On Advantages of Grid Computing for Parallel Job Scheduling

Carsten Ernemann, Volker Hamscher, Uwe Schwiegelshohn, Ramin Yahyapour
Computer Engineering Institute
University Dortmund, 44221 Dortmund, Germany
{*carsten.ernemann,volker.hamscher,uwe.schwiegelshohn,ramin.yahyapour*}*@udo.edu*

Achim Streit
PC^2 - Paderborn Center for Parallel Computing
33102 Paderborn, Germany
streit@upb.de

Abstract

This paper addresses the potential benefit of sharing jobs between independent sites in a grid computing environment. Also the aspect of parallel multi-site job execution on different sites is discussed. To this end, various scheduling algorithms have been simulated for several machine configurations with different workloads which have been derived from real traces. The results showed that a significant improvement in terms of a smaller average response time is achievable. The usage of multi-site applications can additionally improve the results as long as the increase of the execution time due to communication overhead is limited to about 25%.

1 Introduction

Grid computing is intended to offer seamless access to rare and limited resources as e.g. high-performance parallel computers. A computational grid (see [2, 8]) is the cooperation of distributed computer systems where user jobs can be executed either on local or on remote computer systems. The idea is similar to the former metacomputing [17] where the focus was limited to compute resources while grid computing takes a broader approach [13, 7]. On one hand it provides the user with access to locally unavailable resource types. On the other hand there is the expectation that a larger number of resources is available. It is expected that this will result in a reduction of the average job response time. Also the utilization of the grid computers and the job-throughput is likely to improve due to load-balancing effects between the participating systems.

Now, parallel computing resources are usually not exclusively dedicated to grid computing. Furthermore, they are typically not owned and maintained by the same administrative instance. Research institutes are an example for such resource owners, as well as laboratories and universities. Without grid computing local users are usually only working on the local resources. The owners of those computing systems are interested in the consequences of participating in a computational grid and whether such a participation will result in better service for the users by improving the job response time. Therefore, we want to determine the practical benefit of the collaboration of computing sites.

The usage of multi-site applications has been theoretically discussed for quite some time [1]. Multi-site computing is the execution of a job in parallel at different sites. This results in a larger number of totally available resources for a single job. The effect on the average job response time is yet to be determined as there are only few real multi-site applications. This lack of real multi-site applications may be the result of a absence of a common grid computing environment which is able to support allocating resources in parallel on remote sites. In addition many user fear a significant adverse effect on the computation time due to the limitations in network bandwidth and latency over wide-area networks. This overhead depends on the communication requirements between the process parts of a particular application. As WAN networks become ever faster, this overhead may decrease over time. It is therefore a subject of the paper to determine which amount of overhead will still result in an overall user benefit.

To evaluate the effect of multi-site applications in a grid environment, we examine the usage of multi-site jobs in addition to job sharing. To this end, discrete event simulations on the basis of workload traces have been executed for sample configurations. The potential benefit is evaluated if a computing site participates in a computational grid. The paper is focused on the question whether sharing jobs

Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID'02)
0-7695-1582-7/02 $17.00 © 2002 IEEE

between sites and/or using multi-site applications provide advantages in mastering the existing workload.

The paper is organized as follows. First, our scheduling model is discussed in the next section. In Section 3 we present the used algorithms. The simulations and their results are presented and discussed in Section 4. The paper ends with a brief conclusion.

2 Models

2.1 Site Model

As already mentioned before, we assume a computing grid consisting of independent computing sites with their local workloads. That means that each site has its own computing resources as well as local users that submit jobs to the local job scheduling system. In a typical single site scenario all jobs are only executed on local resources.

The sites may combine their resources and share incoming job submissions in a grid computing environment. Here, jobs can be executed on local *and* remote machines. The computing resources are expected to be completely committed to grid usage. That is job submissions of all sites are redirected and distributed by a grid scheduler. This scheduler exclusively controls all grid resources. For a real world application this may be a requirement difficult to fulfill. There are other possible implementations where site-autonomy is still maintained.

2.2 Machine Model

We assume massive parallel processor systems (MPP) as the computing resources where each site has a single parallel machine that consists of several nodes. Each node has its own processor, memory, disk etc. The nodes inside a machine are connected with a fast interconnection network that does not favor any communication pattern inside the machine [4]. This means a parallel job can be allocated on any subset of nodes of a machine. This model comes reasonably close to real systems like an IBM RS/6000 Scalable Parallel Computer, a Sun Enterprise 10000 or a HPC cluster.

