Proc. IPPS Workshop on Job Scheduling Strategies for Parallel Processing, LNCS, No. 1162, pp 65-88, 1996

Architecture-Independent Request-Scheduling with Tight Waiting-Time Estimations*

Jörn Gehring, Friedhelm Ramme

Paderborn Center for Parallel Computing
University of Paderborn
Fürstenallee 11, 33102 Paderborn

e-mail: [joern | ram]@uni-paderborn.de

Abstract

In the course of the last few years, the user's interaction with parallel computer-systems has changed. A continuous growth in the number of interactive HPC-applications can be observed. When considering partitionable MPP-systems with exclusive usage of the physically separated regions, issues like the average waiting-time become more dominant for the users than the total system-throughput.
In this paper, we focus on the problem of scheduling an arbitrary mixture of resource-requests for batch and interactive applications in an architecture-independent manner. To help users plan their daily work tight waiting-time estimations are indispensable. However, the resulting scheduling problem interferes with the problem of mapping requests onto certain MPP-architectures to reduce their internal fragmentations. We will show that this conflict can be alleviated by a distributed prover-verifier methodology. At first, we will introduce the distributed resource-management software CCS with its architecture-independent scheduling method. The message-based approach presented is used to verify the pre-calculated schedules with help of the system-dependent mapping instances. Simulations with the accounting data of our center have shown that tight waiting-time estimations can be made while the architecture-independent scheduling approach is still preserved. We will show that by using this methodology the mean error of the predicted waiting-time can be reduced by 76 %. Finally, we will discuss the impact of such a distributed resource-management system on the meta-computing challenge.

*This work was partly supported by the German "Ministerium für Wissenschaft und Forschung" and the research cooperation NRW-Metacomputing. Further grants were provided by the "Stifterverband für die Deutsche Wissenschaft."

1 Introduction

As parallel systems become more commonly used, there is a growing awareness for the need of a resource management software. Such software is indispensable for MPP-systems where jobs can be isolated from each other, giving each the fiction of a dedicated (virtual) machine, and where accounting functions should be carried out. When talking about problems related to the resource management task one should bear in mind some important observations:

- Reliable parallel computer systems exist and various industrial codes have been ported [4].
- In parallel high-performance computing, interactive applications with 'an engineer in the loop' demand reasonable computing performance, short response-times, and a seamless runtime-environment. Traditional batch programs will be replaced more and more by this type of interactive applications.
- Significant progress was made in the Wide-Area Network technology. Bandwidth and latency were improved by some orders of magnitude that make advanced metacomputer applications possible [18].

One should also notice that currently more than a dozen operating-systems, runtime-kernels, or message-passing libraries are in use. The user's requirements are of an extremely wide range [7]. While analyzing the accounting data of our center, we found that the

requested resources range from a single MPP-node to complete systems that were occupied for some minutes or up to some days. In the profile, no significant pattern could be identified.
Having made this observation, we concluded that any approach to the resource management problem should be highly independent of the system architectures to be managed, and at the same time it should be independent of the programming-environments currently preferred by MPP-users. Both conditions are indispensable to turn the idea of a virtual machine-room into practice [1, 14, 18].

In order to solve the request-scheduling problem for such an environment (see 3.1), one has to think about new methodologies that meet the demands of service providers as well as those of interactively working users. Not only the resources of an MPP-system demand an adequate scheduling mechanism, each MPP-user has to schedule his own time-constraints. During the past, this topic was frequently underestimated [7]. Indeed, the inability to run a program when desired sometimes causes significant user frustration, especially during the development-stage.
A possible solution would be to tell a user the estimated waiting-time after a resource-request is submitted. If this estimation is tight, a user can plan his work much more efficiently. On the other hand, if the estimation is too weak, it may occur that a user goes to lunch while his request is fulfilled. When he returns, the time is expired, the resources are released, and the bill has been made. The frustration will be even greater and the MPP-systems become labeled as being not usable in daily work.
The best estimation of the waiting-time can be made when the request-scheduler considers the constraints of all other users as well as all properties of the MPP-architecture in question and all details adopted from the system-configuration and its surrounding environment. The effort to develop such a scheduler will be tremendous. The resulting software-module will be very complex and highly dependent on architecture and installation. The expected life-time of such a software will be comparably short.

In this paper we will present an architecture-independent request-scheduling method and study the resulting waiting-time estimation problem. We will start with a short description of the management model (Sec. 2.1) and its realization by the Computing Center Software CCS (Sec. 2.2). The basic algorithms used for the request-scheduling task are introduced in Section (3). In Section (4.1), we study the waiting-time estimations when using autonomous scheduling algorithms. A verification protocol is presented in Section (4.2). In Section (4.3) it is shown that the interference between the request-scheduling problem and mapping problem can be exploited to get tight waiting-time estimations while keeping the request-scheduler independent of the system architecture. The simulation runs were performed with the accounting data gathered at our center. The paper will finish with a prospect for the upcomming meta-computer challenge.

2 A Comprehensive Approach

The Paderborn Center for Parallel Computing, PC^2, is equipped with a number of partitionable and freely configurable MPP-systems. The largest machine is a partitionable GCel from Parsytec with 1024 T 805 nodes. The most powerful machine currently is a partitionable GCPP with 128 MPC 601 processors. The systems of the PC^2 are used by more than 250 people in about 80 different projects. More than 50 % of the users are working outside Paderborn. Users from all over Germany have access to the PC^2 by the research-net (WIN) at 34 Mbps. Especially for small and medium sized enterprises, fast modem- and ISDN-connections are provided.
Being one of the Europort benchmark sites, several industrial codes have been ported to these machines [4], with users from seven European nations have been working remotely at the PC^2.

This user-community, combined with unsuited system-software, has been the background for the large Computing Center Software project[1] at the PC^2. Right from the beginning, CCS was not directed to a single machine. The aim was to make various types of MPP-systems transparently, and thus more efficiently usable than before. A comprehensive management model was developed which was implemented as a distributed software system. The subject of the CCS framework is the resource management task-force. On the one hand, CCS is a research package to study different optimization topics, on the other it is a production environment serving our whole machinery.

