Evaluation of Scheduling Structures

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 metasystems or computational grids. Scheduling algorithms and strategies applicable to these structures are introduced and classified. To compare those different approaches simulations haven been executed for different workload and machine configurations. We point out some of the results for the performance of the schedulers in the presented metasystem structures.

1 Introduction

In recent years an increasing number of parallel computers have become part of so called computational grids ([1], [2]) or metacomputers. Such a grid typically contains many computers offering a variety of resources like compute power, storage and installed software. It is the goal of the user to select the machine in this grid on which his job runs best. In large grids this selection process would be very cumbersome for individual users. Therefore, a management and scheduling system is required that generates job schedules for each machine in the grid by taking into consideration user and owner objectives, static restrictions and dynamic parameters. If the grid resources are not free of charge the cost of a resource will also affect the selection process. In this case accounting must be included into this system as well.

Unfortunately, job scheduling for a single parallel computer significantly differs from scheduling in a metacomputer. For instance, the job scheduler of a parallel computer is a single central component that coordinates all jobs submitted to the computer. As there is a single owner for all nodes of the computer there will be only one objective for the resources which is enforced by the job scheduler. In economic terms we can speak of a quasi-monopoly as most users have no other choice but to use this specific parallel computer for their jobs. Therefore, the scheduler of a parallel computer usually arranges the submitted jobs in order to achieve a high utilization. On the other hand a metacomputer can be compared to a market economy with many offers and requests. In this case, schedulers may have to act as brokers.

Also, it is likely that a large metacomputer may be subject to frequent changes as individual resources may join or exit the grid at any time. Therefore, the scheduler will not necessarily be able to provide an up-to-date overview of the system including the schedules for all participating machines. Finally, many users see 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 general it can be assumed that parallel computers will still be equipped with their usual job scheduling system in order to achieve a high degree of flexibility. The metacomputing scheduler must therefore form a new level of scheduling which is implemented on top of the job schedulers. In this paper we discuss architectures for such a scheduling system. To this end, several architectures are presented in Section 3. Next, we show a few simple scheduling algorithms in Section 4 that are used for the performance evaluation. Our first simulation results are discussed in Section 5.

2 Background

Since the term metacomputing was established in 1987 by Smarr and Catlett [13], the concept of connecting computing resources has been subject to many research projects. Some to mention 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.

But, a scheduling infrastructure in a metacomputing system must take those additional requirements into account. For instance, this means special considerations for security and fault-tolerance. Also the independence of resources, especially the different ownership, requires support for the fine-tuning of scheduling policies defined by their providers.

Most current scheduling architectures are based on different infrastructure topologies. Especially the logical structures of the scheduling environment differ to great extend. In the next section, this paper gives an overview of common structures. The presented topologies are classified into centralized and distributed schedulers.

Next, we address the scheduling topologies used by current metasystem implementations. The Globus project is based on a local and decentralized allocation structure [5]. Allocations are locally collected from remote resources by co-allocators. The resources have a component called GRAM (Global Resource Allocation Manager) that interacts with the requests. Note, that this represents an allocation scheme which is not necessarily equivalent with scheduling.

On the other hand, Condor [11] employs a method of advertising resources to a central manager. Jobs are submitted to this central manager, which finds suitable resources by matchmaking of requested and offered resources. Condor is specialized in high throughput computing, with the linked machines usually being workstations. In this case, the aspect of local scheduling is not very difficult.

In Legion the autonomy of sites and users is one of the major aspects of the system [3]. Therefore the scheduling policies are influenced by each of them and handled in a decentralized fashion. In contrast to the earlier mentioned projects, Legion is almost a distributed operating system specialized on high performance computing.

Besides 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 translated to the metasystem level. In this paper, we concentrate on the discussion of scheduling structures in metasystems in combination with some common scheduling algorithms.

3 Scheduling Architecture for Metacomputing

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 computing systems itself, but only on the logical structure of the scheduling process. First, we distinguish centralized and decentralized scheduling architectures.

3.1 Centralized Scheduling

In a centralized environment all machines are scheduled by a central instance. Information on the state of all available systems must be collected here, which may be challenging for large networks. 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. If the local network breaks down generally the hole site is malfunctioning. In this scenario jobs are submitted to the central scheduler. Those jobs, that cannot be started on a machine immediately after submission, are stored in a central job-queue for a later start.

