An Evaluation of Heuristics for SLA Based Parallel Job Scheduling

Viktor Yarmolenko, Rizos Sakellariou
School of Computer Science, The University of Manchester, Manchester M13 9PL, U.K.
{viktor.yarmolenko,rizos.sakellariou}@manchester.ac.uk

Abstract

In the context of SLA based job scheduling for high performance grid computing, this paper investigates the behaviour of various scheduling heuristics to schedule SLA-bounded jobs onto a parallel computing resource. The key objective of this investigation is to evaluate the effectiveness of simple scheduling heuristics using as criteria the maximization of resource utilization (both in terms of time and SLAs serviced) and income. Our results suggest how each SLA constraint ought to be prioritized in order to improve the income.

1 Introduction

There is a growing interest in high performance grid computing to move toward job management based on *Service Level Agreement*s (SLAs) (for example, WS-Agreement (WS-A) [1]). An SLA is essentially a contract outlining service qualities and guarantees. Considering an environment where parallel compute resources are made available through the Grid, the use of SLAs in submitting jobs to those resources may potentially provide more flexibility, increase resource utilisation and accommodate larger numbers of user requests.

SLAs have already been used in the context of providing Internet traffic based services [3]. The use of SLAs job submission on Grid is lagging behind; instead, a traditional queue based system, such as LSF, may be used. These may or may not have a middleware layer that is able to map SLA requests onto a queue scheduler. To be fair, the WS-A specification covers an extensive set of use cases in general.

This paper aims to investigate the behaviour of a number of simple heuristics that could be used to schedule the SLAs onto a parallel compute resource. The evaluation is based on metrics such as resource utilisation and income. Other metrics, used in traditional batch systems, such as the makespan, become irrelevant for the evaluation of an SLA based system. The latter rather uses Quality of Service (QoS) constraints, such as time deadlines, price *etc.* in the form of agreement guarantees embedded in the SLA. The use of time related constraints suggests some similarities with real-time scheduling, whereas other SLA constraints point to work on economic models for Grids.

The paper is structured as follows. Starting with related work in Section 2, it then proceeds to describe the model used (Section 3). Section 4 presents the heuristics used in this study and the scheduling algorithm. This is followed by results and discussion in Section 5 and concluding remarks in Section 6.

2 Related Work

SLAs are not new in the world of customer services; now they are gradually introduced to the world of Grid computing, although relatively little has been done on SLA based job scheduling [5, 7]. In fact, the majority of SLA based scheduling relates to Web Services and networking [4]. The most relevant field to the problem discussed in this paper is real-time scheduling where scheduling decisions, same as with SLAs, are time critical [2]. As seen later, some of the SLA terms used in this study can be mapped onto parameters of real-time tasks which are dynamically optimised.

3 Model Design

We consider a simple model, which consists of the *Client*, the *Provider* and the *Agreement* (SLA) between the two. The Client can represent individual users submitting their requests to the Provider or a broker that negotiates a usage of the resource. The Provider (of a service) represents the owner of a parallel compute resource, which allocates a Client's job based on the Agreement. The Agreement is a coherent contract between the Client and the Provider which is reached as a result of a negotiation. The actual process of the negotiation is not studied here. The role and goals of the three high-level components are highlighted below.

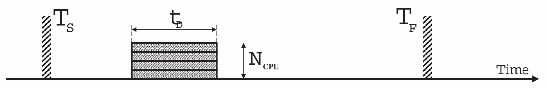

Figure 1. SLA Terms in pictures.

Terms of Agreement An ideal agreement captures unambiguously the requirements of the client as well as capacity and goals of the provider. Adopting the terminology of *WS–Agreement* [1], the important guarantee terms of an SLA can be divided in Service Level Objectives and Business Value List. The former captures requirements, constraints and level of service whilst the latter describes the importance of such requirements individually or as an integrated value indicator. The following guarantee terms are considered in this study:
SLO: Earliest job start time, T_S
SLO: Latest job finish time, T_F
SLO: Reserved time for job execution, t_D
SLO: Number of CPU Nodes required, N_{CPU}
BVL: Final Price Agreed, V_{tot}

Figure 1 provides a graphical representation of the SLO terms. The time line in the figure represents wall-clock time, referenced by the scheduler when booking the job. The deadlines T_S and T_F are values in wall-clock time units and indicate the agreed limits within which the job must be executed. The job duration, t_D, is an estimate for the execution time needed by the job; in this study, we assume that this estimate is equal to the actual time needed by the job. Finally, the number of CPU nodes, N_{CPU}, required for each job imposes a constraint in an orthogonal dimension. The last two SLOs explicitly define the size of the job, or amount of work that is required to be performed by the resource. Later, we define and use the integrated job size, A.

