Reduced to the MAX - A Simplified Resource-Management System for large Workstation-Pools

Achim Streit[1]

Paderborn Center for Parallel Computing,
University of Paderborn,
33095 Paderborn, Germany
streit@uni-paderborn.de

Abstract. In this paper, we present a concept for a lean resource-management system used in heterogeneous pools of workstations. A reduced infrastructure in combination with improved scheduling concepts leads to an easy-to-use, highly-available, fast and effective system. This is achieved by using a simplified architecture and new scheduling algorithms, mapping the monitored information-data to mathematical functions and merging existing technologies in one product.

1 Introduction

In recent years the number of workstations in educational and economical branches have increased. Reasons for that are increasing peak performances and decreasing prices of all sizes of workstations. The grown infrastructure leads to a variety of different hardware-vendors, operating-systems, CPU speeds and sizes of memory. In fact older workstations are still fast enough for certain applications. Generally a user can login to any of the available workstations, but primary works on a specific one. Generally workstations are used as desktop-computers and only a few of them as compute servers residing in an air-conditioned room.

Thus an unbalanced usage of the computational power takes place. Some users are unsatisfied with long execution times of their processes evoked by other users working on the same workstations. They are unaware that other workstations are idle at the same time. If they start their processes on different workstations they spent less time waiting for the end of a process. For example, two users compile on the same workstation A at the same time. If one compile process is started on workstation B, both compile processes end faster. This saves time and therefore costs.

Avoiding this situation an intelligent resource-management system has to be installed. To achieve a distinct benefit, all available workstations have to be managed by the system, otherwise only suboptimal improvement is reached. The management system has to suffice certain restrictions. In general it must be reliable and fast, by providing only a small amount of overhead for balancing the processes on different machine instead of leaving all processes on one machine in unbalanced state.

The necessary software-infrastructure for the resource-management system is kept very simple to avoid possible errors, unavailability and to decrease development costs. Thereby the scheduling module of the system gets more important in order to achieve better performance. This paper invents a concept for the development of similar resource-management systems and is seen as a white paper.

2 Why another resource-management system ?

Recently many research projects connected geographically distributed supercomputers to one big virtual supercomputer called metacomputer ([7], [10]). This optimization took place at the upper end of computing trying to increase the number of FLOPS (floating operations per second). This is often called HPC (High Performance Computing). Generally the peak performance was increased by delivering a high amount of computational power over a short time. Usually users need computational power over longer periods of time. Therefore the term HTC (High Throughput Computing) was invented.

Generally it was tried to run applications, with a high demand on computing power, faster by a shared usage of workstations which were used for everyday work. As an example Condor ([8], [11]) tries to utilize workstations which are currently not used by their owners (e.g. during the night or the coffee break). If the user gets back to the desk and starts to work on the machine, Condor checkpoints its jobs and tries to run them on other available workstations. If no other workstation can be found the job is stored in a queue until an idle machine is found. This concept works fine with batch jobs, but if a user interacts with a job by e.g. constantly producing keyboard input, it is not acceptable that a job may reside in a queue waiting for execution. Therefore the resource-management system guarantees that a job is only checkpointed for migration, if another workstation is found where the job can be restarted. Furthermore a job has to be started immediately after submission to the system without any waiting without any waiting in a queue for an appropriate workstation.

The following aspects represent the main development goals of the new resource-management system:

– simplicity of the infrastructure: the usage of standard mechanisms prevents of possible errors, bottlenecks and unexpected development problems.
– information storage based on mathematical functions: the huge amount of monitored informations of each workstation is stored by using mathematical functions with adjustable parameters. Newly monitored information values are used to improve the settings. Reduced complexity of the information storage system is gained.
– merging of innovations: several innovations were invented by different research projects, like e.g. improved scheduling algorithms and infrastructures,

accounting systems, resource brokerage or the ability of advanced reservation. The combination of all features leads to a product with outstanding advantages and thereby a more efficient utilization of available resources.

3 Architecture Elements

Some general elements of the system are described in the following. An information retrieval mechanism and a scheduler is necessary to provide the basic functionality of the resource-management system. Other tools for easier usage, administration or visualization are described later.

For the information retrieval a master-slave paradigm is used. A Monitor-Slave runs on every workstation and fetches all necessary information with a distinction between static and dynamic. Static data needs to be retrieved exactly once. The hostname and IP-number are two possible static values as well as the number of processors installed or the total size of physical memory. Generally these values never change.

Of more importance are dynamic values, which describe the recent performance of the monitored workstation. Possible values are the current percentage of CPU and memory usage, the amount of free harddisk-space, the available network bandwidth, or the number of processes or logged-on users. A major problem of dynamic information values is their actuality. If the monitored values are up to date at every time, the interval for monitoring the information is infinitesimal small. A high amount of processor usage of the monitor-slave comes along with that, but the monitor-slave itself shouldn't effect the performance of the workstation noticeable. The solution is a bigger time-out between to updates of monitored information values.