2.1 The Model

There is more to building up a virtual machine-room than just linking the HPC-systems by a fast interconnection network. It is a seamless environment that provides transparent access to various system architectures, it supports different runtime

[1] The URL of the CCS home-page is:
http://www.uni-paderborn.de/pcpc/ccs

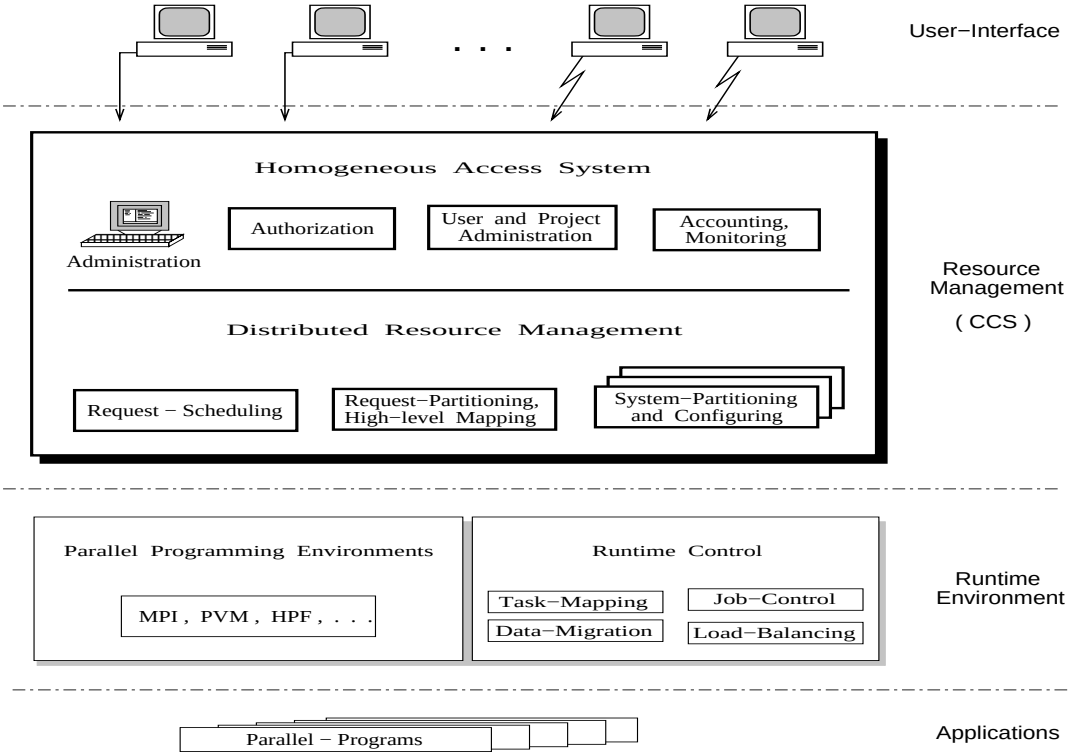

Figure 1: A comprehensive metacomputer management model

environments, uncountable user demands and also all the necessary administrative requirements. Taking this comprehensive view, four research domains can be identified which can be treated more or less independently (Fig. 1). While the CCS focus is on the resource management, a complementary project is initiated to improve the runtime environment [9].
In this paper we will study the interference between the architecture-independent request-scheduling task and the impact resulting from partitioning configurable MPP-systems.

The request-scheduling problem: When dealing with this problem, the utmost abstraction level can be assumed. Only the number of requested processors and an estimation of the occupation-time is taken into account. When the time is expired the resources are released. No assumptions are made about the facilities of the operating-system or the runtime-environment. Furthermore, we abstract from all constraints added by the architectures. Thus, the task is to determine a well-suited order of the hardware-requests to

- improve the overall utilization,
- reduce the average waiting-time,
- support interactive and batch applications simultaneously,
- support a priority mechanism while fairness has to be guaranteed,
- provide a high-level mechanism to decide if a resource-request can currently not be configured due to system internal fragmentations or whether this request can never be fulfilled by any machine under control. This decision should be made without knowledge of the architecture nor of the detailed system-configuration.

Another user-demand, not considered yet, is to support guaranteed reservations.

The architecture dependent mapping problem: This problem reflects the fact that most MPP-systems in public operation are variable- (or at least fixed-) partitionable. Thus, before a parallel program can be activated, a suited partition must be determined and the system must be configured accordingly. The shape of the partition might be subject of further optimizations. The goal of the mapping process is to reduce the internal fragmentation while fulfilling all of the architecture dependent constraints, (e.g.: each parti-

tion must have at least one I/O-link for host-access). An Intel iPSC, for example, must be partitioned into hypercubes of smaller dimensions, a Cray T3D into 3D-subcubes, a Parsytec GCel (GCPP) into rectangular subgrids. Such a subgrid must consist of so called atomic units. Due to hardware-internal restrictions, an atomic unit of a GCel is a 4x4 grid while an atomic unit of a GCPP is a 2x2 grid. While a GCel has some I/O-links into the inner area of its 2D-architecture, a GCPP can only be accessed from its outer border. Links and processor-nodes are dedicated to exclusive usage. With freely configurable MPP-architectures, completely different criteria come into question [8]. As there is (generally) no knowledge about the resource-requests of the future, further burdens are placed to the mapping decision.

2.2 The CCS Framework

The approach developed by PC^2 to tackle the resource management problem led to a distributed management-software running in the Unix environment in front of the HPC-systems. Its underlying multi-agent model was implemented as a system of communicating Unix daemons.

On the one hand, this gives the user the view of a virtual machine-room while on the other it offers various possibilities to optimize the selection of best suited machines, to improve the architecture dependent request mapping, and to optimize some of the conflicting scheduling goals. As a basic understanding of the distributed management model is necessary for this paper, a short description of CCS is given in the following. A detailed discussion of the software-package is presented in [16].

The essential daemons of CCS are depicted in Fig. 2. The common user interface to CCS is the so called *Mastershell*. It offers a limited environment for creating Virtual Hardware Environments and running applications in interactive or batch mode. When a Mastershell comes up, a connection to the *Port-Manager Daemon* (PM-D) is established and data identifying the user (e.g. uid, hostname) are transferred. The PM-D uses this information to initiate a first authorization by asking the *Database Daemon* (DB-D). If the authorization failed, the user session is aborted immediately. Otherwise, the user has the whole command language of the Mastershell at his disposal. Currently, a replacement for the Mastershell is under construction. This new interface will turn the user's view into that of a client-server model. The multi-agent model of CCS, however, remains untouched. There are also discussions to develop a user-interface based on Java [2] to exploit the new internet facilities.

As an example, let us assume that a user requests a virtual hardware environment consisting of a number of processors in a certain configuration and exclusive usage for one hour. When such a request is ordered from the Mastershell the PM-D checks the user's limitations first, i.e. the number and kind of resources maximally allowed for the requesting user or project. If the request validation is successful, it is sent to the *Queue-Manager Daemon* (QM-D). The QM-D consists of a waiting-room and a request-scheduling module (Fig. 3). If the scheduler of the QM-D decides that a certain request should be fulfilled, this request is sent to the PM-D to be configured. In cooperation with the selected *Machine-Manager Daemons* (MM-Ds) the PM-D creates the requested hardware environment and supervises the corresponding time limits. For authorization and accounting purposes the PM-D consults the DB-D. This daemon can be linked with two public interfaces, one for user management and the other for request accounting. In this way it is possible to connect commercial database systems or to adapt home-made software packages. When the requested resources are available, the user will be allowed to start an arbitrary application using nearly any type of runtime-environments.

Additionally, the PM-D provides logging facilities and preserves an operator interface for administering the virtual machine-room. Furthermore, the PM-D performs the request synchronization task and is responsible for the high-level request mapping. The MM-Ds perform the architecture dependent request mapping and the online system-configuration. If the MPP-architecture is flexible enough, various heuristics can be applied to reduce the internal fragmentation. An MM-D is the only architecture dependent part of the CCS-software. Thus, integrating a new system family can simply be done by re-implementing the corresponding MM-D. The QM-D is responsible for solving the request-scheduling problem in an architecture independent manner. A possible solution of this problem is outlined in Section (3).

