On the Placement of Reservations into Job Schedules

Thomas Röblitz[1,4], Krzysztof Rzadca[2,3,4]

[1] Zuse Institute Berlin, D-14195 Berlin-Dahlem, Germany
[2] Laboratoire ID-IMAG, Grenoble, France
[3] Polish-Japanese Institute of Information Technology, Warsaw, Poland
[4] CoreGRID Institute on Resource Management and Scheduling

Abstract. We present a new method for determining placements of flexible reservation requests into a schedule. For each considered placement the *what-if* method inserts a placeholder into the schedule and simulates the processing of batch jobs currently known to the system. Each placement is evaluated wrt. well-known scheduling metrics. This information may be used by a Grid reservation service to choose the most likely successful placement of a reservation. According to the results of extensive simulations, the *what-if* method grants more reservations and improves the performance of local jobs compared to our previously used *load* method.

1 Introduction

Reserving resources is an accepted technique for delivering Quality-of-Service (QoS) [1]. Without support for reservations, QoS-levels may be achieved by cancelling conflicting tasks [2], or by using "best effort" strategies like assigning higher priority to QoS-critical tasks [3] or predicting the future utilization of resources [4, 5].

In space-sharing resource management systems the diversity of scheduling policies, the inaccurate estimates of job execution times and unknown future job submissions make predictions of future utilization imprecise. These predictions are clearly not sufficient for supporting QoS formed e.g. by Service Level Agreements.

Furthermore, the autonomy of individual Grid resources makes it difficult to coordinate complex requests such as multi-site jobs or workflows. Such coordination could be managed in a peer-to-peer manner, i.e. the sites' local resource management systems agree on the start time of the parts of a complex request. While the complexity of a single peer is small, the control of the overall behavior of multiple peers is difficult. Thus, we favor another approach which is based on the capability to reserve resources at Grid sites in advance. The coordination is therefore achieved at the Grid level by a reservation service which implicitly solves conflicts between individual sites.

As discussed in our previous work [6] a reservation request should allow fuzzy parameters, such as the requested QoS-level, the requested duration and

the start and end times. Fuzzy parameters reduce the communication between the requestor and the reservation service when certain parameters may not be matched, because the reservation service may try alternative configurations automatically. In addition, the reservation scheme may let the Grid site's local resource management system express its preferences on the parameters of a request, i.e. to enforce local scheduling policies and utilization goals.

This paper presents a superior method (*what-if*) to calculate such preferences from the point of view of a Grid site's local resource management system. The *what-if* method calculates different placements of a reservation into a schedule by simulating the current workload including a placeholder for the reservation and measuring well-known scheduling metrics. Using discrete event simulations based on a workload log from the *SDSC Blue Horizon* supercomputer, we found that the *what-if* method performs better than our previously best – the *load* method.

Algorithms for placing reservations have been studied in previous work. Ernemann et al. [7] describe a method for determining available slots at Grid resources. With our method a Grid site does not only determine a list of available slots, but also calculates its preferences for them. [7] also assumes that all jobs have fixed start and end times. Therefore all jobs are in fact reservations. Our method works in situations when users submit both reservations and normal (movable) jobs. Heine et al. [8] propose two schemes for processing rigid reservations requests (with a fixed start time): (1) denying requests conflicting with jobs and (2) delaying jobs to admit more requests. In our approach requests are flexible wrt. the start time. Also, the *what-if* method allows to adjust the admission policy between the two possibilities proposed by [8]. Thus, it is easier to adapt to the systems' need. Smith et al. [9] study the use of advance reservations for co-scheduling multi-site jobs in the Grid. The reservation request specifies a start time for which the algorithm tries to make a reservation. If it fails, the next available start time is taken. As in [7], a resource does not specify its preferences for alternative start times. Through evaluation [9] concludes that backfilling, stopping and restarting jobs and more accurate execution times decrease the impact of reservations on jobs. In our work, we do not consider stopping and restarting of jobs, because this feature is not available on all systems.

The remainder of this paper is structured as follows. In Section 2 we describe basic entities in our context. Then, we briefly present an architecture for processing reservation requests in Section 3. In Section 4 we describe the methods to calculate the availability of a Grid site for placements of a reservation. We experimentally evaluate these methods by simulating a single cluster with normal jobs and reservations in Section 5. We conclude in Section 6.