On the other side of the information retrieval process the monitor-master represents a bottleneck. At a specific number of monitor-slaves and size of the monitoring interval, the monitor-master can not handle all received data. The available network bandwidth is a problem as well as the speed of information processing. Therefore one solution is the decentralization of the monitor-master. Specific groups of monitor-slaves transmit their data to monitor-masters distributed on different workstations. The easiest way to realize a distribution is the usage of OS-dependent monitor-masters (see figure 1). A further distribution is done by the OS-revision or the number of processors, if needed.

The monitor master itself has to store the information values, so they can be used later by the scheduler and other tools. One possible way is to store all information in a classic database or in separate files, but this leads to a huge amount of data to put in storage. To store the information with mathematical methods is a more convenient way. Either the use of statistical methods to analyze the data is possible as well as a fixed mathematical function with a variety of parameters. New information values are then used to readjust the already computed parameters to make them more exact. A further benefit is the predictability of performance for the future with a specific probability. This knowledge can be used by the scheduler to optimize the decisions. Possible day/night behaviour is

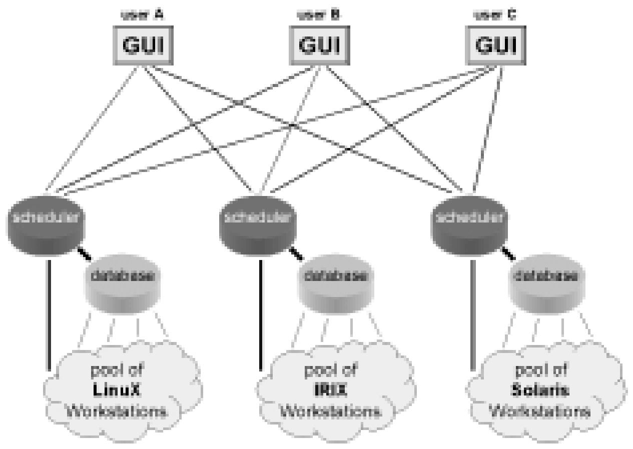
user A
GUI
user B
GUI
user C
GUI
scheduler
scheduler
scheduler
database
database
database
pool of LinuX Workstations
pool of IRIX Workstations
pool of Solaris Workstations

Fig. 1. resource-management architecture

detected and used for a better utilization of the hole system (e.g. running batch process at night).

The scheduler bases its decisions on the stored informations from the monitoring system. A detailed description of the functionality of the scheduler is given later (see Section 4). The job-start is realized by using standard UNIX commands for remote job starting (e.g. RSH). Therefore it is necessary that users can access all available workstations. A global user management and file-system is necessary, but this is common in most cases (e.g. by using NFS). To provide security other standard applications like SSH can be used instead (Note: this may influence the available network bandwidth). It is implied that the system is only used for internal usage and thereby certain security-mechanisms are provided automatically, like the usage of firewalls and gateways, which is common in most future operational areas. The advantage of using standard UNIX applications is the simplicity and the aspect, that no further development is necessary (ready-to-use).

If only batch jobs are used, which retrieve their input from files and also write their output to files, no more mechanisms are necessary. In the case that jobs with user I/O are used, mechanisms are used to redirect this I/O to the user. Generally the user works in front of a computer with a graphical user interface and therefore uses processes with a GUI. The easiest way of redirecting graphical I/O is the usage of the UNIX-shell variable *DISPLAY*, which has to be set appropriately. The non-graphical output of processes can be redirected to log-files which then can be viewed by the user at anytime. A different approach is the usage of X-Terminals for the text-based I/O. Common to all is the necessity of local X-Servers running on the users desktop computer.

Another important aspect is the description of resources which should be managed by the system. For easier administration of the system, a resource description language is used. Different solutions have been developed so far ([1]), but for more standardization XML ([5]) is chosen. Thereby new defined tags are used to describe workstations of different manner. Tags come with attributes which are then used to represent certain workstations. An example could look like the following:

```
<SUN hostname="oscar" IP="91.235.83.207" OS="Solaris 7" />
```

The resource description language can also be used for specifying some static information values like speed indices. Thereby it is made possible to classify available workstations by the computational power of the processors.

4 Scheduling

The structure of the resource-management system should be invented in a way, that different scheduling algorithms can be used. Therefore a scheduler interface makes it possible to use new and different scheduling modules. This makes it easier to develop and test new scheduling algorithms adjusted to specific objectives of the workstations-owner. Due to the fact that the effectiveness of the system is directly depending on the algorithm and efficiency of the scheduler, the scheduler is the most important part of the hole system.

