Job Superscheduler Architecture and Performance in Computational Grid Environments

Hongzhang Shan, Leonid Oliker
Computational Research Division, Lawrence Berkeley National Laboratory, Berkeley, CA 94720

Rupak Biswas
NASA Advanced Supercomputing Division, NASA Ames Research Center, Moffett Field, CA 94035

Abstract

Computational grids hold great promise in utilizing geographically separated heterogeneous resources to solve large-scale complex scientific problems. However, a number of major technical hurdles, including distributed resource management and effective job scheduling, stand in the way of realizing these gains. In this paper, we propose a novel grid superscheduler architecture and three distributed job migration algorithms. We also model the critical interaction between the superscheduler and autonomous local schedulers. Extensive performance comparisons with ideal, central, and local schemes using real workloads from leading computational centers are conducted in a simulation environment. Additionally, synthetic workloads are used to perform a detailed sensitivity analysis of our superscheduler. Several key metrics demonstrate that substantial performance gains can be achieved via smart superscheduling in distributed computational grids.

1 Introduction

Grid computing [1, 8] holds the promise to effectively share geographically distributed heterogeneous resources in a seamless and ubiquitous manner. The development of computational grids and the associated middleware has therefore been actively pursued in recent years. There are many potential advantages to utilizing the grid infrastructure, including the ability to simulate applications whose computational requirements exceed local resources, and the reduction of job turnaround time through workload balancing across multiple computing facilities. However, many major technical (and political) hurdles stand in the way of realizing these gains. Among the myriad research issues to be addressed is the problem of distributed resource management and job scheduling for computational grids. Although numerous researchers have proposed scheduling algorithms for parallel architectures [5, 6, 7, 9, 13, 15], the problem of scheduling jobs in a heterogeneous grid environment is fundamentally different. This is the focus of our work in this paper.

Job scheduling on computational grids is conducted via autonomous local schedulers that cooperate through a *superscheduler* [16] using grid middleware. Since the superscheduler (or grid scheduler) does not have control over the resources of the distributed computing centers, it depends on the individual local batch queuing systems to initiate and manage job execution. The superscheduler is thus responsible for discovering grid resources, monitoring system utilization, and intelligently migrating workloads to the local queues of distributed resource centers.

SC'03, November 15-21, 2003, Phoenix, Arizona, USA

In this paper, we first investigate the architectural requirements of a superscheduler. Although various aspects of its infrastructure have been studied before [3, 4, 10, 14], a number of important issues remain unaddressed. These include the superscheduling algorithm, interaction between the superscheduler and various local schedulers, selection of jobs for migration, and destination choice for the transferred jobs (also known as the location policy). The superscheduler algorithm is basically a job transfer policy that determines if there is a need to migrate jobs from one computing server to another. Using system and workload requirements, the grid scheduler determines when a server becomes eligible to act as a sender (transfer a job to another server) or as a receiver (retrieve a job from another server). The location policy selects a partner server for a job transfer transaction. In other words, it locates complementary computing nodes to/from which another node can send/receive workloads to improve critical performance metrics. Since these issues are important for effective grid scheduling, we propose a novel distributed superscheduler architecture and three job migration algorithms in this paper. We then compare their performance in terms of several key metrics with ideal, central, and local schemes in a simulation environment.

The other distinguishing aspect of this research is the set of real and synthetic workloads used in our experiments. We obtained real workload data (binary compatible) from three leading computational centers over the same six-month period of 2002. Since the trace data is for the same period of time, we are able to evaluate the potential benefits of allowing jobs to migrate between distributed compute nodes. By examining real data, we accurately demonstrate the substantial performance improvement, in terms of average waiting time and average response time, that can be achieved via smart superscheduling in computational grid environments. Additionally, we present simulation results based on heavy and light synthetic workloads that are derived from the real workloads using the hyper-Erlang distribution of common order [11, 12]. By varying the model parameters, synthetic workloads allow us to conduct a detailed sensitivity analysis of superscheduling architectures and algorithms under different conditions, such as over-/under-subscription and additional compute servers.

Our overall results demonstrate that intelligent superscheduling can deliver substantial performance gains compared to locally isolated machines. However, it is important to note that this preliminary study does not attempt to address many complex questions related to computational grids. Future research will build on our simulation environment to address issues such as job migration overhead, grid network costs, superscheduler scalability, fault tolerance, multiple-resource requirements, and architectural heterogeneity.

The remainder of the paper is organized as follows. Section 2 describes the distributed superscheduler architecture and the three job migration algorithms that we developed. Section 3 discusses the simulation environment, including the real and synthetic workloads, and various performance metrics. Detailed performance analysis, including the effects of local scheduling policy on overall grid performance, is reported in Section 4. Finally, Section 5 concludes the paper by summarizing this work and providing a preview of future research in this area.

2 Superscheduler Architecture

This section presents the three job superscheduling architectures examined in this study. We first describe the distributed architecture and three job migration algorithms: *sender-initiated*, *receiver-initiated*, and *symmetrically-initiated*. Next we present a centralized architecture that uses a single global queue to schedule jobs in a grid environment. Finally, we introduce an idealized strategy to establish an upper bound on performance.

2.1 Distributed

The distributed architecture for the grid job superscheduler is depicted in Figure 1. It is composed of a collection of autonomous local schedulers that cooperate with the superscheduler through grid middleware. A new job is first submitted to a *grid queue* (GQ), which then forwards the job's resource requirements to the *grid scheduler* (GS). In the distributed architecture, the GS is assumed to have an affinity to a particular *local scheduler* (LS). The GS queries the LS via the *grid middleware* (GM) for the *approximate wait time* (AWT) that the job would have stay in the *local queue* (LQ) before beginning execution on the local system. The LS computes the AWT based on the local scheduling policy and the LQ status. If the local resources cannot satisfy the requirements of the job, an AWT of infinity is returned. If the AWT is below a minimal threshold ϕ, the job is moved from the GQ directly into the LQ without any external network communication. Otherwise, one of the three distributed job transfer algorithms is invoked by sending workload information to a *partner set* of computing facilities connected via the grid. The pseudo-codes for all three algorithms are shown in Figure 2. For the simulations in this paper, the partner set contains all of the available machines on the grid. However, in a large computational grid setting, each machine would intelligently organize and dynamically update a subset of the available partners to keep the system efficient and scalable. The management of partner sets will be the subject of future work.