The final price agreed, V_{tot}, depends on all SLOs described here. It can be determined as a fixed price for the service or as a function of any parameter. Later on, we will be looking into both these pricing approaches. In addition, we will evaluate the model in view of three simple pricing policies, which also influence the final value of V_{tot}; in other words, various pricing policies will define the integrated importance of each SLO, by specifying the relationships between the latter.

Client and Provider The main role of the Client is to generate job requests. Each request has a set of constraints such as: N_{CPU}, t_D, T_S and T_F, that are used to form an SLA. The Provider is a module that participates in the negotiation of the job request, controls the resource and applies heuristics to derive a schedule. In this study we only concentrate on the scheduling aspects of the Provider.

3.1 Pricing

A pricing policy can reasonably adequately represent the Provider's goals. On the other hand, any constraints that the client imposes on pricing can also unambiguously define client's priorities. Next we attempt to define two simple constraints on pricing imposed by the Client and a few pricing policies of the Provider. These together would influence the final price for each job done.

3.1.1 The Client's Constraint on Pricing

We choose two different, mutually excluding, pricing constraints imposed by the client which represent either a simple limitation or more complex (such as those advocated in [6]). We refer to those as *Rigid* and *ASAP*. Both types depend only on one temporal parameter, T_{ex} - the execution start time of the job. It is assumed that the job duration time, t_D specified by the Client is indeed the time it takes for the job to complete. Therefore the price of the job as a function of the job start, $V(T_{ex})$ can adequately represent the two constraints:

$$V_{CL}^{j}(T_{ex}) = \begin{cases} 1, T_S \geq T_{ex}^{j} \leq (T_F - t_D) \\ \\ 0, T_S < T_{ex}^{j} > (T_F - t_D) \end{cases} \tag{1}$$

$$V_{CL}^{j}(T_{ex}) = \begin{cases} \left(1 - \frac{t_{ex} - T_S}{T_F - t_D - T_S}\right), T_S \geq T_{ex}^{j} \leq (T_F - t_D) \\ \\ 0, T_S < T_{ex}^{j} > (T_F - t_D) \end{cases} \tag{2}$$

In the first constraint — Rigid (Equation (1)) — the value of $V(T_{ex})$ experiences two discontinuities at $t = T_S$ and $t = T_F - t_D$. In other words, the price of the job is constant between these two t-points, i.e. $V(T_{ex}) = 1$ and is zero otherwise.

The second pricing rule (Equation (2)) — ASAP (As Soon As Possible) — reflects time critical SLAs in which any delay imposes penalty or discount on the final price of the service. Such penalty has (Equation (2)) bears a linear form, but, in principle can be expressed as any analytical function. Effectively, to encapsulate the latter in SLA, one must use a continuous function to describe an SLA guarantee term.

Despite the popularity of the Rigid constraints, there is a growing need for more sophisticated way to describe terms of agreement [6], whilst the general notion itself, of which ASAP rule is a particular case, is not alien for many business models.

3.1.2 The Pricing Policy

Three different and simple pricing policies are discussed here. Together with the Client's pricing constraint they determine the final value of the service recorded in each SLA.

Each of the three pricing rules uses either N_{CPU} or t_D or both parameters to set a fixed price for each job upon its completion. The policies are:

SLA Pricing is based solely on a flat rate charge per SLA:

$$V^j_{SLA} = 1 \equiv const \tag{3}$$

This model represents services that charge on per-access basis, such as phone directory service, when customer pays a flat fee per enquiry irrespective of the time spent to satisfy this enquiry.

Usage Pricing is based on the amount of CPU-hours used with no regard for overheads associated with negotiating the agreement, accessing the service, managing and starting each job, *etc.*:

$$V^j_{CPU} = kA_j = k\left(t^j_D N^j_{CPU}\right) \tag{4}$$

where k is the price unit for the smallest job unit.

