Evaluation of Job-Scheduling Strategies for Grid Computing

Volker Hamscher[1], Uwe Schwiegelshohn[1], Achim Streit[2], and Ramin Yahyapour[1]

[1] Computer Engineering Institute, University of Dortmund, 44221 Dortmund, Germany
[2] Paderborn Center for Parallel Computing, University of Paderborn, 33095 Paderborn, Germany

Abstract. In this paper, we discuss typical scheduling structures that occur in computational grids. Scheduling algorithms and selection strategies applicable to these structures are introduced and classified. Simulations were used to evaluate these aspects considering combinations of different Job and Machine Models. Some of the results are presented in this paper and are discussed in qualitative and quantitative way. For hierarchical scheduling, a common scheduling structure, the simulation results confirmed the benefit of Backfill. Unexpected results were achieved as FCFS proves to perform better than Backfill when using a central job-pool.

1 Introduction

In recent years an increasing number of parallel computers have become part of so called computational grids or metacomputers [1], [2]. Such a grid typically contains many computers offering a variety of resources. The scheduling system is responsible to select best suitable machines in this grid for user jobs. In large grids it is very cumbersome for an individual user to select these resources manually. The management and scheduling system generates job schedules for each machine in the grid by taking static restrictions and dynamic parameters of jobs and machines into consideration.

The job scheduling for a single parallel computer significantly differs from scheduling in a metacomputer. The scheduler of a parallel machine usually arranges the submitted jobs in order to achieve a high utilization. The task of scheduling for a metacomputer is more complex as many machines are involved with mostly local scheduling policies. The metacomputing scheduler must therefore form a new level of scheduling which is implemented on top of the job schedulers. Also, it is likely that a large metacomputer may be subject to more frequent changes as individual resources may join or exit the grid at any time. Note that many users take a special advantage of a computational grid in the potential combination of many resources to solve a single very large problem. This requires the solution of various hardware and software challenges in several areas including scheduling.

In this paper we discuss several architectures and scheduling policies for such a system. To this end, we are presenting a brief overview on this topic in Section 3. Next, we show a few simple scheduling algorithms for these architectures in Section 4. These algorithms are subject of the performance evaluation in Section 5 where preliminary simulation results are presented.

2 Background

The term metacomputing was established in 1987 by Smarr and Catlett [11]. The concept of connecting computing resources has been subject to many research projects. Some to be mentioned are Globus [5], Condor [10] and Legion [6].

In the area of metacomputing the topic of scheduling is an important part for building efficient infrastructures. As already mentioned, the requirements of scheduling in a metacomputing environment significantly deviate from those for scheduling of jobs on a single parallel machine. One important difference is the inclusion of network resources. Additionally, metasystems are geographical distributed and often belong to several institutions and owners. A scheduler on a single parallel machine must not cope with system boundaries and can manage the given resources independently of external restrictions.

A scheduling infrastructure in a metacomputing system must take those additional requirements into account. Therefore special mechanisms for security and fault-tolerance are needed. Also the independence of resources, especially the different ownership, requires support for the fine-tuning of scheduling policies defined by their providers.

In the next section, this paper gives an overview of common structures in metacomputing environments. The presented topologies are classified into centralized and decentralized schedulers. The structure of the scheduling infrastructure, the used algorithms and strategies are very important for the quality and performance of the system. Many of those scheduling algorithms, starting from simple FCFS strategies to improvements like backfilling [4] known from scheduling on a single parallel machine, can be adapted to the metasystem level. In this paper, we concentrate on the discussion of scheduling structures in metasystems in combination with some common scheduling algorithms. In the following, example architectures for scheduling infrastructures are presented. Note, that this list should not be considered complete, but gives an overview on common structures in computational grids and metacomputing networks. Further we do not elaborate on the architecture of the parallel computing systems itself, but only on the logical structure of the scheduling process. First, we distinguish centralized and decentralized scheduling architectures.

