Enhanced Algorithms for Multi-Site Scheduling

Carsten Ernemann[1], Volker Hamscher[1], Ramin Yahyapour[1], and Achim Streit[2]

[1] Computer Engineering Institute, University of Dortmund, 44221 Dortmund, Germany,
({carsten.ernemann,volker.hamscher,ramin.yahyapour}@udo.edu)
[2] Paderborn Center for Parallel Computing, University of Paderborn, 33102 Paderborn, Germany,
(streit@upb.de)

Abstract. This paper discusses two approaches to enhance multi-site scheduling for grid environments. First the potential improvements of multi-site scheduling by applying constraints for the job fragmentation are presented. Subsequently, an adaptive multi-site scheduling algorithm is pointed out and evaluated. The adaptive multi-site scheduling uses a simple decision rule whether to use or not to use multi-site scheduling. To this end, several machine configurations have been simulated with different parallel job workloads which were extracted from real traces. The adaptive system improves the scheduling results in terms of a short average response time significantly.

1 Introduction

Computational grids gained much attention in recent years as shown by current initiatives like the Global Grid Forum [15] or the implementations of the Globus project [12]. The idea of joining resources that are geographically distributed leads to the expectation to enable access to limited resources. In addition, potentially a larger number of resources is available for a single job. This is assumed to result in a reduction of the average job response time. Moreover, the utilization of the grid computers and the job-throughput is likely to improve due to load-balancing effects between the participating systems.

The management of the grid environment, especially the scheduling of the computational tasks, becomes a rather complex problem. The computational resources in a grid are often high-performance computers as for example parallel machines or clusters. Without grid computing local users are typically only working on the local resources. These resources have their own local management systems. Job scheduling for parallel computers has been subject to research for a long time. However, most real systems still use some kind of list scheduling, as e.g. backfilling [11]. It is the task of a grid scheduler to utilize the local management systems for scheduling jobs on a machine. Previous works discussed several strategies for such a grid scheduler. One approach is the transition and modification of conventional list scheduling strategies for grid usage [16, 5, 6]. Alternatively, economic models are for a grid schedulers are being discussed [2, 3]. In this paper we continue the examination of a non economic grid scheduler with support for multi-site job execution.

The usage of multi-site applications [1] for the grid scenario has been examined in previous works. Multi-site computing is the execution of a job in parallel at different sites. Under the condition of a limited communication overhead, the results from [5] showed that for distinct workloads derived from real machine traces multi-site applications improve the overall average response time. This overhead mainly results from the communication between the job parts. It has been modelled by extending the execution time of a job by a certain percentage.

If a single site job start was possible immediately the applied algorithm always tried to initiate a multi-site execution. In this paper we examine whether the potential benefit of multi-site can be improved by applying heuristics and limitations on the multi-site scheduling and adding a first approach of an adaptive decision process to our algorithm. As in [6], discrete event simulations on the basis of workload traces have been executed for sample configurations.

The results show that the proposed strategy is a significant improvement in terms of average weighted response time.

The following sections are organized as follows. Section 2 gives a short overview on our model for the grid scenario and the applied algorithm are presented in section 3. The evaluation is presented in Section 4. The paper ends with a brief conclusion in Section 5.

2 Model

In this work our scenario is based on independent computing sites with existing local workload and management system. We examine the impact from a job scheduling point of view if the computing sites participate in a grid environment [5, 6, 13]. The sites combine their resources and share incoming jobs. That is, job submissions of all sites are redirected to and then distributed by a grid scheduler. This scheduler exclusively controls all grid resources. We assume that each participating site in the computational grid has a single MPP (Massive Parallel Processor system), which consists of several nodes. Nodes (resources) are single-processor systems with local memory and disc. The nodes are linked with a fast interconnect that does not favor any specific communication pattern inside the machine [10]. The actual mapping of a job on the nodes is neglected. The machines support space-sharing and run the jobs in an exclusive fashion. Moreover, the jobs are not preempted nor time-sharing is used. Therefore, once started a job runs until completion. Furthermore, a job that exceeds its allocated time is cancelled.

Typically a preselection phase takes place ahead of the actual scheduling process and generates a reduced se of resources, that are all suitable for executing the job request. In our study we neglect this preselection and focus on the job scheduling process. Therefore, sites only differ in the number of nodes and all resources are of the same type and all jobs can be executed on all nodes.

The jobs are executed without further user interaction. In our grid computing scenario, a job can be transmitted and executed at a remote site without any additional overhead. In real implementations the transport of data requires additional time which is taken into account in our model as part of the communication overhead, that extents the job's run time. This effect can often be hidden by pre- and postfetching before and after the execution. In this case the resulting overhead is not necessarily part of the scheduling process. The data management is subject to further research.

Jobs are submitted by independent users on the local sites. This produces an incoming stream of jobs over time. Therefore, the scheduling problem is an on-line scenario without any knowledge on future job submissions. Our evaluations are restricted to batch jobs, as this job type is dominant on most MPP systems. Interactive jobs are often executed on dedicated partitions of the system, where the effects of scheduling algorithms are limited [17]. A job request consists of several parameters as e.g. the requested run time and the requested number of resources. After submission a job requests a fixed number of resources that are necessary for starting the job. This number is not changed during the execution of the job. That is jobs are not moldable or malleable [7, 8]. It is the task of the scheduling system to allocate the jobs to resources and determine the starting time.

