Prospects of Collaboration between Compute Providers by means of Job Interchange

Christian Grimme, Joachim Lepping, and Alexander Papaspyrou

Robotics Research Institute, Dortmund University, 44221 Dortmund, Germany
email: {christian.grimme, joachim.lepping, alexander.papaspyrou}@udo.edu

Abstract. This paper empirically explores the advantages of the collaboration between different parallel compute sites in a decentralized grid scenario. To this end, we assume independent users that submit their jobs to their local site installation. The sites are allowed to decline the local execution of jobs by offering them to a central job pool. In our analysis we evaluate the performance of three job sharing algorithms that are based on the commonly used algorithms First-Come-First-Serve, EASY Backfilling, and List-Scheduling. The simulation results are obtained using real workload traces and compared to single site results. We show that simple job pooling is beneficial for all sites even if the local scheduling systems remain unchanged. Further, we show that it is possible to achieve shorter response times for jobs compared to the best single-site scheduling results.

1 Introduction

In recent years the demand for computing power has increased significantly as a growing number of science areas rely on extensive simulations, like biotechnology, high energy physics [15], or climate research [13]. In order to provide an on-the-spot support for high performance computing power many Massively Parallel Processing (MPP) systems have been put into service. Those machines are extensively shared by a local user community that submits jobs which are adapted to the local site capabilities, for instance in the jobs' degree of parallelism. As these user communities are still growing, the utilization of such systems is typically very high and computing power is becoming a limited resource.

This, however, is twofold: on the one hand, a high utilization is desirable for the system provider as it allows a profitable operation of the MPP system. On the other hand, this entails longer wait times for the execution of the users' jobs, decreasing customer satisfaction. To ameliorate this, system providers introduce additional queues, restrictive quotas and partitions, or even employ more sophisticated scheduling concepts that allow a flexible prioritization of users, see Franke et al. [11, 10]. However, all those approaches favor a certain group of users at best, but do not prevent congestions induced by the overall high demand for compute power.

This has initiated the concept of Computational Grids: the loose coupling [8] of independent and typically diverging MPP installations to a federated source

of ubiquitous computing power. All the more, the world wide growing network infrastructure allows an interchange of jobs among sites with an almost evanescent time delay. In this way locally submitted jobs may—in case of a high local load— also be executed on another MPP system in the federation, utilizing idle resources. However, new scheduling problems arise in the Grid context as interchanging jobs between different sites while keeping or enhancing the performance for all local user communities is a non-trivial problem.

In this paper we study the advantages for local user communities and system providers induced by joining a decentralized Computational Grid where jobs can be interchanged between different MPP installations. We evaluate the concept of a global job pool which can be used by all participating sites to offer or accept available jobs and restrict our evaluation to a small set of sites with simple job-sharing enabled adaptations of common scheduling algorithms.

The rest of the paper is organized as follows. Current developments in the area of research are reviewed in Section 2. Scheduling concepts for local and distributed computing are presented in Section 3. Our model of MPP systems in a decentralized Grid along with job policies and employed algorithms is introduced in Section 4. The experimental setup is described in Section 5, and a detailed evaluation of different heterogeneous site configurations is presented in Section 6. Our paper ends with a conclusion and an outlook on future work in Section 7.

2 Background

Since the first ideas of metacomputing have been formulated by Smarr and Catlett [21], many research activities deal with the development of concepts to connect world wide distributed resources. Nowadays, metacomputing is replaced by the new ideas of Grid Computing [9] which facilitates the aggregation and sharing of heterogeneous resources distributed over large geographical areas. Although the original Grid idea extends the term "'resource"' beyond[1] MPP systems, the area of Computational Grids as the federations of several high performance compute sites is supposed to be the most developed one. To satisfy the growing need for compute power Grids are considered an integral part of the usual business in the near future [22].

Admittedly, every system provider in such a federation aims to retain the control over his system. Futhermore, his main interest lies in reaching short response times for the local user community while keeping the utilization of his machine as high as possible. Against this background it is unrealistic to assume a model where a global resource management system is allowed to decide over the local resources. As such, it is widely agreed that Grid Computing can only be applied beneficially if the resource management and scheduling is appropriately adapted to the environment. Therefore, almost all current Grid projects spend special effort in the development of smart scheduling strategies.

[1] For example, storage systems and network links, as well as special purpose devices such as telescopes or satellites.

However, there is a lack of studies that empirically identify potential advantages of the collaboration of compute sites using a decentralized structure and an indirect communication model. A first study by Hamscher et al. [14] shows potential advantages that result from different job sharing strategies. However, their empirical results are not meant to be complete as they mainly focus on the conception of job interchange procedures. Their paper only considers two real workloads and they do not give exclusive single-site scheduling results for comparison. Further, Ernemann et al. identify improvements for the average weighted response time objective assuming hierarchical centralized scheduling structures [4] in multi-site computing [3].

Among the large variety of studies on load balancing Lu et al. [17] developed algorithms for distributed load balancing in a grid environment assuming an additional communication delay. However, their approach requires that the whole local installation is controlled by a central scheduler and their evaluation is founded on statistical workload models that can hardly be compared with real workload traces originating from real-world job submissions. The latter also applies to England and Weissman [2], who analyzed costs and benefits of load sharing, limiting themselves to site configurations and reviewing average slowdown only.

On the theoretical side, Schwiegelshohn [19] proves that the quality of two schedules can be improved concerning the total completion time objective if an interchange of jobs is allowed during the creation of the schedules.

