In: Proc. of the 1999 Int. Conf. on Parallel and Distributed Processing Techniques and Applications, Las Vegas, 1999.

Batch Queueing in the WINNER Resource Management System

Olaf Arndt[1], Bernd Freisleben[1], Thilo Kielmann[2], Frank Thilo[1]
[1]Dept. of Electrical Engineering and Computer Science, University of Siegen, Germany
{arndt|freisleb|thilo}@informatik.uni-siegen.de
[2]Dept. of Computer Science, Vrije Universiteit, Amsterdam, The Netherlands
kielmann@cs.vu.nl

Abstract *Resource management systems (RMS) for networks of workstations exploit unused computing capacities by automatically assigning jobs to idle workstations. Until now, the* WINNER *RMS developed by our group only provides support for interactive jobs. In this paper, an approach to seamlessly integrate a queueing system for batch jobs is presented. The basic design decisions, the queueing system's functionality, and the benefits of its use within our workstation environment are described.*

Keywords: networks of workstations, resource management, batch queueing system

1 Introduction

Networks of workstations (NOWs) are by now ubiquitous general purpose computing platforms. Since console users of workstations hardly utilize the processing capabilities of their machines (e.g. while editing text, reading mail, browsing the web, or being physically absent), the idle time of workstations is often as high as 95% [10]. Simultaneously, there is high demand for computing power driven by applications from fields like simulation and optimization [11, 17].

Software packages that assign such compute intensive applications to idle workstations are known as *resource management systems* (RMS) [4]. Such systems have to cope with two conflicting objectives. First, interactive tasks demand *minimal completion time* to satisfy human users waiting in front of a workstation. Second, non-interactive background tasks (sequential as well as parallel batch jobs) demand *maximal throughput* to complete as many jobs as possible within given time constraints.

While interactive tasks have to be scheduled immediately on the currently best suited machine capable to service a given request, execution of batch jobs may be delayed to better meet these objectives. Whenever the set of pending batch jobs exceeds the available computational resources, jobs have to be executed in a coordinated manner to avoid performance penalties caused by interferences between multiple concurrent jobs. Coordinated execution of batch jobs is called *scheduling*. With scheduling systems, jobs, once submitted, are waiting in queues, while the scheduler selects jobs for execution as resources become available.

Existing RMS can be divided into two groups: First, there are batch processing systems with little or no support for interactive applications like the *BATCH system* [15], the *Generic Network Queuing System (GNQS)* [9], *Distributed Queuing System (DQS)* [8], *Condor* [11], and *CODINE* [5].

Second, some projects do provide support for both batch and interactive jobs. For example, GNU *Queue* [16] essentially replaces the *rsh* command by allowing both batch and interactive jobs to be started automatically on a particular set of machines. *LSF* [18] and *GLUnix* [7] both support interactive jobs as well as batch jobs, but treat them in a totally different manner.

Until now, the WINNER RMS as described in [1, 2] has concentrated on the transparent execution of interactive jobs. In this paper, the process of extending WINNER by a batch queueing system is described. The design decisions leading to a seamless integration of the concept of queues into the system are presented. Special emphasis is put on WINNER's extensibility to further kinds of jobs and platforms.

The paper is organized as follows. Section 2 outlines WINNER's overall design. In Section 3, the usefulness of a batch queueing system is motivated, before its integration into WINNER is presented. First experiences gained during the use of the system are described in Section 4. Section 5 concludes the paper and outlines areas of future research.

2 WINNER System Design

WINNER has been designed for typical Unix NOW environments, consisting of a central server and several workstations. For a WINNER NOW, the server is required to provide shared file systems and user accounts for all connected workstations. The various tasks of the WINNER system are performed by three kinds of *manager* processes: system managers, node managers, and job managers (see Fig. 1). Additionally, there are several user–interface tools, e.g. for status reports and for controlling access to each workstation.