Newly submitted jobs can be executed on the grid either on a single site/machine or they are split up into several job fragments for *multi-site* execution. These job fragments are executed synchronously. That means a job can run in parallel on a set of nodes distributed over more than one site. This allows the execution of large jobs requiring more nodes than available on a single machine. In this case the impact of bandwidth and latency has to be considered as wide area networks are involved. For our modelling, we summarize the overheads caused by communication and data migration as an increase of the job's run time.

3 Algorithm

In previous research [5, 6, 16] we have examined the potential benefits of using multi-site scheduling in detail. The algorithms used within these projects were extended for this study.

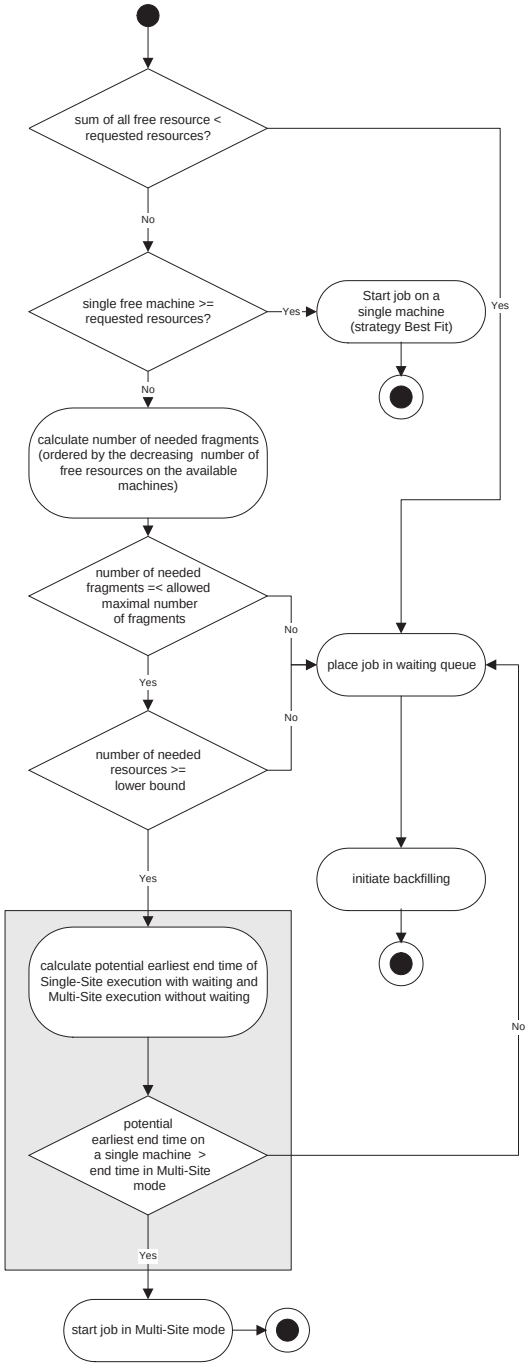

Fig. 1. Algorithm for Multi-Site Scheduling.

Earlier, the effects of using multi-site in comparison to single site scheduling and job sharing have been analyzed [5]. The single site scenario describes the case that all local submitted jobs have to be calculated on the local resources. The job sharing scenario described an environment where jobs can be exchanged between a collection of sites. The results clearly indicated that using multi-site scheduling improves the scheduling results in terms of a better average weighted response time significantly.

The analysis was extended in order to study the effects of an increasing run time for multi-site scheduling resulting from the overhead due to data management and communication [6]. If a job is executed in multi-site, the run time of the job is extended by the overhead which is specified by a parameter **p**. Therefore the new run time r^* is:

$$r^* = (1 + p) \cdot r \tag{1}$$

Where r is the run time for the job running on a single machine. This modelling of the communication overhead is used within this work in all multi-site cases. The results showed that considering the communication overhead multi-site scheduling is still advantageous if the overhead is limited to about 40 % of the run time. Alternative modellings of the overhead calculations were presented in [16].

In the work presented in this paper, we examined the multi-site scheduling behavior by applying constraints for the job fragmentation during the multi-site scheduling. To this end two parameters were introduced for the scheduling process. The first parameter *lower bound* restricted the jobs that can be fragmented during the multi-site scheduling by a minimal number of necessary requested processors. The second parameter was implemented as a vector describing the maximal number of job fragments for certain intervals of processor numbers. This lead to a further improvement of the scheduling process. Based on those results the algorithm was extended by the ability to react on the current status of the system.

The basic scheduling process in the grid scheduler works as described in Figure 1. In a first step the algorithm determines whether or not there are enough free resources at this point in time. If the job cannot be started in single nor multi-site mode immediately the job is added to a waiting queue.

Jobs are scheduled from this queue by a First-Come-First-Serve algorithm in combination with Backfilling ([18, 14]) where each step consists of the method described above. Otherwise a single machine with enough free resource is searched. Upon a successful search the job is started on this machine, which is chosen by using the BestFit-strategy [9], immediately. A search failure leads to the calculation of the needed number of fragments by using a machine list ordered by the decreasing number of free resources. This allows to minimize the number of fragments (job parts running multi-site) for a job. If the number of fragments is larger than a given maximal allowed number of fragments the job is placed in the waiting queue and again backfilling is initiated. Next the parameter *lower bound* is checked e.g. a value of 8 implies that only jobs requesting more than 8 resources may be split up.