3 Scheduling Concepts for Local and Distributed Computing

In this section, we present the scheduling concepts underlying our evaluations. First, three standard scheduling strategies are briefly recapitulated for single-site MPP systems. Then, we introduce three very simple job-sharing algorithms that are based on the previously introduced standard approaches.

3.1 Local Job Scheduling

The single site scenario has been discussed in research for quite a long time. Various algorithms are available that try to minimize the response time for jobs and to utilize the system properly. The algorithms that we chose for evaluation purposes are most commonly applied in practice. These algorithms are in detail:

First Come First Serve (FCFS) starts the first job of the waiting queue whenever enough idle resources are available. Despite the very low utilization that is produced in the worst case this heuristic works well in practice [20].

List-Scheduling (LIST) as introduced by Graham [12] serves as a template for our LIST algorithm. List-Scheduling is based on a list that is ordered according to static job weights. The scheduler selects from the list the job with the highest weight. In our case, the job arrival time (or release date)

is used as the list's weight. The scheduler scans the list in ascending order until finding a job that can be processed immediately on the currently idle resources. Therefore, this algorithm is similar to the *First-Fit* strategy [1].

EASY Backfilling (EASY) requires that a runtime estimation is provided for each job by its user [16]. If the first job of the waiting queue cannot be started immediately, the algorithm allows an allocation of another job if it does not delay the earliest possible start time of the *first job*.

3.2 Distributed Job Scheduling

For local scheduling systems, the aforementioned algorithms have been established as quasi-standard, since they work well for a single parallel computer but they are not optimized for any Grid interaction.

Scheduling in Grid systems can mainly be divided into two distinct layers. The local layer is responsible for the scheduling of jobs in the current waiting queue onto the available local resources. This allows for example the realization of priorities for different user groups which can be formulated by the system provider. Albeit in a very simplistic way, the aforementioned local job scheduling algorithms cover this area, and work well for the single parallel computer view to such an extent that they have been established as quasi-standard for many MPP installations.

The second, higher layer is responsible for the interaction with other Grid participants and might perform (among others) discovery, negotiation, and the migration of jobs to remote sites. This functionality, however, is not provided by the local scheduling algorithms and is subject to research [22].

For the structure of these two layers, a large variety of concepts exists. These can be classified into the following two categories:

Hierarchical Scheduling The hierarchical scheduling structure utilizes a central scheduler—the so- called meta-scheduler—which accepts the submission of workload independent of its origin and decides on the distribution among available sites. After the decision is made, the given workload is assigned to the local schedulers which in turn are responsible for the assignment on local resources. The advantage of this approach is that different strategies and policies may be applied for the meta-scheduler and the local scheduling systems. The main disadvantage arises from the centralization concept of this approach: it inherently lacks scalability and, since having a single-point-of-failure, has a very bad fault-tolerance—if the meta-scheduler breaks down, workload handling ceases completely.

Decentralized Scheduling In decentralized scheduling systems the distributed local schedulers interact with each other in a direct or indirect fashion. As there is no single-point-of- failure, the system is more reliable: possible breakdowns of one participant does not necessarily impair the performance of the whole system. The possibility of realizing different policies on the local and global scheduling

layers is given as in the hierarchical case, admittedly requiring more sophisticated scheduling concepts due to the lack of a global system view.

As mentioned above, there exist two possibilities to establish the communication of distributed schedulers. In *direct communication* schedulers can transfer/accept jobs to/from remote sites directly. This may imply searching for remote sites and/or keeping a list of potential communication partners. In case that local job execution is not possible, the scheduler may then delegate the execution one of the known remote sites.

The second model is *indirect communication* by sharing a central repository. Whenever local execution is not possible, jobs are offered this repository —for example realized as a job pool— which then can be used by the other participants as a second queue for occupying currently idle local resources. Since this concept is used for the evaluations in this paper, the details are given in Section 4.3.

4 Model

In this Section, we detail the setup and actual environment model that we use for our evaluation. First, we give an description of the site scenario. Then, we describe the scheduling problem for a single site and specify the assumptions that are made for the computational jobs and the formal description.

4.1 Site and Machine Environment

We assume a Computational Grid consisting of $|K|$ independent MPP systems, following referred to as *sites*. Each site $k \in K$ is modeled by m_k parallel processors which are identical such that a parallel job can be allocated on any subset of these machines. Splitting jobs over multiple sites (multi-site computation) is not allowed.

Moreover, we assume that all sites only differ in the number of available processors, but not in their speed. As we focus on the job-sharing algorithms, the differences in execution speeds are neglected.

Still, every site has its own local user demand for computational resources which is reflected by the sites' originating workload. This includes the submission characteristics, but also the adaptation of the submitted jobs' resource demand to the local configuration. That is, jobs that are submitted to the local site scheduler may not be accepted for execution elsewhere because of their resource demand being oversized for some or all of the other sites.

4.2 Job Model

We assume rigid[2] parallel batch jobs for our analysis, which are dominant on most MPP systems. The user provides the number of required machines m_j at

[2] Neither moldable nor malleable, requiring concurrent and exclusive access to the requested resources.

the release date r_j of the job. The completion time of job $j \in \pi_k$ in schedule S_k on site k is denoted by $C_j(S_k)$. As preemptions are not allowed in many MPP systems [6], each job starts its execution at time $C_j(S_k) - p_j$.

Job scheduling on MPP systems is an online problem as jobs are submitted over time and the processing time p_j of job j is not available at the release date r_j. As the release and processing times are unknown, the problem is often classified as non-clairvoyant online scheduling, see Motwani et al. [18]. In the following, we denote the set of jobs that are submitted to a *local* site k by τ_k. Further, system administrators often require users to provide worst case estimates $\bar{p}_j$ of the processing time p_j to determine faulty jobs whose processing times exceed the estimate.