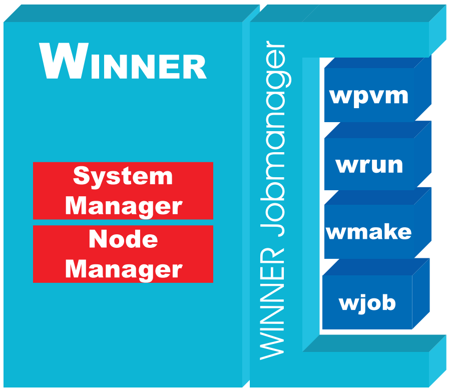

Figure 1: Manager processes in a WINNER cluster.

The system manager is the central server process of a WINNER network. Its duties include (a) collecting the load information of all respective workstations, (b) managing the currently active jobs, and (c) assigning hosts to job requests. This assignment is based on the estimated processing performance that each workstation can provide to a newly started process. This estimation is computed from each workstation's load-free speed, its current workload, and the number of processors. Furthermore, the presence of the workstation owner or additional resource constraints can reduce the set of hosts which are suitable for a given job.

On every host participating in a WINNER network, a node manager performs the tasks related to its machine. First of all, it periodically measures the host's utilization and reports it to the system manager. Furthermore, node managers are responsible for starting and controlling WINNER processes on their nodes, like e.g. reducing process priorities to protect console users [1].

System and node managers run as daemon processes. In contrast, job managers are invoked by users to execute sequential and parallel jobs. Thus, job managers are part of WINNER's user interface. Their duties are (a) acquiring resources from the system manager, (b) starting processes on the acquired nodes via the respective node managers, and (c) controlling and possibly redirecting input and output of the started processes.

WINNER's simplest job manager is called wrun. It allows the execution of a sequential (possibly interactive) job. It works almost like the standard Unix command rsh, except that the job is automatically started on the currently best suited workstation in a network. *wjob* allows non-interactive execution of sequential tasks. It provides support for process checkpointing and migration using the user-level checkpointing library libckpt [14]. Parallel PVM applications [6] are supported by *wpvm*. This job manager registers several processes into the PVM infrastructure to avoid changes to the PVM system or application programs. It automatically selects a number of appropriate hosts for the PVM virtual machine (usually a tedious task when performed manually), starts the user's parallel application, and improves PVM's scheduling by utilizing WINNER's load distribution mechanism. Furthermore, with *wmake* a parallel and distributed variant of GNU make has been implemented which can significantly reduce the time needed for compiling and linking large software projects. For an in-depth description of these different job managers, see [2].

WINNER's modular structure yields a system that is easily extensible and adaptable to further workstation platforms or kinds of jobs. Its manager processes and user–interface tools are implemented in C++. They communicate with each other by a reliable message–exchange layer on top of UDP sockets. Currently, WINNER runs on Digital Unix, Linux, FreeBSD, and Solaris.

3 Batch Queueing in WINNER

Until now, WINNER provides support for the automatic distribution of possibly interactive processes on the fastest available nodes in a NOW. Although this is very useful for daily work, this scheme is unsatisfactory for users having to start hundreds or thousands of batch jobs, because each submitted job is either started immediately (as long as computational resources are available) or will be rejected. This leads to a situation where jobs are either not executed at all or many jobs compete for the resources offered by each of the workstations, significantly degrading the overall performance. Obviously, some kind of of queueing mechanism is necessary.

The basic idea of introducing queues to WINNER is to provide a simple mechanism for submiting bulks of jobs without having the users explicitly coordinate their execution. To minimize the competition between queued jobs, WINNER guarantees that at most one queue job per processor of each workstation is executing at the same time. When all nodes are busy executing queue jobs, any further batch jobs will remain in a waiting state until one of the active queue jobs completes.

This situation is shown in Fig. 2. In this example, there are two queues, both containing currently running jobs as well as jobs waiting to be scheduled as soon as suitable nodes become available. Each queue can be restricted to a particular subset of nodes. In the example shown in Fig. 2, queue 1 can assign jobs to nodes A, B, and C, while queue 2 can assign jobs to nodes C, D, E, and F, thus sharing host C between both queues. All nodes except node F are executing queue jobs. Node F is idle because (assuming a FIFO scheduling algorithm) the next job of queue 2 that is to be scheduled requires three nodes to run. A variety of scheduling policies have been implemented in WINNER, a detailed description of which can be found in [3].