In the following we must distinguish between the adaptive and the non-adaptive case. The non-adaptive algorithm directly executes the job in multi-site mode. In the adaptive case the *lower bound* is set to zero and the number of fragments is not limited. Additionally the boxed, greyed area in Figure 1 is inserted. In this part of the algorithm an earliest potential end time of the job running on a single machine is compared to the end time of a multi-site execution with its additional overhead. The solution with the earliest end time is chosen. Note that the real starting time of the job may vary depending on the jobs ahead in the queue.

The algorithm is adaptive as the decision whether to run the jobs in multi-site mode or to wait for a single site execution is done automatically and depends on the given situations in terms of machine schedules and the submitted jobs.

4 Evaluation

For the comprehensive evaluation of the described algorithms for different structures a discrete event simulation was performed. The machine and workload configurations that were examined during these simulations are described in more detailed in [6, 16]. For a better understanding a brief overview of our notation is given in the following two subsections.

4.1 Job Configurations

For our scenario with different independent machines with local workload we use real traces of MPP systems to model this workload. Currently, there are is no real grid workload traces or grid workload model available. Therefore, we use real traces to emulate the workload of single machines participating in a grid environment. To this end workload traces from the 430 nodes IBM RS/6000 SP of the Cornell Theory Center (CTC) are used. Further information on the CTC-trace are available in [21]. The used workloads and their identifiers are summarized in Table 1. Note that the actual workload in future grid environments may vary in the consistence and characteristics. Nevertheless, the used workloads give a good impression of the nowadays demand on computing resources at a MPP.

identifier	description
10_20k_org	An extract of the original CTC traces from job 10000 to 19999.
30_40k_org	An extract of the original CTC traces from job 30000 to 39999.
60_70k_org	An extract of the original CTC traces from job 60000 to 69999.
CTC-syn	The synthetically generated workload derived from the CTC workload traces.

Table 1. Job Configurations

4.2 Resource Configuration

The resource configuration is highly influenced by the job model. As we use workload traces that are derived from a real machine installation with 430 nodes, the resources configuration has to be selected accordingly. The partitioning of the simulated configurations is shown in Table 2. We use configurations with a total of 512 resources (nodes) in order to ensure comparability. A larger grid environment would require additional scaling of the workload without improving the evaluation validity. Note, that the combination of workload and resource configuration impacts the backlog by submitted and not yet started jobs. The existence of this backlog is important for the achievable scheduling quality; e.g. the backfilling strategy relies on the availability of queued jobs to choose from.

identifier	configuration	max. size	number of machines	sum
m64	$4 \cdot 64 + 6 \cdot 32 + 8 \cdot 8$	64	18	512
m64-8	$8 \cdot 64$	64	8	512
m128	$4 \cdot 128$	128	4	512
m256	$2 \cdot 256$	256	2	512
m256-5	$1 \cdot 256 + 4 \cdot 64$	256	5	512
m384	$1 \cdot 384 + 1 \cdot 64 + 4 \cdot 16$	384	6	512
m512	$1 \cdot 512$	512	1	512

Table 2. Resource Configurations

4.3 Performance Metrics

The resource consumption of a job can be described as the product of the job's run time and the number of requested resources. Therefore large jobs have a larger resource consumption than smaller jobs. The resource consumption of a single job j and the total resource consumption can be defined as follows:

$$\text{Resource\_Consumption}_j = (\text{reqResources}_j \cdot (\text{endTime}_j - \text{startTime}_j)) \tag{2}$$

$$\text{Total\_Resource\_Consumption} = \sum_{j \in \text{Jobs}} \text{Resource\_Consumption}_j \tag{3}$$

One measure for the schedule quality is the average response time weighted by the job's resource consumption [19]. We define it as follows:

$$\text{AWRT} = \frac{\sum_{j \in \text{Jobs}} (\text{Resource\_Consumption}_j \cdot (\text{endTime}_j - \text{submitTime}_j))}{\text{Total\_Resource\_Consumption}} \tag{4}$$

With weighting the response time of each job with its resource consumption, therefore all jobs with the same resource consumption have the same influence on the schedule quality. Otherwise a small, often insignificant job would have the same impact on this metric as a big job with the same response time.

As the efficiency of backfilling algorithms is highly dependent on a sufficient backlog [18, 14], the average weighted wait time is used to survey the average weighted backlog of the system.

$$\text{AWWT} = \frac{\sum_{j \in \text{Jobs}} (\text{Resource\_Consumption}_j \cdot (\text{startTime}_j - \text{submitTime}_j))}{\text{Total\_Resource\_Consumption}} \tag{5}$$