Furthermore, data management of any files is neglected in this paper. In our multi-site scenario, a job can be transmitted to a common pool without any communication penalty while in a real implementation the transport of data requires additional time. The communication cost can often be hidden by pre- and postfetching before and after the execution. In such a case the overhead is not necessarily part of the scheduling process.

4.3 Job-Sharing Algorithms

In order to establish indirect communication between sites we introduce a central job pool from/to which local schedulers can receive/transfer jobs. All sites joining this collaboration scenario are connected to the central pool as shown in Figure 1. In our model, the pool is realized as a global job queue where all jobs are ordered according to their insertion time and regardless of their local submission time. The interaction with the pool requires a policy that decides, whether a job is offered to the job pool or is kept in the local queue. For all our job-sharing algorithms we apply the following universal policy:

> *Whenever a job from the local queue cannot be executed immediately on the local processors, it is offered to the global job pool and removed from the local queue.*

In the following, we detail how the interaction between local queue and global job pool is implemented by the different job-sharing (JS) algorithms. Our three job-sharing algorithms are based on the template which is described in Algorithm 1. In the beginning, FCFS, EASY, or LIST is applied locally (see Line 2). Following (see Line 4), all jobs that have been considered for the current schedule during the previous step —including all potential backfilling candidates—but could not be scheduled immediately are moved to the pool. If no job from the local queue could be scheduled, the algorithm is executed again, using the global job pool (see Line 7).

For the modified FCFS policy, only the first job of each queue is considered respectively. In the case of adapted EASY policy, the application of the algorithm is completely separate for each queue. As such, backfilling is repeated on the global job pool if and only if local scheduling failed. Note that only the currently

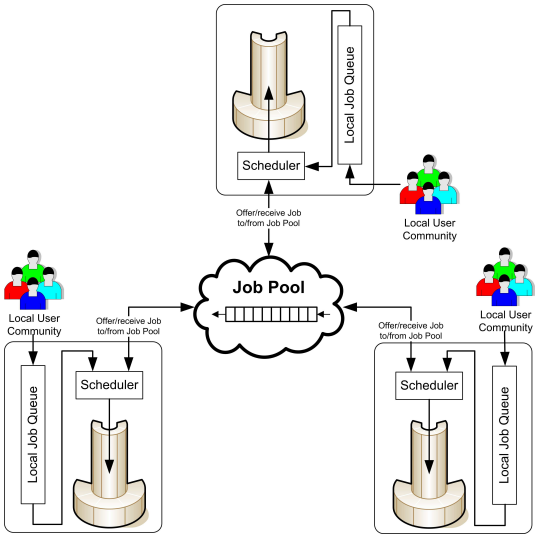


Fig. 1. Decentralized scheduling scenario sharing a central job pool.

Algorithm 1 General template for job-sharing algorithm.

Preconditions: a local job queue l, a global job pool g

1: *algorithm* $\in$ {FCFS, EASY, LIST}
2: apply *algorithm* to l
3: **for all** considered, but locally non-schedulable jobs **do**
4: offer job to g
5: **end for**
6: **if** no job could be scheduled locally **then**
7: apply *algorithm* to g
8: **end if**

treated queue is used for finding backfilling candidates, and not the union of locally and globally available jobs. The changed LIST policy also iterates over the queue stopping as soon as a schedulable job has been found, and also resorts to the global job pool if no local job could be fit into the schedule.

In the remainder of this paper we will denote the FCFS based job-sharing algorithm as **FCFS-JS**, the EASY based as **EASY-JS**, and the LIST based algorithm as **LIST-JS** respectively.

5 Experimental Setup

Before presenting our detailed evaluation results we give an overview on how our experiments have been setup and introduce performance objectives and used input data.

5.1 Performance Objectives

In order to measure the schedule quality and to quantify the effect on jobs interchange we define several objectives. Remember that we denote the set of jobs that have been submitted locally to site k by τ_k and all jobs that have been actually processed on site k by π_k.

Squashed Area and Utilization The first two objectives are *Squashed Area* SA_k and *Utilization* U_k, both specific to a certain site k. They are measured from the start of the schedule S_k, that is $\min_{j \in \pi_k}\{C_j(S_k) - p_j\}$ as the earliest job start time, up to its makespan $C_{max,k} = \max_{j \in \pi_k}\{C_j(S_k)\}$, that is the latest job completion time and thus the schedule's length.

SA_k denotes the overall resource usage of all jobs that have been executed on site k, see Equation 1.

$$\mathrm{SA}_k = \sum_{j \in \pi_k} p_j \cdot m_j \tag{1}$$

U_k describes the ratio between overall resource usage and available resources after the completion of all jobs $j \in \pi_k$, see Equation 2.

$$\mathrm{U}_k = \frac{\mathrm{SA}_k}{m_k \cdot \left(C_{max,k} - \min\limits_{j \in \pi_k}\{C_j(S_k) - p_j\} \right)} \tag{2}$$

U_k describes the usage efficiency of the site's available machines. Therefore, it is often serving as a schedule quality metric from the site provider's point of view.

However, comparing single-site and multi-site utilization values is illicit: since the calculation of U_k depends on $C_{max,k}$, valid comparisons are only admissible if $C_{max,k}$ is approximately equal between the single-site and multi-site scenario. Otherwise, high utilizations may indicate good usage efficiency, although the corresponding $C_{max,k}$ value is very small and shows that only few jobs have been computed locally while many have been delegated to other sites for remote execution.