For simplicity all nodes in this study are identical. The machines at the different sites only differ in the number of nodes. The existence of different resource types would limit the number of suitable machines for a job. In a real implementation such a preselection is part of grid scheduling and normally executed before the actual scheduling process takes place. After the preselection phase the scheduler can ideally choose from several resources that are suitable for the job request. In this study we neglect this preselection process and focus on the scheduling result. Therefore, it is assumed that all resources are of the same type and all jobs can be executed on all nodes.

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, in our study we do not consider the case that a job exceeds its allocated time. 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 [5, 3].

2.3 Job Model

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.

We restrict our simulations on batch jobs, as this job type is dominant on most MPP systems. For interactive jobs there are usually dedicated machine partitions where the effect of the scheduling algorithm is quite limited. In addition interactive jobs are usually executed on local resources.

It is the task of the scheduling system to allocate the jobs to resources and determine the starting time. Then the job is executed without further user interaction. Data management of any files is neglected in this study. In our grid computing scenario, a job can be transmitted to a remote site without any overhead. In a real implementation the transport of data requires additional 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.

In a grid environment we assume the ability of jobs to run in multi-site mode. That means a job can run in parallel on a node set distributed over different sites. This allows the execution of large jobs that require more nodes as available on a single machine in the grid environment. The impact of bandwidth and latency has to be considered as wide-area networks are involved. In the simulations, we will address this subject by increasing the job length if multi-site execution is applied to a job.

2.4 Scheduling System

In parallel job scheduling on single parallel machines, simple first-come-first-serve (FCFS) strategies have often been applied. As an advantage, these algorithms provide some kind of fairness (see [15]) and are deterministic for the user. Nevertheless, it can result in poor quality if jobs with large node requirements are submitted. To this end, a strategy called backfilling has become standard on most systems. It requires knowledge of the expected job execution time and can be applied to any greedy list schedule. If the next job in the list cannot be started due to a lack of available resources, backfilling tries to find another job in

the list which can use the idle resources. But it will not postpone the execution of the next job in the list. The backfilling algorithm has been introduced by Lifka [12].

In our scenario for grid computing, the task of scheduling is delegated to a grid scheduler. The local scheduler is only responsible for starting the jobs after allocation by the grid scheduler. Note, that we use a central grid scheduler for our study. In a real implementation the architecture of the grid scheduler can differ as single central instances usually lead to drawbacks in performance, fail-safety or acceptance of resource users and owners. Nevertheless, distributed architectures can be designed that act similar as a central grid scheduler.

3 Algorithms

Three scenarios have been examined in this paper: *job-sharing* between computing sites in a small grid environment, in addition with *multi-site* computing and as a reference a scenario with the normal *local job* processing of independent sites.

3.1 Local Job Processing

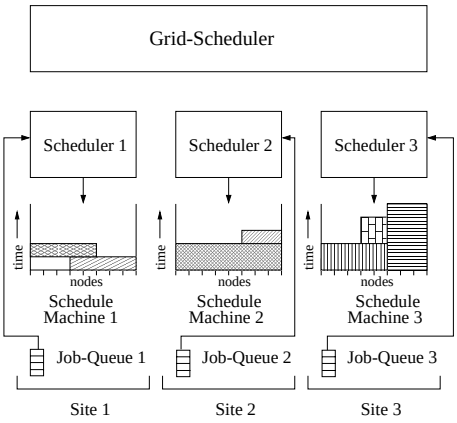

Figure 1. Sites executing all jobs locally

This scenario refers to the common situation where the local computing resources at a site are dedicated only to its local users (see Figure 1). A local workload is generated at each site. This workload is not shared with other sites. In our examination the forementioned backfilling scheduler is applied.

There are 2 variants of backfilling as described by Feitelson and Weil [6]. *EASY backfill* is the original method by Lifka [12]. It has for example been implemented for several IBM SP2 installations [16]. While EASY backfill will not postpone the *projected* execution of the next job in the list, it may increase the completion time of jobs further down the list, see [6]. The other variant is *Conservative backfill* will not increase the *projected* completion time of a job that has been submitted before the job which is currently backfilled [6]. On the other hand conservative backfill requires more computational effort than EASY.