2.1 Centralized Scheduling

In a centralized environment all parallel machines are scheduled by a central instance. Information on the state of all available systems must be collected here. This concept obviously does not scale well with increasing size of the computational grid. The central scheduler may prove to be a bottleneck in some situations (e.g. if a network error cuts off the scheduler from its resources, system availability and performance may be affected). As an advantage, the scheduler is conceptually able to produce very efficient schedules, because the central instance has all necessary information on the available resources.

This scheduling paradigm is useful e.g. at a computing center, where all resources are used under the same objective. Due to this fact the lack of communication bandwidth at the central scheduling instance can be neglected.

In this scenario jobs are submitted to the central scheduler (see Figure 1). Those jobs, that cannot be started on a machine immediately after submission, are stored in a central job-queue for a later start.

We can further distinguish schedulers by the way how resources are combined for a job. This applies to centralized schedulers as well as to their decentralized alternatives that are discussed later.

Single-site scheduling A job is executed on a single parallel machine. This means that system boundaries are not crossed. Well known scheduling algorithms for load balancing (e.g. FCFS, Backfill) can be used. The latency for the in-job-communication is often not subject to scheduling considerations due to the fact that communication inside a machine is usually very fast in comparison to distributed execution.

Multi-site scheduling The described restriction of single-site algorithms is lifted. Now a job can be executed on more than one machine in parallel. As job-parts are running on different machines, the latency for the communication between those parts must be considered. Further, the scheduling system must guarantee that the different job-parts are started synchronously on all machines.

2.2 Hierarchical structure

A possible configuration for a computational grid is the usage of a central scheduler to which jobs are submitted, while in addition every machine uses a separate scheduler for the local scheduling, as shown in Figure 2. Although this structure shows properties of centralized and decentralized scheduling, we would consider it to be a centralized system as there is a single instance to which jobs are submitted.

The main advantage is the fact that different policies can be used for local and global job scheduling. The central scheduler is some kind of a meta-scheduler, that redirects all submitted jobs to the local scheduling queues on the resources based on a policy.

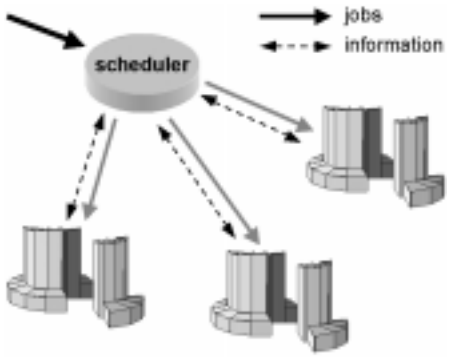

Fig. 1. Centralized Scheduling

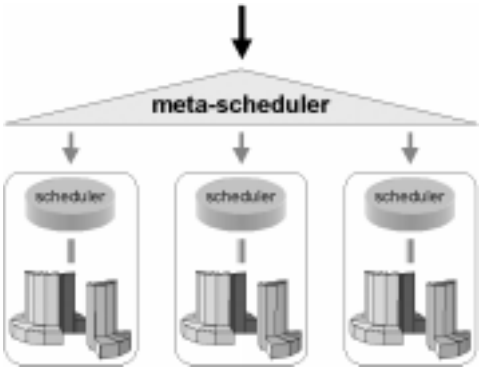

Fig. 2. Hierarchical Structure

2.3 Decentralized Scheduling

In decentralized systems, distributed schedulers interact with each other and commit jobs to remote systems. No central instance is responsible for the job scheduling. Therefore, information about the state of all systems is not collected at a single point. Thus, the communication bottleneck of centralized scheduling is prevented which makes the system more scalable. Also, the failure of a single component will not affect the whole metasystem. This provides better fault-tolerance and reliability than available for centralized systems without fall-back or high-availability solutions.

The lack of a global scheduler, which knows all job and system information at every time instant, usually leads to sub-optimal schedules. Nevertheless, different scheduling policies on the local sites are possible. Further, site-autonomy for scheduling can be achieved easily as the local schedulers can be specialized on the needs of the resource provider or the resource itself.

Unfortunately the support for multi-site applications is rather difficult to achieve. As all parts of a parallel program must be active at the same time, the different schedulers must synchronize the jobs and guarantee simultaneous execution which makes it more difficult to provide optimal schedules.