As such, we additionally introduce the *Change of Squashed Area* $\Delta\mathrm{SA}_k$, which provides a makespan- independent view on the utilization's alteration, see Equation 3.

$$\Delta\mathrm{SA}_k = \frac{\mathrm{SA}_k}{\sum\limits_{j \in \tau_k} p_j \cdot m_j} \tag{3}$$

From the system provider's point of view this objective reflects the real change of the utilization when jobs are shared between site compared to the local execution.

Response Time For our third objective we switch focus towards a more users-centric view and consider the *Average Weighted Response Time* AWRT_k relative to all jobs $j \in \tau_k$ that have been initially submitted to site k, see Equation 4. Note that this also respects the execution on remote sites and, as such, the completion time $C_j(S)$ refers to the site that executed job j.

$$\mathrm{AWRT}_k = \frac{\sum\limits_{j \in \tau_k} p_j \cdot m_j \cdot (C_j(S) - r_j)}{\sum\limits_{j \in \tau_k} p_j \cdot m_j} \tag{4}$$

A short AWRT describes that on average users do not wait long for their jobs to complete. According to Schwiegelshohn et al. [20], we use the resource consumption ($p_j \cdot m_j$) of each job j as the weight. This ensures that neither splitting nor combination of jobs can influence the objective function in a beneficial way.

Migration Rate Finally, we measure the amount of migration in the multi-site scenarios. To this end, we introduce a migration matrix M that shows the ratio of job dissemination with respect to the original submission site, see Equation 5.

$$M = \begin{bmatrix} \frac{|\pi_{kk}|}{|\tau_k|} & \frac{|\pi_{kl}|}{|\tau_k|} & \cdots \\ \frac{|\pi_{lk}|}{|\tau_l|} & \frac{|\pi_{ll}|}{|\tau_l|} & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix} \tag{5}$$

In this matrix we apply the following notation:

- τ_k Total amount of jobs that have been originally submitted to site k.
- π_{lk} Amount of jobs that have been originally submitted to site l but have been executed on site k.

Remember that the rows denote the site where the jobs have been submitted to while columns specify the actual execution sites. Next, we detail the data source utilized for our investigation.

5.2 Input Data

The Parallel Workloads Archive[3] provides job submission and execution traces recorded on real-world MPP system sites, each of which containing information on relevant job characteristics. For our evaluations, we restricted the set of used workloads to those which contain valid runtime estimations, since the EASY and EASY-JS algorithms depend on this data. Furthermore, we applied various pre-filtering steps to the original—partially erroneous—data: we discard jobs with invalid release dates ($r_j < 0$), processing times ($p_j \leq 0$), node requests ($m_j \leq 0$), and estimates ($\bar{p}_j \leq 0$), as well as unsatisfiable resource demands ($m_j > m$). Furthermore, we assume overuse interdiction, effectively disallowing jobs to run longer than the user provided runtime estimation ($p_j = \min\{p_j, \bar{p}_j\}$). Some details of the examined cleaned traces are given in Table 1.

[3] `http://www.cs.huji.ac.il/labs/parallel/workload/`

Identifier	n	m	Months	Site Setup							SA
				1	2	3	4	5	6	7	
KTH-11	28479	100	11	X				X	X		2017737644
CTC-11	77199	430	11						X		8279369703
LANL96-11	57715	1024	11					X			16701881984
LANL96-13	67043	1024	13				X				19467626464
LANL96-24	110769	1024	24		X	X				X	35426509344
SDSC00-11	29810	128	11	X				X	X		2780016139
SDSC00-24	54006	128	24			X				X	6656350775
SDSC03-11	65584	1152	11								23337438904
SDSC03-13	78951	1152	13				X				27410557560
SDSC03-24	176268	1152	24		X					X	54233113112
SDSC05-13	84876	1664	13				X				34347849760

Table 1. Details of the shortened workloads obtained from the Parallel Workload Archive and the corresponding examined site setups 1-7.

Naturally, the total number of available processors differs in workloads which makes it possible to model unequally sized site configurations. Further, the original workloads record time periods of different length. In order to be able to combine different workloads in a multi-site simulations we shortened the workloads to the minimum required length of all participating workloads. To reflect different site configurations (e.g. small machine and large machine) we combine only workloads that represent a long record period to obtain meaningful results. Therefore, we created shortened versions of the LANL96, SDSC00, SDSC03, and SDSC05 workloads[4]. The resulting seven different site configurations are listed in Table 1.

Note that we do not shift the traces regarding their originating timezones. We restrict our study to workloads which are all submitted within the same timezone. Therefore, the known diurnal rhythm of job submission is similar for all sites in our scenario and time shifts cannot be availed to improve scheduling. In a global grid the different timezones even benefit the job scheduling as idle machines at night can be used by jobs from peak loaded sites at noon, see Ernemann at al. [5]. As we cannot benefit from timezone shifts the presented results might be even better in a global grid.

6 Evaluation

We now provide results obtained for the single site case and for the different site setups which allow the interchange of jobs.

