181.95	51.232	245.96	200.166	930.225	4.43877	1.16016
263	3.564	2.156	85.055	308.02	96.979	426.85	391.679	1741.44	3.40985	1.15018
264	3.404	2.275	67.31	248.29	69.241	286.85	273.818	1126.89	3.17919	1.10117
265	3.066	2.125	18.55	65.145	16.95	73.476	75.4703	297.29	8.54864	2.59927
266	3.029	1.938	59.749	214.46	71.451	296.33	267.861	1159.71	7.67104	2.66958
267	3.038	1.909	77.462	282.54	73.475	330.13	270.173	1079.25	6.79849	2.17418
268	3.277	2.064	74.648	282.99	103.08	499.16	306.329	1276.19	5.08973	2.02332
269	3.299	2.109	23.08	72.167	27.169	93.121	99.6	368.659	9.91031	3.21907
270	3.055	1.969	60.762	210.06	60.372	246.19	219.494	856.524	9.31986	3.04798
271	3.154	1.993	76.115	271.93	80.918	357.21	329.981	1425.82	8.48453	3.03525
272	3.026	2.222	71.732	296.09	85.833	469.76	285.627	1373.47	10.8042	2.93746
273	3.347	2.218	20.691	64.304	25.966	100.08	83.9988	302.795	11.6088	3.24162
274	3.179	2.111	71.387	233.17	85.19	353.04	303.055	1154.73	12.2073	3.71968
275	3.231	1.98	69.912	248.36	77.057	363.29	267.3	1120.89	10.9809	3.52276
276	3.058	2.063	72.057	259.72	77.71	481.73	314.044	1298.37	10.9444	3.54502
277	3.62	1.909	16.565	62.52	16.259	84.627	69.269	295.603	15.6439	6.64812
278	3.229	2.1	70.229	239.06	75.621	313.4	284.082	1060.4	10.9474	3.88974
279	3.105	2.001	61.198	226.91	69.442	259.19	252.883	1069.83	14.3141	4.70594
280	3.089	1.982	103.77	360.15	132.39	538.72	405.247	1595.42	10.8857	3.87709
281	3.05	2.083	18.74	58.856	8.4895	32.243	75.7946	290.69	4.49638	2.10363
282	3.408	2.194	58.263	199.18	22.722	83.836	246.15	972.322	3.65188	1.03803
283	3.269	2.046	72.028	291.61	28.189	149.97	316.073	1426.22	3.68794	1.11693
284	3.315	2.186	55.528	239.8	25.612	113.98	223.895	1149.98	3.69223	1.084
285	3.226	2.016	18.323	62.656	8.2768	33.671	74.5339	293.243	8.4777	2.60171
286	3.2	2.229	59.451	207.6	29.71	119.52	254.466	1082.26	7.15932	2.39393
287	3.306	2.253	64.689	226.35	27.443	117.8	285.917	1278.4	7.307	2.26286
288	3.167	1.977	76.683	257.79	30.066	124.95	301.647	1118.9	7.10886	2.29114