2 Modelling Resources, Non-reservation Jobs and Reservations

A *resource* R is described by its number of processors R_N. We assume that the Local Resource Management System (LRMS) of a resource uses *first-come-*

first-serve (FCFS) scheduling with *EASY backfilling* [10] for scheduling jobs to available processors.

A *non-reservation job* (short: job) is described by its submission time j_{sbt}, its estimated execution time j_{eet} and its requested number of processors j_{np}. The LRMS has no knowledge about a job before it is submitted to the waiting queue at time j_{sbt}. A job is started by the scheduler at some time j_{stt} (start time). Usually j_{eet} is overestimated and a job finishes sooner. We denote the actual execution time of a job as j_{aet}.

The request parameters of a *reservation* are its submission time r_{sbt}, its duration r_{dur}, its earliest start time r_{est}, its latest end time r_{let} and its number of requested processors r_{np}. The reservation algorithm may place a reservation in the time interval $[r_{est}, r_{let}]$. Because there may be multiple feasible parameter sets satisfying a request, a user may specify its preferences r_{pref} to let a reservation system automatically decide which set should be chosen. When a reservation was granted, its start time and end time are denoted by r_{stt} and r_{edt}, respectively.

A scheduling event occurs when a new job or a reservation is submitted or an existing one is completed. On each such event, the LRMS schedules jobs and reservations in the following order:

1. The LRMS assigns the earliest possible start time to the first job in the waiting queue, such that it does not conflict with running jobs and existing (granted) reservations, and locks the requested processors j_{np} for the estimated execution time of the job j_{eet}.
2. All submitted (but not yet granted) reservation requests are handled in their submission order. Reservations are granted if they do not conflict with already scheduled workload and if they do not delay too much the remaining waiting jobs.
3. The LRMS assigns start times to the remaining waiting jobs using FCFS scheduling and EASY backfilling.

3 Architecture and Mechanism for Processing Reservation Requests

In this section, we briefly introduce the architecture and the reservation procedure as it was proposed in our previous work [6]. Figure 1 shows the involved components and their interaction.

The *Grid Reservation Service (GRS)* provides an interface to the clients, coordinates the processing of a request and selects the best time slot to be reserved. The *Grid Information Service (GIS)* stores information about resources in the Grid, such as the operating system, the total number of processors, the number of running jobs, etc. The *Cluster Reservation Service (CRS)* provides methods for *probing* status information and for *reserving* a time slot.

Upon reception of a request (step ①), the GRS queries the GIS for appropriate candidate resources (steps ② and ③). Then the GRS sends *probe* requests to these

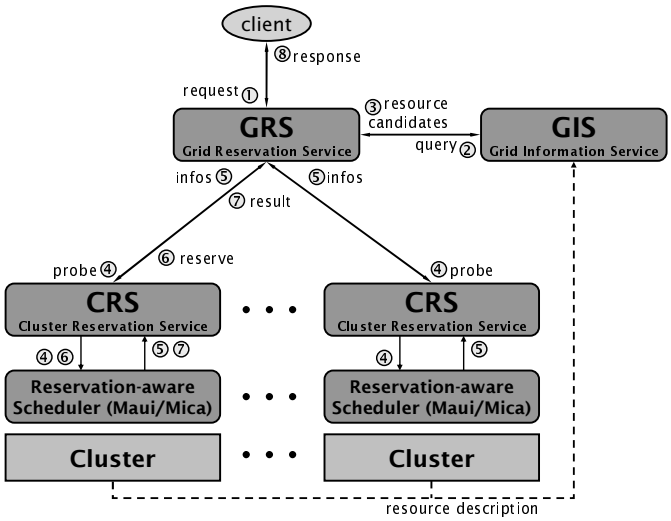

Fig. 1. Architecture and mechanism for processing reservation requests.

candidates to obtain detailed information about their availability (step ④). In each candidate resource, the CRS processes the probe and returns a list of time slots including the requested status information to the GRS (step ⑤). When the GRS has received the responses from the candidates, it orders the time slots according to the user preferences. Next, it tries to reserve the best time slot by sending a *reserve* request to the corresponding CRS (step ⑥). If the request fails (step ⑦), the time slot is removed and the processing continues with step ⑥. If any *reserve* request succeeded or all failed (step ⑦), the GRS sends a response including the result to the client (step ⑧). In case of success, the result contains the agreed start time of the reservation. Otherwise, the result may indicate the reason for the failure.