6.1 Reference Results on the Local Sites

As it is our aim to compare the introduced simple job sharing mechanism to the performance of standard algorithms in a single site environment, we show these results first. In Table 2 the AWRT, U, and C_{max} are given when the three

[4] `http://www-ds.e-technik.uni-dortmund.de/~lepping/workloads/`

standard algorithms FCFS, EASY, and LIST are applied to the 11 different workloads and the corresponding sites given in Table 1. It becomes obvious

	FCFS			EASY			LIST		
Identifier	AWRT	U	C_{max}	AWRT	U	C_{max}	AWRT	U	C_{max}
KTH-11	418171.98	68.67	29381344	75157.63	68.72	29363626	80459.27	68.72	29363626
CTC-11	58592.52	65.70	29306682	52923.79	65.70	29306682	53285.69	65.70	29306682
LANL96-11	13886.61	55.65	29306900	12441.87	55.65	29306355	13112.69	55.65	29307464
LANL96-13	14064.40	56.24	33801984	12467.31	56.24	33802986	13205.34	56.24	33802986
LANL96-24	12265.92	55.54	62292428	11105.03	55.54	62292428	11678.88	55.54	62292428
SDSC00-11	266618.82	71.53	30361813	73606.71	73.94	29374554	74529.44	73.96	29364791
SDSC00-24	2224462.43	75.39	68978708	112050.78	81.78	63591452	120173.69	82.26	63618662
SDSC03-11	72417.87	68.58	29537543	50980.20	68.74	29471588	48007.66	68.87	29413625
SDSC03-13	82149.66	69.92	34030926	54704.54	70.12	33932665	52013.62	70.14	33921090
SDSC03-24	166189.54	73.22	64299555	71021.03	73.91	63696120	72237.20	73.91	63695942
SDSC05-13	72690.44	61.08	33793591	57000.60	61.08	33793591	55190.04	61.08	33793591

Table 2. AWRT (in seconds), U (in %), and C_{max} (in seconds) for the different workloads. The objective values are shown for the three algorithms and only single site execution ($K = 1$).

that EASY outperforms the FCFS algorithm in terms of AWRT and U, what is well known from the literature, see Feitelson and Weil [7]. However, for some workloads the LIST algorithm performs even better than EASY which might be due to the characteristics of some workloads. In general, both queue-iterating procedures are superior to FCFS while there is no significant advantage of LIST scheduling compared to EASY.

6.2 Interchange of Jobs between Two Sites

To investigate the effect of site collaboration applying the job-sharing approach described in Section 4.3 we define three simple setups of two sites respectively connected to the central job-pool. Every site in this setup corresponds to the machine represented by the workload trace it processes.

In Table 3 the results of our examined setup with two collaborating sites are listed. The objectives are given for each job-sharing algorithm compared to the results of the corresponding single-site algorithm (denoted by Δ) and the best single-site algorithm (denoted by Δ_b) respectively. This distinction has been made to illustrate the effect of applying job-sharing algorithms as well as to review whether a general improvement compared with the single-site approach is possible. To analyze the behavior systematically, we first discuss the results compared to the related single-site scheduling algorithms.

For the constellation of two small sites, represented by KTH and SDSC00 workloads, a significant improvement in AWRT with respect to the corresponding local setup can be observed. This is achieved even without affecting the utilization or squashed area significantly. However, this does not hold for two large sites, processing workloads like LANL96 and SDSC03. Especially for the

Site Setup	1		2		3	
Workload	KTH-11	SDSC00-11	LANL96-24	SDSC03-24	LANL96-24	SDSC00-24
m	100	128	1024	1152	1024	128
k	1	2	1	2	1	2
FCFS-JS						
AWRT_k	128043.42	139380.14	33487.04	67584.37	14277.15	40764.89
U_k	70.12	72.94	64.59	65.44	58.91	55.57
$C_{max,k}$	29571802	2957587	62440050	64151102	62303391	63638145
SA_k	2073589120	2724164663	41300491136	48359131320	37584312832	4498547287
Δ AWRT_k	69.38	47.72	100.00	59.33	-16.40	98.17
Δ U_k	2.11	0.63	16.31	-10.62	6.07	-26.71
Δ SA_k	2.77	-2.01	16.58	-10.83	6.07	-32.42
Δ $C_{max,k}$	-0.65	2.59	-0.24	0.23	-0.02	7.74
Δ_b AWRT_k	-70.37	-89.36	-201.55	4.51	-28.56	63.62
Δ_b U_k	2.04	-2.68	16.31	-11.46	6.07	-32.47
Δ_b SA_k	2.77	-2.01	16.58	-10.83	6.09	-32.42
Δ_b $C_{max,k}$	-0.71	-0.68	-0.24	-0.71	-0.02	-0.07
$M =$	82.86 13.95	17.14 86.05	88.69 9.51	11.31 90.49	97.35 15.41	2.65 84.59
EASY-JS						
AWRT_k	66115.56	59015	13579.91	54019.88	12583.95	41441.88
U_k	68.95	74.80	63.20	67.24	58.33	60.28
$C_{max,k}$	29363626	29364791	62303135	63694187	62292428	63618367
SA_k	2024644907	2773108876	40318878368	49340744088	37204951541	4877908578
Δ AWRT_k	12.03	19.82	-22.28	23.67	-13.32	63.01
Δ U_k	0.34	-0.21	13.79	-9.02	5.02	-26.75
Δ SA_k	0.34	-0.25	13.81	-9.02	5.02	-26.72
Δ $C_{max,k}$	0.00	0.03	-0.02	0.00	0.00	-0.04
Δ_b AWRT_k	12.03	19.82	-22.28	23.67	-13.32	63.01
Δ_b U_k	0.34	-0.21	13.79	-9.02	5.02	-26.75
Δ_b SA_k	0.34	-0.25	13.81	-9.02	5.02	-26.72
Δ_b $C_{max,k}$	0.00	0.03	-0.02	0.00	0.00	-0.04
$M =$	79.63 15.04	20.37 84.96	88.69 9.99	13.37 90.01	97.21 17.01	2.79 82.99
LIST-JS						
AWRT_k	61016.52	58354.18	13509.25	51150.19	13001.56	38655.67
U_k	68.98	74.84	63.45	67.01	59.11	54.04
$C_{max,k}$	29363626	29339969	62317990	63694095	62303391	63588898
SA_k	2025505127	2772248656	49172388088	49172388088	37711932960	4370927159
Δ AWRT_k	24.16	21.70	-15.93	29.17	-11.52	66.75
Δ U_k	0.38	-0.19	14.24	-9.33	6.43	-34.30
Δ SA_k	0.38	-0.28	14.29	-9.33	6.45	-34.33
Δ $C_{max,k}$	0.00	0.08	-0.04	0.00	-0.02	0.05
Δ_b AWRT_k	18.82	20.72	-21.65	27.73	-17.08	65.50
Δ_b U_k	0.38	-0.16	14.24	-9.33	6.43	-34.33
Δ_b SA_k	0.38	-0.28	14.29	-9.33	6.45	-34.33
Δ_b $C_{max,k}$	0.00	0.12	-0.04	0.00	-0.02	0.00
$M =$	77.77 19.61	22.23 80.39	83.20 11.35	16.80 88.65	96.62 14.88	3.38 85.12