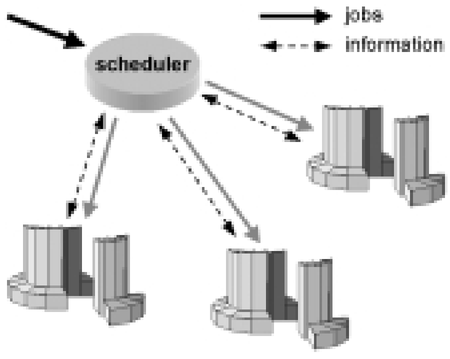

Fig. 1. Centralized Scheduling

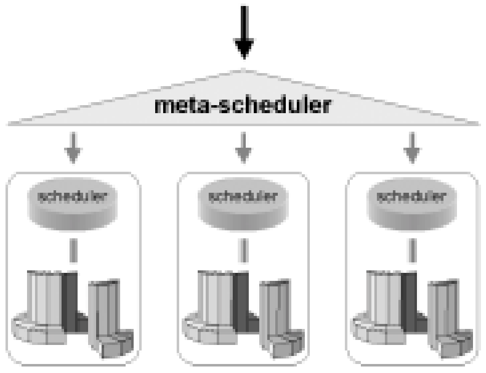

Fig. 2. Hierarchical Structure

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 cannot be distributed over more than one machine. This means that system boundaries cannot be 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 at the same time on all machines.

3.2 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. Although this structure shows properties of centralized and decentralized scheduling, we would consider it more a centralized one as there is a single instance to which a job is submitted.

The main advantage is the fact that different policies can be used for local and global job scheduling. Further, it is possible to keep already installed local schedulers. 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.

Note, that the functionality of the centralized scheduler is sometimes limited to the just-in-time distribution of submitted jobs to the local schedulers. In this case, the central job-queue becomes unnecessary.

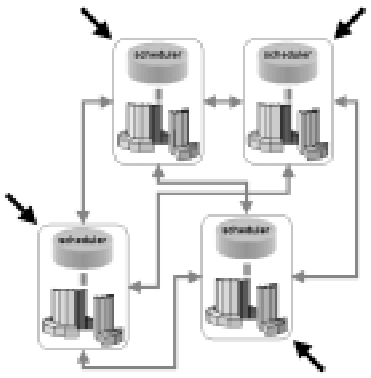

Fig. 3. Decentralized Scheduling with Direct Communication

3.3 Decentralized Scheduling

In the decentralized scenario, 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. This 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.

In contrast, an optimal schedule may not be achievable as current information on all systems is not available. 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. In addition the local schedulers can be specialized on the needs of the resource provider or the resource itself.

On the other hand 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. This makes the ability to provide optimal or good schedules even more complex.

In the following, we present two explicit cases of decentralized architectures that were used for the evaluation shown in Section 5.

Direct communication The local schedulers can send/receive jobs to/from other schedulers directly. Either scheduler has a list of remote schedulers it can contact or there is a directory that provides information of other systems.

We want to highlight the following two alternatives on how to manage those jobs in a decentralized system, which cannot be started immediately.

– The job is stored in a local queue for a future start. Note, that this backlog affects the performance of some scheduling algorithms (s. Section 4.2).
– The local scheduler is searching for an alternative machine/scheduler where an immediate job-start is possible. If such a system has been found the job and

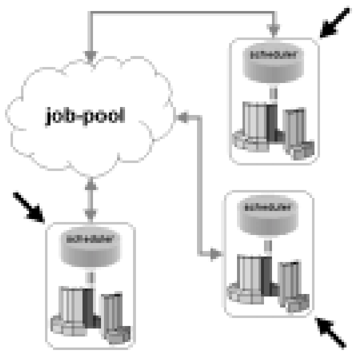

Fig. 4. Using a Job Pool in Decentralized Scheduling

all its data is transferred to the other machine/scheduler. The overhead for this transmission must be considered. In our evaluation, the execution length of the job is modified to reflect this overhead.

Communication via a Central Job Pool Instead of sending a job, that cannot be executed at the moment, to a remote machine, a central job pool is established. In this scenario, systems can *push* jobs into the pool or *pull* jobs out of the pool.

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.

4 Scheduling Algorithms

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

4.1 Selection-Strategies

We define three 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 This takes at random a machine from the lists M_{max} or M_{free}. On average it provides a fair distribution of the jobs on the available machines.

BestFit This 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 The machine with the lowest utilization is chosen to balance the load on all machines [14]. Note, that this strategy does not try to keep larger machines free for larger jobs which may be as a drawback.

4.2 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. The produced schedule is non-deterministic. No job is preferred, but scheduling 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.

5 Evaluation

5.1 Description of the Simulation Environment

For performance evaluation of the different structures and algorithms, we simulated several configurations. To this end, a simulation environment based on discrete event simulation has been used. 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.

