Scheduling a Metacomputer With Uncooperative Sub-schedulers [*]

Jörn Gehring Thomas Preiß

Paderborn Center for Parallel Computing
D-33095 Paderborn, Germany
{joern | tppb}@uni-paderborn.de
http://www.uni-paderborn.de/pc2/

Abstract. The main advantage of a metacomputer is not its peak performance but better utilization of its machines. Therefore, efficient scheduling strategies are vitally important to any metacomputing project. A real metacomputer management system will not gain exclusive access to all its resources, because participating centers will not be willing to give up autonomy. As a consequence, the scheduling algorithm has to deal with a set of local sub-schedulers performing individual machine management. Based on the proposal made by Feitelson and Rudolph in 1998 we developed a scheduling model that takes these circumstances into account. It has been implemented as a generic simulation environment, which we make available to the public. Using this tool, we examined the behavior of several well known scheduling algorithms in a metacomputing scenario. The results demonstrate that interaction with the sub-schedulers, communication of parallel applications, and the huge size of the metacomputer are among the most important aspects for scheduling a metacomputer. Based upon these observations we developed a new technique that makes it possible to use scheduling algorithms developed for less realistic machine models for real world metacomputing projects. Simulation runs demonstrate that this technique leads to far better results than the algorithms currently used in metacomputer management systems.

1 Introduction

Ever since 1992 when the term *metacomputing* was coined by Larry Smarr and Charles E. Catlett [42], researchers all over the world have been working with increasing effort on this promising concept. The most obvious advantage of a metacomputer is its tremendous computing power. Theoretically, a nation wide cluster can provide more FLOPS than any single machine has ever been able to achieve. However, after six years of research the first wave of enthusiasm has passed and researchers realized that many problems have to be solved before the idea of Smarr and Catlett can become reality. Especially the *Information Wide Area Year* [12] has not only demonstrated the potential of the metacomputing concept but also its limitations. Particularly on the application side,

[*] Supported by a grant from the NRW Metacomputing project

there exist only few programs that can fully exploit the accumulated power of a large heterogeneous machine pool. For most real world applications efficiency – and sometimes even performance – decreases dramatically, if they are executed on a cluster of WAN distributed machines. Consequently, the enormous efforts necessary for building a metacomputer cannot be justified by its theoretical peak performance. During the recent past, other benefits of metacomputing have become more and more apparent. One of the most important aspects of a nationwide computational grid is its ability to share load more evenly and achieve better utilization of the available hardware.

Nowadays people show a strong tendency to run their jobs on those machines that are locally available. Only if these resources are absolutely insufficient, they are willing to go through the trouble of applying for an account at a remote center with more powerful machines. If access to a remote machine was as easy as to HPC resources at the local site, users would be more flexible in choosing their target hardware. As a consequence, the total amount of job requests would spread more evenly over the available machines. Even more important: computer systems with special features could more easily be kept free for those applications that can take advantage of these. Software with less demanding hardware requirements could be run on general purpose machines without loosing efficiency while code that was tuned for a particular system could be executed faster on the appropriate hardware. As a consequence, the implementation of a metacomputer could effectively increase the amount of available computing power without the need to add extra hardware.

If the primary goal of building a metacomputer is to optimize resource usage, then its scheduling component becomes a critical part. From the scheduling perspective, a metacomputing management system can be divided into two distinct layers. The bottom layer consists of the primary infrastructure that links the distributed machines together and manages their information exchange. On top of this there is the scheduling layer that decides when, how, and where jobs are executed. Obviously, there has to be close cooperation between the scheduler and the underlying infrastructure. During the past, researchers often worked independently on these two layers. As a consequence, there is now a large gap between research results achieved in both areas [19,32]. The best known scheduling algorithms, were studied using analytical models that are not directly compatible with the infrastructure implemented in the various metacomputing projects. It is therefore the purpose of this work to show a possible way of bridging this gap and to qualify the inevitable performance decreases.

The rest of this paper is organized as follows. In Sec. 2 we give an overview of current research in the field of metacomputer environments and relevant scheduling strategies. We describe our machine model in Sec. 3 and the workload model as well as the evaluation criteria in Sec. 4. Sec. 5 outlines the examined scheduling algorithms and describes the most important simulation results. Furthermore, a new technique is shown for that makes it possible to use scheduling algorithms developed for less realistic machine models for real world metacomputing projects. Finally, in Sec. 7 we summarize the results, draw conclusions for the scheduling community, and provide an outlook on our future work.

2 Background

Many fields of research are contributing to the implementation of a metacomputer. For example, networking, security, interface design, load balancing, mapping, distributed system design, or even multimedia are all important aspects of this field of research. In this paper we concentrate on what has been achieved in infrastructure oriented projects and on scheduling algorithms that are applicable to such an environment.

2.1 Infrastructure

Although there exists a definition of "metacomputing" in [42] that is frequently referenced, there is no broad agreement about how a metacomputer should look like. A lot of projects have been set up since 1992 which rightfully claim to be working in the field of metacomputing. However, some of their approaches and goals differ significantly. We found that the majority of these projects can be divided into five distinct categories:

HPC Metacomputing is the most traditional approach. Projects of this category are targeting at connecting high performance computers via wide area networks. Supercomputers are typically installed at different locations all over a nation. As a consequence, these projects do not only have to deal with technical problems but also with the political aspects arising from the necessary sub-ordination of supercomputer centers under any form of metacomputer management. The machines forming the resource pool are heterogeneous in hardware and software (e.g. operating system). During the past, most representatives of this category of metacomputing systems had a centralized architecture which incorporated a global scheduling module with full control over the available resources. Due to political and technical considerations (e.g. fault tolerance) nowadays most projects favor a more distributed approach. [20,9,36,44,31] are among the most prominent initiatives of this category.

Cluster Computing has developed very fast during the last six years. The goal of these projects is to connect a large number of workstation sized computers as a cheap alternative to supercomputers. Although some are also trying to connect resources of different administrative entities (e.g. universities), a more typical use is a building wide metacomputer. Most of these systems have support of heterogeneous hardware but put restrictions on software heterogeneity. Since there is much economical potential in this area, there are several commercial products available offering a high degree of robustness. However, compared to HPC-metacomputing these systems focus more on distributing sequential jobs and less on efficiently scheduling parallel applications. Some of the best known cluster computing environments are [25,33,23,5].

Single Site Metacomputing is similar to HPC metacomputing. The main difference is the restriction to only one administrative entity. Many supercomputing centers have been enhancing their local management systems to provide metacomputing services on top of the local machine pool. From the algorithmic point of view, this problem is almost identical to HPC metacomputing. However, results can be achieved faster, because there are less political problems and a centralized software architecture is feasible. [21,1,27] are some examples of these projects.

Object Oriented Metacomputing is increasingly gaining importance in the metacomputing community. Applications that have been implemented using an object oriented programming paradigm can easily be ported to the distributed runtime environment of a metacomputer. Especially the availability of the CORBA standard [41] has had strong effects on the development of distributed object oriented environments. CORBA provides full support of heterogeneity in hard- and software as well as a possible technical solution to the interconnection of several sites. Since each site within such a system represents itself by one or more *object request brokers*, CORBA also indicates a possible way to overcome some of the political obstacles. This is not the least reason why similar approaches have been adopted by several HPC metacomputing projects. The major disadvantage of object oriented metacomputing is related to the fact that currently only few relevant applications have been programmed in object oriented languages.