4 Methods for Placing Reservations

When the CRS receives a *probe* request, it determines a limited number of time slots distributed over the time interval $[r_{est}, r_{let}]$. The number is limited by a configurable constant and a constraint on the minimum time gap between two succeeding time slots [6]. For each time slot the CRS calculates properties such as the availability of the resource, the cost for reserving, etc. In this work we concentrate on methods for determining the availability.

In our previous work [6], we presented three methods for calculating the availability of a time slot. The *load* method performed best in all experiments. Thus, we only use the *load* method in the comparison with the newly developed *what-if* method.

Generally, both the *load* method and the *what-if* method work by defining a function, respectively p^L_{res} and p^W_{res}, which ranks each possible slot for a reservation by assigning a real number from the range $[0, 1]$. The higher the value assigned, the minor the expected impact of the reservation placed in this slot on the performance of local jobs.

4.1 Load Method

The *load* method uses information on the current state of the system: running and waiting jobs and active or pending (but already granted) reservations. This information is used to calculate an approximate end time T for the current workload. Given a time slot $\langle ts_{begin}, ts_{end} \rangle$, we define the function p^L_{res} as follows

$$p^L_{res}(ts_{begin}, T) := \begin{cases} 1, \text{ if } ts_{begin} \geq T \\ 0, \text{ if } ts_{begin} < T \end{cases} \quad (1)$$

The approximate end time T is calculated as follows. First, the remaining execution time (per processor) for running and waiting jobs is determined. Then, it is multiplied by an arbitrarily chosen accuracy factor of 0.5 to take into account the overestimation of jobs' execution times. Next, we iteratively increase the temporary T for reservations which may be active between the current time ct and T. For each reservation r, we add to the time T the area occupied by this reservation $r_{np} \cdot (r_{edt} - \max(r_{stt}, ct))$ divided by the total number of processors in the system R_N.

Although the value for T is only a rough approximation, the method proved to be reasonable in our previous experiments. However, time slots between the current time ct and T are not considered for reservations. Existing reservations starting later than T are also not taken into account. The latter may lead to failing reservation attempts in the presence of existing advance reservations.

4.2 What-If Method

The basic idea of the *what-if* method is to let the availability reflect the impact of a reservation on the non-reservation jobs. For this purpose, the local scheduling system must be able to construct execution plans for jobs without executing them. This requirement is, however, not very restricting, as commonly-used cluster-level schedulers, such as Maui [11] or OAR [12], either provide a simulation mode or can operate in planning mode.

The *what-if* method uses three kinds of execution plans – *original (ORG)*, with *reservation placeholder (RSV)* and with *job placeholder (JOB)*.

Original: The execution plan P^{ORG} for the current workload.

Reservation placeholder: Given a time slot $[ts_{begin}, ts_{end}]$ and a requested number of processors r_{np}, it places a temporary reservation for r_{np} processors from ts_{begin} until ts_{end} into the system. Then it determines the execution plan $P^{RSV/ts_{begin}}$. This procedure is repeated for all time slots.

Job placeholder: The execution plan determines the time when a job with the same requirements would have been started. Therefore a temporary job with an estimated execution time j_{eet} equal to the duration of the reservation request r_{dur} and the same number of requested processors is submitted to the system. The resulting execution plan P^{JOB} defines a new time slot by setting its properties as follows: $ts_{begin} := j_{stt}$, $ts_{end} := j_{stt} + j_{eet}$.