Job Model As input data real workload traces from the 430 node IBM RS6000/SP of the Cornell Theory Center are used [7]. Additionally, a workload trace of the Intel PARAGON from the KFA, Jülich.

For the different structures it is necessary to adapt the number of jobs. Therefore, probabilistic data derived from the original traces has been generated. Also, some data traces do not produce enough backlog for the simulated scenarios. This evidently makes scheduling algorithms incomparable. This is avoided by simply doubling workload.

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 the jobs is used.

Machine Model We use a simple abstract machine model where every machine is only defined by its number of available resources (nodes). All nodes have the same processor speed and there are no architectural differences. The communication inside a machine does not prefer any specific communication patterns. Therefore, jobs can be distributed on a machine in any fashion.

In short, every machine is capable of starting every job as long as enough resources are available. The nodes are used in an exclusive manner, that is at every time instant at most one job is active on a node. After the start of a job, the subset of nodes cannot be changed. Therefore, we do not use moldable jobs or support for migration.

Configurations For the evaluation different simple example configurations have been used. First, a set of resources have been modeled according to an existing metacomputing network as found in North-Rhine-Westphalia [12]. Additionally, three generic configurations have been designed, see Table 1.

5.2 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.

Single-Site Scheduling First, we compare FCFS and Backfilling in the single-site scenario in the same configurations. As expected the backfilling strategy 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, large backlogs cause the Backfill scheduler to be more efficient. Interestingly, the simple random strategy performs only slightly worse than Backfill.

Compared to the other selection strategies (see Section 4.1) using the same scheduling algorithm, BiggestFree performs worst. This is probably caused on the above mentioned fact, that resources are allocated without regard of future wide

Name	Sizes of Machines	Total Resources	Machine Number
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

		Job Dataset	Resource Configuration	Scheduler	Selection Strategy
Central	Single-Site	CTC-prob, KFA-prob	all	FCFS, Random, Backfill	BestFit_Free, BiggestFree, Random_Free
	Multi-Site	CTC-prob, KFA-prob	all	-	-
	Hierarchical	CTC-prob, KFA-prob	all	FCFS, Backfill	BestFit_Max, EqualUtil_Max, Random_Max
Decentral	Direct Communication	CTC-prob, KFA-prob	all	FCFS, Backfill	-
	Job Pool	CTC-prob KFA-prob	all	FCFS, Backfill	-

Table 2. Simulations

jobs. 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 multi-site.

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. For all four methods 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 interesting fact is the influence of a given minimum size of a job-segment (minJobSize). This parameter prevents a job from being partitioned in segments smaller than minJobSize. Therefore we expect a larger makespan and a larger ART for bigger minJobSize, but obtained different results. Further research must determine whether there is a turning point at which the originally expected correlation begins. Overall, multi-site scheduling does not perform as well as the single-site one.

Hierarchical Scheduling EqualUtil_Max is slightly better than the Random selection strategy. Both produce fairly good results. EqualUtil_Max performs especially better 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.

Direct Communication In the simulations the increase of the execution length for the overhead, as mentioned in Section 3.3, leads a decrease in performance as to be expected. But the decrease is negligible as only few jobs are affected. Overall the results for distributed structures are highly dependent 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 produce much worse results.

Using a Job Pool 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. In a dynamic environment an information exchange system must allow the adjustment of this size.

The scheduling itself depends to a very high degree 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.

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 has a better performance than Backfill. First simulations verify this effect.

The increase of the execution length for the overhead as mentioned in Section 3.3 shows a decrease in performance as to be expected. Another fact to point out is, that it is beneficial to send jobs into the pool. If enough small jobs are forwarded, the results are comparable to central scheduling.

6 Conclusion

In this paper, we discussed some example scheduling structures that typically occur in metasystems or computational grids. As evaluating such structures highly depend 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. To this end, 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. Steve Chapin, John Karpovich, and Andrew Grimshaw. Resource management in legion. 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.
4. D.G. Feitelson and A.M. Weil. Utilization and Predictability in Scheduling the IBM SP2 wi th 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. Miron Livny and Rajesh 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.
12. Uwe Schwiegelshohn and Ramin Yahyapour. The NRW Metacomputing Initiative. In G. Cooperman, E. jessen, and G. Michler, editors, *Workshop on Wide Area Networks and High Performance Computing*, pages 269–282. Springer–Verlag, Lecture Notes in Control and Information Science LNCIS 249, 1998.
13. Larry Smarr and Charles E. Catlett. Metacomputing. *Communications of the ACM*, 35(6):44–52, June 1992.
14. 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.