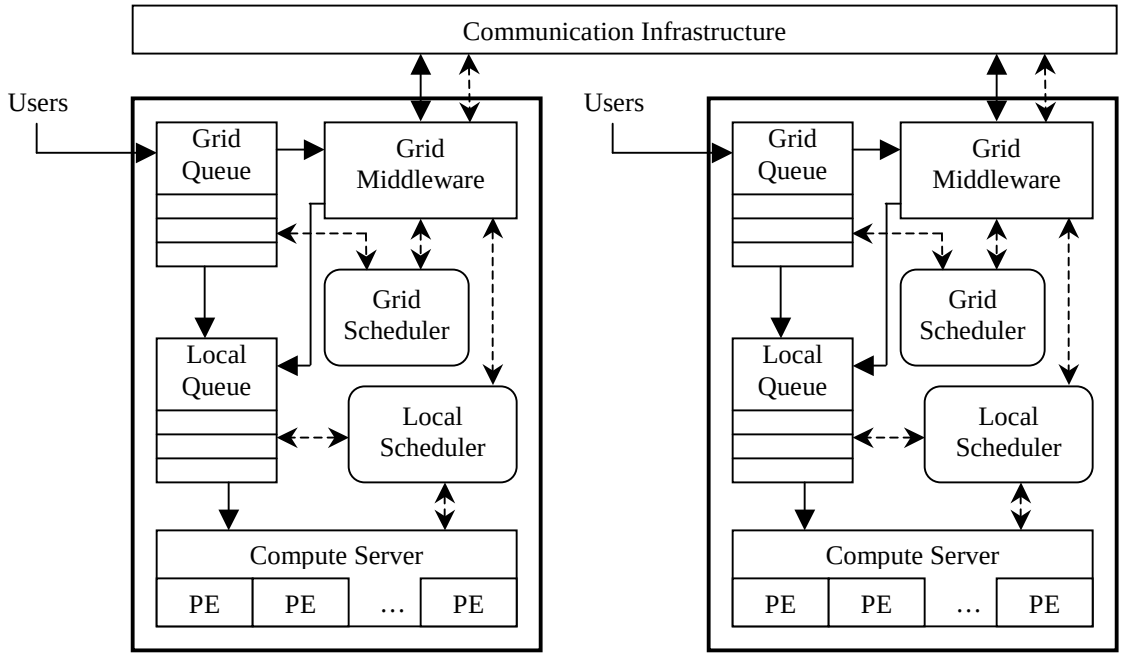

Figure 1: Distributed architecture of the grid superscheduler (solid arrows represent movement of jobs, dashed arrows represent transfer of information).

2.1.1 Sender-Initiated

In the sender-initiated (S-I) strategy, the GS sends the resource demands of the job to the compute server's partner set via the GM. In this study, we only consider the CPU and run time requirements of each job; however, this can be extended to an arbitrary number of resource constraints. In response to the GS query, each partner returns the AWT and *expected run time* (ERT) of the requested job, as well as its personal *resource utilization status* (RUS). Note that the ERT can vary from one computational node to another

All Job Migration Algorithms:

Job J arrives in Global Queue of Machine L (GQ^L)
Compute Approximate Wait Time of J on L (AWT_J^L)
If ($AWT_J^L < \phi$) Move J to Local Queue of L (LQ^L)
Else Call S-I, R-I, or Sy-I

Sender-Initiated (S-I):

Send Resource Requirements of J (RR_J) to Partner Set of L ($PS(L)$)
Compute Turnaround Cost for J on all Machines in $PS(L)$ ($TC_J^{PS(L)}$)
Find Machine M such that $TC_J^M = \min(TC_J^L, TC_J^{PS(L)})$
Move J to LQ^M

Receiver-Initiated (R-I):

Each Machine R checks own Resource Utilization Status (RUS^R) at time interval σ
If ($RUS^R < \delta$) Send Availability Message (AM^R) to Partner Set of R ($PS(R)$)
If (Machine L with Job J in GQ^L receives AM^R)
    Send RR_J to all Machines R
    Compute TC_J^R for all R
    Find Machine M such that $TC_J^M = \min(TC_J^R)$
    If ($TC_J^M < TC_J^L$) Move J to LQ^M

Symmetrically-Initiated (Sy-I):

Call R-I
If (No Availability Messages) Call S-I

Figure 2: Pseudo-codes for the three distributed job migration algorithms.

depending on their architectural designs and program characterizations. If certain partners do not respond within a specified time limit due to traffic congestion or machine failure, they are simply ignored for that request.

Based on the collected information, the GS calculates the potential *turnaround cost* (TC) of itself and each partner. To compute the optimal TC, first the minimum *approximate turnaround time* (ATT) is calculated as the sum of AWT and ERT. If the minimum ATT is within a small tolerance ϵ for multiple machines, the system with the lowest RUS is chosen to accept the job. Thus the TC metric attempts to minimize the user's time-to-solution, while using system utilization as a tiebreaker. We found this approach to be more effective then simply relying on ATT. A more robust TC metric would also consider the communication overhead of data and job migration, and will be considered in future research. The job is then migrated into the LQ of the machine with the minimal TC. The GM is responsible for handling the job transfer to the LQ either locally or across the communication network to a remote site. Note that once a job enters a LQ, it will be scheduled and run based exclusively on the local policy of the LS, and will no longer be controlled by the superscheduler or migrated to another site. When the job is completed, the results are sent back to the compute node where it was originally submitted. In order to avoid message congestion, the GS can only send out a query for a new job after it has received all of the responses from a previous call. During this time, the new job waits in the GQ.

2.1.2 Receiver-Initiated

The receiver-initiated (R-I) algorithm takes a more passive approach to job migration than the S-I strategy. Here, each system in the computational grid checks its own RUS periodically at time interval σ. If the RUS is below a certain threshold δ, the machine volunteers itself for receiving jobs by informing its partner set of its low utilization. Once a partner (say, L) receives this information, it checks its GQ for the first job waiting to be scheduled. If a job is indeed queued, its resource requirements are sent to the volunteer node. The underutilized system then responds with the job's ATT, as well as its own RUS. Based on this data, L computes and compares the TC between itself and the volunteer system. If the TC of the volunteer is lower than that of L, the job is transferred to the LQ of that system through the GM. Otherwise, it continues to wait in the GQ until either its local AWT falls below ϕ (examined at time interval σ), or an available machine volunteers its services.