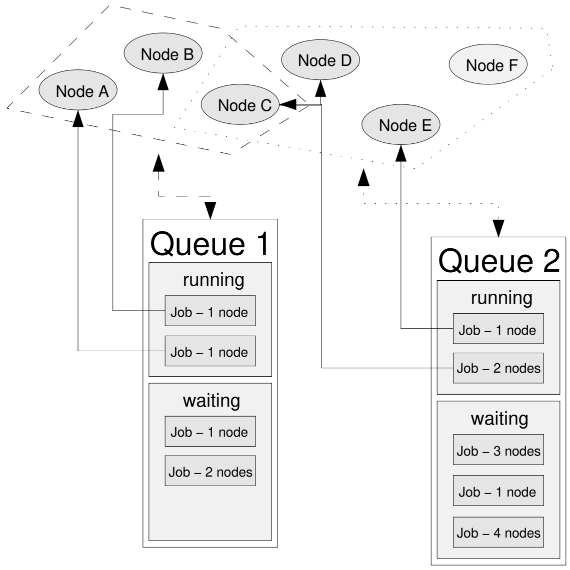

Figure 2: WINNER queues.

Another important advantage of a queueing mechanism is fault tolerance. In WINNER, we distinguish between two situations: jobs that do support checkpointing and those that do not. Since in the most general case jobs do not support checkpointing, we focus on this case first. All information on pending or active queue jobs is stored persistently on the system manager's disk. Thus, a failure of the system manager does not lead to any loss of jobs besides the information which job is running on which nodes. Since the node managers cannot work properly without the system manager, those jobs will be stopped anyway. After a restart of the system manager, the batch jobs will be restarted as well.

Since each node manager regularly sends information about the node's load situation to the system manager, a node failure can be assumed, when load messages have not been received within a particular time period by the system manager. Once such a failure is detected, the system manager restarts all queue jobs running on this node; in the worst case, the computation is done twice.

Those jobs that do provide checkpointing regularly create checkpoint files. When the system manager or a participating node manager fails, those jobs will be restarted from the most recent checkpoint file. This can significantly reduce the amount of lost computation time. The major weakness of WINNER's checkpointing mechanism is that it supports only sequential programs so far; extending it to parallel jobs is a subject of future work.

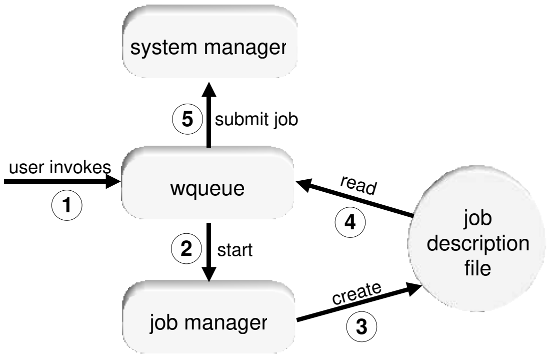

Figure 3: Submission of queue jobs in WINNER.

3.1 System Integration

WINNER has been designed for extensibilty to future kinds of systems and job types. As outlined in Section 2, this has been achieved by identifying three kinds of manager processes that communicate with each other by exchanging messages. Of course, a batch queueing facility has to fit into this structure to maintain WINNER's extensibilty. WINNER's batch queueing facility consists of two major parts: the job scheduler itself and a user interface via which jobs can be submitted. The job scheduler needs information about the available workstations and their current utilization. Hence, it is a natural choice to integrate the scheduler into the system manager process.

Each job manager is responsible for starting a certain class of jobs. It implements the process startup of its sequential or parallel application. Therefore, job managers have to be started by the job scheduler once their job is selected for execution. To make scheduling decisions, the resources required by each batch job have to be known a priori, namely at the time at which jobs are submitted. Because only the job manager knows which resources it will request for executing a given job, the job manager has to provide this information. Therefore, WINNER provides a "meta job manager" *wqueue* that invokes the respective job manager in a probing mode. We will now discuss this in detail.