Table 3. Objective values for site setups 1-3 each with two participants ($K = 2$). AWRT_k and $C_{max,k}$ values are given in seconds while the U_k, SA_k, and all comparative objectives (Δ, Δ_b, and matrix M) are given in %.

FCFS-JS strategy, the AWRT of site 1 increases drastically. It is obvious that in the non-cooperating case and the LANL96 workload short AWRT values are achieved for all standard algorithms. This may be due to a comparably low utilization and a special manner of use where only jobs are allowed that (a) require a number of processors equal to a power of two and (b) use at least 32 processors. For the job-sharing algorithms one may argue that even slight changes in uti-

lization as well as violations of the policies may result in a significant growth of AWRT. Thus, for a low utilized system the job-sharing may be disadvantageous.

A completely different behavior can be observed when a small site (SDSC00) cooperated with a large site (LANL96). Here, the small site migrates a higher percentage of its jobs to the large site than vice versa. Obviously, the AWRT of the small site improves strongly while the utilization decreases. Concerning the large site similar effects as in the case of two large sites, discussed above, are observed. Due to the large amount of jobs migrated from the small to the large site the AWRT becomes longer while the utilization increases. However, the impact of these jobs is smaller than in the previous case which may again result from the use policy of LANL96 so that small jobs from SDSC00 can be used to fill a gap.

The comparison of the job-sharing algorithms with the results for EASY as the best single-site algorithm yields structurally similar results. However, for two small sites, the EASY-JS and LIST-JS approaches even outperform EASY with respect to AWRT. Summarizing we can formulate the following statement from our experiments:

1. Adapting the local algorithms to job-sharing can lead to an overall AWRT improvement for sites with high utilization and rather equal size.
2. Combining a site with low utilization and short AWRT with a more loaded one resembles a load balancing behavior which may involve a significant deterioration concerning AWRT for the less loaded site.
3. For disparate sized sites a significant higher percentage of job migrates from the small to the large site than vice versa. While many jobs from the large site cannot be executed on the small site due to oversized resource demands, jobs from the small site may fill gaps in the schedule of the large site.

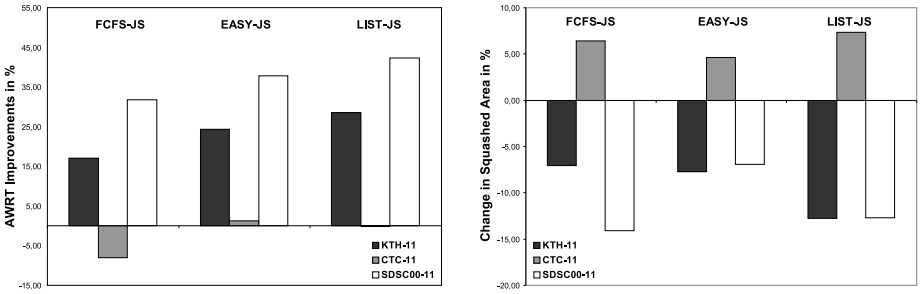
6.3 Interchange of Jobs between Three Sites

For the setup with three collaborating sites we omit a complete presentation of the obtained simulation results and refer to the accompanying Table 4 and 5 in the appendix. Instead we present figures of AWRT improvements and the change in SA compared to EASY as the best single-site algorithm.

Again, we find that job-sharing yields shorter AWRT values for all participating sites. However, the utilization at the largest site (CTC) increases in a load balancing behavior, see Figure 2, but this does not affect the AWRT for EASY-JS and LIST-JS.

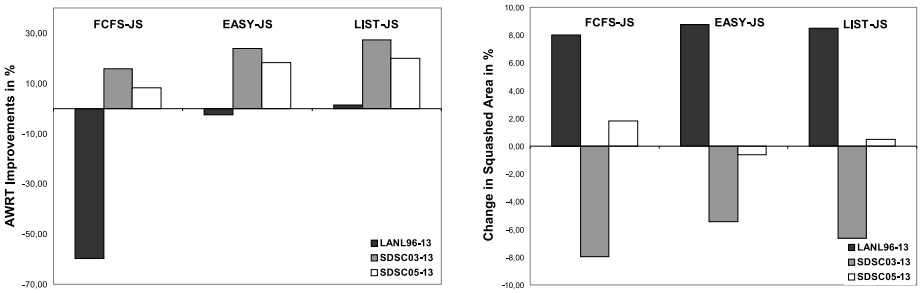
Since the sharing of jobs results in less utilization on two sites and more on one site while preserving or even improving the AWRT, this can be interpreted as benefit of job-sharing: on the less utilized site the system providers have more capabilities to serve additional customers.