2.1.3 Symmetrically-Initiated

Unlike S-I and R-I, the symmetrically-initiated (Sy-I) algorithm works in both active and passive modes. As in the R-I strategy, each machine periodically checks its own RUS and broadcasts a message to its partner set if it is underutilized. The difference occurs when the local AWT of a job exceeds ϕ but no underutilized machine volunteers its services. In the R-I approach, the job passively sits in the GQ while waiting for a volunteer, and periodically checks its local AWT at each σ time interval. However, the Sy-I algorithm immediately switches to active mode and sends a request to its partners using the S-I strategy. The main differences in the three job migration algorithms therefore lie in the timing of the job transfer request initiations and the destination choice for those requests.

2.2 Centralized

In the centralized architecture, all jobs are submitted to a single GQ which does not have an affinity to a specific local system. The GS is responsible for making global decisions and assigning each job to a specific machine. The GS tracks the status of each job and maintains up-to-date information on all available resources, allowing it to compute the TC directly, without the need for any communication. When a job arrives, the GS computes its TC for all systems, selects the one with the minimum TC, and immediately migrates the job to that system. Although communication-free resource awareness is an unrealistic assumption, it allows us to model the potential gain of a centralized architecture. However, it constitutes a single point of failure and thus suffers from a lack of reliability and fault tolerance. Additionally, this approach has severe scalability problems that may result in a performance bottleneck for large-scale grid environments. In contrast, the distributed approach has the potential to be highly scalable and robust, since each computational facility runs its own GS. Detailed superscheduler scalability and fault-tolerance will be addressed in future work.

2.3 Idealized

Finally, we present an idealized superscheduler architecture to establish an upper bound on grid performance. Here, the entire computational grid is viewed as a single virtual machine, where each node is considered to contain exactly one CPU running at 1 MHz. Thus, each CPU in the grid running at X MHz will contribute X nodes to the virtual machine, for a sum total of $\sum_{m \in Servers} \#CPUs_m \times CPUSpeed_m$ nodes. Each submitted job is treated as a modulable workload, i.e. the number of CPUs assigned to the job can be varied arbitrarily according to the machine status, with an assumption of ideal scalability. The idealized GS can therefore perfectly pack the available resources with incoming jobs. For example, if a job requests eight 300 MHz CPUs for 100 seconds, the GS may assign the job to $8 \times 300 \times 100$ CPUs in the virtual machine,

which would complete the computation in one second. Although the performance predicted by this virtual architecture can never be achieved, it establishes an ideal performance upper bound for computational grids.

3 Simulation Environment

The configurations of the computational servers used in our simulations are shown in Table 1. They are six binary-compatible architectures currently deployed and listed in the Top500 [2]. Each system is similar architecturally, consisting of cache-coherent SMP nodes interconnected via a fast proprietary network. However, individual characteristics such as CPU speed, SMP size, node count, and interconnect topology do vary across the machines. Future work will address true server heterogeneity. Currently, a common practice for this type of architecture is that a single node cannot run more than one job simultaneously, regardless of the number of CPUs actually consumed by the job. We therefore implemented the same restriction in our simulation environment. For the experiments in this paper, we also made the simplifying assumption that program performance is linearly related to CPU speed.

Server Identifier	Number of Nodes	CPUs per Node	CPU Speed (MHz)
M_1	192	16	375
M_2	305	4	332
M_3	144	8	375
M_4	8	16	1300
M_5	74	4	375
M_6	180	4	375

Table 1: Configurations of the computational servers.

3.1 Workloads

We used both real and synthetic workloads in our experiments. The real workloads were collected from three supercomputing centers: National Energy Research Scientific Computing Center (NERSC) at Lawrence Berkeley National Laboratory, Lawrence Livermore National Laboratory, and San Diego Supercomputer Center. These three machines are listed as M_1, M_2, and M_3 in Table 1. All three logs started on March 1, 2002 and ended August 31, 2002, and contained 132069, 42339, and 36131 batch jobs, respectively. Interactive jobs were filtered out of the job submissions since they would normally be restricted to run on the local systems. By using real user batch data over the same time period in our experiments, we are able to accurately simulate the potential contribution of a smart grid scheduler.

However, real workloads have certain limitations. First, it is a non-trivial task to obtain log reports from various computing facilities, thus limiting the potential scope of the simulations. It is also difficult to use existing batch data to perform parameter studies of varying workload conditions, such as over- or under-subscribed systems. Therefore, we derived a set of synthetic workloads from the real logs using the methodology described in [11, 12].

In this approach, the real job data is first grouped into different classes based on the number of processors required for each execution. The initial class size is set to the number of CPUs per node for the corresponding machine. If the percentage of jobs within a class is below 2% of the total, the class is merged with the smaller of its neighboring classes. For each class, we then compute the first three non-center moments (μ_1, μ_2, μ_3) separately for the inter-arrival and service times. The three moments essentially capture the generic features

of the workloads. Next, the hyper-Erlang distribution of common order, based on four parameters: n, λ_1, λ_2, and ρ, that fits these three computed moments is selected. An example of a server with hyper-Erlang distribution of common order is a system where a job must pass through only one of two service paths to completion. The parameter ρ is the probability of selecting the first path. In each path, the job passes through n stages, spending a random amount of service time at each stage. The probability density function of service time at each stage of the two paths is an exponential distribution with mean times $1/\lambda_1$ and $1/\lambda_2$, respectively. The parameters for the hyper-Erlang distributions that model the real workloads on M_1, M_2, and M_3 are presented in Table 2, which clearly shows that the three real workloads have significantly different characteristics. Finally, the synthetic job submissions are generated by combining the different class models. We can create different workloads by varying the model parameters that control the inter-arrival rate and service time.