Fair Pricing combines the other two pricing rules. The simplest way this can be expressed is shown below:

$$V^j_F = \frac{1}{2}\left(V^j_{SLA} + V^j_{CPU}\right) = \frac{1 + kA_j}{2} \tag{5}$$

here, k plays the role of a weighting factor between two types of charges. Normally, providers will try to use some form of this pricing model, where access to the service is charged separately from the usage of the service. Most of the modern services adopt some form of such a pricing model. For example, the client pays a line rental (fixed charge for the privilege of accessing the service) then a variable amount based on the usage (so much per minute of usage).

4 Scheduling Heuristics

In this section, we consider a number of simple heuristics for scheduling the SLAs. The main principle of those heuristics is the calculation of a priority value that can be used to prioritize the SLAs. Scheduling, then, falls to a simple routine of picking jobs from the list in this order of priority and fitting them onto the resource. The scheduler performs a single iteration fitting as follows:

- Pick up the first/next job from the sorted list.
- Try to find N_{CPU} CPU nodes which are empty from T_S to $(T_S + t_D)$ virtual hours.
- If failed, try to find N_{CPU} CPU nodes which are empty from $T_S+\Delta t$ to $(T_S+\Delta t+t_D)$ virtual hours.
- Repeat the process while $(T_S + \Delta t + t_D) \leq T_F$.
- If failed to find a sufficient number of CPU nodes between T_S and T_F times , reject the request altogether, i.e. do not schedule the SLA.

In general, the priority value, H, is computed by using a function whose generic form is as follows:

$$H = \min\left(h_1 + w \cdot h_2\right) \tag{6}$$

$$H = \max\left(h_1 + w \cdot h_2\right) \tag{7}$$

Here h_1,h_2 is one of $\{T_F,\ T_S,\ t_D,\ N_{CPU},\ A,\ t_T,\ t_L\}$ and w is a weighting coefficient that can be both positive and negative. Briefly: T_S: earliest job start time; T_F: latest job finish time; t_D: reserved time for job execution; N_{CPU}: number of CPU Nodes required; A: objective job size, measured in CPU-hours, $(A = N_{CPU} \times t_D)$; t_T: the deadline tightness, defined by $(t_T = \frac{t_D}{T_F - T_S})$; t_L : the job laxity, similar to t_T, defined as $t_L = T_F - (T_S + t_D)$.

When $w = 0$, the heuristics prioritize on the basis of a single parameter. In addition, the sign of w determines the order of prioritization (ascending or descending). For example, $H = \min\left(T_F - 5 \cdot t_D\right)$, prioritizes SLA requests by the earliest T_F deadline and by the longest job first, whereas the value of 5 balances out the importance of the two parameters. In the experiments we did all possible combinations for the two parameters h_1 and h_2 were considered.

5 Experimental Evaluation

The aim of the experiment is to evaluate the performance of different scheduling heuristics in the light of resource utilisation and the income generated from the scheduling of SLAs. A job is considered successfully executed if all the constraints specified by its SLA are met.

The Setting We start with the description of specific parameters for the Client used in the experiments. We used a Gaussian distribution function to generate requests with a varied number of processors, N_{CPU}, and job duration, t_D. Figure 2 shows the actual distribution of t_D and N_{CPU} per job taken from ten random job sets. The peaks of these two distributions are chosen to reflect real jobs based on the log files of two parallel farms. The Client generates the required percentage of utilisation with the number of CPUs required per job,

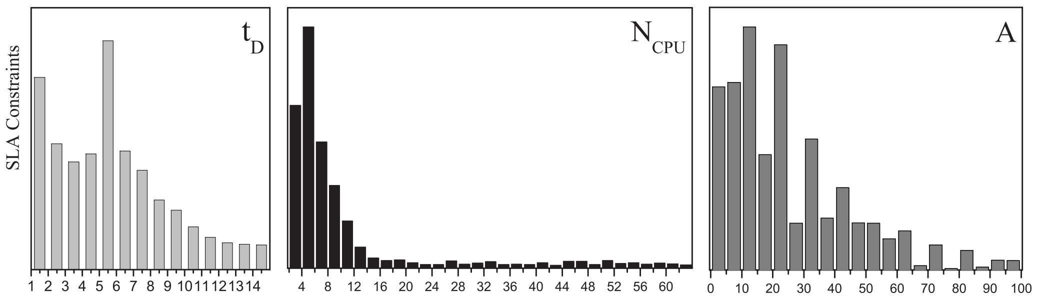

Figure 2. The distribution of t_D, N_{CPU} and A across a typical job set.