In the following, we present two explicit cases of decentralized architectures that were used for the evaluation shown in Section 5. Note that all jobs are submitted locally.

Direct communication The local schedulers can send/receive jobs to/from other schedulers directly (see Figure 3). Either schedulers have a list of remote schedulers they can contact or there is a directory that provides information of other systems.

If a job start is not possible on the local machine immediately, the local scheduler is searching for an alternative machine. If a system has been found, where an immediate start is possible, the job and all its data is transferred to the other machine/scheduler. In our evaluation, the execution length of the job is modified to reflect this overhead.

It can be parameterized which jobs are forwarded to another machine. Note, that this affects the local queue. This can also affect the performance of some

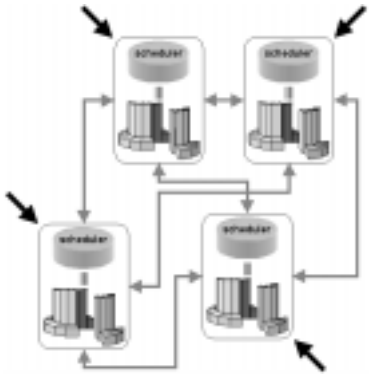

Fig. 3. Decentralized Scheduling with Direct Communication

scheduling algorithms. E.g. the backfilling algorithms (s. Section 3.2) relies on a suitable backlog.

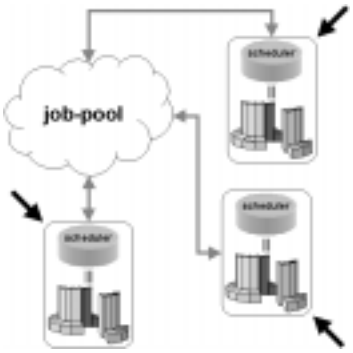

Fig. 4. Using a Job Pool in Decentralized Scheduling

Communication via a Central Job Pool Jobs that cannot be executed immediately are sent to a central job pool instead of a remote machine (see Figure 4). In contrast to direct communication the local schedulers can pick suitable jobs for their schedules. In this scenario, jobs can be *pushed* into or *pulled* out of the pool. A policy is required that all jobs from the pool are executed at some time to prevent job starvation.

This method can be modified, so that all jobs are pushed directly in the job-pool after submission. This way all small jobs requiring few resources can be used for utilizing free resources on all machines.

3 Scheduling Algorithms

The allocation process of a scheduler consists of two parts, the selection of the machine and the scheduling over time.

3.1 Selection-Strategies

We define four strategies for selecting suitable machines for a job request. In the following, M_{max} denotes all machines that are able to execute a specific job in the metacomputer. M_{free} is the subset of machines that have currently enough free resources to start the job immediately.

- *BiggestFree* takes the machine from M_{free} with the largest number of free resources. A disadvantage of this strategy is a possible delay of a wide job, as small jobs may take the critical resources necessary for the next wide job.
- *Random* chooses a machine from the sets M_{max} or M_{free} by random. On average it provides a fair distribution of the jobs on the available machines.
- *BestFit* takes the machine either from M_{max} or M_{free} that leaves the least free resources if the job is started. In comparison to *BiggestFree* this strategy does not unnecessarily fill up larger machines with smaller jobs.
- *EqualUtil* chooses the machine with the lowest utilization to balance the load on all machines [13]. Note, that this strategy does not try to keep larger machines free for larger jobs which may be a drawback.

3.2 Scheduling Algorithms

Most common algorithms in scheduling are based on list-scheduling. In the following three variants are presented that we used for our evaluation [8].

- *First-Come-First-Serve:* The scheduler starts the jobs in the order of their submission. If not enough resources are currently available, the scheduler waits until the job can be started. The other jobs in the submission queue are stalled. This strategy is known to be inefficient for many workloads as wide jobs waiting for execution can result in unnecessary idle time of some resources.
- *Random:* The next job to be scheduled is randomly selected among all jobs that are submitted but not yet started, therefore the schedule is non-deterministic. No job is preferred, but jobs submitted earlier have a higher probability to be started before a given time instant.
- *Backfill:* This is an out-of-order version of FCFS scheduling that tries to prevent the unnecessary idle time caused by wide jobs. Two common variants are EASY- and conservative-backfilling [4, 9]. In case that a wide job is waiting for execution other jobs can be started under the premise that the wide job is not delayed. Note, that the performance of this algorithm relies on a sufficient backlog.