			Inter-arrival time				Service time			
$N_{\min}$	$N_{\max}$	% Jobs	n	λ_1	λ_2	ρ	n	λ_1	λ_2	ρ
Machine M_1										
1	16	43.0	1	2.75E-04	4.71E-03	0.0197	1	9.10E-05	4.55E-03	0.4695
17	32	15.2	1	1.44E-04	2.41E-03	0.0571	2	1.04E-04	2.74E-03	0.3119
33	48	2.12	1	2.37E-05	4.36E-04	0.0847	1	7.22E-05	2.99E-03	0.3319
49	112	27.9	1	2.22E-04	4.16E-03	0.0448	1	7.31E-05	3.94E-03	0.2241
113	240	6.21	1	1.62E-04	1.41E-03	0.2253	1	6.03E-05	3.80E-04	0.4072
241	3072	5.57	1	1.42E-04	3.11E-03	0.2728	1	5.69E-05	7.93E-04	0.2473
Machine M_2										
1	4	19.0	1	3.99E-05	1.64E-03	0.0555	1	1.00E-04	6.79E-03	0.0697
5	12	10.4	1	2.18E-05	9.08E-04	0.0554	1	1.43E-04	5.39E-03	0.1947
13	24	17.0	1	3.14E-05	8.07E-04	0.0319	1	1.47E-04	1.07E-03	0.3335
25	28	2.07	1	1.76E-06	2.59E-04	0.0250	1	1.82E-04	3.13E-02	0.0115
29	44	4.84	1	2.02E-05	3.76E-04	0.1080	1	6.09E-05	2.41E-04	0.0082
45	60	2.75	1	1.14E-05	1.08E-04	0.0561	4	3.65E-04	4.47E-02	0.5000
61	92	10.8	1	2.98E-05	3.49E-04	0.0200	1	4.89E-06	2.48E-04	0.0009
93	104	5.07	1	1.72E-05	1.64E-04	0.0251	2	1.09E-04	3.64E-04	0.0314
105	124	3.16	1	1.59E-06	1.38E-04	0.0074	1	2.23E-04	3.02E-03	0.4179
125	176	13.6	1	1.09E-04	5.27E-04	0.1149	2	8.07E-05	3.19E-04	0.0210
177	188	4.32	2	9.87E-05	1.17E-03	0.3762	3	2.01E-04	2.13E-02	0.6043
189	252	3.67	1	9.15E-06	1.38E-04	0.0293	1	9.26E-05	3.28E-04	0.0603
253	1220	3.41	1	8.73E-06	1.93E-04	0.0532	1	1.97E-04	1.75E-03	0.4098
Machine M_3										
1	8	55.0	1	4.01E-05	1.47E-03	0.0047	1	3.99E-05	1.01E-03	0.0411
9	24	8.09	1	3.77E-05	4.35E-04	0.1281	2	3.92E-05	2.25E-03	0.2113
25	56	9.99	1	4.15E-05	6.33E-04	0.1235	2	4.13E-05	2.16E-03	0.1400
57	120	13.8	1	5.57E-05	5.53E-04	0.0833	1	3.03E-05	6.79E-04	0.2539
121	248	4.99	1	2.73E-05	3.64E-04	0.1731	3	6.91E-05	1.01E-02	0.2479
249	504	5.09	1	2.09E-05	5.08E-04	0.1431	1	1.89E-05	2.64E-04	0.1033
505	1152	2.99	1	7.31E-06	2.63E-04	0.0766	2	6.05E-05	1.13E-02	0.0671

Table 2: Parameters for the inter-arrival and service times of workloads on M_1, M_2, and M_3.

3.2 Performance Metrics

We use several key metrics in our simulations to evaluate the effectiveness of the proposed grid superscheduler and the three distributed job migration algorithms. These metrics are also used to compare performance with local, central, and ideal job scheduling schemes. The local and ideal strategies respectively are expected to provide lower and upper bounds on the performance of a grid scheduler.

Since individual users and center system administrators often have different (and possibly conflicting) demands, no single measure can comprehensively capture overall grid performance. From the users' perspective, key measures of grid performance include the *Average Response Time* and the *Average Wait Time*. These are computed as follows (N is the total number of jobs):

$$\text{Average Response Time} = \frac{1}{N} \sum_{j \in Jobs} (\textit{EndTime}_j - \textit{SubmitTime}_j)$$

$$\text{Average Wait Time} = \frac{1}{N} \sum_{j \in Jobs} (\textit{StartTime}_j - \textit{SubmitTime}_j)$$

where $\textit{SubmitTime}_j$, $\textit{StartTime}_j$, and $\textit{EndTime}_j$ are the times when job j is submitted to the queue, when it commences execution, and when it is completed. The response (or turnaround) time is probably the single most important measure for an individual submitting a job; however, the wait time is also critical to users even though it is usually beyond their control. The wait time is especially important for users running short jobs. Finally, we also examine the *Average Wait Time Deviation* in order to investigate overall fairness and performance variability:

$$\text{Average Wait Time Deviation} = \frac{1}{N} \sqrt{\sum_{j \in Jobs} (\textit{WaitTime}_j)^2 - (\sum_{j \in Jobs} (\textit{WaitTime}_j / N))^2}$$

where $\textit{WaitTime}_j = (\textit{StartTime}_j - \textit{SubmitTime}_j)$.

A system administrator (or funding agency), on the other hand, is more interested in maximizing the utilization of the available computational resources at his/her center. Thus, we present the *Grid Efficiency* metric, which measures the overall ratio between consumed and available computational resources across the distributed grid. It is computed as:

$$\text{Grid Efficiency} = \frac{\sum_{j \in Jobs} (\textit{EndTime}_j - \textit{StartTime}_j) \times \textit{CPUs}_j \times \textit{CPUSpeed}_j}{(\textit{EndTime}_{last\_job} - \textit{SubmitTime}_{first\_job}) \times \sum_{m \in Servers} \textit{CPUs}_m \times \textit{CPUSpeed}_m} \times 100\%$$

where $(\textit{EndTime}_{last\_job} - \textit{SubmitTime}_{first\_job})$ is the duration of the entire simulation; $\textit{CPUs}_j$ and $\textit{CPUSpeed}_j$ are the number of processors used by job j and their clock speed; and $\textit{CPUs}_m$ and $\textit{CPUSpeed}_m$ are the number of processors in machine m and their clock speed. Individual site-specific system utilizations are also reported to understand the effects of superscheduling on local computational centers.