A similar effect can be observed in Figure 3 for three large collaborating sites. Here, however, the aforementioned characteristics of LANL96 seem to influence the results. Nevertheless, for EASY- JS and LIST-JS we can identify a constant



(a) Improvements of AWRT in % (b) $\Delta \mathrm{SA}_b$ in %

Fig. 2. AWRT and Δ_bSA for setup 6 with KTH-11 (m_1=100), CTC-11 (m_2=430), and SDSC00-11 (m_3=128) workloads and all three algorithms FCFS-JS, EASY-JS, and LIST-JS.



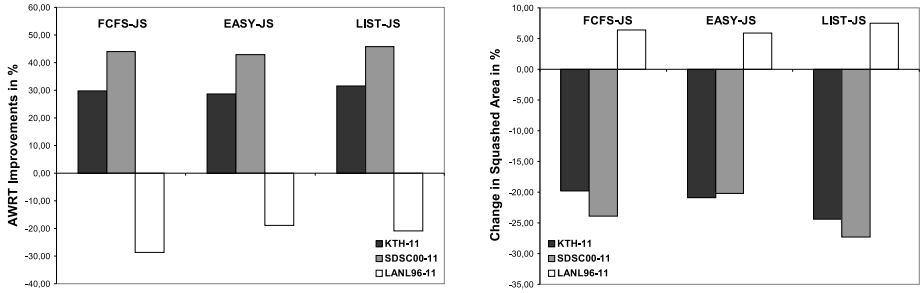
(a) Improvements of AWRT in % (b) $\Delta \mathrm{SA}_b$ in %

Fig. 3. AWRT and Δ_bSA for setup 4 with LANL96-13 (m_1=1024), SDSC03-13 (m_2=1152), and SDSC05-13 (m_3=1664) workloads and all three algorithms FCFS-JS, EASY-JS, and LIST-JS.

or improved AWRT, while U_1 increases for LANL96 and remains rather constant or decreases for the other sites.

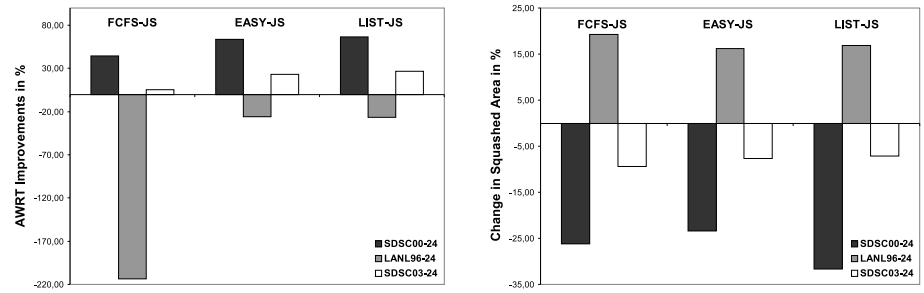
Analogous to the two site setups, the results shown in Figure 4 for two small and one large sites yield a significant improvement of the small sites' AWRT to the expense of the large. It is possible to argue that many jobs are migrated from the small sites with high utilization to the large site with relatively low utilization.

In the last evaluation setting of two large and one small site, see Figure 5, the constellation behaves similar to the two large sites setup, discussed in Section 6.2. Again, LANL96 takes most of the shared jobs which leads to a strong deterioration in AWRT for FCFS-JS. This effect is even intensified by jobs offered by the small site SDSC00. Note that this behavior was also observed in the two-site case with one large and one small site. For the other algorithms the situation is alike while the AWRT deterioration is less strong.



(a) Improvements of AWRT in % (b) ΔSA$_b$ in %

Fig. 4. AWRT and Δ_bSA for setup 5 with KTH-11 (m_1=100), SDSC00-11 (m_2=128), and LANL96-11 (m_3=1024) workloads and all three algorithms FCFS-JS, EASY-JS, and LIST-JS.



(a) Improvements of AWRT in % (b) ΔSA$_b$ in %

Fig. 5. AWRT and Δ_bSA for setup 7 with SDSC00-24 (m_1=128), LANL96-24 (m_2=1024), and SDSC03-24 (m_3=1152) workloads and all three algorithms FCFS-JS, EASY-JS, and LIST-JS.

6.4 Job Pool Size Variation during Interchange

To conclude our evaluation we finally measure the job pool size to determine the amount of jobs residing in the pool during execution. To this end, we determine the pool size every time a job is added to or removed from the job pool for a two site setup. The box plots in Figure 6 show the median, 25% quantile, 75% quantile as well as 5% and 95% quantile (at end of the whiskers) of the measured data. From these results we can conclude that in comparison with the total number of jobs processed the pool is not overcrowded. As the median values are comparably small jobs do not reside in the pool for a long time.