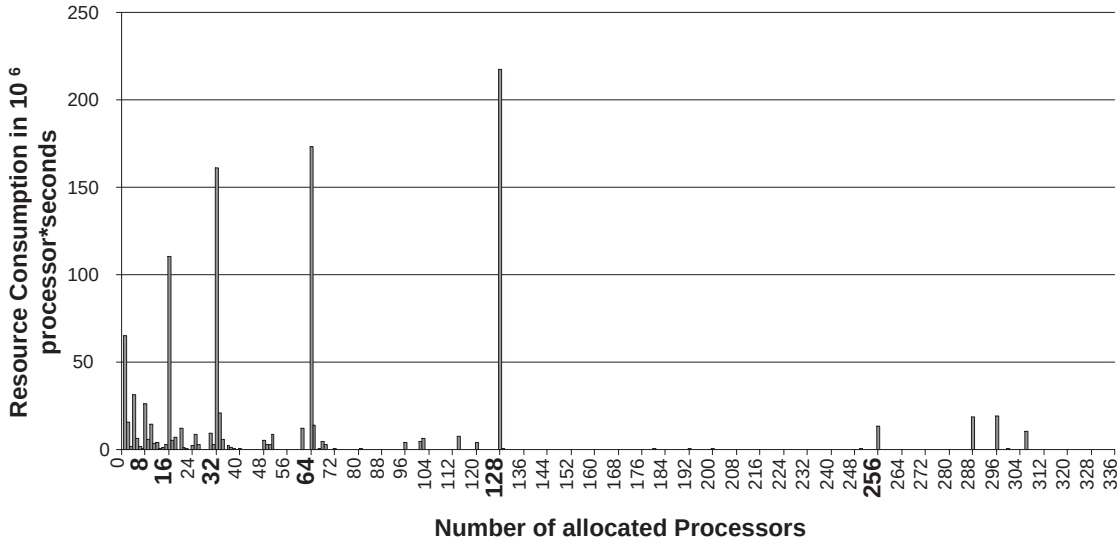

Fig. 2. Resource consumption of jobs in the CTC-syn workload, sorted by requested resources.

4.4 Fragmentation Parameters

The analysis of the used workloads shows a tendency of jobs to size with the power of 2 [4], as seen in Figure 2. Herein the resource consumption, see Equation (3), is summed up for all jobs of a specific width (i.e. the number of requested resources) for the ctc_syn workload. The other workloads show a similar behavior. Because of this "power of 2" focus we also used power of 2 values for the *lower bound* parameter. Jobs that fall short of this *lower bound* can only be scheduled on a single machine and cannot be distributed across several sites. For the maximum number of job fragments we use a function defined over several intervals. We define two configurations with different values for each interval:

1. a *limited* configuration, wherein only necessary fragmentation of a job for the machine configuration with the smallest machines (here: machine configuration m64) is allowed and
2. an *unlimited* configuration, which does not restrict the fragmentation process at all.

For the *unlimited* case the number of fragments is only limited by the maximum number of machines in these configurations and the maximum number of requested resources for a specific job.

requested resources		maximal number of fragments	
lower limit	upper limit	**limited configuration**	**unlimited configuration**
1	4	1	4
5	8	1	8
9	16	1	16
17	32	1	18
33	64	1	18
65	128	2	18
129	512	8	18

Table 3. Maximal fragmentation for the *limited* and *unlimited* configuration

4.5 Evaluation Results

All displayed results were achieved using the CTC_syn workload. As mentioned earlier this workload represents the average behavior of all used data sets. The results vary depending on the used workload, but only within close ranges.

Comparison of *limited* and *unlimited* fragmentation In configurations with smaller machines e.g. *m64*, as shown in Figure 3, and an overhead up to 50% the impact of the *lower bounds* is not as significant as in cases of larger machines or larger overhead. Here the *unlimited* multi-site strategy is far superior to the limited fragmentation strategy.