User Interface

To illustrate the usage of *wqueue*, consider a sample application `app` which is parametrized by a configuration file `conf`. Since it needs some space for temporary files, a resource requirement of at least 10MB of free temp space is specified by using *wjob*'s option `-rt 10`:

```
wjob -rt 10 app conf
```

Invoked in this way, *wjob* registers a new job with the system manager, obtains the fastest available host that meets the resource requirements, and immediately starts `app`. If the user chooses to apply WINNER's batch queueing facility, the call is preceded by *wqueue* and its respective parameters. Here, `-n P300` specifies a queue:

```
wqueue -n P300 wjob -rt 10 app conf
```

Now, the job will first be enqueued by the system manager. It will be started as soon as a free workstation with at least 10MB of free temp space becomes available.

Job Submission

Figure 3 outlines the process of submitting a queue job. First, a user invokes *wqueue* (1). For retrieving the job's resource requirements, *wqueue* invokes the job manager in probing mode (2). The job manager will then write a *job description file* (3) containing all of the job's resource requirements as well as the current environment (including the working directory, environment variables etc.). *wqueue* reads this file (4), submits the job along with all required information to the system manager (5), and terminates.

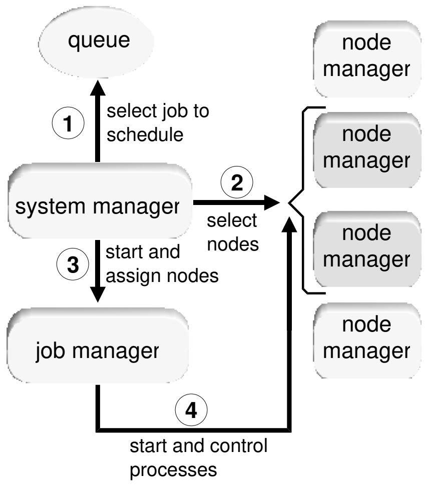

Figure 4: Start of a WINNER queue job.

Job Execution

The start of a queue job is shown in Figure 4. The batch job scheduler (which is part of the system manager) regularly inspects the queues for waiting jobs the resource requirements of which can be met. If there is any such job, it is selected to be started (1). Next, the nodes which are to be assigned to the job are chosen (2). The environment which was stored together with the job is restored, and the job manager is started (3) using the parameters that were originally specified by the user upon invoking *wqueue*. Then, the job manager requests and obtains the preselected set of nodes from the system manager. Finally, the job manager performs its duties like starting and controlling the job's processes as if it was directly invoked by the user.

3.2 Queue Management

In WINNER, queues can be configured in a convenient way using a graphical tool (cf. Fig. 5). Administrators can specify several queue attributes. These include the queue name, a plain-text description, the set of workstations on which jobs will be executed, the set of users having access to this queue, and the priority at which each job of the queue will be executed. This priority corresponds to the scheduling priority of the operating system and affects the percentage of processor time assigned to a job in the presence of competing processes. WINNER provides an additional priority, the queue priority, that affects the order in which jobs of different queues will be scheduled. Finally, it is possible to limit the start of jobs to specific time frames, optionally distinguishing between working days and weekends.

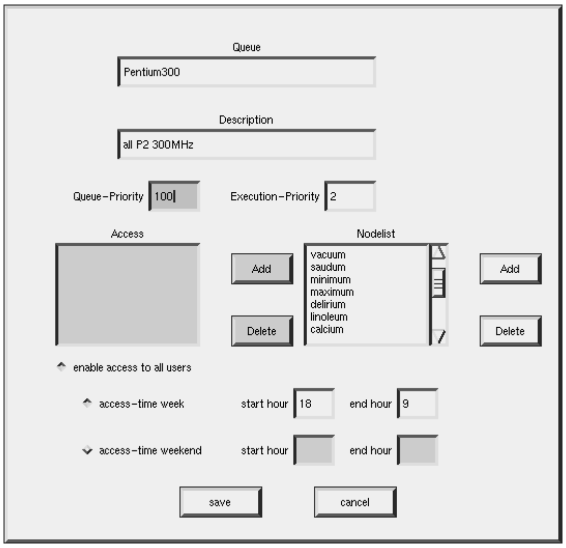

Figure 5: WINNER queue configuration tool.