4 Evaluation

4.1 Description of the Simulation Environment

For performance evaluation of the different structures and algorithms we used a simulation environment based on discrete event simulation. It allows the evaluation

of different configurations by providing results for common evaluation criteria, like schedule-length (makespan), average response-time and utilization of the machines.

Information on workload traces from the 430 node IBM RS6000/SP of the Cornell Theory Center [7] and a workload trace of the Intel PARAGON from the FZ Jülich were used to generate different *Job Sets*. Thereby the jobs contain all relevant information necessary for the scheduling.

The traces were modified to produce larger backlogs, which was done by a duplication of the jobs.

A job consists of a submission time and a requested number of resources. Also for some algorithms (backfilling) the actual or estimated execution length of a job is used.

We use a simple abstract *Machine Model* of homogeneous resources (nodes). The communication inside a machine does not prefer any specific communication patterns. Therefore, jobs can be distributed on a machine in any fashion. Every machine is capable of starting every job as long as enough resources are available. The nodes are used in an exclusive manner. After the start of a job, the subset of nodes cannot be changed and therefore no support for migration is provided here.

Some machine models used in the simulations are presented in Table 1 with information on the size of each machine and the total number of resources. The first Machine Model `nrw` is based on the machines available in the NRW-Metacomputing project [3].

Name	Sizes of Machines	Total Resources	Number of Machine
nrw	10, 12, 16, 32,48, 192, 512, 512	1334	8
equal	256, 256, 256, 256, 256, 256, 256, 256	2048	8
4small_4big	32, 32, 32, 32, 256, 256, 256, 256	1152	8
2powN	2, 4, 8, 16, 32, 64, 128, 256	510	8

Table 1. Resource Configurations

An overview on the evaluated combinations of algorithms and selection strategies is given in Table 2. Each scheduler is simulated with different job and machine models.

4.2 Results

The different combinations of configurations, algorithms and structures produced a large amount of data. In the following, we can only discuss some of the results, while the complete listing is found in [12].

Single-Site Scheduling First, we compare FCFS and Backfilling in the *Single-Site* scenario, see Table 3.

As expected Backfill is much more efficient than FCFS in single-site scheduling and also better than Random. Especially with the BiggestFree strategy, Backfilling is vastly superior to FCFS. As already mentioned before, large backlogs cause

Structure		Scheduler	Selection Strategy
Central	Single-Site	FCFS, Random, Backfill	BestFit_Free, BiggestFree, Random_Free
	Multi-Site	-	-
	Hierarchical	FCFS, Backfill	BestFit_Max, EqualUtil_Max, Random_Max
Decentral	Direct Communication	FCFS, Backfill	-
	Job Pool	FCFS, Backfill	-

Table 2. Simulated combinations

Scheduling Algorithm	Selection Strategy	Makespan	Average Response Time	Utilization
Backfill	BestFit_Free	14.878.363 s	12.445 s	66,28 %
	BiggestFree	14.878.363 s	13.060 s	66,28 %
	Random_Free	14.878.363 s	12.769 s	66,28 %
FCFS	BestFit_Free	16.361.362 s	881.155 s	60,27 %
	BiggestFree	18.086.122 s	1.806.165 s	54,52 %
	Random_Free	17.033.913 s	1.312.609 s	57,89 %
Random	BestFit_Free	14.879.165 s	13.951 s	66,27 %
	BiggestFree	15.639.085 s	40.166 s	63,05 %
	Random_Free	15.240.330 s	31.356 s	64,70 %

Table 3. Exemplary results of single-site scheduling. Machine model `2powN` and job model are based on trace data from CTC.

the Backfill scheduler to be more efficient. Note that, the simple random strategy performs only slightly worse than Backfill.