Finally, we present the *Fraction of Jobs Transferred* for each scheduling approach:

$$\text{Fraction of Jobs Transferred} = \frac{\textit{Number of Jobs Transferred}}{\textit{Total Number of Jobs}}$$

Although our turnaround cost metric TC (defined in Section 2.1.1) does not explicitly incorporate job migration overhead at this time, it is clear that network traffic must be minimized. The fraction of jobs transferred is an initial attempt to capture this cost.

Note that performance, measured by any metric, is highly dependent on the workload requirements. For example, we would not expect an underloaded system to derive much benefit from a superscheduler in terms of grid efficiency, as there may not be much room for improvement.

4 Performance Analysis

This section presents and analyzes the simulation results of our job migration algorithms in terms of the performance metrics described in Section 3.2. We first examine real workload data from three supercomputing centers, over one- and six-month submission periods. Next, we use our synthetic workloads to evaluate a larger, six-machine grid configuration under heavy and light system load conditions. Finally, the effects of the local scheduling policy on overall grid performance is investigated.

4.1 Real Workloads

The real workload data was obtained from the job logs for the same six-month period (March 1, 2002 through August 31, 2002) of the machines M_1, M_2, and M_3 listed in Table 1. We also examine a one-month period (August 2002) to investigate differences in performance trends for shorter workload durations. Note that only batch job data are used in our simulations; interactive submissions have been removed.

Table 3 presents the local run characteristics of the three machines examined. Machine M_1 is the most heavily loaded, with an utilization of over 90%, while M_2 and M_3 have lighter loads and lower utilization. Also notice that the average wait time and response time for M_2 is significantly lower than the other two machines. By examining the workload data, we found bulk jobs that require a relatively large fraction of the computational resources often arrive at approximately the same time, thus preventing one another from being efficiently scheduled. This presents an opportunity for a smart superscheduler to improve the average turnaround times of these large submissions.

	Six-month Workload			One-month Workload		
	M_1	M_2	M_3	M_1	M_2	M_3
Number of Jobs	132,069	42,339	36,131	26,343	5,735	5,974
Local Machine Utilization	91%	72%	79%	92%	72%	73%
Average Wait Time (sec)	8,318	1,955	11,506	7,977	5,173	15,271
Average Response Time (sec)	13,404	5,445	16,660	12,770	9,525	20,075

Table 3: Characteristics of real workloads for local runs.

Figure 3 presents simulation results for the one- and six-month real workload data sets for the five metrics described in Section 3.2. Both the average wait time and the average response time are normalized relative to the performance of the local scheduler. Results are compared among the three distributed job migration algorithms: sender-initiated (S-I), receiver-initiated (R-I), and symmetrically-initiated (Sy-I), as well as with local, central, and ideal strategies.

Notice that the one- and six-month data exhibit similar overall performance trends, indicating that the workload characteristics change little across months and that we do not expect to see a dramatic change in our observations for longer time durations on these systems. The normalized average wait and response times, and the average wait time deviation are all key metrics from an individual user's perspective. These results clearly demonstrate the large potential gain of using a superscheduler, as opposed to relying on traditional local job submission in a grid environment. For example, comparing the local and S-I schemes for six-month data, we see that the average wait time is reduced by a factor of 2.5, along with a 30% improvement in its deviation and a 1.5X reduction in the average response time.

Comparing the individual distributed job migration schemes among themselves, we find the R-I performance to be lower than that of S-I. This is because the R-I approach is the most passive, waiting for machines to advertise themselves and thus migrating the fewest number of jobs. Figure 3 shows that R-I migrates less than 10% of all jobs, while S-I transfers over 40%. Lowering the utilization threshold δ from