Next, the algorithm calculates the availability p_{res}^{W} for the execution plans $P^{\{JOB,RSV/ts_{begin}\}}$. The availability represents the quality of an execution plan wrt. the well-known scheduling metrics. In this work we measure the makespan and the average completion time of the jobs, which are normalized to map onto the interval $[0,1] \subset \mathbb{R}$. The execution plan P^{ORG} is used as reference in the normalization, possibly defining optimal values for the scheduling metrics. Other well-known scheduling metrics such as slowdown or resource utilization can be easily added if needed. If a time slot cannot be reserved by a reservation placeholder – because it conflicts with running jobs, the first job at the head of the waiting queue (EASY backfilling) or existing reservations – its availability is set to zero.

Let K be the number of jobs in the current workload and P be one of the above execution plans. The start time of job j^i in the execution plan P is denoted by the term $stt(P, j^i)$ where $1 \leq i \leq K$.

In order to assess the quality of a simulated execution plan, the makespan and the average completion time is computed. The makespan $C_{max}(P)$ of an execution plan P is defined as

$$C_{max}(P) := \max_{1 \leq i \leq K}(stt(P, j^i) + j^i_{eet}) \tag{2}$$

C_{max}, stating how long the resource will be occupied, ranks proposed execution plans from the point of view of the owner. Let C^*_{max} denote the minimum makespan for all considered execution plans $P^{\{JOB,RSV/ts_{begin}\}}$.

The average completion time $C_{avg}(P)$ of the jobs in an execution plan P is defined as

$$C_{avg}(P) := \frac{1}{K} \sum_{1 \leq i \leq K} (stt(P, j^i) + j^i_{eet} - j^i_{sbt}) \tag{3}$$

C_{avg} expresses how fast on average the jobs are completed and thus rates plans from the point of view of resource's users. Let C^*_{avg} denote the minimum average completion time for all considered execution plans $P^{\{JOB,RSV/ts_{begin}\}}$.

Let $\omega_{C_{max}}$ denote the weight for the makespan ($\omega_{C_{max}} \geq 0, \omega_{C_{max}} \in \mathbb{R}$) and $\omega_{C_{avg}}$ denote the weight for the average completion time ($\omega_{C_{avg}} \geq 0, \omega_{C_{avg}} \in \mathbb{R}$). We require that $\omega_{C_{max}} + \omega_{C_{avg}} = 1$. Considering a time slot $\langle ts_{begin}, ts_{end} \rangle$, the availability $p^W_{res}(ts_{begin}, P)$ of an execution plan is computed as:

$$p^W_{res}(ts_{begin}, P) := \omega_{C_{max}} \cdot \frac{C^*_{max}}{C_{max}(P)} + \omega_{C_{avg}} \cdot \frac{C^*_{avg}}{C_{avg}(P)} \tag{4}$$

The unweighted parts, both for the makespan and for the average completion time are normalized to fit into the interval $[0,1] \subset \mathbb{R}$. Thus the availability is a number in the interval $[0,1] \subset \mathbb{R}$ too. The more a reservation at ts_{begin} delays the execution of the local jobs, the lower is the availability p^W_{res}.

5 Experimental Evaluation

We evaluated the presented methods with a simulation based on a workload log of the *SDSC Blue Horizon* supercomputer published in [13]. Because we found

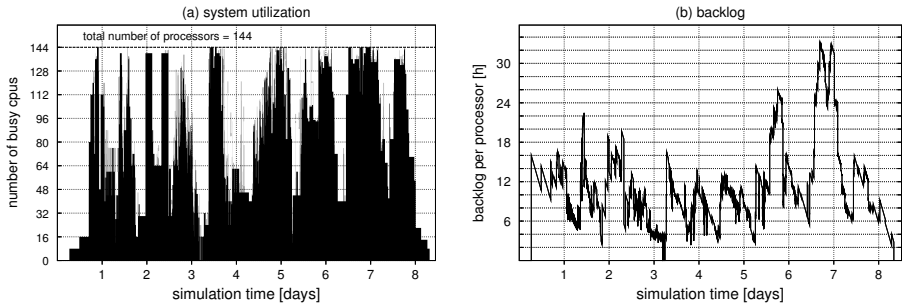

Fig. 2. Non-reservation workload characteristics: (a) system utilization, (b) backlog.

it difficult to implement the *what-if* simulations in Maui, we developed our own scheduler Mica (MICrotus Arvalis, field mouse). In the following sections we will briefly describe the simulation environment including Mica. Then, we describe the workloads used in the experiments. Last, we present the results in detail.