In comparison to the other selection strategies (see Section 3.1) BiggestFree performs worst, because resources are allocated without regard of wide jobs potentially submitted in near future. BestFit_Free unveils the best results in all cases as it leaves less resources idle. Random_Free which effectively represents a mixture of both variants achieves average results.

Multi-Site Scheduling Four computation methods are used exemplary to modify the execution length of jobs running on several machines.

1. $(1+p)^f$
2. $\max_i[(1+p)^{r_i}]$
3. $(1+p)*f$
4. $\max_i[(1+p)*r_i]$

- f specifies the number of job segments
- r_i specifies the number of requested resources by each job part i
- p denotes a unit value for the partitioning overhead

The segmentation of a job to run in parallel on several machines leads to an overhead. The size increases are described by the parameter p. Generally a larger p results in a longer average response time (ART). While the first three procedures show a monotonous behavior in this context, the last one seems not to share this trend.

An additional parameter (minJobSize) prevents a job smaller than minJobSize from being partitioned in segments. Therefore we expect a larger makespan and a larger ART for a bigger minJobSize, but obtained different results. Further research must determine whether there is a turning point at which the originally expected correlation begins. As to be expected, multi-site scheduling does not perform as well as single-site scheduling.

Hierarchical Scheduling The Tables 4 and 5 present the results for different selection strategies and *Machine Models*.

Machine Size	256	256	256	256	256	256	256	256
Random_Max	6,85 %	6,47 %	7,27 %	8,16 %	7,53 %	7,84 %	7,38 %	6,94 %
EqualUtil_Max	7,28 %	7,35 %	7,28 %	7,35 %	7,49 %	7,30 %	7,30 %	7,30 %

Table 4. Exemplary utilization results for each machine with different selection strategies in hierarchical scheduling. Machine model `equal` respectively `2powN` and job model are based on trace data from CTC.

Machine Size	2	4	8	16	32	64	128	256
Random_Max	3,20 %	3,71 %	35,85 %	21,31 %	21,40 %	16,99 %	17,76 %	40,08 %
EqualUtil_Max	14,73 %	14,86 %	24,67 %	24,85 %	24,90 %	24,78 %	24,80 %	28,53 %

Table 5. Exemplary utilization results for each machine with different selection strategies in hierarchical scheduling. Machine model `equal` respectively `2powN` and job model are based on trace data from CTC.

In both cases EqualUtil_Max performs slightly better than the Random selection strategy, nevertheless both produce fairly good results. The use of EqualUtil_Max proves to be advantageous if machine sizes vary. If all machines in the metasystem are equipped with the same number of resources (e.g. 8 machines with 256 resources each) the differences between both strategies are negligible.

Especially in combination with the EqualUtil strategy and heterogeneous structures, the Backfill scheduler is much more effective than FCFS as shown in Table 6.

Direct Communication The increase of the execution length (see Sec. 2.3) leads to an expected decrease in performance (see Table 7). But it is negligible as only few jobs are affected. Overall the results for distributed structures are highly dependent

Scheduling Algorithm	Makespan	Average Response Time	Utilization
FCFS	120.662.772 s	32.836.642 s	8,17 %
Backfill	16.189.537 s	94.377 s	60,91 %

Table 6. Exemplary results for selection strategy EqualUtil_Free in hierarchical scheduling. Machine model `2powN` and job model are based on trace data from CTC.

on the resource configuration. In a configuration where all machines are similar in size as in set *equal*, there is no significant difference between centralized and decentralized scheduling. In an environment of machines with varying size, decentral scheduling produces much worse results.

CTC

extension time of transferred jobs	Makespan	Average Response Time	Utilization
20 s	28.005.954 s	2.615.614 s	34,27 %
50 s	28.775.899 s	4.167.194 s	35,21 %

KFA

parameter p (relative length modification)	Makespan	Average Response Time	Utilization
0.1	23.485.345 s	26.135 s	29,31 %
0.2	23.485.635 s	26.154 s	29,31 %

Table 7. Exemplary results for absolute modification of the execution length using Backfill-Schedulers with Direct Communication. Machine model `2powN` and job model are based on trace data from CTC.