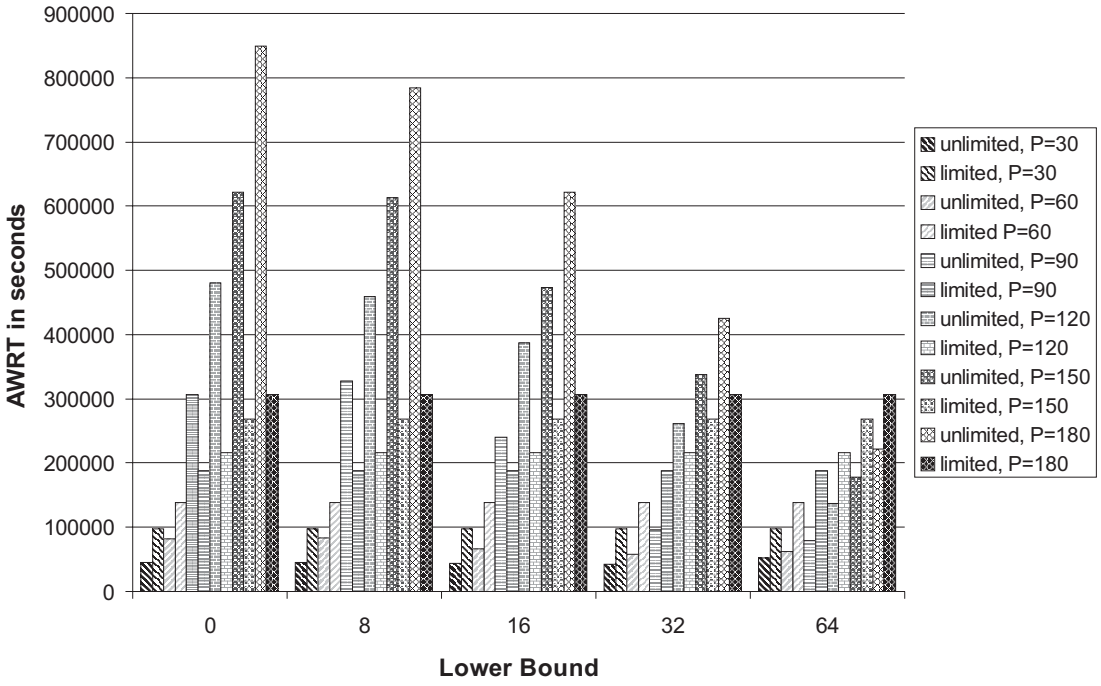

Fig. 3. Comparison of *m64*

The benefit of a higher number of fragments in the *unlimited* configuration ranges at about 60% for an overhead between 10% to 60% and *m64* compared to the *limited* configuration. Whereas the *m64_8* configuration with equally sized machines shows a decrease of the average weighted response time of only around 10% for a higher fragmentation and is more dependent on the *lower bound*. The impact of different machine configurations is described in [6] in more detail.

Especially in resource configurations with smaller machines the average weighted response time scales directly with the overhead parameter p, as major parts of the workload can only be scheduled in multi-site mode, as shown earlier in Figure 2.

Figure 3 verifies the advantages of unlimited fragmentation under the condition of a *lower bound* of 64 and an overhead of up to 180% for small machine configurations. Note, that in this case the average weighted response time may reach values up to twelve times as high as the average weighted response time on a single large machine of 512 nodes.

Impact of lower bounds The influence of the *lower bound* is highly dependent on the additional run time caused by the overhead. Without overhead a multi-site execution would be beneficial in any case. Jobs which run unnecessarily in multi-site manner can be forced to run on a single machine, which then reduces the impact of the overhead on the overall performance. Up to a certain level of overhead the use of multi-site (besides mandatory fragmentation) leads to a better average weighted response time in our simulations.

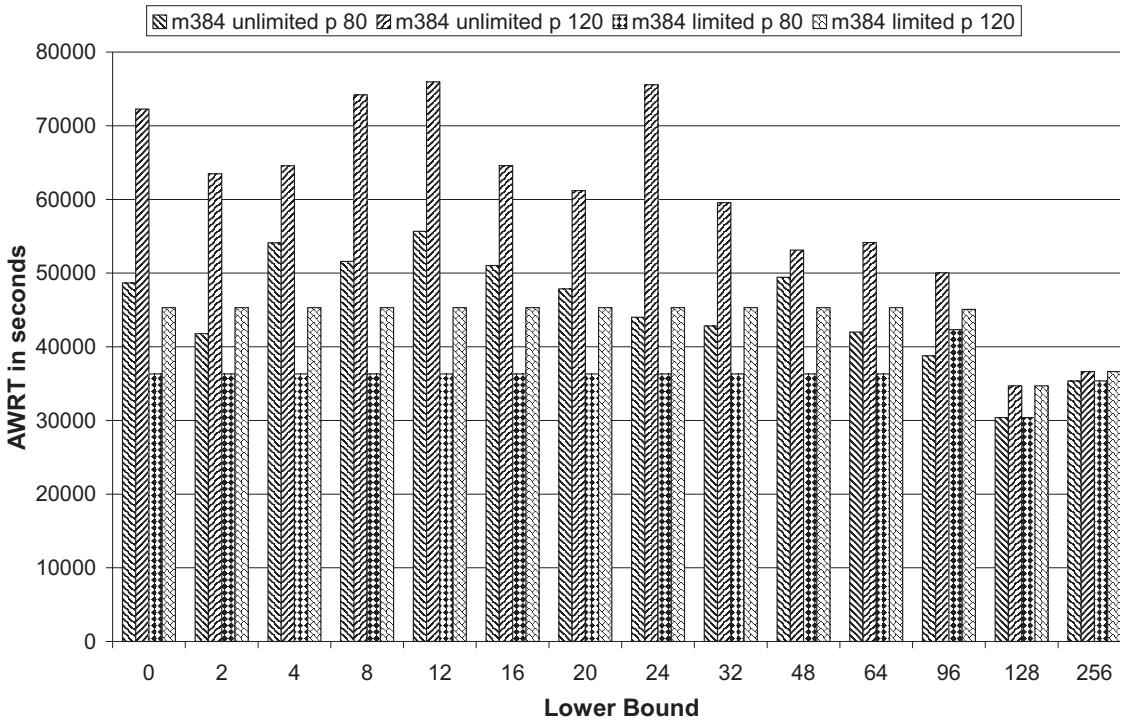

Fig. 4. Influence of the *lower bound* in a *m384* configuration.

This correlation is observed for the *m256* configuration too. Here a *lower bound* of 128 (which is half the size of the biggest machine [20]) proves to be beneficial compared to any other values. For an overhead of 80% a reduction of about 12% is reached and for 120% even 32%.

Figure 4 shows similar results for the *m384* configuration and a *lower bound* of 128: decrease of average weighted response time by 16,3% for p = 80%, and decrease of average weighted response time of 29,9% for p = 120%. Here the limited and the *unlimited* configurations show no difference with regard to the average weighted response time.