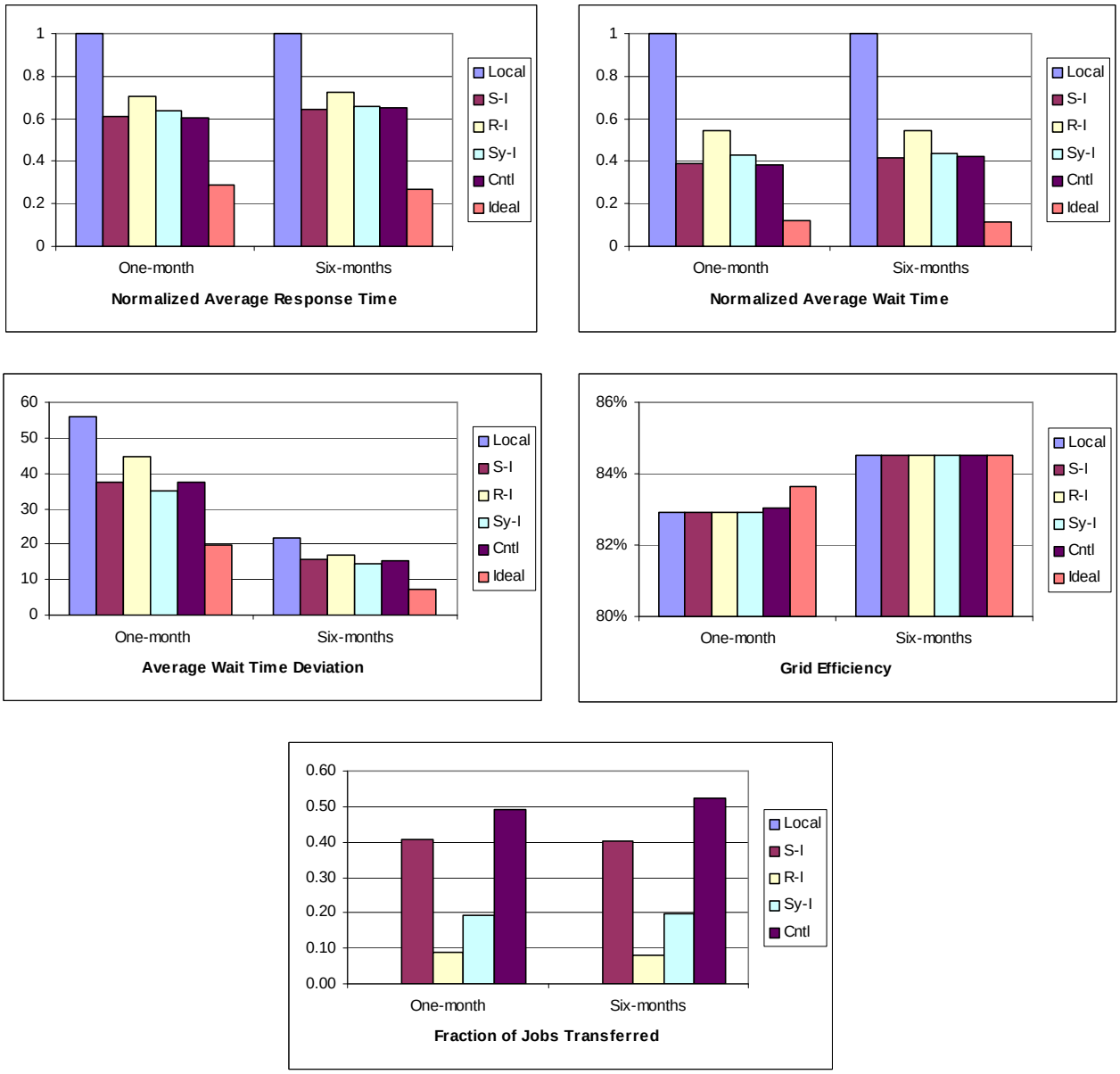

Figure 3: Performance results for the one- and six-month real workloads.

0.7 and/or the time interval σ from 300 secs (see Section 2.1.2) would improve performance but increase the number of jobs transferred. Nonetheless, compared to the local scheme, the average wait time of R-I is still reduced by an impressive 50%. The Sy-I scheme is more flexible than R-I, having the option to passively wait for a machine to advertise their availability, or to actively migrate jobs if no volunteers appear. The Sy-I algorithm strikes a good balance, achieving better performance than R-I while transferring significantly fewer jobs than S-I. Future work will directly incorporate job migration overhead into our cost models.

The central scheme achieves about the same performance as S-I, while transferring a higher fraction of its jobs to a remote site. Recall from Section 2.2 that the centralized architecture has a single grid queue whereas S-I has multiple grid queues. In S-I, a job is considered for migration only if its approximate wait time is larger than a threshold ϕ (see Section 2.1) set to 60 secs; instead all jobs are assigned to machines solely based on turnaround cost in the centralized approach. Therefore, the S-I algorithm is significantly

more conservative in moving jobs. However, observe that not all jobs are transferred in the central scheme. Since input/output data for each job still has an affinity to a particular computational node, we do not consider it a transfer if a job migrates to that node. The central scheme is also too limited in terms of fault tolerance and scalability. Finally, an idealized (unattainable) algorithm is presented to establish upper bounds on performance.

Grid efficiency for the six schemes are also presented in Figure 3. Rather surprisingly, it remains practically unchanged regardless of the scheduling algorithm. On closer inspection, we found that the overall grid resources were under-subscribed, thus allowing little improvement in grid efficiency even in the ideal case. This result further motivated us to explore superscheduling performance under both heavy and light grid load conditions, using synthetically generated data sets. Note that even though there is little change in grid efficiency, individual site utilization is dependent on the specific job migration scheme. For example, comparing local and S-I for the six-month data, utilization changed from 92%, 72%, and 73% to a more "balanced" 86%, 81%, and 78% for M_1, M_2, and M_3, respectively. However, interpreting these results can be rather difficult. For example, if an over-subscribed site's utilization decreases due to grid participation, it may seem like a positive consequence to an individual user; however, the center management may be unhappy with the new outcome since lower utilization may jeopardize future funding.

4.2 Synthetic Workloads

To study superscheduler performance with respect to various workload demands, we generated synthetic job submission data using the methodology described in Section 3.1. The statistical models of the real workload data using the hyper-Erlang distribution of common order are shown in Table 2. Once these parameters are generated, they can be adjusted accordingly to simulate different workload conditions. Our synthetic workloads simulate heavily- and lightly-loaded system conditions for a relatively larger six-machine grid configuration over a two-week period. Synthetic data for machines M_1, M_2, and M_3 are derived from their own individual models, while those for machines M_4, M_5, and M_6 are based on M_1, M_2, and M_3, respectively. Table 4 shows the local run characteristics of all six machines for both the heavy and light workloads.

	Heavy Workload					
	M_1	M_2	M_3	M_4	M_5	M_6
Number of Jobs	10,192	3,342	2,900	336	830	1,658
Local Machine Utilization	94%	83%	88%	33%	72%	81%
Average Wait Time (sec)	254,797	5,871	14,293	2,779	6,872	18,697
Average Response Time (sec)	260,010	9,295	19,554	7,756	10,154	24,460
	Light Workload					
	M_1	M_2	M_3	M_4	M_5	M_6
Number of Jobs	10,432	3,483	2,774	350	864	1,704
Local Machine Utilization	82%	72%	42%	36%	75%	62%
Average Wait Time (sec)	3,064	661	1,241	3,099	7,463	5,509
Average Response Time (sec)	8,266	4,199	6,321	7,466	11,146	10,865

Table 4: Characteristics of synthetic workloads for local runs.

Figure 4 presents simulation results for the heavy and light synthetic workloads for the five performance metrics described in Section 3.2. Observe that, as with the real workloads, superschedulers significantly outperform the local scheme from the users' perspective (in terms of normalized average response and wait