Seamless Computing is the most pragmatic approach and probably not truly metacomputing. Projects of this class are working on standardizations of supercomputer access interfaces. Nowadays, almost any supercomputing center has its own access modalities and users have to learn and remember how jobs are submitted to each of the available machines. Therefore, there is a strong demand for a unified access interface. Some of the projects working in this area are described in [3,28,6].

Our institute, the Paderborn Center of Parallel Computing, is involved in a couple of projects that deal with interconnecting supercomputers [30,22,35]. Thus, our work concentrates mainly on aspects of HPC metacomputing. The model described in Sec. 3 is therefore derived from this category.

2.2 Job Scheduling for Metacomputing

Much research has been done in the field of parallel job scheduling. A good overview about the current state can be found in [19]. Unfortunately, many researchers have been using their own nomenclature for specifying the capabilities of their algorithms. In 1997 Feitelson et al. proposed a common set of terms for describing scheduling strategies [19]. We will use these definitions throughout the rest of this paper.

According to [19], scheduling models can be classified according to partition specification, job flexibility, level of preemption, amount of job and workload knowledge available, and memory allocation. Clearly, a metacomputer is a distributed memory system (although some of its nodes may be shared memory machines). The other criterions are less easy to define. Given the current practice and the results of Smith, Foster, and Taylor in [43], it seems sensible to allow the expected application runtime as the only job related information available to the scheduler. Newer management software packages such as CCS [21] allow much more complex job descriptions, but these systems cannot be expected to be broadly available in the near future. On the machine level handling of partition sizes ranges from fixed partitions on many batch systems to dynamic partitions on modern SMP architectures. On the metacomputer level it seems best to define partition sizes according to job requirements and current load situation before an application is being executed. The effort of implementing mechanisms for re-adjusting partition sizes during run time seems inappropriate considering the small number of applications that currently support such a feature. Therefore, we are using variable partitions in our metacomputer model. The same arguments also hold for job flexibility. Several parallel applications can run on different partition sizes but only few support dynamic re-adjustment during run time. Therefore, we are looking for a scheduler that can efficiently manage a mix of moldable and rigid jobs (strictly speaking, rigid jobs are moldable jobs with a fixed degree of parallelism). The most difficult decision was, whether the metacomputer shall support preemption. Up to now, only few job management systems provide support for preemptive scheduling. However, many long running scientific applications (i.e. applications well suited for metacomputing) have internal checkpointing and restart facilities. Therefore, it would not be too difficult to implement coarse grain preemptive scheduling for these applications. Especially since preemption seems to be a promising concept for scheduling of parallel jobs [38]. Nevertheless, current versions of the major metacomputing environments – including our own MOL system [35] – do not support preemption and therefore we have so far only considered run-to-completion scheduling algorithms.

Typically, scheduling disciplines are distinguished as online or offline algorithms. However, as we will show below, this difference becomes less important in the metacomputer scenario. Therefore, both types of algorithms are potential candidates for a metacomputer scheduler. Since the problem of scheduling parallel tasks on multiple parallel machines has been proven to be computationally hard for most cases, the most commonly used algorithms are based on heuristics. Although several papers have been published on using general-purpose heuristics for scheduling [37,8], better results are usually obtained by more problem specific strategies. The latter can be divided into two categories. On the one hand, there are algorithms derived from analytical considerations like for example those discussed in [29]. Often, these strategies are difficult to be implemented but guarantee a good worst case behavior. On the other hand, there are several fairly simple heuristics that have been tested in simulation environments or sometimes even as part of real world scheduling systems [32,4,10]. These approaches do not have such a good worst case behavior, but the results for "typical" problems are often quite good [29]. We decided to concentrate our examinations on those strategies that are not too complex for implementation and for which good performance has been demonstrated with real workloads. Although there are still a lot of algorithms in this class, many of them are based on three fundamental approaches: FCFS [39], FFIH [34], and ASP [40]. In Sec. 5 we provide a closer look at these strategies and show, if and how they can be applied to a metacomputing scenario.

3 Modeling The Metacomputer Infrastructure

Our main motivation was to find a scheduling algorithm that can be implemented on top of the NRW metacomputing infrastructure [30]. One of the key concepts of this project is to maintain autonomy for the participating supercomputing centers. Especially, it was not possible to replace existing resource management systems like NQS [2], NQE [11], LoadLeveler [26], or CCS [21] by a global scheduling module. The metacomputer management facilities had to be installed on top of the existing environments. It should be emphasized that this is not only a problem of the NRW metacomputer. Other metacomputing projects like GLOBUS [20], POLDER [44,31], or the Northrhine-Westphalian Metacomputer [36] had to use similar approaches. As a consequence the metacomputer scheduling algorithm cannot access the available machines directly but has to submit its jobs to the local queuing systems. Furthermore, users with direct access to the machines cannot be forced to submit all their jobs via the metacomputer. Especially since metacomputer frontends are designed to provide easy access to a heterogeneous machine pool and therefore often do not support the more advanced features of the hardware. Thus, local users will continue submitting jobs to the machines directly, thereby bypassing the metacomputer scheduler. Fig. 1 depicts this model. Each computer is controlled by a local resource management system (RMS). The metacomputer scheduler has no direct access to the hardware. It can only submit (sub-)jobs via each center's local queuing environment. In general, there are two different types of resource requests that can be sent to such a system. On the one hand, these are metacomputer jobs that are submitted to the metacomputing scheduler. On the other hand, there are local requests that are submitted directly to the resource management software without using the metacomputer facilities. The amount of information that is communicated from the local queuing systems to the metacomputer is a parameter of the model. Basically, there are three different levels of information exchange possible:

1. The metacomputer has no knowledge about the locally submitted jobs. It has no information, if (or how many) resources have already been assigned by the RMS and how much is left for the metacomputer (*scheduling with no control*).
2. The meta-scheduler can query the local queuing systems about the resources assigned to local jobs. This may include jobs that have already been submitted but are not yet scheduled. This type of information can be unreliable,

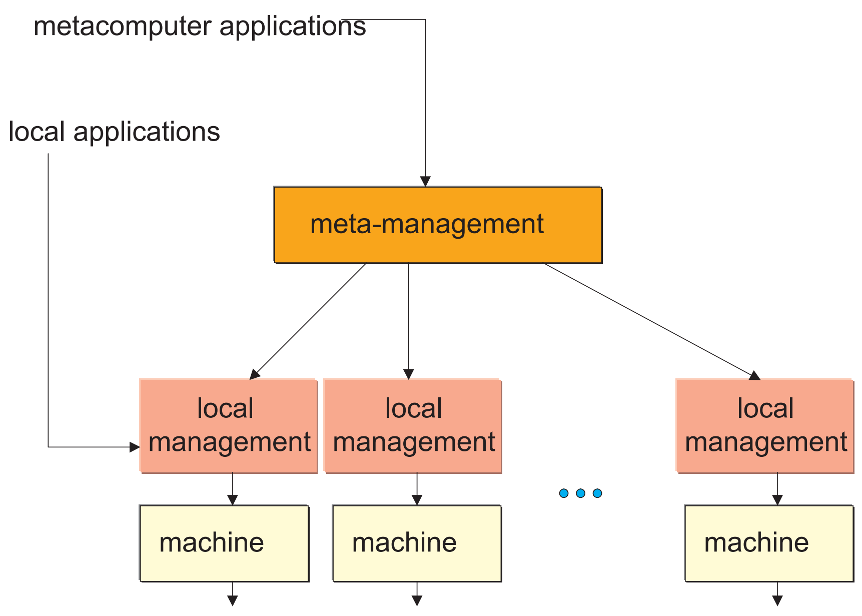

Fig. 1. The metacomputer accesses the machines through local resource management systems

because the jobs' resource requirements are typically not exactly known in advance (*scheduling with limited control*).