N_{CPU}, and the job duration, t_D specified by the distribution function (Figure 2). In the scenarios presented in this paper the Client generated job requests to satisfy 100% utilisation requirement for each job set. The resulting job sets of the requests had the same distribution of N_{CPU} and t_D and could potentially fill 100% of the resource within the specified time period. The sum of all integrated job sizes, A_i, in every job set is constant. The number of jobs, N_{job}, in each job set, however, is varied slightly. On average, around 384 job requests, were generated each time to be scheduled by the Provider. The provider itself consists of a homogeneous parallel resource of 64 CPU nodes. Each set of job requests is generated in such a way that at least one solution exists where the Provider can fit all jobs from the set and therefore reach 100% SLA and CPU utilisation. The time constraints of the job set form a window of 300 virtual hours between the earliest T_S deadlines and the latest T_F deadlines present in this job set, creating therefore a reference frame of 64×300 in size. A preliminary study showed that for the average job length of 5 virtual hours the boundary of 300 hours is a reasonably large time scale to make the size effects on the system's behaviour negligible.

Finally, T_S and T_F boundaries for each job are calculated based on the duration of the job, t_D, using a fixed value for tightness and randomly positioned at either the very end or the very start of each job's generated execution time.

Each experiment was conducted 100 times using a different job set each time and the result was averaged over 100 experiments. The pricing policy was uniform across the job set.

Metrics To evaluate the utilization performance of each scheduling scenario we used two metrics:
The CPU utilization: the amount of the CPU-hours scheduled normalised by the total value of utilization generated from the set of jobs (i.e. 64×300 CPU-hours).
The SLA factor: the amount of the SLAs formed and jobs scheduled normalised by the total number of job requests from the set of jobs.

To evaluate the success of each scheduling heuristic in terms of income, we used the equation below

$$\langle V \rangle_{tot} = \sum_{j}^{N_{job}} \left(V_{CL}^{j} V_{PR}^{j} \right) \tag{8}$$

where V_{CL}^{j} is one of two functions $V(T_{ex})$ that represent the pricing constraint imposed by the Client on job j (Equations (1),(2)), and $V_{PR}^{j} = \{V_{SLA}^{j}, V_{CPU}^{j}, V_{F}^{j}\}$ represents the pricing policy (Equations (4)-(5)).

The Impact on Utilisation We start the discussion of results considering the utilisation performance of different heuristics mentioned in Section 4, that is both SLA utilization and CPU utilization. Due to the nature of the problem, T_S and T_F are expected to be the most influential parameters; prioritizing T_S or T_F on their own should yield a better scheduling compared with any other single parameter discussed in this paper. The dominant influence of T_S and T_F on the performance is supported by preliminary experiments with the single parameter optimisation (Figure 3). The prioritization based on T_S and T_F appears to produce a slightly better schedule than that based on other prioritization. Figure 3 also provides a good reference point for the integrated (combined) heuristics (Section 4), presented below.

The integrated heuristics, where two parameters and weighting are used in prioritization (Equation (6)) generally improve the performance. For each heuristic the value of w was chosen so that the overall performance

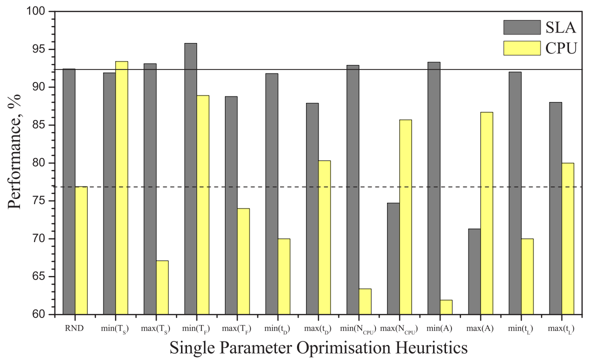

Figure 3. The utilization performance of various single parameter prioritizing heuristics compared to random prioritization. Solid and dash lines conveniently show which parameter prioritization performs better than random job order. The values are in percentage of the complete schedule.