5.1 Simulation Settings

In order to measure the performance of reservations accurately, there was one GRS and one CRS in our testing environment (cf. Figure 1). Instead of interfacing a real system, the CRS communicated with the simulation scheduler Mica. Mica is a simple scheduler capable of FCFS scheduling with EASY backfilling. Mica allows users to submit reservations and normal, parallel jobs. In addition, it can perform what-if simulations for both job and reservation requests.

We used the same workloads as in our previous work [6]. Thus, we were able to compare the behavior of the Mica and Maui schedulers. These workloads were generated as follows. We extracted the first 2000 sound jobs of a workload log of the *SDSC Blue Horizon* supercomputer published in [13]. The workload log contains non-reservation jobs only. Fig. 2 shows (a) the utilization of the processors during the simulation and (b) the backlog[5] of the jobs (right). The complete workload lasts about 12.5 simulation days. Because there are only 5 jobs during the first 4.5 days (388800 s), we only show the data for jobs processed after simulation time 370000 s.

The reservation workload was generated by converting 10% of the jobs into reservations. We split the 2000 jobs into blocks of ten jobs (based on consecutive submission times). From each block we selected a single job, removed it from the non-reservation workload and converted it into a reservation request. In such a request r_{sbt}, r_{np} were copied from the original job. The duration r_{dur} was set to the job's actual execution time j_{aet}. An important parameter of a reservation request is the book-ahead time or advance notice time which is derived as $r_{est} - r_{sbt}$. In order to measure the impact of the reservation's parameters on the algorithm, within a particular workload all reservation requests had the same

[5] The backlog is defined as $\left(\sum_{j \in RUN} (j_{ret} \cdot j_{np}) + \sum_{j \in WAIT} (j_{eet} \cdot j_{np}) \right) / R_N$ with $j_{ret} := j_{eet} - (ct - j_{stt})$ being the remaining execution time of a running job.

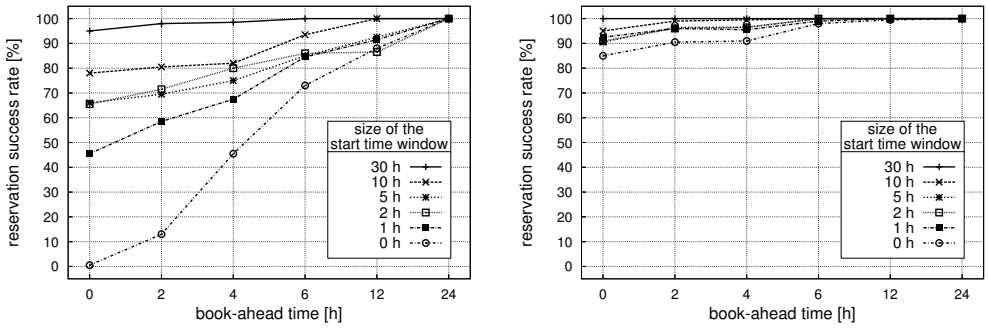

Fig. 3. The reservation success rate for the methods *load* (left) and *what-if* (right).

book-ahead time $bat \in \{0, 2, 4, 6, 12, 24\}$ hours and the same size of the start time window $stw \in \{0, 1, 2, 5, 10, 30\}$ hours. For each request, we set its earliest start time as $r_{est} = r_{sbt} + bat$ and its latest end time as $r_{let} = r_{est} + r_{dur} + stw$. We tested all 36 combinations for each reservation placement method.

5.2 The Reservation Success Rate

The reservation success rate can be defined as a percentage of submitted reservations which were granted by the system. Fig. 3 shows the results in detail for the *load* method (left) and the *what-if* method (right). The average success rate for all workloads is 97% with the *what-if* and 80% with the *load* method. When reservation requests have small start time windows and book-ahead times (in scenarios when both parameters are up to 2 hours), the *what-if* method grants 92% of submitted reservations. The *load* method's success rate is only 38%.