This functional distribution of services gives CCS the power and flexibility to support a wide range of MPP-architectures [14].

The main features of CCS are the following:

- Uncoupled user- and system-views.
- Transparent access to MPP-systems with different architectures.

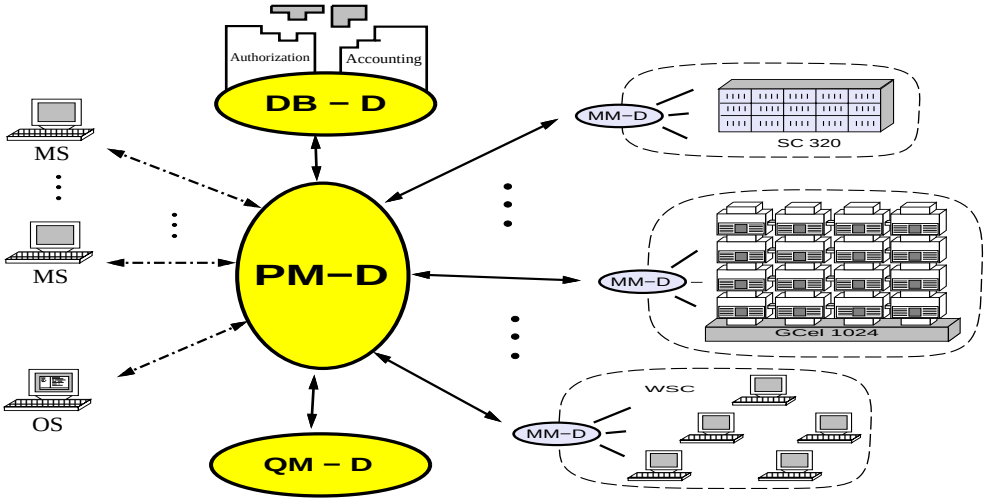

Figure 2: The essential CCS daemons

- Arbitrary mixture of interactive and batch applications within the virtual machine-room.
- Optimized request-scheduling.
- Dynamic partitioning of MPP-systems in order to reduce their internal fragmentations.
- Central authorization, accounting and charging facilities.
- Highly independent of the runtime-environment and the HPC architectures.
- Support for unstable WAN-connections. (Even if a dial-up line breaks down, a parallel application (batch or interactive) will continue running. A user can reconnect his 'old' application by logging in again and typing only two commands.)

The distributed method described turned out to be very flexible. However, doing request-scheduling without knowledge of the later mapping-decision leads to weak waiting-time estimations and thus unpleasant user-reactions. How this problem can be alleviated without sacrificing the architecture-independent approach is shown in (4.2).

3 Architecture-Independent Request-Scheduling

As all MPP-systems of the PC^2 are at least variably partitionable, there is no doubt that in the whole context request-scheduling plays a major role. Similar to Intel's iPSC and Paragon, or Cray's T3D, the Parsytec GC-systems are partitioned into disjoint sets of processors. Afterwards, a parallel job is executed in the physically separated regions of the systems. Thus, each parallel program comes along with an (explicit) request for hardware resources. Furthermore, the maximum duration for which an application is allowed to occupy its resources is given.
These requests are equal for batch and interactive applications. In addition, the interactive one needs a (virtual) terminal for keyboard input and screen output. Both types of applications are strongly competitive. While for batch applications the overall throughput is the first priority, users working interactively count the minutes until their hardware-requests are fulfilled. As a user is already satisfied when he can start working, issues like the total system throughput or the average response-time are less valuable. As the completion-time is subject to change, especially in interactive user-sessions, the average waiting-time seems to be much more important.

3.1 The Scheduling Model

Hundreds of papers have been written about job scheduling in parallel systems (see [6]). However, there is a large discrepancy between what is studied and what can be applied in practise. This discrepancy is even greater to what is delivered by the MPP-vendors. Thus, more sophisticated methods were developed by supercomputing centers serving large MPP systems to a broad user community. For example, an NQS based scheduler for the Intel Paragon was developed by the San Diego Supercomputer Center [20]. A more flexible approach was developed by the Cornell Theory Center to schedule their 512 node IBM SP [17].
In this section we will now motivate a request-scheduling system which alleviates some of the dif-

ficulties mentioned. This method has already proved successful in daily operation. A more detailed description is presented in [15].

The scheduling model we are using for the simulation purpose is very close to the constraints described in the previous sections. We assume that the virtual machine-room consists of a number of MPP-systems with similar characteristics. Each is composed of processing nodes of a certain type and a number of user entries. All resource-requests are dedicated to exclusive usage and are limited by the time a virtual hardware environment can be occupied by the application program.
The resulting request-scheduling problem can now be conceived as an n-dimensional bin-packing problem. One dimension corresponds to the continuous time flow and $(n\text{-}1)$-dimensions are representing general system characteristics (e.g. the number of processors of a certain type or the number of user-entries). As a high-level decision mechanism must be provided at the QM-D to determine whether a request can ever be fulfilled, we are restricted to level-oriented scheduling algorithms (see [15]). If the first request at a scheduling-point (which corresponds to a new packing-level) is rejected by the MM-Ds, it can never be fulfilled and thus must be removed. To keep the model manageable, only the expected occupation-time and the number of requested processors are considered. All other (architecture-dependent) properties are on the mapping-modules of the MM-Ds. Since the scheduler has the view of ideal MPP-systems it may happen that single machines are over-booked. Thus, an MM-D will reject one or more requests, because they can not be configured any more. Now, it is up to the protocol between the QM-D and the MM-D to solve this conflict.

While doing this, four aims have to be considered: A priority mechanism must be provided, fairness must be guaranteed, the average waiting-time must be small, and the total utilization high. It is up to the scheduler to optimize the latter two. Handling priorities is mainly a question of the waiting-room organization. Fairness in this context means that no resource-request should pend forever. Furthermore, it must be possible to give an estimation of the waiting-time for each pending request. This topic interferes with both the waiting-room and the scheduling-module. Thus, both parts are integrated into the QM-D, whereby the algorithms of the scheduler can be switched by an operator or automatically, as it is done by the Implicit Voting System IVS (3.3). The basic structure of the QM-D is shown in Fig. 3.