3. Whenever a job is submitted to a local management system, it is forwarded to the metacomputer for scheduling (*scheduling with full control*). This is the model that has been used in most theoretical papers on metacomputer scheduling.

It is obvious that a scheduling algorithm designed for model 1 cannot perform better than the same strategy working in scenario 2. The same holds for models 2 and 3. Consequently, scenario 1 can be expected to deliver the worst results. However, this approach is currently used in most real world metacomputer management systems [20,36,31]. The most promising model is the approach that gives full control to the metacomputer scheduler. Unfortunately, due to the political reasons explained above, this approach is unlikely to become reality. Therefore we examined the performance decreases that happen to scheduling algorithms developed for the full control model, if they are used in a no control environment. Since worst case analysis can sometimes be misleading, if the performance of algorithms applied to real world problems shall be determined [19,24], we base our comparisons on simulation runs. In order to obtain results that are useful for other researchers, too, we derived our simulated environment from the suggested standard workload model that was proposed by Feitelson and Rudolph in [18]. A detailed description of the workload model is given in Sec. 4.

Besides the scheduling hierarchy, the configuration of the heterogeneous machine pool is another important aspect of the metacomputer model. For our work, we employ the concept of *application centric metacomputing* described in [22]. Therefore, we may assume that in principle any job can be fulfilled by each machine. The only heterogeneity that is visible to the metacomputer scheduler is caused by performance differences between the computing nodes. For simplicity, we will use performance and number of processors of a node as synonymous for the rest of this paper (see [22]). What still remains to be defined is the distribution of machine sizes (i.e. performances) within the metacomputer. Since we are researching on metacomputing with supercomputers, we used the *Top 500 Report* of Jack Dongarra et al. as a reference [13]. Fig. 2 depicts the performance distribution of the world's 500 fastest supercomputers. As can be seen, perfor-

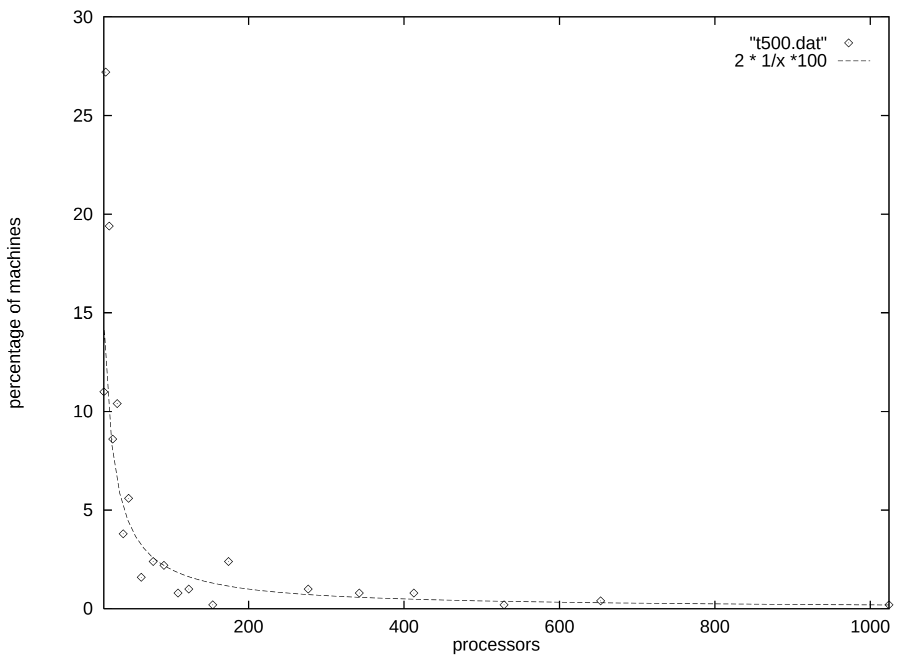

Fig. 2. Performance distribution in Top500 list

mance follows roughly a uniform logarithmic distribution and the probability of a machine having less than P processors can be estimated by

$$p(Procs \leq P) = 2\, ln(P) + c \qquad (1)$$

4 Workload Model And Evaluation Criteria

The model for evaluating our scheduling strategies is derived from the suggestion made by Feitelson and Rudolph in [18]. Their work was a first step towards a unified benchmark that will make it easier to compare the quality of different scheduling algorithms. Currently, there exists a broad variety of working scenarios and evaluation metrics used by different authors to prove the quality of their algorithms. This often makes it difficult – if not impossible – to compare performance values of different algorithms. However, the suggested model does not provide a clearly defined benchmark but contains several topics that are still subject of ongoing research. In order to implement a simulation environment, we had to fill these gaps with what seemed to be the best possible solution at this time. In the following we give a formal description of the workload model and the criteria that were used for evaluating the metacomputer scheduling techniques.

4.1 Job Arrivals

As in [18] we use an open on-line model where jobs are continuously fed into the system while the scheduler is working. The rate at which new jobs enter the metacomputer is modeled by a polynomial of degree 8 as described in [7]. Thus, the arrival rate at any given time t is

$$\lambda(t) = 3.1 - 8.5t + 24.7t^2 + 130.8t^3 + 107.7t^4 + 804.2t^5 + 2038.5t^6 + 1856.8t^7 + 4618.6t^8 \tag{2}$$

with $-0.5 \leq t \leq 0.5$ representing the time span from 8:30 AM to 6:00 PM. During the remaining time, we estimate an arrival rate of one job per hour for normal load situations. In order to obtain scheduling performance as a function of load, we multiply this arrival rate by a compression factor in order to simulate different load situations.

Obviously, this way of modeling job arrivals is derived from observations made on stand-alone supercomputers. At a nation wide or even global metacomputer jobs will probably arrive in a more uniform distribution.

4.2 Job Sizes

Estimation of job parallelism is based on the suggestions made in [18] and [14]. This means that a distinction is made between jobs using a number of processors that is a power of two and the remaining ones. Both [18] and [14] observed that powers of two are much more often requested than other processor numbers. Hence, it was suggested to use a model where the probability of using fewer than n processors is roughly proportional to $log n$ and modify it to contain extra steps where n is a power of two. The height of these steps is derived from the workload parameter p_2 that determines the fraction of jobs using partition sizes that are powers of 2. Feitelson and Rudolph observed values of p_2 around 81% but since it is unclear, if this is going to decrease in the future, it was suggested to integrate it as a parameter rather than a constant.