In this work we just present the results for EASY backfill algorithm as its performance proved to be more effective in our simulations than conservative backfilling.

3.2 Job Sharing

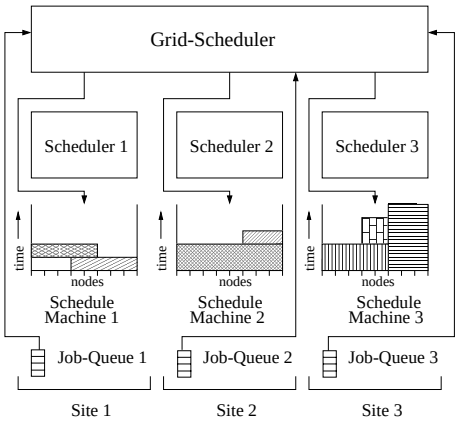

Figure 2. Sites sharing jobs and resources

In the *job-sharing* scenario all jobs submitted at any site are delegated to the grid scheduler as seen in Figure 2. In our examination the scheduling algorithms in grid computing consist of two steps. In the first step the machine is selected and in the second step the allocation in time for this machine takes place.

Machine Selection:

There are several methods possible for selecting machines. Earlier simulations results (presented in [9]) showed good results for a selection strategy called *BestFit*. Here, the machine is selected on which the job leaves the least number of free resources if started.

Scheduling Algorithm:

Here, the backfilling strategy is applied for the single machines as well. This algorithm has shown best results in previous studies.

3.3 Multi-Site Computing

This scenario is similar to *job sharing*: a grid scheduler receives all submitted jobs. Additionally, jobs can now be executed crossing site boundaries (see Figure 3).

There are several strategies possible for multi-site scheduling. For this work we use a scheduler which first

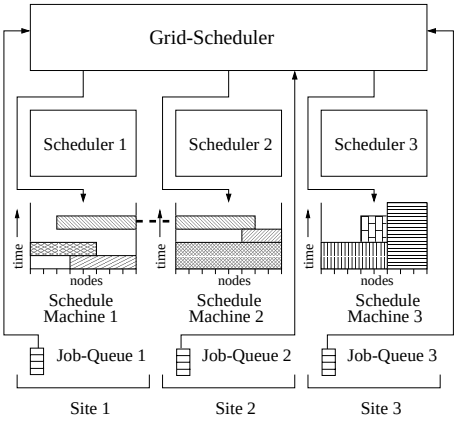

Figure 3. Support for multi-site execution of jobs

tries to find a site that has enough free resources for starting the job. If such a machine is not available, the scheduler tries to allocate the jobs on resources from different sites. To this end the sites are sorted in the descending order of free resources and allocating the free resources in this order for a multi-site job. In this case the number of combined sites is minimized. If there are not enough resources free for a job, it is queued and normal backfilling is applied.

Spawning job parts over different sites usually produces an additional overhead. This overhead is due to the communication over slow networks (e.g. a WAN). Consequently, the overall execution time of the job will increase depending on the communication pattern. For jobs with limited communication demand there is only a small impact. Note, without any penalty for multi-site execution, the grid would behave like a single large computer. Hence, multi-site scheduling will outperform all other scheduling strategies. In this study we examine the effect of multi-site processing on the schedule quality under the influence of a communication overhead. To this end, the influence of the overhead is modelled by extending the required execution time r_i to r_i^* for a job i that runs on multiple sites by a constant factor: $r_i^* = (1 + p) \cdot r_i$ with p = 0 .. 40 % in steps of 5%.

4 Evaluation

For the evaluation of the different structures and algorithms a discrete event simulation was performed. Several machine configurations have been examined for the forementioned algorithms.

4.1 Machine Configurations

All configurations use a total of 512 resources. Those resources are partitioned in the various machines as shown in Table 1. The configurations *m128* and *m256* are representations of several sites with equal machines. They are balanced as there is an equal number of resources at each machine. The configuration *m384* represents a large computing center with several client sites. The configuration *m64* represents a cluster of several sites with smaller machines.

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

Table 1. Resource Configurations

Finally, a reference configuration *m512* consists of a single site with one large machine. In this case no grid computing is used and a single scheduler can control the whole machine without any need to split jobs.