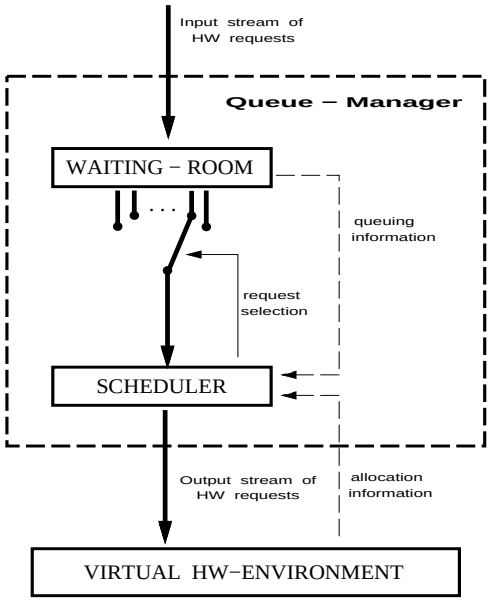

Figure 3: Structure of the QM-D

The waiting-room is organized in n priority-queues. If the scheduler runs out of work a finite set of pending requests is extracted from the waiting-room. The number of requests to be extracted from each queue ($E_x(p_i) := \left\lceil \frac{N(p_i) * E(p_i)}{E_m} \right\rceil$) depends on the extraction-order assigned to each queue by the administrator ($E(p_i)$) and its current filling-degree ($N(p_i)$). It is relative to the filling-degree of other priority-classes ($E_m = \max\{1, \max\{E(p_i) \mid N(p_i) \neq 0;\ 0 \leq i \leq n\}\}$). Using this formula, fairness is guaranteed for the priority-queues of the waiting-room. If fairness can be guaranteed by the scheduler as well (that means even if the scheduler has to handle temporarily rejected requests, non of those requests can be shifted forever), the whole system will behave fairly. If the scheduling algorithm used behaves deterministically, if no new requests of higher priority overtake others in the meantime, if the estimated occupation-times are accurate, and if there is an ideal hardware environment, then we can directly calculate the waiting-time for each request.

However, practise looks somewhat different. Thus, we can only estimate the expected waiting-times. The problem of overtaking within the waiting-room can not be avoided because we have an online system without knowledge of the future. The time-limit given with each resource-request can be treated as an upper bound. But it may happen that an interactively working user will release his resources earlier than

promised. If now the management software keeps that partition idle, the time estimations are still valid. We prefer to use this free resources again, which however can result in shorter waiting-times than previously told. Thus, the error-values of the predicted waiting-time (see 4.1 and 4.2) will become negative. In practise this can be alleviated up to a certain degree by also accounting for reserved but unused resources. During the simulations this situation does not occur, as we are using the accounting data of our center as input sequence.
Thus, in our scheduling model we consider a real hardware environment, not known in detail, a waiting-room within which new requests with higher priority can overtake others, together with a deterministic scheduling algorithm.

3.2 Basic Scheduling Primitives

Having introduced the combined priority and queuing scheme, we now concentrate on optimization aspects. Due to the proposed structure, we can use the advantage that the number of requests extracted from the waiting-room is always finite. First of all, let us assume that the MPP-systems in question are not saturated, that there are pending requests, and that the old schedule is done. In this case, most or all of the requests of the subset can be configured immediately. There is no need for sophisticated computations. Passing on the requests is the best we can do. Thus, from the user's view the whole QM-D should behave transparently. This mode is simply called the *First-Come-First-Serve* (FCFS) mode of the scheduler.

From now on, we deal with the case where the MPP-systems are saturated up to a certain degree, there are pending requests, the old schedule is done, and a finite set of pending requests was extracted from the waiting-room. Furthermore, we assume that this set is ordered by the corresponding upper bound of time each hardware request is allowed to occupy its resources. We will refer to this ordered set by the term *request-list* (RL). Each level of a bin-packing algorithm indicates a situation where at a point of time the whole system is empty. This is the case with all vertical lines on the time-axis and if these lines do not cross any rectangle (Fig. 5 – 6). Such a level is called a scheduling-point. If the first (requeued) request can not be configured at a scheduling point, it will never be fulfilled at all and must be removed. By spreading scheduling points into each schedule we can handle temporarily rejected requests resulting from a real hardware environment. At each scheduling-point, at least one decision can be made. Thus the scheduler behaves fairly, too.

The *First-Fit-Decreasing-Height* (FFDH) algorithm [3] computes a sequence of scheduling levels by working through the RL in non-increasing order. All rectangles are placed with their left side resting at one of the scheduling points. The first level is simply the bottom of the bin. At any point of the packing sequence, the next rectangle to be packed is placed on the top of the lowest level on which it will fit, justified to its left border. If none of the current levels will accommodate this rectangle, a new level is created.

To illustrate the behavior of the basic scheduling primitives, we will use the following example. This example is chosen as a wild mixture, ranging from large and short-running to small and long-running requests. Thus, it corresponds to the observed user behavior. Let i denote the request-number as it is inserted into

i	1	2	3	4	5	6	7	8	9	10
n_i	6.25	100	6.25	100	12.5	50	12.5	50	25	25
t_i	25	50	10	5	20	40	20	10	15	30

Figure 4: Sample request sequence

the waiting-room and n_i (t_i) being the corresponding percentage of the processors (number of time-units) requested. Ordering the requests by the non-increasing time results in $RL = (2, 6, 10, 1, 5, 7, 9, 3, 8, 4)$. Applying the FFDH-algorithm to the example of Fig. 4 results in the schedule shown in Fig. 5 with a total system utilization of

$$UTL_{FFDH}(RL) = \frac{\sum_{i=1}^{N} n_i * t_i}{FFDH(RL)} = \frac{9843.75}{115} = 85.6\,\%$$

and an average waiting-time of

$$AWT_{FFDH}(RL) = \frac{1}{N} * \sum_{i=1}^{N} \triangle(i) = \frac{1}{10} * 630 = 63.0\,\mathrm{m},$$

with $\triangle(i)$ denoting the absolute waiting-time of request i, and N the length of RL.

If there are two contradictory criteria to be optimized (UTL vs. AWT), it is quite often a good idea to turn things around. Thus, the request list was considered in non-decreasing order. As this modified algorithm increases the height of the bin, it is called *First-Fit-Increasing-Height* (FFIH) algorithm [15]. Applying the FFIH algorithm to the modified list results in packing short rectangles close to the bottom of the bin, and broad rectangles at relatively high x-values. Thus, we get

$$UTL_{FFIH}(RL) = \frac{9843.75}{115} = 85.6\,\%$$

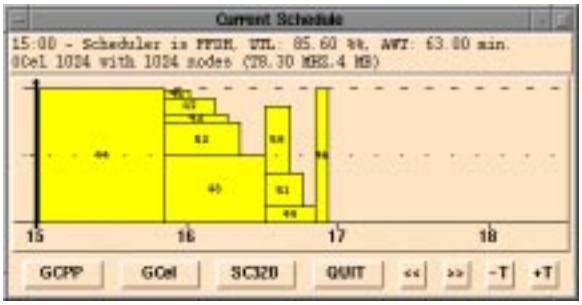

Figure 5: FFDH packing

and

$$AWT_{FFIH}(RL) = \frac{1}{10} * 185 = 18.5\,\mathrm{m}.$$

The resulting schedule is shown in Fig. 6.