The second dimension of a job's size is its execution time. Results reported in [16] and [14] indicate that runtimes of jobs on large systems have a wide distribution with a high degree of variation. This may be the reason why there is currently no model available that leads to convincing results for all the accounting data that we have examined [21,17]. But there is evidence that the traces made available by some supercomputers show a mixture of two different job classes. On the one hand, there are short running jobs that have only a small degree of parallelism. These jobs are typically generated during the development cycle. On the other hand, there are long running production runs that require large amounts of resources. This effect cannot be seen equally well on all available traces and especially for a metacomputer we expect only a small number of development runs. Consequently we decided to follow the suggestion made in [18] and [14] for the simulation of batch production runs and assume uniform logarithmic distribution for the execution times. This means that for any given job requesting A processors the probability that its execution time on a monolithic machine lasts t units is

$$p(t) = \frac{1}{10t} \tag{3}$$

4.3 Internal Job Structure

The model of the internal job structure proposed in [18] is very generic and needs some refinement in order to be applicable to our scenario. In the metacomputing context the most important properties of a job are scalability and sensitivity towards slowdown of selected communication links. Preliminary surveys among users of the *Metacomputer Online Initiative* [35] and the *Northrhine-Westphalian Metacomputer Task-force* [36] indicate domination of two distinct job classes:

1. rigid jobs with frequent barrier synchronization or grid structured communication and
2. malleable applications with only few communications that are programmed using a client/server paradigm.

In the following, we will refer to class 1 as *synchronous* jobs and to jobs of class 2 as *asynchronous*. The fraction p_{sync} of synchronous jobs to all jobs is a parameter of the model.

For our scheduler it is important to be able to calculate the expected runtime of a job according to the assigned amount of processors and the number of machines used. The latter is relevant because the number of machines determines the number of slow WAN network connections a job has to cope with. Besides the fact that synchronous jobs imply a fixed degree of parallelism, the main difference between the two job classes is the dependency between available communication performance and expected run time. Due to the frequent synchronizations performed by synchronous jobs, the runtime performance is determined by the speed of the slowest communication link. This means that, if a synchronous job is mapped onto multiple computers, the actual number of machines used has no effect on the expected execution time. If there is one wide area network link involved, it dominates the communication behavior of the application completely. Hence, the parallel execution time of such a request r that uses c_i processors on each machine $i \in \{0, \ldots, M-1\}$ can be derived as

$$Time_{Sync}\left(r, (c_0, \ldots, c_{M-1})\right) \ := \\ \frac{Seq(r)}{S_r\left(\sum_{j=0}^{M-1} c_j \cdot P_j\right)} \cdot \left[1 + Comm(r) \cdot \left(\frac{BW_{MPP}}{BW_{WAN}} - 1\right) \cdot \phi_{Sync}\right]$$

with

$$\phi_{Sync} := \min\left(\ Card\left(\{c \in \{c_0, \ldots, c_{M-1}\} \ | \ c > 0\}\right) - 1 \ \ , \ \ 1\ \right) \tag{4}$$

where P_j denotes the number of processors available on machine j, BW_{WAN} represents the bandwidth of a wide area communication link, BW_{MPP} the bandwidth within each of the parallel machines, and $S_r(n)$ the speedup for n processors. $Seq(r)$ denotes the runtime on a single processor and $Comm(r)$ the fraction of the overall execution time that is consumed by communication, if the job is executed on a number of processors that corresponds to its average parallelism. Since it is unknown, how the distribution of $Comm(r)$ in a representative set of metacomputer jobs will look like, we decided to assume a uniform distribution within the range of $0 < Comm(r) \leq 0.5$.

As a consequence of (4.3) the scheduler should avoid splitting a synchronous job onto multiple machines wherever possible. However, if it has to be split, then the algorithm is free to choose an arbitrary number of different machines.

The effect of a partial communication slowdown on an asynchronous job is less dramatic than in the synchronous case. Here we assume a star-like communication on the WAN links. In the client/server paradigm this means that all the work packages are generated on one single machine from which they are distributed to the worker nodes. Arbitrary communication topologies can be used within each of the machines. Therefore, the parallel runtime of a synchronous job is less sensitive towards a partitioning onto multiple computers. The slowdown in communication is proportional to the amount of data that has to be sent across wide area links. Assuming that this is again proportional to the amount of processors on the other side of link, the expected execution time can be obtained from

$$Time_{Async}\left(r, (c_0, \ldots, c_{M-1})\right) \ := \\ \frac{Seq(r)}{S_r\left(\sum_{j=0}^{M-1} c_j \cdot P_j\right)} \cdot \left[1 + Comm(r) \cdot \left(\frac{BW_{MPP}}{BW_{WAN}} - 1\right) \cdot \phi_{Async}\right]$$

with

$$\phi_{Async} := \frac{\left(\sum_{j=0}^{M-1} c_j \cdot P_j\right) - \max_{0 \leq j < M}\left(c_j \cdot P_j\right)}{\sum_{j=0}^{M-1}\left(c_j \cdot P_j\right)} \tag{5}$$

So far we have not explained, how the speedup function $S_r(n)$ shall be defined. Our solution follows the suggestions made by Downey in [15]. His model compares well with real life workload traces but incorporates only few free parameters. Downey characterizes the speedup behavior of a program by its average parallelism A and the variance in parallelism σ with

$$S_r\left(n\right) := \begin{cases} \frac{An}{A+\frac{\sigma(n-1)}{2}} & 0 \leq \sigma \leq 1\ ,\ 1 \leq n \leq A \\ \frac{An}{\sigma(A-\frac{1}{2})+n(1-\frac{\sigma}{2})} & 0 \leq \sigma \leq 1\ ,\ A < n \leq 2A-1 \\ \frac{nA(\sigma+1)}{A+A\sigma-\sigma+n\sigma} & \sigma > 1\ ,\ 1 \leq n \leq A+A\sigma-\sigma \\ A & else \end{cases} \tag{6}$$

Typical values for σ are in the range of $0 \leq \sigma \leq 2$. Although no extensive study has been made on the distribution of σ in real workloads, Downey suggests in [14] to use a uniform distribution.

4.4 Evaluation Criteria

Although much research has been done on scheduling, there is still no final answer to the question, which metrics shall be used for measuring the quality of a schedule [18,19]. This is because different people have different requirements. For example, managements of computing centers have strong interests in optimizing the overall throughput of their machines. The users of these centers however, like to obtain good response times for their jobs. Since managers of computing centers should also be interested in satisfying the needs of their users, it might be a good idea to search for optimal response time schedules with good throughput. However, response time measurement can lead to irritating results since it tends to overemphasize the effects of small jobs. Nevertheless, we decided to focus primarily on optimizing response time since it is used in many papers dealing with on-line scheduling and it is at least considered a sensible metrics in [18]

5 The Algorithms

An analogy that is often used for describing scheduling algorithms represents machines as two dimensional buckets. The width of such bucket is related to the total amount of resources on the corresponding machine and the height represents time (see Fig. 3). Each request that has been scheduled for a single machine can therefore be represented by a rectangle defined by the number of processors allocated to that job (width of the rectangle) and the expected run time (height of the rectangle). For the rest of this paper we will use $Free_m(t)$ as the amount of free resources on machine m at time t. Furthermore, we define $Free^*_m(t)$ as

$$Free^*_m(t) := \min_{t' \geq t}\{Free_m\left(t'\right)\} \tag{7}$$

$Free^*_m$ will also be called the *surface* of m. In Fig. 3 the surface corresponds to the dashed line.