4.2 Workload Model

Unfortunately, no real workload is currently available for grid computing. For our evaluation we derived a suitable workload from real machine traces. These traces have been obtained from the *Cornell Theory Center* and are based on an IBM RS6000/SP parallel computer with 430 nodes. For more details on the traces and the configuration see the description of Hotovy [10]. The workload is available from the standard workload archive [18].

In order to use these traces for this study it was necessary to modify the traces to simulate submissions at independent sites with local users. To this end, the jobs from the real traces have been assigned in a round-robin fashion to the different sites. It is typical for many known workloads to favor jobs requiring a power of 2 nodes. The CTC workload shows the same characteristic. The modelling of configurations with smaller machines would put these machines into disadvantage if the number of nodes is not a power of 2. To this end, our configurations consist of 512 nodes. Nevertheless, the traces consist of enough workload to keep a sufficient backlog on all systems (see [9]). The backlog is the workload that is queued at any time instance if there are not enough free resources to start the jobs. A sufficient backlog is important as a small or even no backlog indicates that the system is not fully utilized. In this case there is not enough workload available to keep the machines working. Many schedulers, e.g. the mentioned backfilling strategy, require that enough jobs are available for backfilling in order to utilize idle resources. This case usually leads to a bad scheduling quality and unrealistic results.

Over all the quality of a scheduler is highly dependent on the workload. To minimize the risk to achieve singular

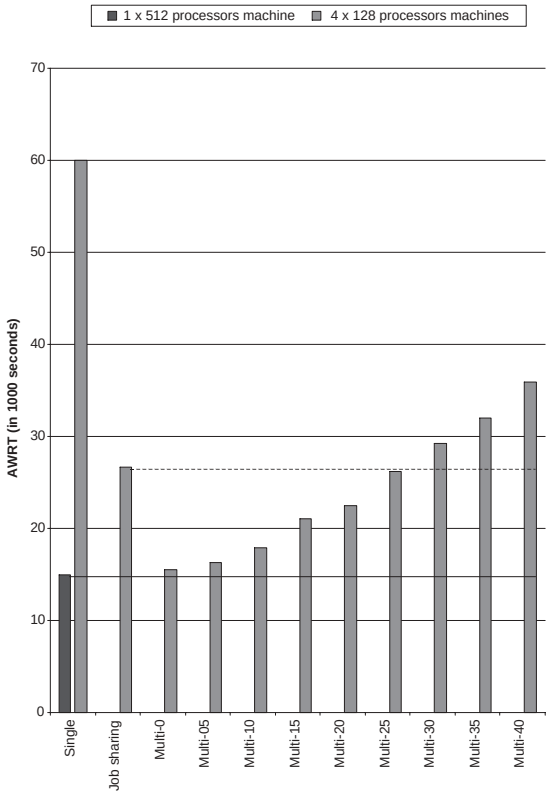

Figure 4. Average Weighted Response Time for the *m128* configuration and workload *ctcsyn* with modification *2*

effects the simulations have been done for 4 workload sets:

- A synthetic probabilistic generated workload on the basis of the CTC traces.
- 3 extracts of the original CTC traces.

The synthetic workload is very similar to the CTC data set [11]. It has been generated to prevent that these singular effects in real traces do not affect the accuracy of the result. Also the usage of 3 extracts of the real traces are used to get information on the consistency of the results for the CTC workload. Each workload set consists of 10000 jobs which corresponds in real time to a period of more than three months.

A problem of such simulations is the handling of wide jobs contained in the original workload traces. The widest job in the CTC traces e.g. requests 336 processing nodes. On one hand these jobs can be used in simulations with multi-site. Here jobs can be split over different sites to get more resources than available at a single site. On the other hand, some of these jobs cannot be started in simulations of scenarios with only local execution or job sharing.

To permit the validity of comparing simulation results, jobs cannot be neglected. Therefore we assume that the workload of wide jobs is still generated at a single site. Wide jobs are split up into several parts of the machine size to allow their execution. Accordingly, the job size is limited by the size of the largest machine in the *job-sharing* scenario. Here, users can submit jobs that are wider than the local machine size. To allow the comparison of different scenarios the following modifications have been applied to each forementioned workload.
Workload Modifications:

1. Wide Jobs are split in parts of local machine size,
2. Wide Jobs are split in parts of largest machine size in the configuration,
3. Wide jobs are unchanged.