Similarly, we observed that the *what-if* method yields significantly better results than the *load* method when the system load is high. For each reservation request r we have defined the load L_r as the backlog of jobs and reservations at the time r_{sbt}. We have ordered the reservations according to the load L_r. Considering the top 20% reservation requests (i.e. submitted at the highest load situations), the *what-if* method granted 92% of such requests, whereas *load* method – only 58%.

5.3 The Impact of Reservations on Jobs

We use two metrics to demonstrate the impact of reservations on jobs: the makespan (more system oriented) and the waiting time of jobs (more user oriented). The makespan resulting from admitting reservations scheduled by either method was extended by 1% to 8% (approx. 60000 s), compared with the makespan of the workload without any reservation. We found that the *what-if* method performed better than the *load* method in most combinations of book-ahead time and start time window. We do not present the full results because of space limitations.

The waiting time of jobs expresses the performance of the system from the users' point of view. The more jobs are delayed wrt. to their waiting time if no

Table 1. Waiting time results for delayed jobs in simulation runs with book-ahead times (BAT) = 0, 2, 4, 6, 12, 24 in hours and a start time window of 30 hours. All reservation requests (200) were successful.

Experiment	BAT [h]	#jobs	∅ Waiting time [s]		BAT [h]	#jobs	∅ Waiting time [s]	
			original	affected			original	affected
load-Maui	0	186	3191	9296	2	276	2225	6442
load-Mica	0	284	2532	7853	2	341	2138	11572
what-if-Mica	0	172	4155	8112	2	280	3594	8744
load-Maui	4	408	1525	7514	6	356	1939	8436
load-Mica	4	372	1782	11167	6	357	1984	11036
what-if-Mica	4	319	3491	9193	6	429	2426	11081
load-Maui	12	383	1901	8957	24	422	1872	17051
load-Mica	12	390	2244	15405	24	411	1638	12832
what-if-Mica	12	393	1926	16649	24	407	1646	13011

reservations were admitted, the lower is the acceptance of reservations by the users of a system. We only present results for the workloads with a start time window of 30 hours (and different book-ahead times), because all reservation requests were granted with these parameters.

Table 1 shows the number of delayed jobs (#jobs), the average of the original waiting time for these jobs and the average of the affected waiting time. For each book-ahead time we show the results for the experiments load-Maui, load-Mica and what-if-Mica.

The results for the schedulers Maui and Mica using the *load* method differ significantly. Both schedulers show the tendency of delaying more jobs with larger book-ahead times. While the average original waiting times are similar, the difference in the average affected waiting times is much larger. We account the simplified nature of Mica for this.

Comparing the results for the experiments load-Mica and what-if-Mica, the *what-if* method performs better for small book-ahead times (0, 2 and 4 hours). Both the number of delayed jobs and the additional waiting time for these are significantly smaller. With a book-ahead time of 6 hours, more jobs were delayed by the *what-if* method, but the additional waiting time per job was similar to the *load* method. The *load* method performed slightly better than *what-if* in the experiment with 12 hours book-ahead time. With a book-ahead time of 24 hours the results were nearly identical for both experiments.

5.4 Detailed Analysis of the Job Delays

Fig. 4 shows the cumulated additional waiting time of the delayed jobs for the methods *load* (left) and *what-if* (right) using the same experimental settings as in Section 5.3. For each job the additional waiting time was cumulated at the start time of the job.

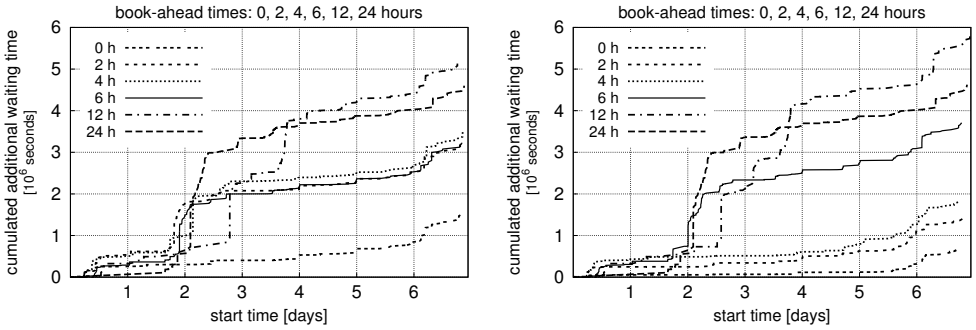

Fig. 4. Cumulated additional waiting time of delayed jobs for the methods *load* (left) and *what-if* (right). The additional waiting time of a job is cumulated at its start time.