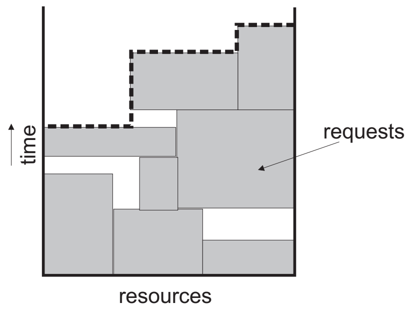

Fig. 3. Job schedule on a parallel machine and the corresponding surface

The same analogy can be used for the metacomputer scenario (see Fig. 4). For this case we define the surface of the whole machine pool M as $Free^*_M(t)$ with

$$Free^*_M(t) := \sum_{m \in M} Free^*_m(t) \tag{8}$$

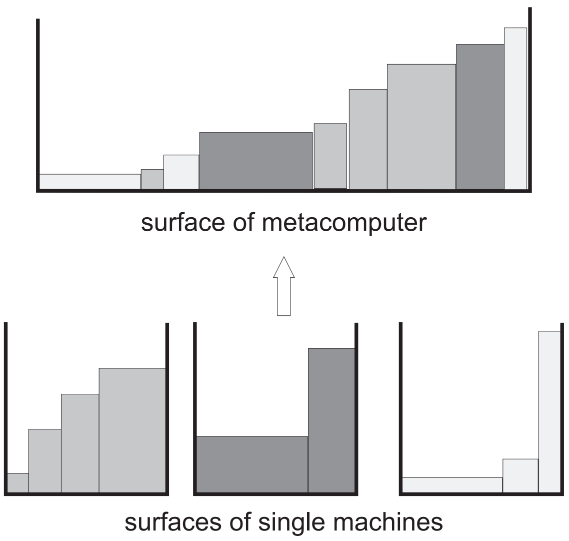

Fig. 4. Constructing the metacomputer's surface from machine surfaces

5.1 Scheduling With Full Control – The Idealized Scenario

A scenario that offers full control to the metacomputing scheduler can be achieved by denying direct submission of jobs to the local queuing systems. As a consequence, the scheduler has complete knowledge about all jobs in the system which makes metacomputer scheduling in this scenario similar to scheduling a single parallel machine. If the speed of the Internet connections between the machines was the same as internal communication, the problem would be identical to the job management of a single parallel machine. However, Internet communication is much slower and the algorithm has to take this into account. As explained in Sec. 2.2 we have chosen scheduling algorithms based on FCFS, FFIH, and ASP for further investigation.

FCFS *First Come First Serve* is a very simple approach. When a request for r resources is submitted to the system at time s, it will be configured at time t with $t = \min\{t' \geq s \mid Free^*_M(t') \geq r\}$. The choice which – and how many – machines shall be used is based upon a greedy strategy. If the request represents an asynchronous application it is divided into as few parts as possible in order to decrease its overall execution time. If the program to be launched belongs to the synchronous class, the algorithm tries to fit the request into one single machine. If this is not possible, it is divided into as many parts as possible thereby using up all those small partitions that would otherwise slow down subsequent asynchronous (or probably also synchronous) applications. This is a feasible approach since the communication costs of synchronous jobs depend on the speed of the slowest network connection.

For the metacomputing scenario, we derived a variant called **FCFS*** that does not choose t as the earliest possible starting time but as that time that assures the earliest possible finishing time. This differs from the original algorithm, because in the metacomputing scenario a later starting time can lead to an earlier result because it could possibly be mapped on less machines (see equations (4.3) and (5)).

Although FCFS is a very simple algorithm, experience has shown that its results are usually not as bad as one could have expected [39].

FFIH *First Fit Increasing Height* is a two-dimensional variant of the well known List Scheduling algorithm [34]. The basic idea of this approach is to sort all requests in order of increasing resource demands and then perform an FCFS scheduling with the sorted list. Strictly speaking, FFIH is an offline algorithm. However, if a waiting room is introduced [21], it can be used for online scheduling, too. Since FFIH is derived from FCFS, it shows similar behavior. However, it is typically better for minimizing the average response time, since small jobs are scheduled first and therefore the number of jobs with long waiting times is reduced.

ASP *Adaptive Static Partitioning* is one of the most promising disciplines for scheduling moldable jobs on parallel machines. Although up to now it has only rarely been used in practice, the algorithm is simple, easy to be implemented, and shows good performance on monolithic machines, if applied to real workloads [29]. ASP stores incoming requests in a waiting room until there are free processors available. It then assigns as many requests to these processors as possible. If the waiting room is sufficiently filled, this means that asynchronous (i.e.

moldable) jobs may be scaled down to only a few processors in order to be able to start more jobs at the same time. For jobs with a high degree of parallelism this can lead to extremely long execution times. Hence, we introduced a slight modification of the original algorithm which we call ASP*. This strategy assigns at least αA processors to any job for which A processors were requested, whereby α is a constant of the algorithm with $0 < \alpha \leq 1$.

5.2 Scheduling With No Control - The Reference Case

Scheduling with no control models the way job management is currently done by most metacomputing environments. Local jobs are submitted directly to the machines without notifying the metacomputer scheduler. Consequently, the scheduler mainly concentrates on good mapping and partitioning strategies. Furthermore, heuristics can be integrated to estimate the total workload on each machine of the metacomputer. In the following, we describe how the scheduling strategies were modified for this scenario.

Since FFIH, ASP, and ASP* need information about the contents of the local queues (the waiting room is only drained when the local queues are at least partially empty), these algorithms are not directly applicable in a scenario where no information is available. Hence, we chose only the FCFS strategy for examination in such an environment. FCFS with no control over local queues first sores all n machines in random order in a vector $(m_0, \ldots, m_n)$. Then each request r_i in the waiting queue is mapped onto machines $m_{i_1}, \ldots, m_{i_2}$ with $\sum_{l=i_1}^{i_2} Size(m_l) \geq A_{r_i}$, whereby $Size(m_l)$ gives the total amount of processors of machine l and A_{r_i} gives the overall number of resources requested by r_i. For any two jobs r_i, r_j with $i < j$ the mapping is created in a way such that $i_1 \leq i_2 \leq j_1 \leq j_2$.

Randomizing the machine vector for each scheduling round works as a simple heuristics to achieve balanced work distribution over the complete machine pool.

5.3 Scheduling With Minimum Control - The Compromise

Scheduling with minimum control is the compromise we propose as a feasible way to adapt scheduling algorithms designed for monolithic parallel machines to a metacomputing environment. The fundamental idea of this approach is to use the surfaces of all underlying machines for scheduling the metacomputer requests. Local jobs can still be submitted to the queuing systems directly. However, from time to time the metacomputer performs a scheduling round. When this happens, each machine informs the metacomputer about the current shape of its surface (or probably only a subset of the complete surface, depending on local administration policies). During the short period while the metacomputer performs its scheduling, the local queuing systems must not alter the shape of their surfaces. This can for example be achieved by delaying jobs, that arrive during this time, in a special waiting room.