One exemplary approach for developing an efficient scheduler is the use of an objective function. Static information values like the available network bandwidth and latency are used, as well as speed indices for classifying the workstations. Dynamic information values like the CPU and memory usage are seen as the basics for using an objective function.

The single steps of the scheduling process are:

1. process submission: The user specifies on with sort of workstations the process should run (e.g. SUN workstation with OS-revision 5.6, or all workstations with a speed-index more than x). Furthermore starting parameters can be specified which are used by the process.
2. subset composing: The scheduler uses the named restrictions from the first phase two sort out all workstations which can not fulfill the demands of the process specified by the user (e.g. type of operating system). Generally the quantity of eligible workstation is reduced here significantly.
3. objective value computing: At this point the scheduler uses the objective function to compute an objective value for every workstation from the subset.
4. decision phase: The workstation with the smallest objective value is picked.

After a decision is made, the scheduler starts the process with all necessary startup-parameters on the chosen workstation by using standard mechanisms.

It can be thought of other scheduling principles. By gathering more information about the running processes on every workstation in conjunction with the owner of the process, mechanisms can be used to foresee the possible execution time and demands of a process. This is used in the scheduling algorithm to produce more effective solutions. Another way of foresight is done, if the user specifies an estimated execution time of the process.

The use of a centralized scheduler implies the risk of possible bottlenecks, because the scheduler can run in a time-out due to a network error or an overload. Two solutions are possible to prevent this situation. At first a back-up (slave-)scheduler is used, which works in parallel to the master-scheduler. The slave-scheduler regularly checks if the master-scheduler is still running and comes to correct decisions in time. If not the slave-scheduler becomes the master-scheduler and resumes the scheduling. The old master-scheduler then is restarted possibly on a different workstation and used as the slave-scheduler in the future. This mechanism of two independent schedulers constitutes redundancy. For a faultless scheduling-process it is necessary that both scheduling processes run on different workstations to eliminate hardware errors. Another back-up scheduler increases the amount of safety and availability.

A second approach is the usage of a decentralized scheduling system. As it was done with the partitioning of the database subjected to the operating system an similar partitioning is done with the scheduler. One scheduler for each operating system (see figure 1). In conjunction with the information databases the risk of bottlenecks is reduced. The big advantage of OS-dependent partitioning is, that a specific process can only be executed on the operating system it was compiled for. The user has to specify the OS at the submision, which substantially reduces the amount of necessary computations in the scheduler.

5 Features

A graphical user interface is provided for easier access to the system. The GUI is independent of any operation-system, so users can start processes from any available computer. Java 2 ([2], [3], [4]) can provide the necessary functionality and is established in the branch of OS-independent graphical user interfaces. The major feature of the GUI is the submission of a process-command to the resource-management system. Therefore the user specifies the process-name and certain command-line parameters like it is done in a standard UNIX-shell. Furthermore the user decides which operating system is needed to run the process. A list of all available operating systems is provided by the GUI. Another list shows all available workstations in case the user wants to start the process on a specific workstation. Although then the scheduler isn't needed anymore, this option is provided for flexibility. The list of workstations include static and dynamic information values for every workstation like hostname, IP-address, size of memory, type of operating system, number of processors, logged-in users, processes, or usage of CPU and memory. Various filters and different orders are used for a more convenient presentation of all informations. The command for starting a job is submitted as a regular string to the resource-management-system. A conversion in special data-structures like an Abstract Job Object ([9]) is not necessary.

Basic OS-commands like ls, mkdir or cp are used via X-Terminal. In this case only the X-Terminal is scheduled to an appropriate workstation and all further processes are started from inside this X-Terminal.

As described earlier the infrastructure of the resource-management system should be implemented as easy as possible to prevent possible errors and to ensure reliability/availability, safety/security and redundancy especially at possible bottlenecks. A step towards an easier extensibility of the system can be guaranteed by providing a set of API's. As already mentioned a scheduler API is used for the development of other schedulers to provide a standard interface to the resource-management system, for retrieving informations from the system and for starting a process on a specific workstation. A second API is used for providing graphical user interface elements which can be integrated in other applications.

For a better utilization of all available workstations it is sometimes necessary to migrate jobs from on workstation to another. The core dump mechanism is used for that. After a process is interrupted the core dump can be transferred to another workstation where it is restarted. This implies that the operating systems on both workstations is the same. A migration from one workstation with e.g. Intel-Linux to a Sun-Solaris workstation cannot work due to e.g. incompatible byte ordering. The scheduler decides, if the costs of a migration are less than the possible benefit ([12]). Migrations are prevented, if the scheduler is capable to foresee possible conflict. The migration of a batch process with no user I/O is quite easy to achieve. However a process with possibly graphical user I/O makes a migration more complicated, because the I/O has to be redirected. Additionally the user can declare some processes as "not movable" if desired.