The sharp increases of the curves show situations in which blocking jobs or reservations finished and many waiting jobs could start in parallel. Studying the logs of the experiments we found a specific pattern for these situations illustrated in Fig. 5. At the current time 'now' the jobs RJ_1, RJ_2, RJ_3 and RJ_4 are being executed. Already granted (advance) reservations are shown along the horizontal axis (boxes with 'RSV'). These reservations were small in their duration and the number of processors. The waiting jobs (WJ_1, WJ_2 and WJ_3) are planned to start after the currently last reservation.

Note, in particular, the very large job WJ_1 which is planned to start at the end of the last reservation. The number of processors this job requested is close to the total available number of processors R_N. Therefore the job is either blocked by already running jobs or by the sequence of reservations. With EASY backfilling, the job is planned into the schedule to prevent its starvation, after it has advanced to the head of the waiting queue (cf. Section 2). The utilization decreases as the time approaches the scheduled start of the large job. The remaining waiting jobs require more processors than the large job leaves available and their estimated execution time is larger than the time remaining to the start of the large job. Therefore, if they were executed before the large job, the large job would be delayed.

Generally, the further in the future reservations may be placed into the system (latest time is given by the book-ahead time plus the size of the start time window), the later a large job will be planned in. Thus, the unused area in the front of a large job will increase as well. The situation may be improved by the following means:

Job size limitation: We repeated the experiments with changed non-reservation workloads. We restricted all jobs to request less than 89 processors. The job delays were significantly improved and the limitation affected less than 10 jobs (out of 1800).

Improving execution time estimates: The influence of inaccurate execution time estimates was extensively studied in other work [14]. Obviously, jobs with more accurate execution time estimates could fit in the hole before a large job. The situation observed will not disappear completely, but the number of delayed jobs and the extent of their delays could be reduced.

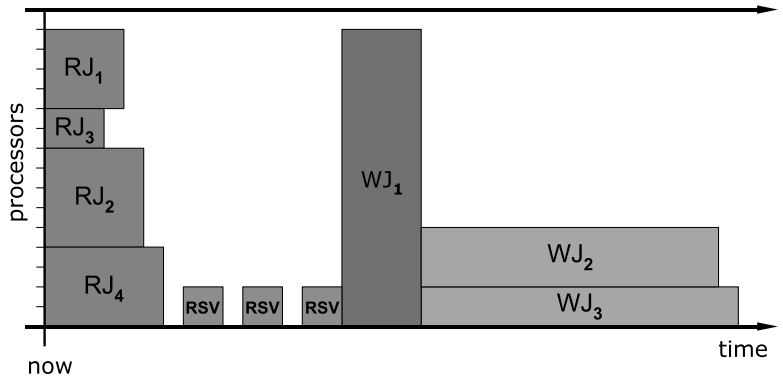

Fig. 5. Pattern causing significant delays of jobs.

6 Conclusion

We presented a new method for placing reservations into a schedule. The *what-if* method simulates the schedule with different placements of the reservation and ranks them according to well-known scheduling metrics, such as makespan and average completion time.

The performance of the *what-if* method was compared with the best method from our previous work, the *load* method, wrt. the reservation success rate, the number of delayed jobs, the average waiting time of delayed jobs and the distribution of the delays. In general, the *what-if* method performs better than the *load* method, except for large book-ahead times and start time windows, where both methods perform equally well. More reservations are granted and, at the same time, the performance of local jobs is better. Clearly, the main advantage of the *what-if* method is the ability to place reservations into holes in the schedule.

A detailed analysis of the simulation data revealed a characteristic pattern when a job requesting a large fraction of processors delays a number of smaller jobs. This pattern accounts for a large fraction of the job delays. When the largest jobs (which represented less than 1% of the load) were shrank, the cumulated additional waiting time of the delayed jobs was significantly decreased. Beside shrinking the size of jobs, more accurate estimates for the execution time of jobs could lower the impact generated by this blocking pattern.