This solution offers the freedom of the full control scenario during each scheduling round without imposing significant restrictions onto the local queuing systems. It is therefore a) feasible and b) a much better basis for advanced scheduling algorithms like for example those described in [29]. We call this technique *Interleaved Hybrid Scheduling* or IHS for short, because each machine is alternatingly managed by two different schedulers (local and meta scheduler). A drawback of IHS is that the surfaces presented to the metacomputer scheduling algorithm are usually not flat but rather form a more or less steep staircase. Hence, many of the algorithms found in literature cannot be applied directly.

The two possible solutions are either to schedule on top of a flat line placed above the highest step of the surface or to find an algorithm that can efficiently handle uneven surfaces. So far, we have only used the IHS technique with the algorithms described in Sec. 5.1. In the future, we are planning to search scheduling strategies that are better tuned to the IHS scenario.

There exist several possible schemes for deciding, when the metacomputer shall trigger a new scheduling round. For this paper we have examined two different approaches. The first solution was to wait a fixed time interval Δ_t between the termination of the last metacomputing job of the previous round and the invocation of a new one. This solution is attractive for the machine owners, since it guarantees a fixed time interval during which their resource are not used by the metacomputer. For the results presented in Sec. 6 we used a value of $\Delta_t = 4$ hours.

The second and more flexible approach is to also consider the amount of resource requests that are pending in the waiting room of the metacomputer. In order to achieve this, we triggered a scheduling round of the metacomputer when

$$\frac{1}{\sum_{m \in MachinePool} Size\,(m)} \cdot \sum_{r \in WaitingRoom} A_r \cdot Time\,(r, A_r) \cdot Waited\,(r) \cdot \gamma \tag{9}$$

Where $Time\,(r, A_r)$ is the expected execution time of r on A_r processors of a monolithic machine, $Waited\,(r)$ is the amount of time units the request has so far been waiting, and γ is a constant weight factor that determines, how much the overall decision is influenced by the waiting time.

6 Results

The following results were all obtained from a wide range of simulation runs using the model described in Sections 3 and 4. We measured the accumulated response time of the first $c \cdot M$ jobs that were completed by the simulated metacomputer, whereby M was the total number of machines in the pool and c was set to 1000. As can be seen in Figures 5, 6, and 7, $c = 1000$ is not completely sufficient to eliminate all the effects of a randomized workload. Although these simulations were performed on the large parallel machines of the Paderborn Center for Parallel Computing, it took several weeks to collect the necessary data. Therefore, we believe that $c = 1000$ represents a good compromise between precision of the data and resources needed to obtain the results.

It should be pointed out that jobs are arriving throughout the whole simulation run. This means that the queues are not being drained towards the end of simulation which would be an unrealistic scenario. However, as we will point out later, this does not leave the results unaffected and has to be considered, when the simulation data is being interpreted.

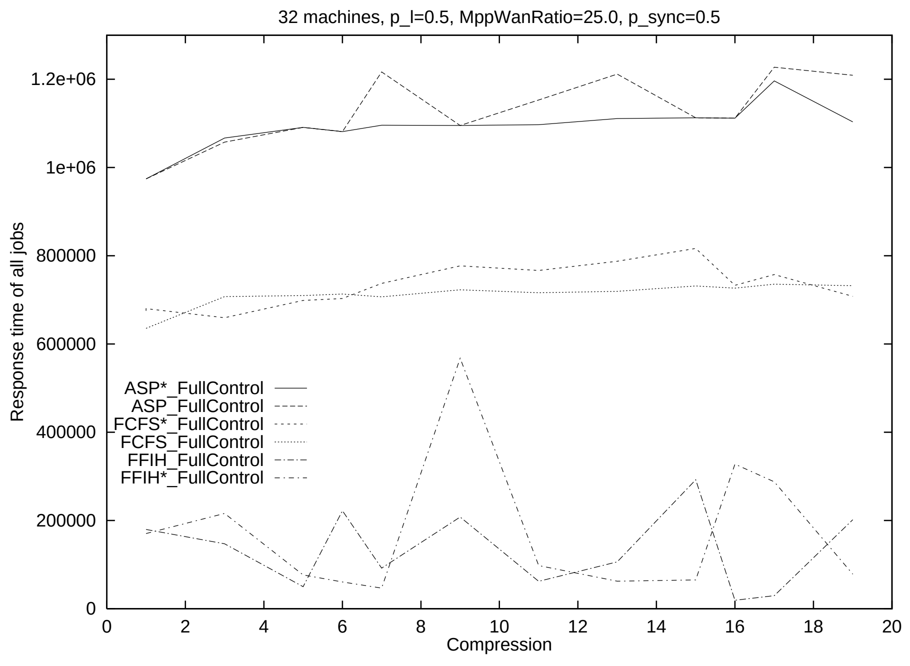

Fig. 5. Performance of different algorithms in the idealized scenario

Fig. 5 depicts the simulation results for algorithms running in the idealized scenario that does not allow direct submission of jobs to local queues (full control). In this diagram compression ranges up to a value of 19, meaning that within a certain time interval there are 19 times more job arrivals than normal. The metacomputer contained 32 machines, Internet communication was assumed to be 25 times slower than communication within the parallel machines, the ratio between synchronous and asynchronous workload was 1:1, and there were as many local jobs as jobs submitted through the metacomputer.

First of all it can be observed that all curves are converging for increasing load situations. This is due to the fact that the job queues were not drained towards the end of a simulation run. Thus, if the arrival rate reaches a certain algorithm dependent threshold, a large fraction of the jobs remains in the queues and has no or only marginal effect on the performance of the algorithm.

Furthermore, it is remarkable that the ASP-based algorithms show the weakest performance. This is due to the fact that ASP has a tendency to allocate very few processors even to large jobs. This effect is a little bit reduced by the ASP* variant. However, since ASP* becomes identical to FCFS if α gets close to 1, we have chosen $\alpha = 0.5$. This was not large enough to prevent the algorithm from creating extremely long running jobs and thereby preventing the start of large synchronous (i.e. rigid) jobs for a long time. We suppose that ASP would perform better, if the job mix did only contain asynchronous jobs.

FFIH, on the other hand, demonstrates good results in our simulation environment. However, the plots indicate that this strategy is very sensitive towards the job sequence to be scheduled. The reason for this becomes clear, if we imagine the worst possible case. This is an alternating sequence of jobs that either use few processors and run for a long time or jobs that block the whole metacomputer but can be finished almost instantly. Since these requests have similar resource demands such sequences are likely to be created by the sorting step of FFIH.