The workloads with modification *1* were executed in all 3 scenarios. The workloads with modification *2* were simulated on scenario *job-sharing* and *multi-site* while modification *3* was only used for the *multi-site* scenario. Note, that all of these modification do not alter the overall workload. We assume that a certain amount of workload exists at a local site. Depending on the scenario a user may submit jobs larger then the local machine. The simulations allow the examination of the impact caused by wider multi-site jobs on the schedule.

4.3 Results

The simulation results show that job-sharing provides significant improvement over local job execution for the user. As a measure the average weighted response time is used in this study. The response time of each job is the difference between the completion time and the submission time. The response time of each job is weighted by its resource consumption. The average weighted response time is the sum of all weighted response times divided by the sum of the resource consumptions of all jobs.

Average Response Time weighted by Width:

$$\text{AWRT} = \frac{\sum_{j \in \text{Jobs}} (\text{j.reqResources} \cdot (\text{j.endTime} - \text{j.submitTime}))}{\sum_{j \in \text{Jobs}} \text{j.reqResources}}$$

Note that the mentioned weight prevents any prioritization of small over wider jobs in regard to the average weighted response time if no resources are left idle [14]. The average weighted response time is a mean for the schedule quality from the user perspective. A shorter AWRT indicates that the users have to wait less for the completion of their jobs.

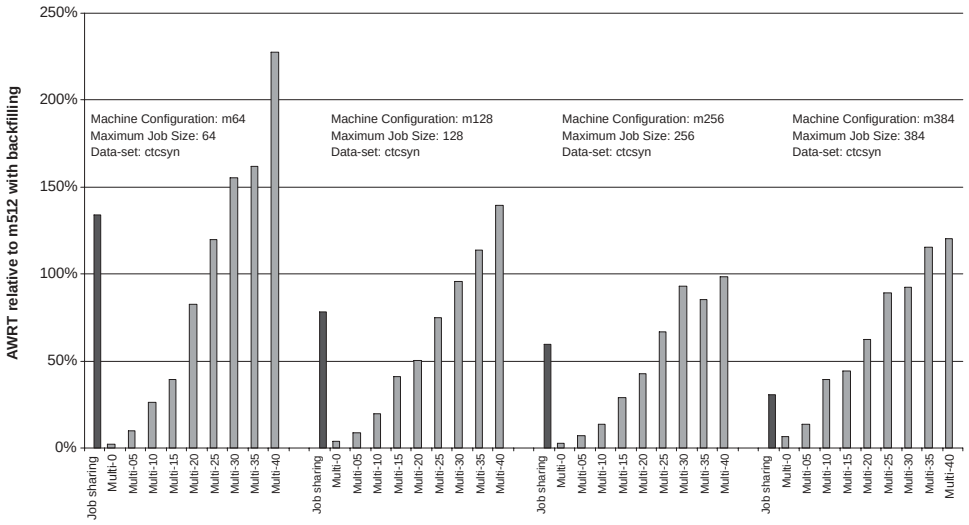

Figure 5. Results for different resource configurations compared to configuration *m512* with backfilling(0%)

The mentioned improvement of job-sharing can be seen in the results for all configurations and workload sets. Figure 4 shows the average weighted response time for the *m128* configuration with an improvement of over 50%. Similar results can be found in the other simulations. The average weighted response time of locally executed jobs certainly depends on the local workload modelling. The results shown in this paper for single-site execution and job-sharing are generated using an EASY backfill scheduler. In contrast to job-sharing single-site execution is restricted to keep the workload locally. That means that no job is transferred to a remote site. As mentioned before large jobs that are wider than the local machine have been split up into smaller jobs which are sequentially executed on the local system. This leads to a significant increase of the AWRT due to the applied weight on each job-part. Job-sharing on the other hand allows the transfer of jobs to remote machines.

Further improvements of the AWRT can be achieved by using multi-site execution. As a reference the result for a single machine with 512 nodes is also given in Figure 4. The result of this *m512* configuration gives the lower bound for the backfilling algorithm. In this configuration no machine partitioning has to be taken into account contrary to any other configuration. Figure 4 shows simulation results for multi-site execution with different run-time overhead for split jobs (0%..40%). As expected, the average weighted response time without overhead for multi-site is near the *m512* result. In this case splitting a job for multi-site execution causes no penalty.