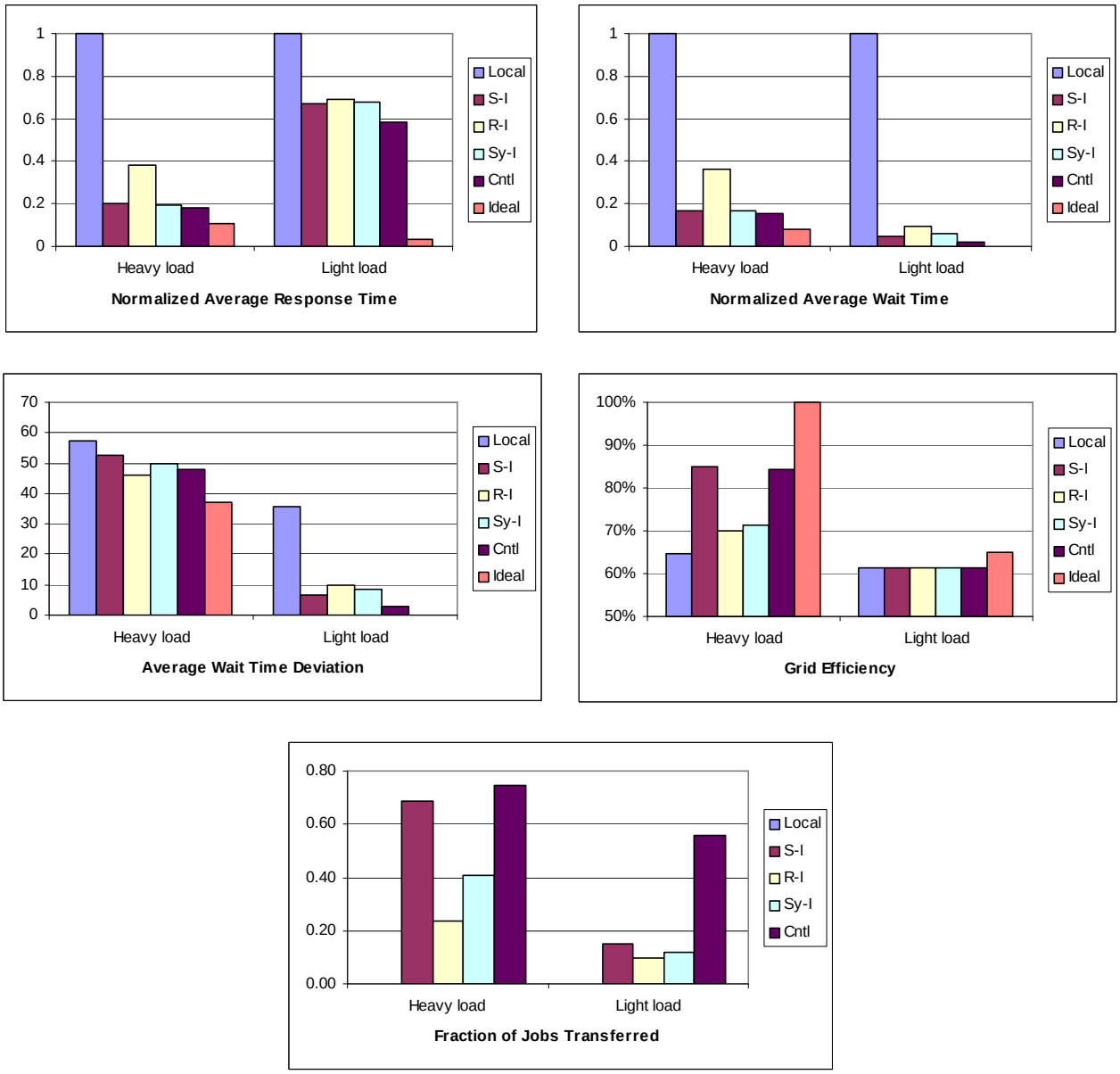

Figure 4: Performance results for the heavy and light synthetic workloads.

times, and average wait time deviation). Furthermore, as the number of machines grows from three to six, the advantages of a grid scheduler become more pronounced even for the lightly-loaded case. For example, compared to local scheduling, the S-I approach improves the average wait time by factors of 5.9 and 21, and the average response time by factors of 5.0 and 1.5 for the heavy and light workloads, respectively. For the heavily-loaded simulation, there is a more dramatic improvement in the average response time when compared with the real workload results in Figure 3. The key difference is the introduction of machine M_4, whose 1300 MHz clock allows the simulated computations to complete approximately 3.5 times faster than the other machines in our study. This highlights the dramatic potential gain that could be attained within a large-scale heterogeneous grid configuration.

Grid efficiency in Figure 4 shows that for the lightly-loaded test case, there is almost no change in performance relative to the local algorithm. This is consistent with the results for the real (under-subscribed)

workload data. However, the heavily-loaded configuration demonstrates that for over-subscribed systems, grid efficiency can be improved through the use of an intelligent superscheduler. For example, the S-I strategy achieves 85% grid efficiency, compared with 65% for the local approach. In fact, the idealized case achieves 100% efficiency in this example. As discussed previously in Section 4.1, each of the grid scheduling algorithms offers a tradeoff between performance and the number of transferred jobs. Overall our simulation results demonstrate the tremendous potential of using a superscheduler, for both individual users and system administrators.

4.3 Effects of Local Scheduler

The simulation results presented in Sections 4.1 and 4.2 assume that the local scheduling policy of each individual machine is the popular first-come-first-serve with backfilling (FCFS+BF). However, the local scheduling algorithm will definitely affect overall grid behavior. Since the superscheduler has no control over local scheduling policies, we evaluate grid performance using two alternative local scheduling policies: first-fit (FF) and shortest-job-first (SJF). Figure 5 examines the effects of the different local schedulers using the sender-initiated distributed job migration algorithm for the one-month real workload data set.

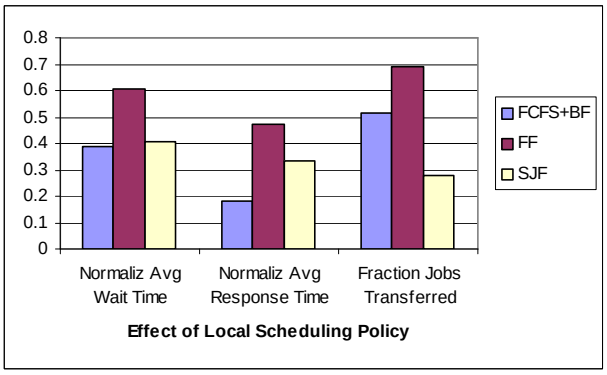

Figure 5: Effects of local scheduler policy on grid performance for the S-I job migration algorithm.

Results indicate that the choice of local scheduler has a significant effect on grid performance. For example, FCFS+BF minimizes the average wait and response times for our test workload; however, SJF transfers the fewest number of jobs. Grid efficiency (not shown) is not affected by the local scheduling policy since the workload is under-subscribed, as discussed in Section 4.1. Nevertheless, even the slowest local scheduler (SJF) with superscheduling still outperforms the local run by more than a factor of two in terms of average response time.