A feature which is not very common nowadays is advanced reservation. It allows users to reserve certain workstations for specific time-slots in the future. During these time-slots the users can switch the workstations in a sort of single-user mode, if desired. With this the user can block the workstation from other users. From the scheduler side of view this specific workstation is not usable for a definite time. This mechanism is a major availability problem, e.g. if a user blocks most of the workstations and other user can't work with them. In this case it is thought of limited advanced reservation for normal users by limiting the number of reserved workstations or the reserved time.

A full blown advanced reservation mechanism is used only for maintenance work. The time where the maintenance should take place is reserved by the system-administrator and the resource-management system guarantees that no user-processes are running on the specified workstations at that time which prevents dissatisfied users.

In conjunction with the advanced reservation system, an accounting system should be integrated. This allows external users to use the system. Furthermore different departments e.g. of a company can sell computing time to other departments. Therefore a better return of investment (ROI) can be achieved. In this case it is necessary that the scheduling algorithm supports accounting.

Am automatic process start mechanism is developed in similar to the UNIX cron/at commands. This gives the user the ability to start processes at certain times of the day, e.g. during night, in order to achieve better performance. Processes with high demand of computational power, e.g. simulations, can run during the night without interfering the daytime work of normal users. Therefore a better utilization of the workstation especially at night can be reached.

A separate tool for administration purposes is used e.g. to observe all available workstations and to produce usage-statistics ([6]). In fact these statistics might be used by some scheduling algorithms to provide an more optimal scheduling, if specific workstations are less utilized. The administration tool is also used to ease the work of the system-administrator by providing e.g. an easier way of logging-in, process-killing or rebooting single machines.

As an optional part the same GUI which is used to start processes on workstations is used to start parallel applications on supercomputers. Therefore interfaces to common supercomputer management software is provided. One may think of CCS ([6]) or UNICORE ([9]).

6 Conclusion

This paper presents a concept for a future resource-management system which leads to an effective usage of available workstations for the everyday work. The simplicity of the infrastructure prevents of useless overhead produced by the resource-management system and gains more effectiveness of all available resources. Mathematical functions for information storage and the merging of innovative ideas in various parts of existing resource-management system are the key elements of the presented concept.

References

1. M. Brune, J. Gehring, A. Keller, and A. Reinefeld. RSD - resource and service description. In *Proc. of the Intern. Conf. on High-Performance Computing Systems HPCS 98, Edmonton, Canada*. Springer-Verlag, 1998.
2. M. Campione, A. Huml, and K. Walrath. The Java Tutorial Continued: The Rest of the JDK. http://java.sun.com.
3. M. Campione and K. Walrath. The Java Tutorial Second Edition: Object-Oriented Programming for the Internet. http://java.sun.com.
4. M. Campione and K. Walrath. The JFC Swing Tutorial: A Guide to Constructing GUIs. http://java.sun.com.
5. D. Connolly. Extensible Markup Language (XML). http://www.w3c.org/xml.
6. A. Keller and A. Reinefeld. CCS resource management in networked HPC systems. In *Proceedings of 7th Heterogeneous Computing Workshop HCW'98 at IPPS, Orlando Florida*, pages 44–56. IEEE Comp. Society Press, 1998.
7. G. Lindahl, A. Grimshaw, A. Ferrari, and K. Holcomb. Metacomputing - what's in it for me? Technical report, University of Virginia, Computer Science Department, 1998.
8. M. Litzkow, M. Livny, and M. Mutka. Condor - A Hunter of Idle Workstations. In *Proceedings of the 8th Intl Conf. on Distributed Computing Systems*, pages 104–111, 1988.
9. M. Romberg. The UNICORE Architecture: Seamless Access to Distributed Resources. In *Proceedings of the 8th IEEE International Symposium on High Performance Distributed Computing,*, pages 287–293, 1999.

10. L. Smarr and C. Catlett. Metacomputing. *Communications of the ACM*, 35(6):44–52, June 1992.
11. The Condor Team. Condor Version 6.1.9 Manual. http://www.cs.wisc.edu/condor, November 1999.
12. G. D. van Albada, J. Clinckemaillie, A. H. L. Emmen, J. Gehring, O. Heinz, F. van der Linden, B. J. Overeinder, A. Reinefeld, and P. M. A. Sloot. Dynamite - blasting obstacles to parallel cluster computing. In P. M. A. Sloot, M. Bubak, A. G. Hoekstra, and L. O. Hertzberger, editors, *High-Performance Computing and Networking (HPCN Europe '99), Amsterdam, The Netherlands*, number 1593 in Lecture Notes in Computer Science, pages 300–310, Berlin, April 1999. Springer-Verlag.