Acknowledgments

The work of the first author was partly funded by German BMBF project AstroGrid-D. Krzysztof Rzadca was partly funded by French CNOUS grant 20045874. This collaborative research work started in the CoreGRID Institute on Resource Management and Scheduling funded by the European Commission (Contract IST-2002-004265). We thank Alexander Reinefeld and the anonymous reviewers for their helpful comments to improve the quality of the paper.

References

1. Foster, I., Kesselman, C., Lee, C., Lindell, R., Nahrstedt, K., Roy, A.: A Distributed Resource Management Architecture that Supports Advance Reservations and Co-Allocation. In: Proceedings of the International Workshop on Quality of Service, IEEE Press: Piscataway, NJ (1999) 27–36
2. Burchard, L.O., Heiss, H.U., Linnert, B., Schneider, J., Kao, O., Hovestadt, M., Heine, F., Keller, A.: The Virtual Resource Manager: Local Autonomy versus QoS Guarantees for Grid Applications. In: Future Generation Grids. Volume 2 of CoreGrid. (2006) 83–98
3. Blake, S., Black, D., Carlson, M., Davies, E., Wang, Z., Weiss, W.: An Architecture for Differentiated Service. RFC 2475 (Informational) (1998) Updated by RFC 3260.
4. Dinda, P.A.: A Prediction-Based Real-Time Scheduling Advisor. In: IPDPS '02: Proceedings of the 16th International Parallel and Distributed Processing Symposium, Washington, DC, USA, IEEE Computer Society (2002) 10–17
5. Downey, A.B.: Using Queue Time Predictions for Processor Allocation. In: IPPS '97: Proceedings of the Job Scheduling Strategies for Parallel Processing, London, UK, Springer-Verlag (1997) 35–57
6. Röblitz, T., Schintke, F., Reinefeld, A.: Resource Reservations with Fuzzy Requests. Concurrency and Computation: Practice and Experience (to appear)
7. Ernemann, C., Yahyapour, R.: Applying Economic Scheduling Methods to Grid Environments. In: Grid Resource Management - State of the Art and Future Trends, Kluwer Academic Publishers (2003) 491–506
8. Heine, F., Hovestadt, M., Kao, O., Streit, A.: On the Impact of Reservations from the Grid on Planning-Based Resource Management. In: Proc. of the International Workshop on Grid Computing Security and Resource Management (GSRM 2005). Volume 3516 of Lecture Notes in Computer Science., Atlanta, USA, Springer-Verlag (2005) 155–162
9. Smith, W., Foster, I., Taylor, V.: Scheduling with Advanced Reservations. In: Proceedings of the 14th International Symposium on Parallel and Distributed Processing, Cancun, Mexico, Washington, DC, USA, IEEE Computer Society (2000) 127–132
10. Lifka, D.A.: The ANL/IBM SP Scheduling System. In: IPPS '95: Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing, London, UK, Springer-Verlag (1995) 295–303
11. Jackson, D.B., Snell, Q., Clement, M.J.: Core Algorithms of the Maui Scheduler. In Feitelson, D.G., Rudolph, L., eds.: JSSPP. Volume 2221 of Lecture Notes in Computer Science., Springer-Verlag (2001) 87–102
12. Capit, N., Costa, G.D., Georgiou, Y., Huard, G., Martin, C., Mouni, G., Neyron, P., Richard, O.: A batch scheduler with high level components. In: Proceedings of the IEEE International Symposium on Cluster computing and Grid 2005 (CCGrid05). Volume 2. (2005) 776–783
13. Feitelson, D.G.: Parallel Workloads Archive. http://www.cs.huji.ac.il/labs/parallel/workload/ (2006)
14. Chiang, S.H., Arpaci-Dusseau, A.C., Vernon, M.K.: The Impact of More Accurate Requested Runtimes on Production Job Scheduling Performance. In: JSSPP '02: Revised Papers from the 8th International Workshop on Job Scheduling Strategies for Parallel Processing, London, UK, Springer-Verlag (2002) 103–127