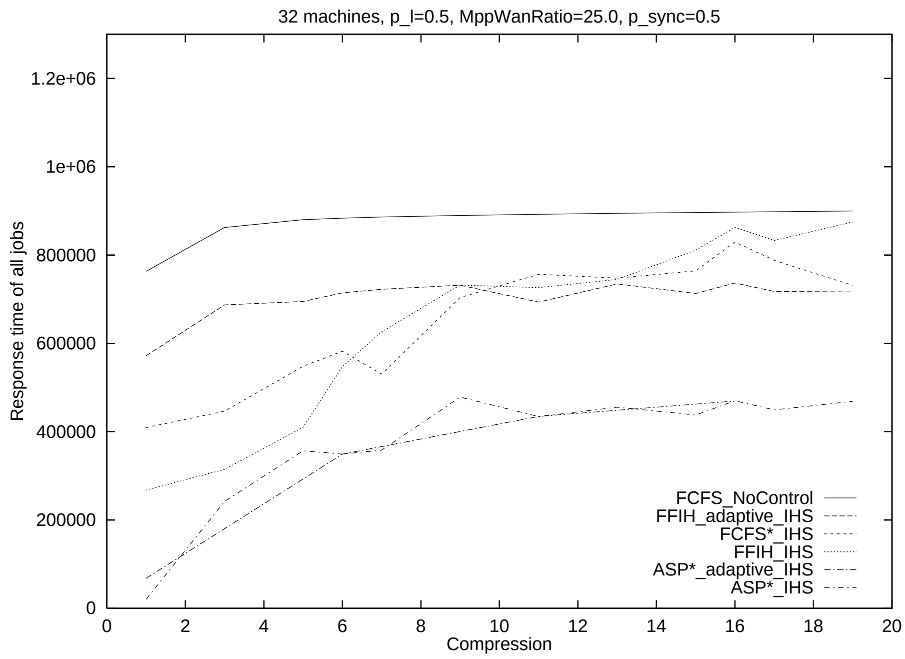

Fig. 6. Performance of different algorithms in a real world scenario

The results presented in Fig. 6 indicate what can be achieved in a real world scenario, if the IHS technique is applied. It can be seen that for smaller and therefore more realistic compression values all IHS variants show significantly better performance than an FCFS algorithm that uses no information about the local queues. For these load values the FFIH and FCFS strategies even seem to be better then they are in the scenario with full control over the local queues. This is caused by the fact that in the idealized scenario there are no locally submitted jobs, which are restricted to the size of the corresponding machines and therefore are smaller than many of the metacomputing jobs. As a consequence, care has to be taken when comparing the plots of Figures 5 and 6. We think that the slight difference in the incoming job mix corresponds to the users' behavior, but this assumption still has to be proven.

At a first glance, it is astonishing that the ASP algorithm seems to be the best strategy for the real world scenario. Having a closer look at Fig. 7, it can be seen that the result plots for ASP in Fig. 6 are mainly dominated by the response times of those jobs submitted by local users. In other words, the execution times of metacomputing jobs became so long that only few of them terminated among the first $c \cdot M$ jobs. Hence the good results of APS have been achieved by providing a much worse quality of service for multi-site applications. Hence, for scheduling a metacomputer we propose to use IHS either with FCFS* or with FFIH. The latter is best for moderate workloads while FCFS tends to behave slightly better, when the system is extremely heavy loaded.

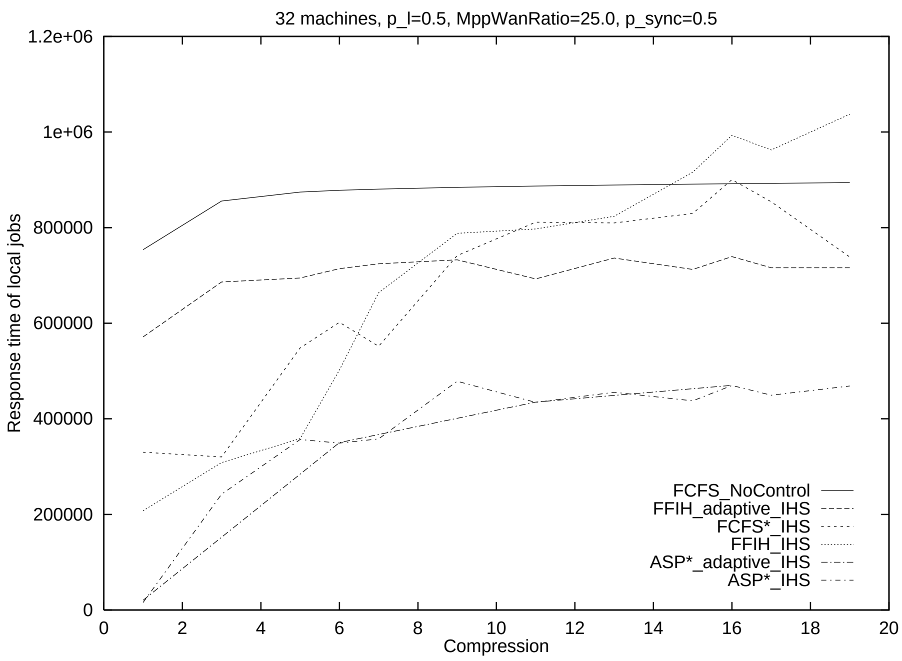

Fig. 7. Average response time of jobs that were submitted locally

7 Conclusion

Our motivation was – and still is – to find the best possible scheduling algorithm for the NRW metacomputer. Looking at what was available in the literature exposed a large gap between algorithms that have been studied analytically and those found in existing environments. Therefore, we extended the model of Feitelson and Rudolph towards metacomputer scheduling in order to obtain a tool for measuring the quality of different algorithms. Care was taken to derive a model that reflects the real world scenario as close as possible. Only in the second line, we tried to keep it simple enough for analytical studies. We think, that our model is still close enough to the proposal of Feitelson and Rudolph to allow comparison with other results obtained from the same approach. Much effort has been spent on the implementation of this model. Therefore, we tried to create a generic simulation environment, which we are making publicly available. This may also be useful to researchers, who study scheduling of monolithic parallel machines, since a metacomputer can be seen as a generalization of this concept.

We found that an important factor for metacomputer scheduling is the existence of local queues to which jobs are submitted without being passed through the metacomputer system. Hence, we developed the IHS technique as a feasible approach to use well known scheduling algorithms for the prototype of a working metacomputer. Our results indicate that for moderate workloads use of the IHS technique with the FFIH algorithm decreases the average response times significantly. However, more effort should be spent on examining more powerful algorithms with IHS.

Another important aspect of metacomputer scheduling that was not dealt with in this paper is the reliability of job descriptions. So far we assumed that everything the scheduler is being told about a job is absolutely precise. However, in reality this is usually not the case. Hence, in our future work, we plan to pay closer attention to the effects of unreliable information on scheduling algorithms for metacomputing.

References

1. Academic Computing Services Amsterdam. The SARA Metacomputing Project. WWW Page. http://www.sara.nl/hec/projects/meta/.
2. Carl Albing. Cray NQS: production batch for a distributed computing world. In *Proceedings of the 11th Sun User Group Conference and Exhibition*, pages 302–309, Brookline, MA, USA, December 1993. Sun User Group, Inc.
3. J. Almond and D. Snelling. *UNICORE: Secure and Uniform Access to Distributed Resources via the World Wide Web*, 1998. http://www.kfa-juelich.de/zam/RD/coop/unicore/.
4. Stergios V. Anastasiadis and Kenneth C. Sevcik. Parallel application scheduling on networks of workstations. *Journal of Parallel and Distributed Computing*, 43(2):109–124, June 1997.
5. T. E. Anderson, D. E. Culler, and D. A. Patterson. A case for NOW (Networks of Workstations). *IEEE Micro*, 15(1):54–64, February 1995.
6. R. Baraglia, R. Ferrini, D. Laforenza, and A. Lagana. Metacomputing to overcome the power limits of a single machine. *Lecture Notes in Computer Science*, 1225:982ff, 1997.