289	3.21	1.974	20.397	67.112	6.1774	25.026	78.6661	308.996	10.191	3.16133
290	3.257	2	70.245	250.15	33.082	135.17	313.078	1332.24	9.90565	3.21687
291	3.271	2.099	79.193	272.17	34.346	149.4	290.428	1059.18	10.2363	2.93831
292	3.228	2.109	62.438	222.65	24.704	109.61	293.83	1357.91	9.22485	3.0958
293	3.14	2.011	25.592	76.637	11.27	39.423	107.564	378.258	10.3001	4.05014
294	3.344	2.104	82.203	260.45	38.325	143.87	313.828	1123.02	10.5304	4.08722
295	3.422	2.132	76.122	262.42	39.993	160.77	296.831	1132.09	9.46022	3.70022
296	3.303	2.167	87.257	303.25	35.589	145.66	333.425	1200.37	10.941	3.44694
297	3.281	2.102	21.769	69.438	9.5508	36.903	83.4385	308.709	12.9552	4.8955
298	3.225	2.191	59.99	195.65	20.417	76.829	249.28	991.715	14.3774	4.58003
299	3.247	2.029	81.339	290.47	31.263	144.45	341.697	1468.07	13.898	3.85616
300	3.224	2.042	66.693	240.93	26.954	110.77	299.715	1300.31	14.8269	3.90866
301	3.316	2.027	18.779	63.021	3.0192	12.476	77.7203	314.355	4.76148	4.07894
302	3.099	2.007	62.49	225.53	8.6245	35.903	243.922	1055.43	3.27047	1.07145
303	3.111	2.008	71.237	307.98	10.774	60.594	305.397	1458.39	5.02762	1.50339
304	3.082	2.011	75.164	287.75	11.535	49.441	320.697	1570.22	4.38046	1.27144
305	3.328	2.103	22.583	72.027	3.2569	12.77	86.7076	328.603	7.67734	2.04496
306	3.14	2.028	68.857	232.45	8.3187	33.96	282.126	1129.9	8.33738	2.70829
307	3.294	2.035	53.674	257.83	6.9087	36.575	199.104	1082.92	7.1832	2.62073
308	3.195	2.08	69.416	237.52	10.608	44.667	308.235	1275.02	6.73988	2.28689
309	3.292	2.14	22.597	71.425	3.5483	12.942	94.4634	346.916	9.26026	3.35297
310	3.155	1.937	63.287	223.97	10.005	39.316	257.592	1118.55	9.18361	3.32405
311	3.237	2.126	75.027	263.14	10.643	39.416	288.863	1104.24	9.7372	3.50061
312	3.517	2.166	54.851	210.2	9.6452	42.759	233.712	1115.38	11.7918	3.93634
313	2.989	2.009	18.481	62.927	3.2621	12.903	85.9488	378.764	12.3778	3.55584
314	3.32	2.183	65.609	210.88	10.316	36.291	249.223	860.828	10.1662	3.93252
315	3.042	1.95	81.425	308.96	13.9	53.065	338.359	1508.11	12.3519	4.24635
316	3.036	1.94	76.274	276.91	14.035	56.571	275.32	1084.67	12.5178	3.91699
317	2.942	2.049	22.804	74.287	3.7338	14.41	82.3779	310.261	13.6835	5.90245
318	3.31	2.126	78.792	237.32	14.027	49.897	334.387	1196.86	12.3313	3.84699
319	2.989	1.996	56.521	194.23	9.7053	36.368	222.28	858.432	12.6125	4.77716
320	3.191	2.138	64.651	232.66	10.73	42.244	247.251	972.889	14.3119	4.6184

Table A.1: Systematic Workload Statistics

Appendix B

Characteristics of Random Workloads Set

Work-load	Avg. N_{CPU}	Std Dev N_{CPU}	Avg. runtime (t_D) *min.*	Std Dev runtime (t_D) *min.*	Avg. laxity (t_L) *min.*	Std Dev laxity (t_L) *min.*	Avg. Jobsize (A)	Std Dev Jobsize (A)	Avg. Arrival Rate Per Sec	Std Dev Arrival Rate Per Sec
1	3.59	2.249	45.991	97.4532	171.85	508.001	186.565	434.09	0.00503	0.001003
2	3.104	1.945	67.52	229.112	30.5962	124.479	249.364	893.689	3.8188	1.119648
3	2.963	2.018	74.326	245.129	12.7627	50.179	306.674	1243.83	0.56244	0.091026
4	3.145	2.188	71.887	274.93	8.40726	41.3072	302.439	1462.2	17.1296	4.655591
5	3.34	2.165	79.375	264.773	241.045	1063.07	307.564	1237.27	12.1497	3.652674
6	3.115	2.043	99.685	305.256	341.453	1282.08	411.998	1496.41	10.2489	3.491132
7	3.376	2.243	71.081	257.86	221.066	1151.3	328.507	1322.9	0.22891	0.05464
8	3.292	2.117	106.43	320.809	298.36	1470.42	479.73	1720.79	0.15899	0.026516
9	3.238	2.055	62.45	245.778	179.174	1007.69	235.062	987.428	0.45924	0.062085
10	3.336	2.058	57.351	185.157	9.28659	38.7031	228.966	884.708	0.80351	0.360315
11	3.659	2.217	40.443	86.5828	20.6774	61.4207	155.027	343.935	0.00752	0.004878
12	3.019	1.999	49.474	108.432	8.03594	22.4355	180.65	477.57	0.00707	0.002131
13	3.107	2.063	82.419	294.214	226.097	1009.45	403.993	1734.87	0.34373	0.050239
14	3.549	2.289	32.851	80.832	26.8374	94.7703	144.687	428.071	0.00597	0.008489
15	3.542	2.185	37.98	87.3084	16.2023	49.5785	164.404	423.696	0.00469	6.81E-04
16	3.072	2.005	57.866	110.627	9.66665	21.3179	209.635	466.045	0.00758	0.003696
17	3.487	2.133	37.142	92.8821	70.7838	292.263	151.045	450.304	0.00417	9.38E-04
18	3.291	1.992	31.675	81.915	43.9146	153.024	115.603	322.137	0.00624	5.64E-04
19	3.458	2.171	45.285	96.1585	18.4039	43.537	187.297	523.587	0.00421	0.001026
20	3.322	2.171	92.639	321.789	311.197	1326.19	375.218	1373.72	6.34223	2.06904
21	3.342	2.05	67.047	267.619	177.845	958.932	286.746	1298.24	1.83899	0.636601
22	3.303	2.179	80.555	312.835	37.49	166.666	314.468	1208.44	2.47565	0.621638
23	3.241	2.015	21.387	64.3356	3.28444	11.4714	87.8145	298.482	2.55884	1.146773
24	2.947	2.006	27.022	75.2776	3.87773	13.0707	105.762	373.036	0.58373	0.468722