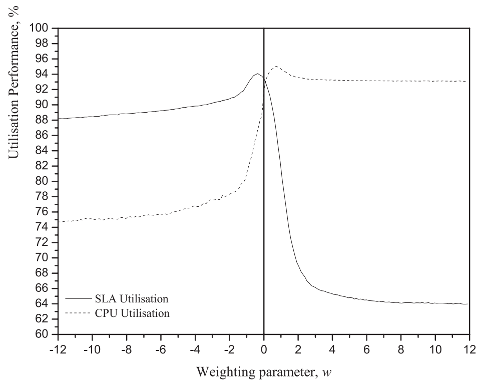

Figure 4. The Resource Utilisation dependence on w for $H = T_S + wN_{CPU}$ heuristics. The solid curve represents SLA utilisation; the dashed curve represents CPU utilisation.

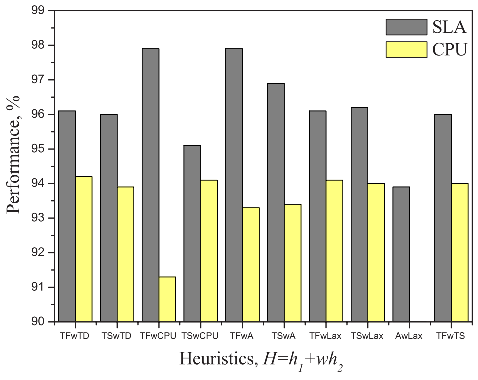

Figure 5. Average utilisation performance (SLA and CPU) for selected integrated heuristics.

of the schedule is maximized. The typical dependence of both SLA utilization and CPU utilisation on w is shown in Figure 4. It can be seen that the optimal value of w is different for maximum SLA utilization and maximum CPU utilization.

Figure 5 presents the best average performance for all the heuristics based on the combination of two parameters with a weight. Again, the performance is expressed as a percentage of the maximum performance for an average job set for both SLA utilization and CPU utilization. The percentage values indicate the highest performance for each heuristic using the most favourable value of w. The precise performance values and the respective values for w are shown in Table 1.

It is noted that the performance of each heuristic depends, to a certain extent, on the size of the resource relative to the average CPU requirement of a job. The preliminary study of this size effect, based on two heuristics (Figure 6), showed that most heuristics favour job requests that require a number of CPUs which is about one eighth ($\frac{1}{8}$) of the resource capacity. The entire behaviour of the w-curve in both these cases does not seem to change much but shifts altogether up or down, depending on the CPU distribution in a job set. All heuristics that use time based variables show similar behaviour. The heuristics based on N_{CPU} and A change in a very nontrivial way with the change of the distribution of CPU requests.

Another important factor that affects performance is the job laxity. For example, when the job laxity was increased five times the performance of $H = T_F + wt_D$ CPU utilisation was increased from 94.2% to 97.5%. On the other hand the efficiency of this very heuristics hardly changed for SLA utilisation, remaining at just over 96%. A further increase of job laxity by slightly over one order of magnitude eventually saturates the performance figures, rendering further increase in laxity redundant. This is generally true for all time based heuristics whereas heuristics based on N_{CPU} and A show some unusual behaviour.

The results suggest that integrating any two heuristics with an appropriate sign and weight, the overall performance of the schedule cannot decrease. In fact,

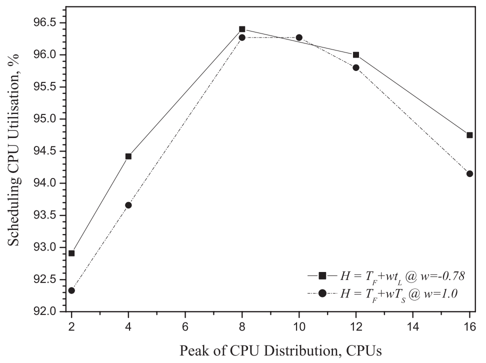

Figure 6. Dependence of the Utilisation performance on the CPU distribution in job requests.

the schedule should improve every time a new parameter is added to the heuristics relationship (e.g. Figure 7). To illustrate this we show utilisation performance optimised using three parameters comparing the results to the respective two-parameter heuristics in Figure 7.

The Impact on Income The second part of our study looks at the effectiveness of the heuristics discussed from an income point of view. We look first at the case of a rigid constraint on pricing in which $V_{CL}(T_{ex}) = const$ (Equation (1)). The typical example of the dependence of the income on the weighting factor of heuristics used is shown in Figure 8. There are two points to be made here. First, there is a direct linear relationship between the income performance and utilisation performance. Unfortunately, the exact maximum possible income cannot be identified due to the specifics of the job generating process. However a rough 100% bar can be placed somewhere just above 384 units for all pricing policies discussed in this paper. The second point is that for all cases of income performance discussed here a fair (V_F) pricing policy yields the same results as the half sum of the other two, which supports a commutative nature of these three policies (Figures 8-9). Therefore, Figure 5 represents adequately the relative performance of income generating effectiveness of that or other heuristics.