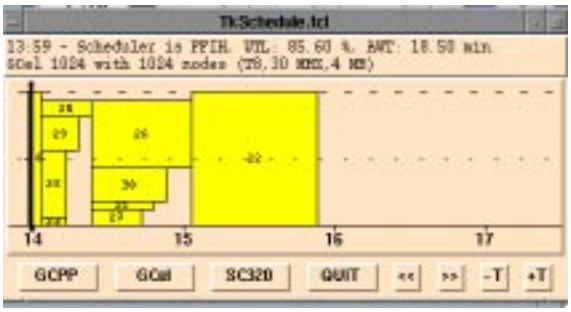

Figure 6: FFIH packing

All algorithms presented so far can be improved without violating the scheduling-point condition. At first, we consider the FFDH algorithm. Assume that we are working through the RL in non-increasing order and that we are computing a location within the bin to place an arbitrary request. Now we abolish the condition that each rectangle must be placed conclusive to the leftmost scheduling point. We allow an arbitrary placement on top of an already placed request without exceeding the scheduling point at the right side, determined by the request which is placed at the lowest y-value. An example of applying this modified algorithm, now called FFDH*, is shown in Fig. 7. This results in

$$UTL_{FFDH^\star}(RL) = 89.49\,\%$$

and

$$AWT_{FFDH^\star}(RL) = 60.5\,\mathrm{m}.$$

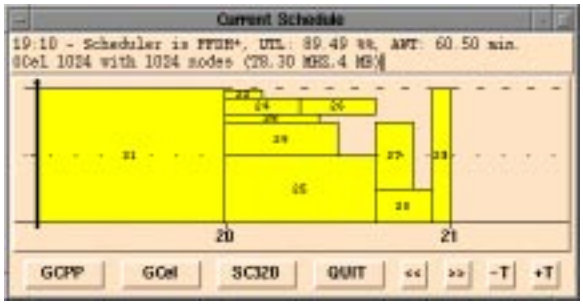

Figure 7: FFDH* packing

By applying the same idea to FFIH, (called FFIH*), we can now allow short-running requests to be enqueued dynamically. When preferring these requests to fill the gaps without violating the scheduling points, the average waiting-time and the overall utilization is improved simultaneously. This idea scales to the FCFS algorithm, too. If a new resource request

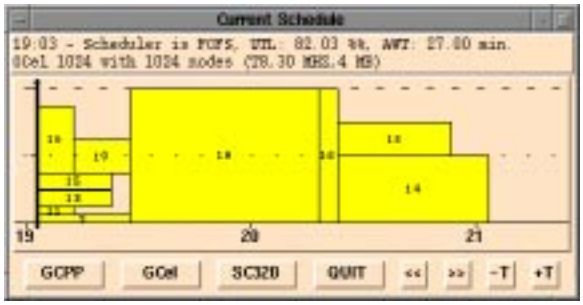

Figure 8: FCFS* packing

has been enqueued into the waiting-room, it is first checked if this request fits into a gap of the FCFS-schedule. In this case, the request to be planned is allowed to skip all pending requests within the waiting-room. This modification is called FCFS*. An example of applying the FCFS* algorithm results in

$$UTL_{FCFS^\star}(RL) = 82.03\,\%$$

and

$$AWT_{FCFS^\star}(RL) = 27.0\,\mathrm{m}$$

(see Fig. 8). However, note that the FCFS* algorithm violates the scheduling-point condition. Thus, when using this algorithm directly, fairness can not be guaranteed anymore.

```
IVS()
    BEGIN LOOP
      CASE
        (the MPP-systems are not saturated)
            DO switch to the FCFS*-mode and configure immediately OD ;  BREAK
        IF (most of the relevant requests were submitted for batch jobs)
            DO switch to the FFDH* algorithm to improve the overall system utilization OD ;  BREAK
        IF (most of the relevant requests submitted are for interactive usage)
            DO switch to the FFIH* algorithm to reduce the average waiting time OD ;  BREAK
    END LOOP
```

Figure 9: Outline of the Implicit Voting System

3.3 The Implicit Voting System

Commonly used scheduling and queuing systems are in general very static. Queues are dedicated to physical parts of the machinery or schedulers rely on time sharing operating-systems running on the resources to be scheduled. Switching between batch and interactive modes is done at a fixed time every day. These custom solutions provide specific capabilities at best, but are not suitable to solve the problem we address. Due to the reasons discussed above, we derive that the users themselves (i.e. the most relevant resource requests of the users) should vote dynamically on the characteristics of their favored resource scheduling method. However, they should not vote explicitly. Thus an Implicit Voting System (IVS) was developed to schedule the MPP-systems of a virtual machine-room [15].

The main pre-condition to build such a system is that the set of the requests in question and their main properties can be determined easily and in advance. Using the approach presented, this condition is fulfilled by the scheduling model of (3.1) and the request-list RL. Thus, the basic scheduling algorithms can be dynamically turned by the current mixture of the requests to tradeoff utilization versus response-time.
An outline of IVS is given in Fig. 9. Using the accounting data gathered at the PC^2, IVS was compared to the original FFDH algorithm in [15]. It was shown that using IVS, the system utilization could be improved up to 30 %, relatively to the FFDH scheduling. Simultaneously, the average waiting time could be reduced by 9 %.

We now focus on the methodology to get tight waiting-time estimations. Doing this, we are not fixed on a certain scheduling algorithm, as long as the constraints discussed above are satisfied.

4 Estimating the Expected Waiting-Time

Supercomputers usually show a very high degree of utilization. On machines without time-sharing support, some resource requests may not be fulfilled at once, but have to wait until the required hardware / software is available. Our experience has shown that these waiting-times can last from tens of minutes up to several hours on normal working-days. Since computing time on a supercomputer is very expensive, the availability of results from batch jobs or the start of an interactive parallel program are important events to the user. This is why users have to plan their day at least partially according to the estimated waiting-times of their resource requests, what makes these estimations a very critical parameter.

Unfortunately, there are several reasons why a scheduler can not predict the waiting-time exactly. First, the users' estimations about the expected execution times of their applications are not precise. As a scheduler has to rely on this information without any possibility of verification, we will assume for the rest of this paper that these estimations are correct. This is admissible, because users usually have to pay for their reserved time and are therefore encouraged to minimize the gap between their estimations and the actual execution time. Another reason for estimated waiting-times being inexact is the existence of different priorities. New requests can overtake others within the priority-queues and thus can be configured before requests for which a time

prediction was already made. As the previously estimated waiting-time in this case is too short, the resulting error-values are positive (Fig. 10). Finally, an architecture-independent scheduling algorithm has difficulties in finding tight waiting-time estimations due to its limited knowledge of the underlying hardware. The schedule as it is planned by the scheduler may not be valid because of hardware-dependent constraints like limited numbers of I/O-nodes or restricted partition shapes, which further increases the error-values.

In the following, we will show how a scheduler can overcome most of these problems in order to provide the user with waiting-time estimations as tight as possible.

4.1 Using autonomous scheduling algorithms

An autonomous scheduler is a scheduler that sends a request to be configured to the target machine without any knowledge whether they can be fulfilled or not. If, for example, the machine currently has a high internal fragmentation or all I/O-nodes are in use by other applications, the MM-D will have to reject the new request. Afterwards the QM-D has to requeue the rejected request until enough resources are available. If fairness is to be guaranteed, this request will have to be shifted to the next scheduling-point. The time-span for the pending requests within the scheduler (Fig. 3) gets lengthened and the overall throughput gets reduced. Fig. 10 depicts the normalized error-values of the waiting-time estimations that were calculated at the time a request enters the waiting-room. Fig. 11 shows these error-values for estimations made at the time a request enters the scheduler of Fig. 3.