25	3.109	2.081	84.487	249.409	13.007	41.9937	352.334	1454.09	0.54941	0.06265
26	3.1	1.913	89.097	286.407	109.815	420.204	363.009	1295.99	0.0852	0.017304
27	3.15	1.993	75.806	244.469	167.356	728.692	294.151	1124.59	0.07149	0.022458
28	3.293	2.035	127.9	340.321	43.5458	122.947	550.255	1771.97	0.07581	0.022129
29	3.319	2.016	71.486	240.448	77.8566	355.74	290.941	1051.23	0.40021	0.101686
30	3.742	2.312	118.37	343.121	16.0835	51.5605	586.331	2198.19	0.01159	0.001308
31	3.457	2.019	197.97	480.802	88.2604	281.973	737.212	1804.19	0.01006	0.001165
32	3.242	2.084	201.3	361.977	27.8203	60.9059	681.988	1492.36	0.00932	0.0019
33	3.597	2.244	130.61	296.463	389.765	1136.79	630.399	1663.48	0.00926	0.011369
34	3.758	2.072	141.78	355.406	21.1042	53.0238	602.413	1689.46	0.01261	0.051055
35	3.237	2.022	75.563	264.406	10.6647	44.6513	322.254	1416.66	0.30217	0.067271
36	3.326	2.107	62.341	243.73	174.33	1079.12	262.023	1193.35	2.31808	0.505158
37	2.921	1.994	67.026	239.57	29.1526	115.364	240.116	987.433	2.36223	0.559588
38	3.224	2.182	103.65	379.913	260.63	1075.35	397.535	1683.23	9.32318	2.936176
39	3.152	1.984	78.872	290.717	218.558	1315.32	287.858	1090.65	0.33786	0.088738
40	3.383	2.091	71.338	283.161	12.114	52.9259	321.63	1556.63	3.05755	0.943335
41	3.153	1.953	21.859	70.9881	24.2399	97.2963	81.3179	300.167	3.89265	0.936234
42	3.052	2.012	17.243	53.7749	2.73968	10.6125	65.2958	225.718	3.48016	1.051419
43	3.227	2.346	39.89	99.3658	132.514	416.669	163.017	474.84	0.00653	0.008195
44	3.484	2.263	49.276	196.767	7.48337	33.8707	196.745	904.572	3.63025	1.023174
45	3.747	2.268	104.81	238.63	304.033	952.998	423.074	914.849	0.00578	9.39E-04
46	2.983	2.059	84.604	282.574	14.4858	59.5142	361.057	1526.53	12.5785	3.931187
47	3.276	2.177	93.614	325.45	251.209	1113.88	377.458	1556.65	10.8587	3.856099
48	3.303	2.098	72.081	266.235	32.1522	151.905	268.403	1126.23	12.8733	4.484932
49	3.499	2.164	145.14	393.913	21.3719	59.3946	544.259	1743.25	0.01209	0.135134
50	3.208	2.166	81.999	288.672	13.3206	57.3113	319.607	1337.84	12.8791	5.84499
51	3.211	2.04	60.428	182.133	136.695	560.485	262.044	974.695	11.7181	3.694076
52	3.206	2.111	52.57	167.793	24.9641	94.1981	224.285	863.365	13.9804	4.407415
53	3.259	2.18	41.982	141.602	6.12409	24.732	163.616	632.641	13.89	4.399634
54	2.948	1.916	48.824	147.172	8.53051	28.1482	175.948	590.31	16.8347	5.629126
55	3.017	1.904	77.646	199.265	11.6854	34.6795	286.085	807.236	0.09952	0.087681
56	3.202	2.061	120.22	331.214	16.0457	50.1878	486.627	1519.39	0.0695	0.059894
57	3.658	2.277	106.59	291.33	136.361	468.914	436.576	1337.89	0.07005	0.069837
58	3.384	2.088	118.49	320.47	122.105	462.821	511.721	1579.56	0.08171	0.011801
59	3.222	2.113	107.29	319.71	260.665	1031.93	413.184	1431.98	0.07616	0.008025
60	3.175	2.124	112.15	285.159	19.6328	66.2204	486.938	1350.98	0.08168	0.118266
61	2.947	1.92	75.419	301.677	10.5563	50.0139	311.295	1350.17	13.2699	5.481165
62	3.334	2.156	96.756	294.414	251.703	1131.27	354.108	1144.49	10.7857	3.465453
63	3.019	1.994	81.063	269.986	111.119	500.261	312.635	1194.51	10.0621	3.676101
64	3.588	2.324	70.58	273.091	10.3259	48.1261	288.186	1389.8	12.8791	4.252279
65	3.514	2.222	57.027	220.307	72.6302	382.63	263.797	1291.59	2.71843	0.78222
66	3.327	2.161	67	228.643	9.4458	43.0395	260.364	1071.09	3.13149	1.396323
67	3.363	2.022	83.363	282.465	214.213	1001.99	372.719	1543.82	0.36414	0.093705
68	3.127	1.961	68.495	209.723	10.4722	37.2411	273.678	1015.69	0.48761	0.097955