For the users, *wqueue* provides some additional functionality. In addition to enabling the submission of jobs to a queue, *wqueue* can be used to display the current status of all queues, print a listing of active and waiting jobs, and delete pending jobs.

3.3 Exclusive Execution

Interactive workstation users can be protected from disturbance by batch jobs via queue priorities and time frames. Additionally, batch jobs may need exclusive access to a workstation, e.g. when their precise runtimes have to be measured. Therefore, the job manager *wjob* has been extended to provide two additional process control mechanisms.

(1) In the default configuration, WINNER does not start jobs on workstations with active console users. To further minimize the probability of other jobs interfering with measurements, jobs may also

be delayed until the workstation's load falls below a given threshold (usually close to zero).

(2) The above mechanism is quite useful, but cannot help when a user logs into a machine once a job had already been started. In this case, the job can be suspended whenever a user logs in. It will be resumed as soon as the user logs out again. This mechanism assures that the job has exclusive access to the CPU. While the process is suspended, it does not consume any CPU time. Thus, measurements which are based on the amount of consumed CPU time will yield correct results.

4 A Use Case Study

To evaluate the basic functionality and possible benefits of our queueing system, a series of experiments has been performed. As test jobs, several genetic algorithms [12, 13] have been chosen. To compare the quality of these algorithms, each of them was run to approximate the solution of a given NP-hard combinatorial optimization problem within a pre-determined amount of time.

A full set of runs consists of a number of independent jobs: First, one job is created for each algorithm. Since genetic algorithms are randomized optimization heuristics, each run has to be repeated for a number of times to get meaningful average results. Furthermore, each algorithm must solve a number of different problem instances. The resulting number of jobs is the product of the number of algorithms, the number of problem instances, and the number of times each job is repeated. Given that a single job takes 5 minutes to 2 hours to complete (depending on which parameters are used), a complete batch run needs a huge amount of CPU time due to the vast number of jobs involved.

Obviously, a single workstation can not provide sufficient computational power for such a problem setting. Parallel execution on several machines is a must. To achieve meaningful results, the job execution has to be coordinated such that the individual jobs interfere neither with each other nor with processes started by other users. Using a batch scheduling system with the features described so far, turns out to be the only feasible way of performing such a set of computations in a NOW environment.

Table 1: Sets of example batch jobs.

Jobs	avg. per job	time seq.	ws	time queue	utilization
3960	600 s	660 h	13	52.9 h	96%
3960	300 s	330 h	12	35.0 h	79%
100	7200 s	200 h	14	16.0 h	89%
1020	600 s	170 h	14	13.0 h	94%
1800	600 s	300 h	15	32.5 h	62%
512	1200 s	171 h	13	13.3 h	99%

WINNER's queueing system significantly simplifies such a task. The user submits each single job to WINNER without having to worry about machine failures, time slots, competing users or which workstations to use. When many similar or identical jobs are needed, a simple shell script that repeatedly calls *wqueue* is sufficient.

We have run several sets of example batch jobs, each executing genetic optimization algorithms. For all runs, the queue was configured to contain a maximum of 15 identical Pentium II 300MHz Linux workstations. Table 1 lists statistics of these runs. Each line describes a test set containing the number of individual jobs, the average completion time per job, the accumulated sequential completion time, the number of workstations available when the test was performed, the completion time using the queueing system, and the achieved CPU utilization of the workstations. The table clearly shows that the WINNER batch queueing system achieves high degrees of utilization. Of course, CPU utilization varies over time because WINNER gives priority to interactive jobs. Batch jobs only consume otherwise unused CPU time. But still, as Table 1 shows, batch jobs can exploit large fractions of the total CPU capacity.

5 Conclusion

In this paper we presented an approach to seamlessly integrate a batch queueing system into the WINNER resource management system for networks of workstations. The general design decisions, the functionality of the queueing system, and first results of its use in our workstation environment were described.

The core features of WINNER have been used on a daily basis by both the regular staff and students for well over a year in an environment consisting of about 80 networked workstations. The system proved to be useful because it is easy to use and decreases the average response time of many applications like computationally intensive compiler runs, $\TeX$ text formatting or other kinds of jobs. This is achieved without burdening users with the task of locating a most suitable workstation for a given task.