Moreover, multi-site execution is beneficial compared to job-sharing even for an overhead on execution time of about 25%.

Figure 5 shows the improvement for other configurations with jobs limited to the maximum machine size. Note, that the configurations with equal sized machines show better results than for the *m384* or *m64* configurations. Here, the overhead can be even larger on the equal-sized machines for multi-site to still be beneficial.

Figure 6 shows the average weighted response time for some sample workloads. Note, that the workload as shown in Figure 4 produces the least effective improvements. The average weighted response time in other configurations delivered better results. Here, the overhead on multi-site executed jobs can even be larger.

The example given in Figure 4 represents the workload where all original wide jobs with node requirements larger than 128 were split up into jobs requesting 128 or less nodes. As mentioned before simulations were computed for multi-site execution with the original job size. Figure 7 shows that submitting these wide jobs does not increase the average weighted response time significantly. The improvement over job-sharing is valid even as these wider jobs are actually more complex to schedule as the job start time has to be synchronized between all job parts. Note, that this simulation cannot be computed for the job-sharing scenario as these wide jobs can only be executed in a multi-site scenario.

5 Conclusion

The results show that the collaboration between sites by exchanging jobs even without multi-site execution significantly improves the average weighted response time. This is

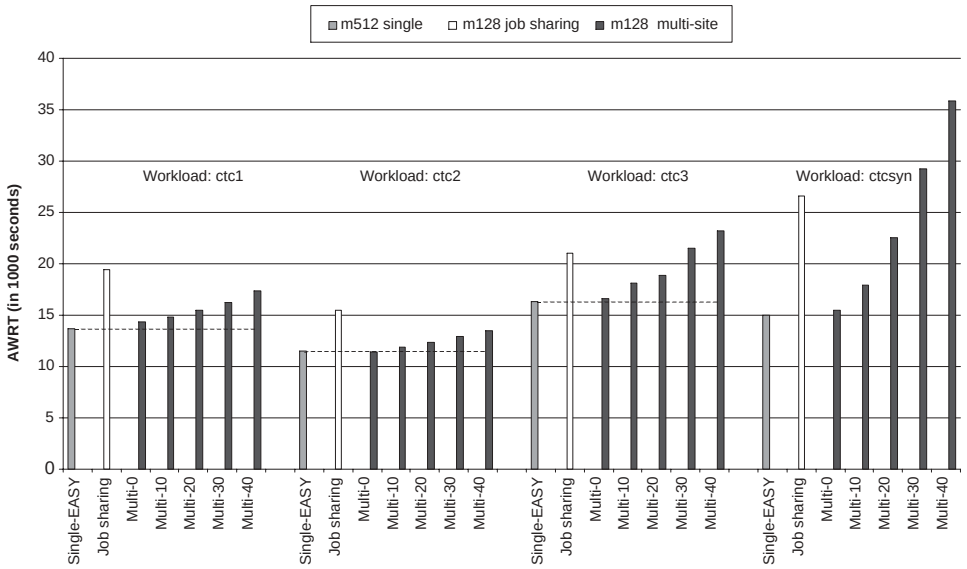

Figure 6. Results for different workloads

already achieved with a simple algorithm of a central scheduler as used in this work.

Furthermore, the usage of multi-site applications leads to even better results under the assumption of a limited increase on job execution time due to communication overhead. Even an increase of their execution time by 25% multi-site proved to be benefical compared to job-sharing. WAN networks are in terms of latency in the order of 2-3 magnitudes slower than common fast interconnection networks between nodes inside a parallel computer, e.g. an IBM SP Switch. Therefore it cannot be concluded that multi-site is suitable for all applications. Whereas multi-site is beneficial for applications with a limited demand in communication.

As grid environments and networks are becoming more common, it seems reasonable for resource owners to participate in such initiatives. Simple strategies like job sharing significantly improve the average weighted response time and therefore the quality of service to the users. Also the research and effort in developing multi-site programs for suitable applications with limited demand in network communication can provide even better results. Furthermore, multi-site applications can effectively use more resources for a single job than available at any single machine. The drawback due to submitting a wider instead of several smaller jobs was limited in our simulations. Of course this may vary with the amount of wide jobs in a workload.