Choosing an unappropriate *lower bound* may lead to a performance decrease, e.g. a *lower bound* of 256 results in an increased average weighted response time of 16% and a *lower bound* of 96 increases the average weighted response time by at least 29 % compared to a *lower bound* of 128 in the *m384* configuration, as shown in Figure 4.

The Adaptive Multi-Site Algorithm In Figure 5 the average weighted response time for the resource configurations m512 and *m384* for the best non-adaptive and the adaptive scheduling results are shown. The improvements by using adaptive scheduling in comparison to the non-adaptive case can clearly be identified. The average weighted response time using the adaptive scheduling process increaes only at about 35 % in comparison the the single site machine (m512)

by an overhead of 300 %. In opposite, the average weighted response time of the non-adaptive scheduling system increases of about 244 % with the same overhead of 300 %.

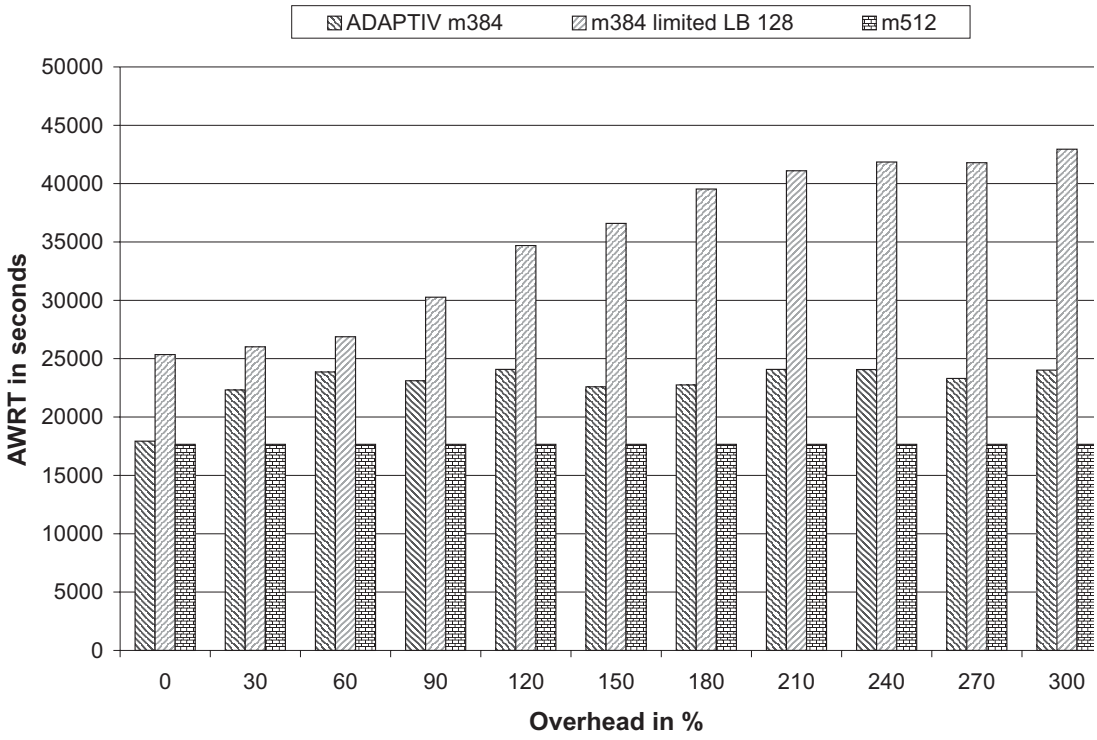

Fig. 5. Comparison of adaptive and best non-adaptive schedules for m384

Note, that using resource configuration *m384* even for the largest jobs multi-site scheduling is not necessary, as the largest job within the workload requests only 336 nodes and can therefore be executed on the largest machine which consists of 384 nodes. Even with an overhead of 300 % using resource configuration *m384* the adaptive scheduling system executes about 4 % of all jobs in multi-site mode. This is only a 42 % reduction compared to an overhead of 30 % in the same configuration. Whereas the resource consumption of the multi-site jobs decreases to 30 % in the same case. This indicates a shift towards the use of smaller, in the sense of resource consumption, jobs for multi-site scheduling. All over, increasing the overhead 10 times from 30 % to 300 % only results in an increase of the average weighted response time of about 7 % in this configuration.

In the best non-adaptive case increasing the overhead 10 times form 30 % to 300 % results in a by 66 % higher average weighted response time. Note, that in the *m384* configuration with any or even without overhead the best non-adaptive multi-site scheduling system performs worse than the adaptive system with an overhead of 300 % as seen in Figure 5. A similar behavior can be observed in all other resource configurations.