Let us now consider the performance of the same heuristics but for SLAs constrained by ASAP (Section 3.1.1). The pricing affected by ASAP constraint is truly dynamic and, ideally, requires dynamic scheduling methods. On the other hand, the Rigid pricing constraint discussed earlier is quasi-dynamic - the income variation on scheduling time is quantised moving the entire scheduling problem closer to the domain of static scheduling.

$H = P(h_1 + wh_2)$	SLA%	w	CPU%	w
$\min(T_F + wt_D)$	96.1	-0.8	94.2	-7.2
$\min(T_S + wt_D)$	96.0	9.4	93.9	2.8
$\min(T_F + wN_{CPU})$	97.9	0.24	91.3	-0.84
$\min(T_S + wN_{CPU})$	95.1	0.72	94.1	-0.36
$\min(T_F + wA)$	97.9	0.03	93.3	-0.135
$\min(T_S + wA)$	96.9	0.135	93.4	0.0
$\min(T_F + wt_L)$	96.1	-0.12	94.1	-0.78
$\min(T_S + wt_L)$	96.2	5.0	94.0	0.3
$\min(T_F + wT_S)$	96.0	-1.0	94.0	2
$\min(A + wt_L)$	93.9	0.06	78.1	$-\infty$
$\max(A + wt_L)$	90.6	$-\infty$	86.7	0
$\min(t_D + wN_{CPU})$	93.8	0.1	84.5	$-\infty$
$\max(t_D + wN_{CPU})$	92.9	$-\infty$	87	0.7
$\min(N_{CPU} + wt_L)$	93.9	1.35	78.4	$-\infty$
$\max(N_{CPU} + wt_L)$	90.7	$-\infty$	87	0.15

Table 1. Table of the highest utilisation performance by various heuristics. The first two columns show the utilisation of SLAs as a percentage of the total possible, followed by the value of w at which the best performance is observed. The next two columns show the same data for CPU utilisation.

The dynamics of this scheduling problem is entirely temporal. Therefore, the maximum SLAs fit or maximum utilisation strategy may conflict with the attempt to schedule each job as close to its T_S deadline as possible. On the other hand, a longer job with the same deadlines may generate more income than the two shorter ones, scheduled sequentially, even for the SLA based pricing policy. Another example is when utilisation may suffer greatly for the sake of maximum income, or, indeed, vise versa, income plunged down when the optimisation of the schedule is geared toward the utilisation. The scheduling technique used does not compare the income of two jobs that can be scheduled at the same time in order to choose the higher income. Neither does it try to swap two jobs with the same deadline to see if it yields a higher income or perform any other income related optimisation, but simply schedules the job at the first available time. Let us now look at the behaviour of these heuristics and whether they can still generate sufficiently good income. The typical behaviour is shown in Figure 9(a), whereas Figure 9(b) shows the reverse performance behaviour. Basically, all heuristics produce reasonable income only when the weighting coefficient takes it away from the region where the job deadlines, T_S and T_F, have some influence. Only $H = A + wt_L$ (Figure 9(b)) and such like (which does not include the job deadlines) seem to

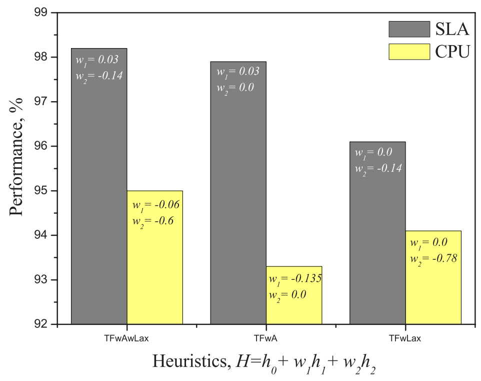

Figure 7. Performance for three-parameter heuristics: $H = T_F + w_1 A + w_2 t_L$ (left), compared to $H = T_F + w_1 A + 0 t_L$ (middle) and $H = T_F + 0A + w_2 t_L$ (right) two-parameter heuristics. Both SLA and CPU utilisation are improved in the three-parameter case.