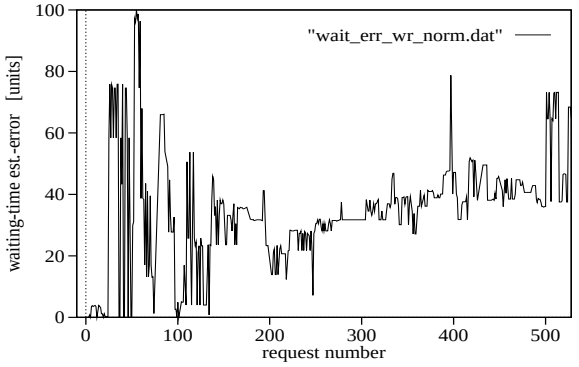

Figure 10: Estimation errors when entering the waiting-room (without verify-protocol)

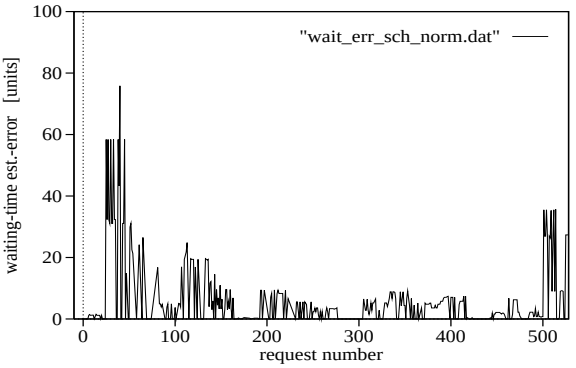

Figure 11: Estimation errors when entering the scheduler (without verify-protocol)

Calculating the expected waiting-time for requests already scheduled is straight forward. To determine the waiting-time for a request which is still in the waiting-room (the priority-queues), the system is virtually frozen and the waiting-room is successively cleared, using the formula in (3.1). Each portion extracted is virtually scheduled afterwards, using the IVS (3.3). Thus, we can also estimate the waiting-time for pending requests which are still in the waiting-room. It is obvious that errors resulting from reordering already scheduled requests have the largest impact on the first waiting-time estimation made (Fig. 10), which is also the most important one for the users.
By analyzing the simulation runs for the autonomous scheduling mode, we can see that the error-values resulting from the first 150 requests vary strongly (Fig. 10). Afterwards, the curve stabilizes at a relatively high level. The errors occurring inside the scheduler are of a much lower level. This is due to the fact that there are much more resource requests to be considered from within the waiting-room than from within the scheduler. Note that we have started our simulation with an empty system. Thus, for the first 20 requests nearly no wrong estimations were made.

4.2 Using a verification protocol

In order to overcome the problems of autonomous scheduling algorithms while still being hardware-independent, we introduce a verification protocol between the QM-D and the MM-Ds. The QM-D sends the preplanned schedule to the corresponding MM-D. Afterwards, the MM-D verifies whether this schedule can be configured in the given order on the given hardware or not. In case of conflicts, the MM-D reports the affected requests back to the QM-D and makes

a suggestion, for solving the conflicts. The scheduler can then reorganize its schedule according to these suggestions and send it back to the MM-D in order to be verified again. These steps will be iterated until there are no more conflicts within the current schedule. In this way, the predicted waiting-times become deterministic for each set of requests that has been taken out of the waiting-room and inserted into the scheduling-module (Fig. 3). Fig. 12 depicts the basic

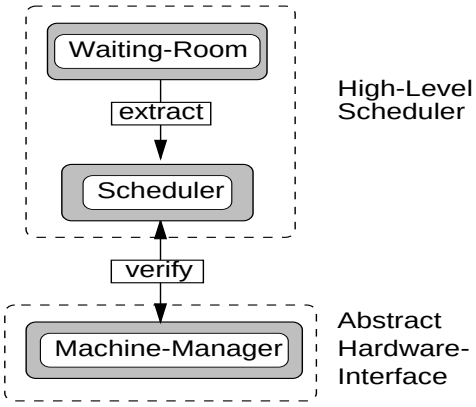

Figure 12: Scheduling single MPP-systems

structure of this verification protocol as it is currently implemented in CCS. Note that the protocol between the scheduler and the hardware-interface supports the mapping process of the MM-D as well as the scheduling decisions of the QM-D. Since the QM-D sends a complete set of requests to the MM-D for verification purposes, the mapping module of an MM-D can optimize the request placement according to this limited knowledge of the future. On the other hand, the QM-D can improve its abstract request-schedule by the hardware-related suggestions received from the MM-D. *Thus both, the mapping and the scheduling unit can benefit from this additional knowledge.*
Furthermore, the schedule is deterministic for those resource requests which have left the waiting-room. Therefore, it is possible to extract the next block of applications from the waiting-room when the preplanned time drops below a certain threshold. Of course, the newly extracted block must not be mixed with the applications that have already entered the scheduler. Otherwise, fairness could no longer be guaranteed. It should be noted that the priority mechanism may be weakened, if this threshold is very high. On the other hand, this gives the administrator a good mechanism to adjust the behavior of the management system according to his needs.

Fig. 13 depicts the normalized error-curve of the waiting-time estimations. The measurements were made when the requests were entering the waiting-room and the verification protocol was used. Comparing this diagram with Fig. 10, we can see that there are still some high deviations but most predictions were fairly accurate. Requests that leave the waiting-room and enter the scheduler can now rely on their predicted configuration-times. Thus, the waiting-time estimations made for verified resource requests are absolutely tight.

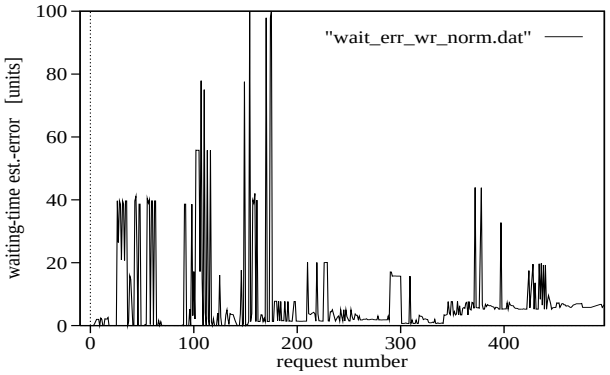

Figure 13: Estimation errors when entering the waiting-room (with verify-protocol)

In practise, it is very important that this protocol enables CCS to manage reservations, too. Daily use has shown that users frequently want to reserve parts of a dedicated machine for a fixed time. Maybe they want to do a presentation or maybe there is a deadline to be met. The handshake between QM-D and MM-D allows the scheduler to use the remaining parts of the machine and be still able to guarantee that the reservation can be fulfilled at the desired time. CCS contains a graphical tool to display the verified part of the schedule. This tool enables the users to find out when their requests will be configured and if there are free slots in the current schedule. This encourages the users to submit requests that will fit into free slots of the schedule and thereby to increase the overall throughput.
Another important property is that the scheduling / mapping / verifying activities do not result in additional runtime-overhead for application programs. Only the startup-time of an application may increase by a few seconds. This is because the CCS software is running somewhere in front of the MPP-systems to be managed and not on the compute-nodes themselves.