In Figure 6 all resource configurations with the adaptive scheduling system are compared. The average weighted response time for the resource configuration m512, given as a reference, is always constant as no multi-site scheduling is invoked.The results of configurations *m64* and *m64_8* show an increasing disadvantage which almost scales with the overhead from 30 % to 300 % up to a factor of about 15. Whereas the influence of the overhead on the overall performance for all other resource configurations is not as significant. The configurations *m128* , *m256* and *m256_5* show a similar behavior in terms of performance. For overheads between 30 % and 300 %, none of these

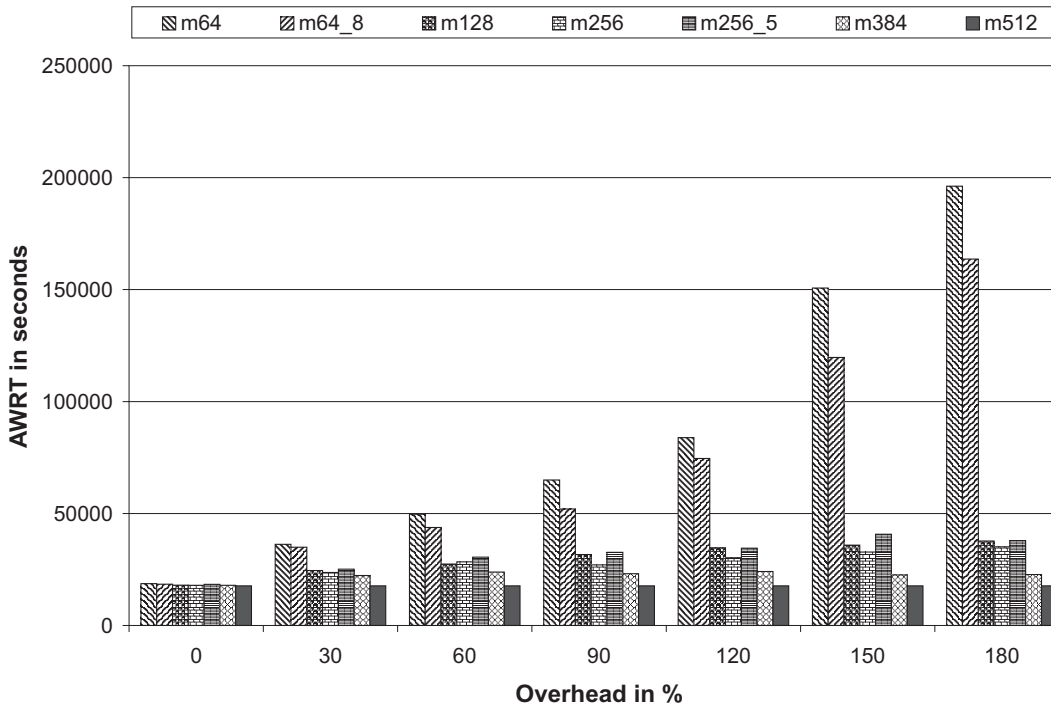

Fig. 6. Comparison of Adaptive Schedules.

three configurations clearly outperforms any of the other two in the adaptive case as shown in Figure 6.

resource configuration	number of multi-site jobs		Δ Jobs	Δ resource consumption
	overhead 30 %	overhead 300 %	in %	in %
m64	1069	430	-60	+176
m64-8	815	387	-53	+132
m128	478	337	-29	+88
m256	336	189	-44	≈ 0
m256-5	589	411	-30	+90
m384	637	368	-42	-71

Table 4. Comparison of adaptive configurations with overheads of 30 % to 300 % in regard to the alternation of the number and the resource consumption of multi-site jobs

This Figure also displays the influence of different machine configurations. The equal partitioned configuration *m64_8* outperforms its non-balanced counterpart *m64*. This is due to the fact that more larger machines are available for the execution of large jobs, leading to more flexibility for the overall scheduling process. For example, the *m64_8* with an overhead of 180 % shows a 20 % lower average weighted response time than the configuration m64. A similar, but not as significant behavior can be observed between the *m256* and *m256_5* configurations.

The difference of the average weighted response time between the configurations *m128* and *m256* is about 7 % at an overhead of 180 %. Whereas the resource configuration *m384* produces with the same overhead much better scheduling results, e.g. for an overhead of 180 % the average weighed response time is 55 % smaller than for configuration *m256* and 66 % smaller than for configuration *m128* . The most significant impact of the adaptive scheduling system can be observed in the results of the resource configuration *m384* as most resources are combined in a single machine and therefore no multi-site scheduling is necessary. Here the multi-site scheduling is only used to enhance the quality of the overall schedule.

5 Conclusion

In this paper we examined three enhancements to a multi-site scheduler for grid computing environments. If a job cannot be placed directly on a single machine, the multi-site scheduler splits the original job in several fragments which are then started synchronously on multiple sites/machines. The communication between job fragments and the additional effort of data migration is considered in extending the jobs runtime by a given percentage. We used a simulation environment to evaluate and compare the performance of different multi-site grid scheduling strategies in terms of average weighted completion time. As job input we extracted job sets from real trace logs. Additionally different machine configurations were evaluated in order to measure the impact of different machine sizes on the way multi-site jobs are distributed across the grid.

In the first enhancements a *lower bound* parameter has been introduced restricting the multi-site execution to jobs requiring at least a certain number of resources. Setting the *lower bound* resulted in a significant improvement of up to 30% in the average weighted response time for certain scenarios. Additionally we restricted the number of fragments to be generated by the multi-site scheduler. For communication overheads with over 60% of the original execution time it proved to be beneficial to limit the fragmentation process.

A different approach was examined using an adaptive version of the multi-site scheduler. Here the scheduler compares the completion times of single- and multi-site execution. For single-site execution the waiting time until the job can be started has to be considered, whereas for multi-site the overhead has to be added to the runtime. Depending on the result the job either is added to the queue or is directly started in multi-site mode. The evaluation showed that the adaptive version substantially improved the performance of the multi-site scheduler, regardless what overhead percentage for the multi-site execution is chosen. Therefore adaptive multi-site scheduler seem to be the best algorithm for a multi-site environment. As the presented adaptive algorithm is a basic implementation, more sophisticated adaptive algorithms may further increase the performance.