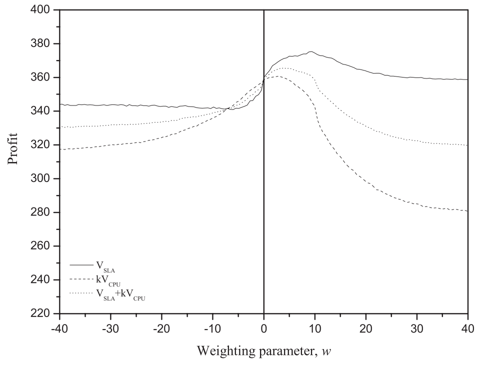

Figure 8. Dependence of the generated income on w for $H = T_S + w t_D$ heuristics. Solid curve represents SLA pricing, dashed curve represents CPU pricing, and dotted curve - Fair pricing (Section 3.1).

benefit from integrated heuristics from Equation (6).

As was expected, the usual strategies, that perform well for static and quasi-static problems, do not seem to work for the new, dynamic, scenario. Analysing graphs from all heuristics it appears that minimising either T_S or T_F only decreases the income that we are aiming to maximise. The problem lies in the fact that many

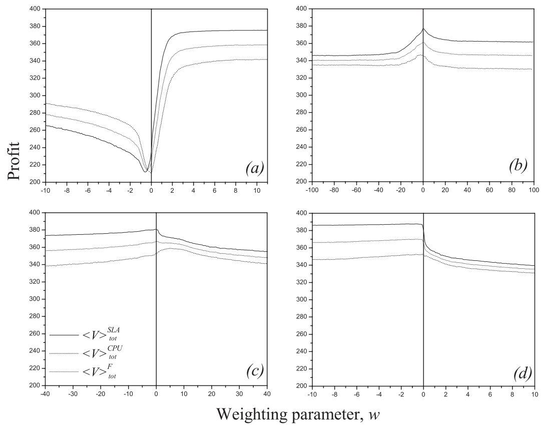

Figure 9. Income made for three different pricing policies ($\max(V_{tot}) \approx 384$) with ASAP constraint. (a) $H = \min(T_S + wN_{CPU})$, (b) $H = \min(A + wt_L)$, (c) $H = \max(T_S + wt_D)$, (d) $H = \max(T_S + wN_{CPU})$.

of the job deadlines are overlapping. These deadlines are purposely moved randomly to one of the extremes to make a scheduling job harder or rather make sure that the deadlines do not accidentally introduce a bias that potentially may improve the performance of an algorithm. Since each job set was generated as a perfect fit, T_S and T_F deadlines that are spread in a uniform way may reveal the set execution times of each job that produces a 100% utilisation. Spreading the deadlines in a random way is an attempt to simulate the job requests in real life.

Going back to the scheduling process, let us look what happens after the first few jobs have been scheduled. The first batch of prioritized jobs would be scheduled at the beginning of their T_S deadline, achieving maximum possible income per job. The next batch of jobs is scheduled back to back with the previous batch to achieve maximum income, however this maximum will no longer be the highest possible income as the T_S deadline overlaps with the jobs that are already scheduled, and so on. It is expected that the income constrained by ASAP will be less than in the case of Rigid constraint.

Table 2 presents income values for all heuristics discussed here. This data supports the earlier discussion regarding the prioritization of T_F and T_S in order of the largest deadlines as a way to maximise the income when ASAP based pricing policies are used. Maximis-

$H = P(h_1 + wh_2)$	$\langle V_{SLA} \rangle$	$\langle V_F \rangle$	$\langle V_{CPU} \rangle$
$\max(T_F + wt_D)$	381	367	359
$\max(T_S + wt_D)$	380	367	359
$\max(T_F + wN_{CPU})$	384	367	357
$\max(T_S + wN_{CPU})$	388	370	352
$\max(T_F + wA)$	385	368	357
$\max(T_S + wA)$	388	370	352
$\max(T_F + wt_L)$	381	367	359
$\max(T_S + wt_L)$	380	366	359
$\min(A + wt_L)$	377	361	347
$\min(t_D + wN_{CPU})$	385	366	346
$\min(N_{CPU} + wt_L)$	385	366	350

Table 2. Table of the highest income performance by various heuristics for three pricing policies.