4.3 Results

Sections (4.1) and (4.2) have shown that the user can more heavily rely on waiting-time estimations, if the high-level scheduler performs a verification handshake with the hardware interface. Thus, the next step is to find out exactly how much better the predictions will be if the extended scheduler is used. In order to do this, there are two parameters to be examined. The first is the frequency of significant errors and the second is the average difference between the predicted and real configuration times. Therefore, we have analyzed the behavior of both schedulers with simulation series of more than 500 resource requests each. The frequencies of the different error-classes are depicted in Fig. 14 and Fig. 15.

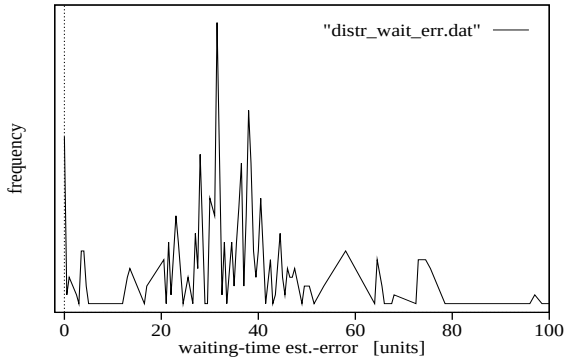

Figure 14: Error distribution (without verify-protocol)

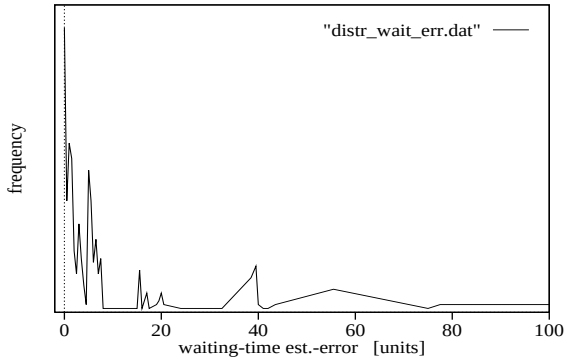

Figure 15: Error distribution (with verify-protocol)

Without the verification protocol, most of the differences are located around 35 % of the maximum error. In contrast, the extended scheduler has nearly the same maximum error but is able to place most of its predictions within a range of 10 % of the maximum error. Thus, there are only few significant errors if the extended scheduler is used.

There are two reasons for estimation-errors to occur, even if the verification protocol is applied. The first one is because we have to schedule an online system. Thus, new enqueued requests with a higher priority can overtake others and therefore interfere with the predictions already made.

The second one is because only the calculated schedule for the requests inside the scheduler (Fig. 3) is verified. Schedules resulting from cleaning the virtually frozen waiting-room, in order to estimate the waiting-time just after request-submission are treated as within the autonomous scheduling mode (see 4.1). Therefore, the predictions made get worse with increasing ratio between the number of pending requests within the waiting-room and those which are already inside the scheduler. Of course one could verify the schedules for all requests in the whole system at any time. In that case, however, we have to pay for the tighter time-values by a significantly increased computation time for the verification task. The compromise we have chosen by only verifying the schedule for requests which are already inside the scheduler can be adapted to the administrator's needs by simply changing the extraction order $E(p_i)$ for the priority-queues (see 3.1).

In the further evaluation of the verification methodology, we have calculated the average difference between predicted and real configuration times. Again, we used a sample of more than 500 requests for both schedulers. For users working interactively the waiting-time estimations become of particular interest, when a request is submitted. Therefore, we have examined the data presented in Fig. 10 and Fig. 13. The normalized mean error-values

$$EV = \frac{100}{N * E_{max}} * \sum_{i=1}^{N} \mid t_{configured}(i) - t_{estimated}(i) \mid$$

with

$$E_{max} = \max\{ \mid t_{configured}(i) - t_{estimated}(i) \mid ; 1 \leq i \leq N \}$$

for all requests were determined and compared. Using the autonomous scheduling approach, a value of $EV = 34.923$ was calculated. This was decreased to $EV = 6.428$ by the verification protocol. Thus, we were able to reduce the uncertainty by 76 % while still maintaining the hardware independence of the algorithm.

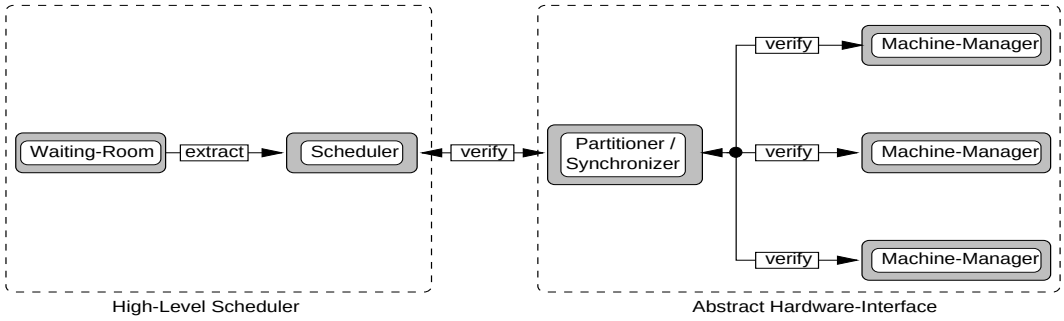

Figure 16: Scheduler layout for a metacomputer

5 Towards the Metacomputer Challenge

Metacomputing in its ideal form involves spreading a single application across several HPC systems, allowing a heterogeneous collection of computers to work in concert on a single problem [10, 18]. However, intermediate steps in metacomputer evolution, e.g. using a number of different MPP-systems of a distributed virtual machine-room to achieve a global load-alliance, seem to be more realistic for most applications [12]. Activities are already in process to determine whether such ideas can be transferred into practise and how that might work [1, 5, 13, 19].

Building a metacomputer is probably one of the greatest challenges to computer scientists. The required hardware has been available for several years, but we are still far away from a really working metacomputer. In the course of the last years, the WAN technology has made rapid progress. Encouraging global file systems (e.g. AFS, DFS) are emerging. Thus, some important preconditions are fulfilled to establish a general purpose metacomputer. What is needed now is a resource-management layer which can be scaled to these new challenges. By comparing the requirements for such an environment to those of a monolithic HPC system, several additional problems have to be solved. For example, synchronized scheduling and mapping methods using the heterogeneous and architectural possibilities have to be developed. Suitable methods to divide an abstract user-request into subrequests and to select the best suited systems from a virtual machine-room afterwards, while taking the WAN performance into account, are required [9]. Typical applications for a heterogeneous machine-pool will need to have each of their modules running on the best suited computer-system. Some modules may require a vector computer and others may perform best on an SMP machine. A metacomputer scheduler receiving such a request must be able to guarantee that all these modules will be running at the same time. This must of course not decrease the overall throughput to intolerable values.