Using a Job Pool The scheduling depends mostly on established (fairness) policies. For instance, a job is only forwarded to the job pool, if it cannot be handled locally. Therefore, the balancing between the local and the remote queue is of major importance to prevent jobs to accumulate in one of them.

Scheduling Algorithm	Makespan	Average Response Time	Utilization
FCFS	23.465.816 s	1.036 s	9,55 %
Backfill	23.466.498 s	1.075 s	9,54 %

Table 8. Comparing FCFS and Backfill with equal settings of parameters. Machine model `nrw` and job model are based on trace data from KFA.

Backfill schedulers prefer the local queue for their backfilling, whereas FCFS based schedulers always use the job-pool to utilize their idle times. Therefore, FCFS has a wider variety of jobs to choose from, if enough backlog exists. Under this

circumstances FCFS shows a slightly better performance than Backfill. First simulations verify this effect as presented in Table 8.

The increase of the execution length for the overhead as mentioned in Section 2.3 shows a decrease in performance as to be expected. If enough small jobs are forwarded to the central pool, the results are comparable to central scheduling.

Besides the scheduling aspect the use of a common job-pool requires certain management features. Jobs exceeding the maximum size of any resource set of the system must be rejected.

5 Conclusion

In this paper, we discussed some scheduling structures that typically occur in metasystems or computational grids. As evaluating such structures highly depends on the used algorithms and strategies of the scheduling itself, a selection of them has been presented. Besides the discussion of scheduling structures, simulations were used to evaluate their run-time performance. Discrete-event simulation has been used with workload from real machine traces and sample machine configurations. The results are not meant to be complete, but give an overview on the methodology and some interesting relations. Future work will extend the studies to more architectures and include more detailed parameter and configuration variation. This is important as the current results show that the performance of the examined algorithms for the scheduling structure are highly dependent on the parameters, machine configurations and workload.

References

1. European grid forum, http://www.egrid.org.
2. The grid forum, http://www.gridforum.org.
3. C. Bitten, J. Gehring, R. Yahyapour, and U. Schwiegelshohn. The NRW-Metacomputer: Building blocks for a worldwide computational grid. In *Heterogeneous Computing Workshop 2000 at IPDPS 2000*, Cancun, Mexico, May 2000.
4. D.G. Feitelson and A.M. Weil. Utilization and Predictability in Scheduling the IBM SP2 with Backfilling. In *Procedings of IPPS/SPDP 1998*, pages 542–546. IEEE Computer Society, 1998.
5. I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. 11(2):115–128, 1997.
6. A. Grimshaw, A. Wulf, J. French, A Weaver, and P. Reynolds. Legion: The next logical step toward a nationwide virtual supercomputer. Technical Report CS-94-21, University of Virginia, Computer Sciences Department, 1994.
7. 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.
8. 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*, Lectures Notes in Computer Science, pages 17–42, 1999.

9. 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.
10. M. Litzkow, M. Livny, and M. Mutka. Condor - a hunter of idle workstations. In *Proceedings of the 8^{th} Intl Conf. on Distributed Computing Systems*, pages 104–111, 1988.
11. L. Smarr and C. E. Catlett. Metacomputing. *Communications of the ACM*, 35(6):44–52, June 1992.
12. A. Streit. Evaluation of Scheduling-Algorithms for Metacomputing (in German). In *Diploma Thesis at CEI*. University of Dortmund, Germany, 1999.
13. G. D. van Albada, J. Clinckemaillie, A. H. L. Emmen, J. Gehring, O. Heinz, F. van der Linden, B. J. Overeinder, A. Reinefeld, and P. M. A. Sloot. Dynamite - blasting obstacles to parallel cluster computing. In P. M. A. Sloot, M. Bubak, A. G. Hoekstra, and L. O. Hertzberger, editors, *High-Performance Computing and Networking (HPCN Europe '99), Amsterdam, The Netherlands*, number 1593 in Lecture Notes in Computer Science, pages 300–310, Berlin, April 1999. Springer-Verlag.