69	3.133	2.188	63.114	222.751	65.9646	264.74	237.044	1007.93	3.2825	0.808553
70	3.33	2.162	95.492	310.505	216.168	895.272	389.254	1639.61	9.70954	3.137518
71	3.27	2.148	90.826	351.138	103.671	573.538	359.739	1540.06	7.13698	2.613133
72	3.07	1.907	77.43	302.218	11.7814	51.9388	298.523	1327.14	8.00975	2.678849
73	3.273	2.122	86.359	305.66	95.4304	433.445	346.543	1436.07	7.70054	2.080032
74	3.284	2.23	78.536	255.127	33.4747	135.462	337.162	1266.63	9.943	3.122662
75	3.249	2.106	72.115	275.123	19.081	78.0002	290.793	1417.84	5.98049	2.310591
76	3.291	2.006	145.38	358.093	14.4446	46.2215	486.814	1223.4	0.03386	0.00659
77	3.508	2.173	129.36	310.795	197.172	555.721	536.058	1423.82	0.02408	0.003475
78	2.883	1.934	88.302	291.883	151.619	753.451	393.131	1546.9	0.03605	0.007039
79	2.818	1.918	80.069	288.479	62.1797	231.491	284.199	1013.58	0.07618	0.080144
80	3.079	2.089	70.01	262.628	141.73	694.917	262.68	1168.17	2.55698	1.434012
81	3.047	2.012	88.503	319.394	232.366	978.1	338.836	1498.96	10.3377	5.030122
82	3.14	2.07	74.387	250.016	41.3082	142.748	322.938	1364.69	11.8348	4.00564
83	3.11	2.05	84.216	293.825	52.005	211.669	321.273	1216.22	13.0613	4.474759
84	3.329	2.168	55.716	233.694	133.722	640.356	198.927	853.196	2.97462	0.912973
85	3.219	2.197	76.436	293.736	42.1165	171.736	314.441	1320.54	12.4352	4.80175
86	3.001	2.042	73.194	243.24	108.197	452.001	295.896	1098.86	10.9183	4.26806
87	2.99	1.898	77.043	300.744	35.9663	151.733	350.331	1613.15	12.3139	4.261488
88	3.115	2.013	71.152	254.756	70.0994	358.026	300.969	1267.58	13.5588	5.365936
89	3.268	2.166	27.377	83.5353	15.0334	50.2428	109.669	399.375	12.594	4.23427
90	3.193	1.984	58.236	230.388	4.93652	21.3407	217.867	924.873	13.6516	3.739017
91	3.449	2.149	209.55	546.895	16.8745	43.5931	881.41	3559.6	2.10E-04	0.001855
92	3.601	2.204	128.85	332.911	53.2957	154.127	475.363	1287.51	1.43E-04	3.09E-04
93	3.337	2.321	119.02	313.102	240.208	899.505	494.983	1536.75	1.38E-04	3.44E-05
94	3.242	2.163	102.26	295.194	192.96	702.815	461.236	1744.15	0.10172	0.023514
95	3.025	2.195	127.91	402.193	11.7583	42.0271	523.957	1724.55	0.06797	0.015651
96	3.192	2.035	38.001	120.522	3.17662	13.0068	150.732	560.322	2.87543	0.888252
97	3.335	2.133	44.968	148.809	48.7575	214.772	170.913	639.228	2.69158	0.779042
98	3.161	2.228	37.507	140.007	16.0323	69.5828	152.2	696.301	3.77843	0.923093
99	3.247	1.921	22.364	71.6786	10.4236	38.1663	94.8338	362.614	2.5975	0.818791
100	3.102	1.88	69.616	271.69	74.7902	330.725	280.295	1561.28	0.58943	0.130462

Table B.1: Random Workload Statistics