The CCS approach presented offers a simple and user-friendly interface to the MPP-resources of a service provider. Due to the high-level view of the physical resources, a framework was drawn up to establish a virtual machine-room locally, thus giving a similar shape to various HPC sides.
Fig. 16 shows how the layout of the CCS-scheduler can be extended to meet some of the additional requirements.
The high-level scheduler remains the same as in the single computer case depicted in Fig. 12. The abstract hardware interface now consists of several different MM-Ds, one for each system in the virtual machine-room. The scheduler indirectly communicates with these MM-Ds via a new module which performs the partitioning and synchronizing task. When the scheduler receives a request that consists of several subrequests that can – or perhaps even must – be executed on different machines, it is sent to the partitioner just like any other request. The partitioner notices that all subrequests have to be fulfilled synchronously. It chooses the best-suited machines and sends the subrequests to these machines for verification. The required multi-agent structure is very similar to the structure depicted in Fig. 2, if the partitioning and synchronizing task is assigned to the PM-D, too. The protocol used now is similar to the one between the high-level scheduler and the abstract hardware-interface. The MM-Ds report to the partitioner that the request can be fulfilled or make a suggestion when this will be possible.
The partitioner iterates these steps until it has found the right machines and a suitable time-slot for the synchronous requests. Then, it reports to the scheduler that the schedule is valid or it presents the alternative suggestion. The scheduler itself can later use this additional information as described in (4.2).

Despite the topics purely related to scheduling, there are other important requirements a resource management system has to meet in order to be suitable for a metacomputing environment. For example, there have to be well defined access points for third party products. A metacomputer built from supercomputers of many different vendors must have its own vendor-independent runtime and development tools like debuggers, performance analyzers or load-balancers. Therefore, it is vitally important that the management system provides standardized access points for these tools. However, it takes knowledge about the requirements of these tools to establish the interfaces in a useful way. Thus, computer scientist working on different areas of research have to combine their knowledge in order to build a metacomputer. We think that the design of CCS is a good focal point for these activities.

Acknowledgment

We would like to thank everybody involved in the CCS project. Especially Christian Hellmann for implementing the scheduling strategies and for running the simulations. Finally, we thank the anonymous referees for their helpful comments that improved the presentation.

References

[1] A. Bachem, B. Monien, F. Ramme : *Der Forschungsverbund NRW-Metacomputing "Verteiltes Höchstleistungsrechnen",* Technical Report, Paderborn, 1996

[2] M. Campione, K. Walrath : *The Java Language Tutorial: Object-Oriented Programming for the Internet,* ISBN 0-201-63454-6, expected July 1996

[3] E.G. Coffman, M.R. Garey, D.S. Johnson, R.E. Tarjan : *Performance bounds for level-oriented two-dimensional packing algorithms,* SIAM J.Comput., Vol. 9, No. 4, pp. 808-826, Nov. 1980

[4] A. Colbrook, M. Lemke, H. Mierendorff, K. Stüben, C.A. Thole, O. Thomas : *EUROPORT – ESPRIT European Porting Projects,* Int. Conf. on High-Performance Computing and Networking, Proc. of the HPCN Europe, Springer-Verlag 1994, LNCS No. 796, Vol. I, pp. 46-54

[5] E=MC^2 Consortium c/o R. McConnell : *The European Meta Computer Utilizing Integrated Broadband Communications (E=MC^2) Project,* Int. Conf. on High-Performance Computing and Networking, Proc. of the HPCN Europe, LNCS, Springer-Verlag 1995 pp. 54-59

[6] D.G. Feitelson : *A Survey of Scheduling in Multiprogrammed Parallel Systems,* Research Report RC 19790 (87657), IBM T.J. Watson Research Center, Oct. 1994

[7] D.G. Feitelson, L. Rudolph : *Toward Convergence in Job Schedulers for Parallel Supercomputers,* In IPPS'96 Workshop on Job Scheduling Strategies for Parallel Processing, April 1996

[8] R. Funke, R. Lüling, B. Monien, F. Lücking, H. Blanke-Bohne : *An optimized reconfigurable architecture for Transputer networks,* Proc. of 25th Hawaii Int. Conf. on System Sciences (HICSS 92), Vol. 1, pp. 237-245

[9] J. Gehring, A. Reinefeld : *MARS – A Framework for Minimizing the Job Execution Time in a Metacomputing Environment,* To appear in spring issue of FGCS 1996

[10] A.S. Grimshaw, J.B. Weissman, E.A. West, E.C. Loyot : *Metasystems: An Approach Combining Parallel Processing and Heterogeneous Distributed Computing Systems,* Journal of Parallel and Distributed Computing, Vol. 21, 1994, pp. 257-270

[11] R.L. Henderson : *Job Scheduling Under the Portable Batch System,* IPPS Workshop on Job Scheduling Strategies for Parallel Processing, D.G. Feitelson and L. Rudolph (eds), Springer-Verlag 1995, LNCS No. 949, pp. 279-294

[12] A.A. Khokhar, V.K. Prasanna, M.E. Shaaban, Cho-Li Wang : *Heterogeneous Computing: Challenges and Oportunities,* IEEE Computer, Vol. 26, No. 6, 1993, pp. 18-27

[13] Reagan Moore : *NSF MetaCenter: A White Paper,* San Diego Supercomputing Center, 1995

[14] F. Ramme : *Building a Virtual Machine-Room – a Focal Point in Metacomputing,* Future Generation Computer Systems (FGCS), Elsevier Science B. V., Aug. 1995, Special Issue on HPCN, Vol. 11, pp. 477-489

[15] F. Ramme, K. Kremer : *Scheduling a Metacomputer by an Implicit Voting System,* 3rd IEEE Int. Symposium on High-Performance Distributed Computing, San Francisco, 1994, pp. 106-113

[16] F. Ramme, T. Römke, K. Kremer : *A Distributed Computing Center Software for the Efficient Use of Parallel Computer Systems,* Int. Conf. on High-Performance Computing and Networking, Proc. of the HPCN Europe, Springer-Verlag 1994, LNCS No. 797, Vol. II, pp. 129-136

[17] J. Skovira, W. Chan, H. Zhou, D. Lifka : *The EASY – LoadLeveler API Project,* In IPPS'96 Workshop on Job Scheduling Strategies for Parallel Processing, April 1996

[18] L. Smarr Ch. E. Catlett : *Metacomputing,* Communications of the ACM, Vol. 35, No. 6, June 1992, pp. 45-52

[19] *HIPERCON – High-Performance Computing Network –,* W. Zimmer (ed.), Eine Analyse zum Aufbau und Betrieb eines Höchstleistungsrechnerverbundnetzes in der Bundesrepublik Deutschland, GMD-First, Berlin 1995, im Auftrag des BMBF

[20] M. Wan, R. Moore, G. Kremenek, K. Steube : *A Batch Scheduler for the Intel Paragon with a Non-contiguous Node Allocation Algorithm,* In IPPS'96 Workshop on Job Scheduling Strategies for Parallel Processing, April 1996