Generally, the results indicate that the usage of small systems in combination with multi-site scheduling can not perform as well as single large machines with the same amount of resources. The presented algorithms decrease this difference significantly. In any case the participation in a grid environment seems to be beneficial, as the usage of multi-site computing enables the execution of jobs that consume more resources than available on the largest single machine in such a grid environment.

References

1. M. Brune, J. Gehring, A. Keller, and A. Reinefeld. Managing clusters of geographically distributed high-performance computers. *Concurrency - Practice and Experience*, 11(15):887–911, 1999.
2. R. Buyya, D. Abramson, J. Giddy, and H. Stockinger. Economic models for resource management and scheduling in grid computing. *Special Issue on Grid Computing Environments, The Journal of Concurrency and Computation: Practice and Experience (CCPE)*, May 2002.
3. R. Buyya, J. Giddy, and D. Abramson. An evaluation of economy-based resource trading and scheduling on computational power grids for parameter sweep applications. In *The Second Workshop on Active Middleware Services (AMS 2000), In conjuction with Ninth IEEE International Symposium on High Performance Distributed Computing (HPDC 2000)*, Pittsburgh, USA, August 2000. Kluwer Academic Press.
4. A. B. Downey and D. G. Feitelson. The Elusive Goal of Workload Characterization. Technical report, Hebrew University, Jerusalem, March 1999.
5. C. Ernemann, V. Hamscher, U. Schwiegelshohn, A. Streit, and R. Yahyapour. On Advantages of Grid Computing for Parallel Job Scheduling. In *Proceedings of the 2nd IEEE International Symposium on Cluster Computing and the Grid (CC-GRID 2002)*, pages 39–46, 2002.
6. C. Ernemann, V. Hamscher, A. Streit, and R. Yahyapour. On Effects of Machine Configurations on Parallel Job Scheduling in Computational Grids. In *International Conference on Architecture of Computing Systems, (ARCS 2002)*, pages 169–179. VDE-Verlag, April 2002.

7. D. G. Feitelson, L. Rudolph, U. Schwiegelshohn, and K. C. Sevcik. Theory and practice in parallel job scheduling. *Lecture Notes in Computer Science*, 1291:1–34, 1997.
8. D.G. Feitelson. A Survey of Scheduling in Multiprogrammed Parallel Systems. Research report rc 19790 (87657), IBM T.J. Watson Research Center, Yorktown Heights, NY, February 1995.
9. D.G. Feitelson. Packing Schemes for Gang Scheduling. *Lecture Notes in Computer Science*, 1162:89–101, 1996.
10. D.G. Feitelson and L. Rudolph. Parallel job scheduling: Issues and approaches. In *IPPS'95 Workshop: Job Scheduling Strategies for Parallel Processing, pages 1-18. Springer Verlag, Lecture Notes in Computer Science LNCS 949, 1995.*, 1995.
11. D.G. Feitelson and A.M. Weil. Utilization and predictabillity in scheduling the IBM SP2 with backfilling. In *Proceedings of IPPS/SPDP 1998*, IEEE Computer Society, pages 542–546, 1998.
12. I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *The International Journal of Supercomputer Applications and High Performance Computing*, 11(2):115–128, Summer 1997.
13. I. Foster and C. Kesselman, editors. *The GRID: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1998.
14. R. Gibbons. A Historical Application Profiler for Use by Parallel Schedulers. In *Job Scheduling Strategies for Parallel Processing*, pages 58–77. Springer, Berlin, Lecture Notes in Computer Science LNCS 1291, 1997.
15. The grid forum, http://www.gridforum.org, June 2002.
16. V. Hamscher, U. Schwiegelshohn, A. Streit, and R. Yahyapour. Evaluation of job-scheduling strategies for grid computing. *Lecture Notes in Computer Science*, 1971:191–202, 2000.
17. S. Hotovy. Workload Evolution on the Cornell Theory Center IBM SP2. In D.G. Feitelson and L. Rudolph, editors, *IPPS'96 Workshop: Job Scheduling Strategies for Parallel Processing*, pages 27–40. Springer–Verlag, Lecture Notes in Computer Science LNCS 1162, 1996.
18. D. A. Lifka. The anl/ibm sp scheduling system. In D. G. Feitelson and L. Rudolph, editors, *IPPS'95 Workshop: Job Scheduling Strategies for Parallel Processing*, pages 295–303. Springer, Berlin, Lecture Notes in Computer Science LNCS 949, 1995.
19. U. Schwiegelshohn and R. Yahyapour. Analysis of First-Come-First-Serve Parallel Job Scheduling. In *Proceedings of the 9^{th} SIAM Symposium on Discrete Algorithms*, pages 629–638, January 1998.
20. U. Schwiegelshohn and R. Yahyapour. Fairness in parallel job scheduling. *Journal of Scheduling, 3(5):297-320. John Wiley*, 2000.
21. Parallel Workloads Archive. http://www.cs.huji.ac.il/labs/parallel/workload/, June 2002.