Note, that the applied algorithms for scheduling in this work are simple extensions of backfilling and node selection strategies. Additional research on more sophisticated scheduling algorithms is necessary which may produce even better results. It has to be kept in mind, that the quality of a schedule depends on the actual configuration and workload. The improvements presented in this paper were achieved using example configurations and workloads derived from a real trace. The outcome may vary in other settings. Nevertheless, the results show that job-sharing and multi-site execution in a grid environment are capable of significantly improving the scheduling quality for the users.

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] European GRID Forum, http://www.egrid.org, October 2001.

[3] 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.

[4] 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.

[5] D.G. Feitelson, L. Rudolph, U. Schwiegelshohn, and K. C. Sevcik. Theory and Practice in Parallel Job Scheduling. pages 1–34. Springer–Verlag, Lecture Notes in Computer Science LNCS 1291, 1997.

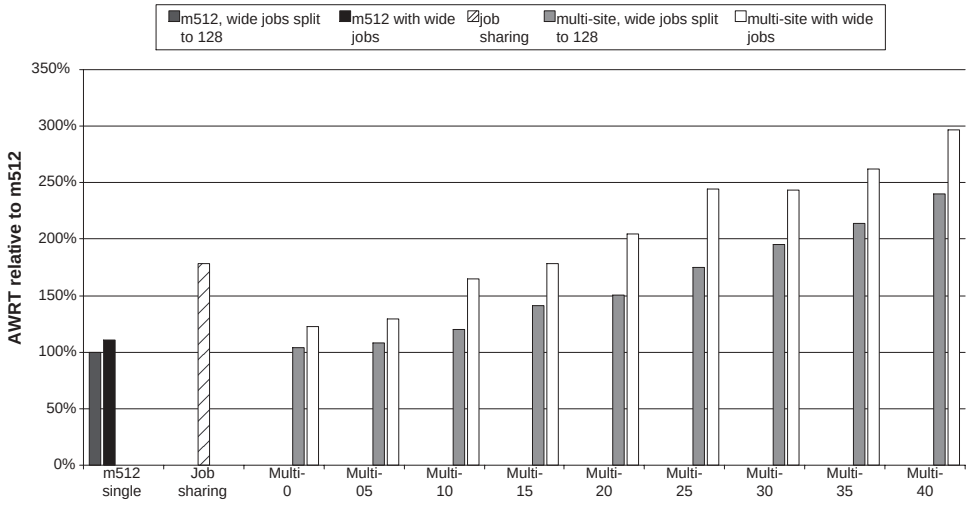

Figure 7. Comparing workloads with original jobs split to 128 parts against keeping wide jobs

[6] 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.

[7] I. Foster and C. Kesselman, editors. *The GRID: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1998.

[8] The grid forum, http://www.gridforum.org, October 2001.

[9] V. Hamscher, U. Schwiegelshohn, A. Streit, and R. Yahyapour. Evaluation of Job-Scheduling Strategies for Grid Computing. pages 191–202. Springer–Verlag, Lecture Notes in Computer Science LNCS 1971, 2000.

[10] 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.

[11] J. Krallmann, U. Schwiegelshohn, and R. Yahyapour. On the Design and Evaluation of Job Scheduling Algorithms. In *Fifth Annual Workshop on Job Scheduling Strategies for Parallel Processing, IPPS'99; San Juan, Puerto Rico; April 1999*, pages 17–42. Springer–Verlag, Lecture Notes in Computer Science LNCS 1659, 1999.

[12] 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–Verlag, Lecture Notes in Computer Science LNCS 949, 1995.

[13] M. Livny and R. Raman. High-throughput resource management. In I. Foster and C. Kesselman, editors, *The Grid - Blueprint for a New Computing Infrastructure*, pages 311–337. Morgan Kaufmann, 1999.

[14] 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.

[15] U. Schwiegelshohn and R. Yahyapour. Fairness in Parallel Job Scheduling. *Journal of Scheduling, 3(5):297-320. John Wiley*, 2000.

[16] J. Skovira, W. Chan, H. Zhou, and D. Lifka. The EASY — LoadLeveler API Project. volume 1162, pages 41–47. Springer–Verlag, Lecture Notes in Computer Science LNCS 1162, 1996.

[17] L. Smarr and C. E. Catlett. Metacomputing. *Communications of the ACM*, 35(6):44–52, June 1992.

[18] Parallel Workloads Archive. http://www.cs.huji.ac.il/labs/parallel/workload/, October 2001.