Pricing	T_F	T_S	t_D	N_{CPU}	A	t_L
ASAP SLA	$\triangle$	$\triangle$	$\triangledown$	$\triangledown$	$\triangledown$	$\triangledown$
ASAP Fair	$\triangle$	$\triangle$	$\triangledown$	$\triangledown$	$\triangledown$	$\triangledown$
ASAP CPU	$\triangle$	$\triangle$	$\triangle$	$\triangledown$	$\triangledown$	$\triangle$
Rigid SLA	$\triangledown$	$\triangledown$	$\triangledown$	$\triangledown$	$\triangledown$	$\triangledown$
Rigid Fair	$\triangledown$	$\triangledown$	$\triangle$	$\triangle$	$\triangle$	$\triangle$
Rigid CPU	$\triangledown$	$\triangledown$	$\triangle$	$\triangle$	$\triangle$	$\triangle$

Table 3. Table of optimisations for different parameters and different pricing policies. $\triangledown$ means that respective parameter has to be minimised in order to achieve maximum respective income, whereas $\triangle$ means that this parameter has to be maximised in order to achieve maximum respective income.

ing of T_F or T_S parameters with the proper weighting usually improves the income of otherwise single optimisation heuristics (Figures 9(c),(d)). This can be explained by the simple example. As job requests are prioritized by the latest T_S there is more chance that most of them will be scheduled at $T_{ex} = T_S$ and not $T_{ex} > T_S$. There is a poorer CPU utilisation in general, however, the jobs scheduled are scheduled immediately on T_S deadline maximising therefore the total income.

6 Conclusions

We performed an evaluation of various scheduling heuristics and presented a comparative analysis of their performance for both resource utilisation and income performance. We showed that using any additional parameter in heuristics never diminishes but often improves the overall performance, subject to weighting and signs. Generally, minimising such parameters as t_D, N_{CPU}, A, t_L leads to an improved V_{SLA} income, whereas maximisation of these and/or minimisation of T_S and T_F deadlines decreases the income. Table 3 shows the rest of the parameters and which prioritization is suitable for which pricing policy or constraint. These can play a role of guidelines showing which of the SLA terms ought to be maximised or minimised in order to achieve the desired performance. Our results indicate that when the job laxity reaches $\approx 0.5 \cdot 10^2$ of t_D the job deadlines no longer affect the performance of the scheduling algorithm. The optimal ratio of the average value of N_{CPU} and the size of the resource is roughly 1:8 in the case of algorithm and heuristics used. The results suggest that integrating any two heuristics with the correct signs and weighting in the prioritization heuristics only increases the overall scheduling performance. The pricing policy can dramatically alter or diminish the effects of various heuristics. Thus for some pricing policies it is irrelevant, to a degree, which heuristic is used.

Acknowledgement This research has been funded by EPSRC, UK (grant reference GR/S67654/01) whose support we are pleased to acknowledge.

References

[1] 'WS-Agreement', White Paper to GGF, Draft 18. 14 May 2004.

[2] S. Aldarmi and A. Burns. Dynamic value-density for scheduling real-time systems. In *11th Euromicro Conference on Real-Time Systems*, Dec. 1999.

[3] M. D'Arienzo, A. Pescapè, S. P. Romano, and G. Ventre. The Service Level Agreement Manager: Control and Management of Phone Channel Bandwidth Over Premium IP Networks. In *ICCC '02: Proceedings of the 15th International Conference on Computer Communication*, pages 421–432, Washington, DC, USA, 2002.

[4] Z. Duan, Z. Zhang, and Y. Hou. Service Overlay Networks: SLAs, QoS, and Bandwidth Provisioning. In *IEEE/ACM Trans. on Networking*, 11(6), 2003.

[5] C. Dumitrescu and I. Foster. GRUBER: A Grid Resource SLA-based Broker. In *Euro-Par 2005*, 2005.

[6] R. Sakellariou and V. Yarmolenko. On the Flexibility of WS-Agreement for Job Submission. In *Workshop on Middleware for Grid Computing 2005 (MGC2005)*, Grenoble, 2005.

[7] V. Yarmolenko, R. Sakellariou, D. Ouelhadj, and J. M. Garibaldi. SLA Based Job Scheduling: A Case Study on Policies for Negotiation with Resources. In *AHM2005*, Nottingham, UK, 20-22 Sep, 2005.