5 Conclusions and Future Work

Computational grids hold great promise in utilizing geographically separated heterogeneous resources to solve large-scale complex scientific problems. However, a number of major technical hurdles, including distributed resource management and effective job scheduling, stand in the way of realizing the true potential of grid computing. In this work, we proposed a novel superscheduler architecture and investigated its performance across a number of key metrics in a simulation environment. Three distributed job migration algorithms were introduced: sender-initiated (S-I), receiver-initiated (R-I), and symmetrically-initiated (SY-I). The S-I approach actively attempts to migrate jobs whose resource requirements cannot be quickly satisfied on the local system. R-I scheduling, on the other hand, uses a more passive strategy where queued jobs must wait for remote systems to advertise their availability. The SY-I algorithm is a hybrid scheme,

combining elements of both passive and active job migration. We also investigated a centralized architecture to compare distributed performance with a global approach; however, this methodology has practical limitations in terms of fault tolerance and scalability. Finally, an idealized (and unattainable) algorithm was presented to establish an upper bound on superscheduler performance.

A critical aspect of this research was the set of real and synthetic workloads used in our experiments. Real workloads were collected from three leading supercomputing centers over the same six-month period, allowing us to accurately simulate the potential contribution of an intelligently implemented superscheduler. Additionally, sophisticated statistical methods were used to generate synthetic data for parameter studies of varying workload conditions.

Several key metrics were used in our experiments to evaluate the effectiveness of the proposed superscheduler and job migration algorithms. Results demonstrated the tremendous potential of an effectively implemented grid environment, even for a small number of participating architectures. For example, comparing the local scheme with S-I for six-month data, the average wait time was reduced by a factor of 2.5, along with a 30% improvement in deviation and a 1.5X reduction in the average response time. Comparing individual job migration schemes, we found that the SY-I approach struck the best balance between optimizing performance and reducing job transfers.

The synthetically generated workload data allowed us to perform experiments for both heavily- and lightly-loaded system conditions. Results demonstrated that for a larger heterogeneous six-machine configuration, the advantages of the superscheduler becomes more pronounced, even for the lightly-loaded case. For example, compared to local scheduling, the S-I approach improved the average wait time by factors of 5.9 and 21, and the average response time by factors of 5.0 and 1.5 for the heavy and light workloads, respectively. Furthermore, grid efficiency increased from 65% to 85% under heavy workload conditions.

Finally, we investigated the relationship between the superscheduler and three different local scheduling policies. Results showed that first-come-first-serve with backfilling gave the best performance in terms of average wait and response times; however, all three local scheduling approaches together with a superscheduler improved overall performance compared with locally isolated systems. Our results demonstrated that superscheduling can deliver substantial performance gains; however, it is important to realize that many important questions have not been addressed in this preliminary study. Future work will build on our simulation environment to include critical parameters, such as job migration overhead, grid network costs, superscheduler scalability, fault tolerance, multi-resource requirements, and architectural heterogeneity. Additionally, we plan to investigate the practical implementation requirements necessary to deploy a distributed superscheduler into a real-world grid environment.

Acknowledgements

The authors would like to gratefully thank LBNL, LLNL, and SDSC for providing the batch job trace files. The first two authors were supported by Director, Office of Computational and Technology Research, Division of Mathematical, Information, and Computational Sciences of the U.S. Department of Energy under contract number DE-AC03-76SF00098.

References

[1] Global Grid Forum. http://www.gridforum.org.

[2] Top500 Supercomputer Sites. http://www.top500.org.

[3] M. Arora, S.K. Das, and R. Biswas. A de-centralized scheduling and load balancing algorithm for heterogeneous grid environments. In *Workshop on Scheduling and Resource Management for Cluster Computing*, pages 499–505, 2002.

[4] C. Ernemann, V. Hamscher, U. Schwiegelshohn, A. Streit, and R. Yahyapour. On advantages of grid computing for parallel job scheduling. In *2nd International Symposium on Cluster Computing and the Grid*, pages 39–46, 2002.

[5] D.G. Feitelson. Packing schemes for gang scheduling. In *2nd Workshop on Job Scheduling Strategies for Parallel Processing*, volume LNCS 1162, pages 89–100, 1996.

[6] D.G. Feitelson, L. Rudolph, U. Schwiegelshohn, K.C. Sevcik, and P. Wong. Theory and practice in parallel job scheduling. In *3rd Workshop on Job Scheduling Strategies for Parallel Processing*, volume LNCS 1291, pages 1–34, 1997.

[7] D.G. Feitelson and A.M. Weil. Utilization and predictability in scheduling the IBM SP2 with backfilling. In *12th International Parallel Processing Symposium*, pages 542–546, 1998.

[8] Ian Foster and Carl Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, San Francisco, CA, 1999.

[9] H. Franke, J. Jann, J.E. Moreira, P. Pattnaik, and M.A. Jette. A evaluation of parallel job scheduling for ASCI Blue-Pacific. In *Proc. SC99*, CD-ROM, 1999.

[10] V. Hamscher, U. Schwiegelshohn, A. Streit, and R. Yahyapour. Evaluation of job-scheduling strategies for grid computing. In *1st International Workshop on Grid Computing*, volume LNCS 1971, pages 191–202, 2000.

[11] J. Jann, P. Pattnaik, H. Franke, F. Wang, J. Skovira, and J. Riordan. Modeling of workload in MPPs. In *3rd Workshop on Job Scheduling Strategies for Parallel Processing*, volume LNCS 1291, pages 95–116, 1997.

[12] M.A. Johnson and M.R. Taaffe. Matching moments to phase distributions: Mixtures of Erlang distributions of common order. *Stochastic Models*, 5(4):711–743, 1989.

[13] J. Krallmann, U. Schwiegelshohn, and R. Yahyapour. On the design and evaluation of job scheduling algorithms. In *5th Workshop on Job Scheduling Strategies for Parallel Processing*, volume LNCS 1659, pages 17–42, 1999.

[14] W. Leinberger, G. Karypis, V. Kumar, and R. Biswas. Load balancing across near-homogeneous multi-resource servers. In *9th Heterogeneous Computing Workshop*, pages 60–71, 2000.

[15] R.D. Nelson, D.F. Towsley, and A.N. Tantawi. Performance analysis of parallel processing systems. *IEEE Transactions on Software Engineering*, 14(4):532–540, 1988.

[16] J.M. Schopf. Ten actions when superscheduling. http://www.gridforum.org/documents/GFD, 2001.