In contrast, utilization of WINNER queues for batch jobs has begun only recently. However, WINNER queueing facilities have already been exploited by various projects within our research group. The implementation has proven to be robust; user feedback has been quite positive.

The concept of a general queue submission tool that supports all kinds of jobs by making use of the corresponding job manager allows us to support both sequential jobs and parallel PVM jobs with minimal effort. In the near future, we plan to support additional job types such as parallel applications based on the MPI standard. Furthermore, we are investigating the effects of the batch queueing system on the interactive users of our workstation pool in a more elaborate manner.

References

[1] O. Arndt, B. Freisleben, T. Kielmann, and F. Thilo. Dynamic Load Distribution with the Winner System. In *Proc. Workshop Anwendungsbezogene Lastverteilung (ALV'98)*, pages 77 – 88, Munich, Germany, 1998.

[2] O. Arndt, B. Freisleben, T. Kielmann, and F. Thilo. Scheduling Parallel Applications in Networks of Mixed Uniprocessor/Multiprocessor Workstations. *Proc. ISCA PDCS-98*, Chicago, IL, 1998, pp. 190-197.

[3] O. Arndt, B. Freisleben, T. Kielmann, and F. Thilo. *A Comparative Study of Online Scheduling Algorithms for Networks of Workstations*. Submitted for publication, 1998.

[4] M. A. Baker, G. C. Fox, and H. W. Yau. A Review of Commercial and Research Cluster Management Software. Technical Report NPAC TR SCCS-748, NPAC, Syracuse University, NY, 1995.

[5] CODINE 4.1.1, Computing in Networked Environments. GENIAS Software GmbH, 1996.

[6] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam. *PVM: Parallel Virtual Machine – A Users Guide and Tutorial for Network Parallel Computing*. MIT Press, 1994.

[7] D. P. Ghormley et al. GLUnix: A Global Layer Unix for a Network of Workstations. http://now.cs.berkeley.edu/Papers2/recent.html.

[8] T. P. Green. DQS 3.0.2 Readme/Installation Manual. *Supercomputer Computations Research Institute*, Florida State University, 1995.

[9] S. Herbert. Generic NQS - Free Batch Processing for UNIX. Academic Computing Services, University of Sheffield, UK. http://www.shef.ac.uk/uni/projects/nqs/Generic-NQS/.

[10] P. Krueger and R. Chawla. The Stealth Distributed Scheduler. *Proc. IEEE 11th Int. Conference on Distributed Computing Systems*, pp. 336-343, 1991.

[11] M. Livny and R. Raman. High-Throughput Resource Management. In: *The GRID: Blueprint for a New Computing Infrastructure* pp. 311–337, Morgan Kaufmann, 1998.

[12] P. Merz and B. Freisleben. Fitness Landscapes, Memetic Algorithms and Greedy Operators for Graph Bi-Partitioning. To appear in *Evolutionary Computation*, MIT Press, 1999.

[13] P. Merz and B. Freisleben. A Comparison of Memetic Algorithms, Tabu Search and Ant Colonies for the Quadratic Assignment Problem. To appear in *Proc. Congress on Evolutionary Computation*, IEEE Press, 1999.

[14] J. S. Plank, M. Beck, G. Kingsley, and K. Li. Libckpt: Transparent Checkpointing under Unix. *Proc. Usenix Winter 1995 Conference*, 1995.

[15] S. R. Presnell. *The Batch Reference Guide*. 3rd Ed., Version 4.0, University of California, March 1994.

[16] GNU *Queue*. Free Software Foundation. http://www.gnu.org/software/queue/queue.html

[17] M. S. Warren, D. J. Becker, M. P. Goda, J. K. Salmon, and T. Sterling. Parallel Supercomputing with Commodity Components. *Proc. PDPTA'97*, pp. 1372–1381, 1997.

[18] S. Zhou. LSF: Load Sharing in Large-Scale Heterogenous Distributed Systems. University of Toronto, 1992. http://www.platform.com