7. M. Calzarossa and G. Serazzi. A characterization of the variation in time of workload arrival patterns. *IEEE Transactions on Computers, Vol. C-34, :2, 156-162*, 1985.
8. Olivier Catoni. Solving scheduling problems by simulated annealing. *SIAM Journal on Control and Optimization*, 36(5):1539–1575, September 1998.
9. Steve J. Chapin, Dimitrios Katramatos, John Karpovich, and Andrew S. Grimshaw. Resource management in legion. Technical Report CS-98-09, Department of Computer Science, University of Virginia, February 11 1998. Wed, 19 Aug 1998 17:14:25 GMT.
10. Su-Hui Chiang, Rajesh K. Mansharamani, and Mary K. Vernon. Use of Application Characteristics and Limited Preemption for Run-To-Completion Parallel Processor Scheduling Policies. In *Proceedings of the 1994 ACM SIGMETRICS Conference*, pages 33–44, February 1994.
11. Cray Research. NQE. commercial product.
12. Thomas A. DeFanti, Ian Foster, Michael E. Papka, Rick Stevens, and Tim Kuhfuss. Overview of the I-WAY: Wide-area visual supercomputing. *The International Journal of Supercomputer Applications and High Performance Computing*, 10(2/3):123–131, Summer/Fall 1996.
13. Jack Dongarra and Hans Meuer and Erich Strohmaier. Top 500 Report. WWW Page, 1998. http://www.netlib.org/benchmark/top500/top500.list.html.
14. Allen B. Downey. A parallel workload model and its implications for processor allocation. Technical Report CSD-96-922, University of California, Berkeley, November 6, 1996.
15. Allen B Downey. A model for speedup of parallel programs. Technical Report CSD-97-933, University of California, Berkeley, January 30, 1997.
16. D. G. Feitelson. Packing schemes for gang scheduling. *Lecture Notes in Computer Science*, 1162:89ff, 1996.
17. D. G. Feitelson and B. Nitzberg. Job characteristics of a production parallel scientific workload on the NASA ames iPSC/ 860. *Lecture Notes in Computer Science*, 949:337ff, 1995.
18. D. G. Feitelson and L. Rudolph. Metrics and benchmarking for parallel job scheduling. *Lecture Notes in Computer Science*, 1459:1ff, 1998.
19. D. G. Feitelson, L. Rudolph, U. Schwiegelshohn, and K. C. Sevcik. Theory and practice in parallel job scheduling. *Lecture Notes in Computer Science*, 1291:1ff, 1997.
20. 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.
21. J. Gehring and F. Ramme. Architecture-independent request-scheduling with tight waiting-time estimations. *Lecture Notes in Computer Science*, 1162:65ff, 1996.
22. J. Gehring, A. Reinefeld, and A. Weber. PHASE and MICA: Application specific metacomputing. In *Proceedings of Europar 97, Passau, Germany*, 1997.
23. Genias Software GmbH, Erzgebirgstr. 2B, D-93073 Neutraubling. *CODINE User's Guide*, 1993. http://www.genias.de/genias/english/codine/.
24. Hoare. Quicksort. In *C. A. A. Hoare and C. B. Jones (Ed.), Essays in Computing Science, Prentice Hall.* 1989.
25. Chao-Ju Hou and Kang G. Shin. Implementation of decentralized load sharing in networked workstations using the Condor package. *Journal of Parallel and Distributed Computing*, 40(2):173–184, February 1997.
26. IBM Corporation. *Using and Administering LoadLeveler – Release 3.0*, 4 edition, August 1996. Document Number SC23-3989-00.

27. K. Koski. A step towards large scale parallelism: building a parallel computing environment from heterogenous resources. *Future Generation Computer Systems*, 11(4-5):491–498, August 1995.
28. Robert R. Lipman and Judith E. Devaney. Websubmit - running supercomputer applications via the web. In *Supercomputing '96*, Pittsburgh, PA, November 1996.
29. Walter T. Ludwig. Algorithms for scheduling malleable and nonmalleable parallel tasks. Technical Report CS-TR-95-1279, University of Wisconsin, Madison, August 1995.
30. The NRW Metacomputing Initiative. WWW Page. http://www.uni-paderborn.de/pc2/nrwmc/.
31. B. J. Overeinder and P. M. A. Sloot. Breaking the curse of dynamics by task migration: Pilot experiments in the polder metacomputer. *Lecture Notes in Computer Science*, 1332:194ff, 1997.
32. E. W. Parsons and K. C. Sevcik. Implementing multiprocessor scheduling disciplines. *Lecture Notes in Computer Science*, 1291:166ff, 1997.
33. Platform Computing Corporation. LSF Product Information. WWW Page, October 1996. http://www.platform.com/.
34. F. Ramme and K. Kremer. Scheduling a metacomputer by an implicit voting system. In *Int. IEEE Symposium on High-Performance Distributed Computing.* Paderborn Center for Parallel Computing, 94.
35. A. Reinefeld, R. Baraglia, T. Decker, J. Gehring, D. Laforenza, F. Ramme, T. Römke, and J. Simon. The MOL project: An open, extensible metacomputer. In Debra Hensgen, editor, *Proceedings of the 6th Heterogeneous Computing Workshop*, pages 17–31, Washington, April 1 1997. IEEE Computer Society Press.
36. V. Sander, D. Erwin, and V. Huber. High-performance computer management based on Java. *Lecture Notes in Computer Science*, 1401:526ff, 1998.
37. M. Schwehm and T. Walter. Mapping and scheduling by genetic algorithms. *Lecture Notes in Computer Science*, 854:832ff, 1994.
38. Uwe Schwiegelshohn. Preemptive weighted completion time scheduling of parallel jobs. In Josep Díaz and Maria Serna, editors, *Algorithms—ESA '96, Fourth Annual European Symposium*, volume 1136 of *Lecture Notes in Computer Science*, pages 39–51, Barcelona, Spain, 25–27 September 1996. Springer.
39. Uwe Schwiegelshohn and Ramin Yahyapour. Analysis of first-come-first-serve parallel job scheduling. In *Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 629–638, San Francisco, California, 25–27 January 1998.
40. S. Setia and S. Tripathi. A comparative analysis of static processor partitioning policies for parallel computers. In *Internat. Workshop on Modeling and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pages 283–286, January 1993.
41. Jon Siegel. *CORBA: Fundamentals and Programming.* John Wiley & Sons Inc., New York, 1 edition, 1996.
42. Larry Smarr and Charles E. Catlett. Metacomputing. *Communications of the ACM*, 35(6):44–52, June 1992.
43. W. Smith, I. Foster, and V. Taylor. Predicting application run times using historical information. *Lecture Notes in Computer Science*, 1459:122ff, 1998.
44. A. W. van Halderen, Benno J. Overeinder, Peter M. A. Sloot, R. van Dantzig, Dick H. J. Epema, and Miron Livny. Hierarchical resource management in the polder metacomputing initiative. submitted to Parallel Computing, 1997.