7 Conclusion and Future Work

In this work we explored the behavior of collaborating MPP systems in a decentralized scenario. In this context, we assumed independent users who submit

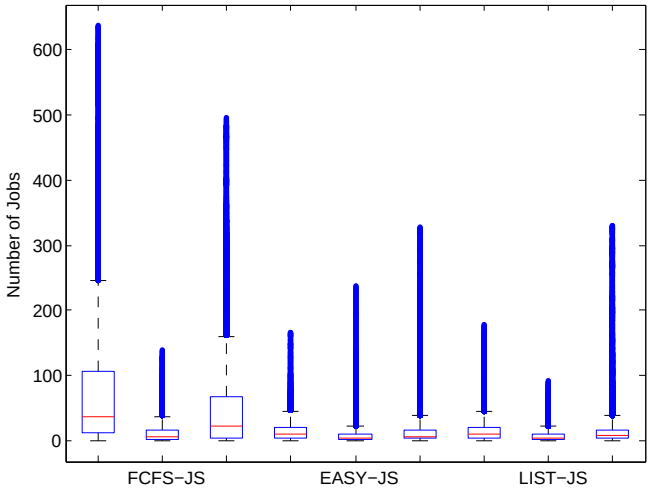


Fig. 6. Box plots of the job pool size for the two-site case considering the three job-sharing algorithms. For each algorithm three box plots are depicted, from left to right, KTH-11/SDSC00-11, LANL96-24/SDSC03-24, and LANL96-24/SDSC00-24.

jobs to their local compute sites, while the site schedulers were allowed to place released jobs into a global job pool instead of starting them locally. Respectively, the site schedulers may incorporate not only local, but also globally offered jobs into their scheduling decision.

To this end, we analyzed common performance metrics such as average weighted response time and utilization for the single-site, non-cooperating case, using standard algorithms. Then, we compared them to the results for collaborating setups in varying configurations. To this end, we defined three simple job-sharing algorithms that are based on modified variants of the standard algorithms, adding the ability to manage the interaction with the job pool.

Our results show that in case of an unchanged local scheduling algorithm many configurations benefit from the ability to additionally use a global job pool, especially in the cases of machines with a high local load. Furthermore, it could be shown that in some cases the job-sharing algorithms yield even better results than the best standard algorithm. Against this background it is noteworthy that our proposed job-sharing algorithms are very simple and leave ample room for improvements.

Regarding the extension of our analysis, the possibilities are twofold: first, an extension within the given model could be considered, including more sophisticated algorithms that make better use of the global job pool as well as dedicated distribution policies for local schedulers regarding non-schedulable jobs. An extension to the model itself could for example impose incentives and penalties for the runtime of jobs that are migrated between sites with highly diverging performance characteristics.

In our next steps, we will continue to examine our current setup, introducing more sophisticated distribution strategies and extend our evaluation to other standard algorithms such as Greedy-Scheduling with different queue ordering policies. On the medium term we plan to replace the indirect communication model we currently use by a direct communication model where each site scheduler may not only decide whether to migrate a job at all, but also to which target machine, allowing a fully decentralized scheduling model for computational jobs.

Acknowledgement

The authors would like to thank Christian Fisseler and Cesare Foltin for their extensive support in creating and verifying the simulation results.

Joachim Lepping is member of the Collaborative Research Center 531, "Computational Intelligence", at the Dortmund University with financial support of the Deutsche Forschungsgemeinschaft (DFG).

References

1. Kento Aida. Effect of Job Size Characteristics on Job Scheduling Performance. In D. G. Feitelson and L. Rudolph, editors, *Proceedings of the 6th Job Scheduling Strategies for Parallel Processing*, volume 1911 of *Lecture Notes in Computer Science (LNCS)*, pages 1–17. Springer, 2000.
2. Darin England and Jon B. Weissman. Cost and Benefits of Load Sharing in the Computational Grid. In D. G. Feitelson, L. Rudolph, and U. Schwiegelshohn, editors, *Proceedings of the 10th Job Scheduling Strategies for Parallel Processing*, volume 3277 of *Lecture Notes in Computer Science (LNCS)*, pages 160–175. Springer, 2004.
3. Carsten Ernemann, Volker Hamscher, Uwe Schwiegelshohn, Achim Streit, and Ramin Yahyapour. Enhanced Algorithms for Multi-Site Scheduling. In M. Parashar, editor, *Proceedings of the 3rd International Workshop on Grid Computing (GRID02)*, volume 2536 of *Lecture Notes in Computer Science (LNCS)*, pages 219–231. Springer, 2002.
4. Carsten Ernemann, Volker Hamscher, Uwe Schwiegelshohn, Achim Streit, and Ramin Yahyapour. On Advantages of Grid Computing for Parallel Job Scheduling. In *Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID02)*, pages 39–46. IEEE Press, May 2002.
5. Carsten Ernemann, Volker Hamscher, and Ramin Yahyapour. Benefits of Global Grid Computing for Job Scheduling. In *Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing (GRID'04)*, pages 374–379. IEEE Computer Society, 2004.
6. Dror G. Feitelson, Larry Rudolph, Uwe Schwiegelshohn, Kenneth C. Sevcik, and Parkson Wong. Theory and Practice in Parallel Job Scheduling. In *Proceedings of Job Scheduling Strategies for Parallel Processing (JSSPP97)*, volume 1291 of *Lecture Notes in Computer Science (LNCS)*, pages 1–34. Springer, 1997.
7. Dror G. Feitelson and Ahuva M. Weil. Utilization and Predictability in Scheduling the IBM SP2 with Backfilling. In *Proceedings of the 12th International Parallel Processing Symposium and the 9th Symposium on Parallel and Distributed Processing*, pages 542–547. IEEE Computer Society Press, 1998.

8. Ian Foster. What is the Grid? - A Three Point Checklist. *GRIDtoday*, 1(6), July 2002.
9. Ian Foster and Carl Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, 1999.
10. Carsten Franke, Joachim Lepping, and Uwe Schwiegelshohn. Greedy Scheduling with Complex Objectives. In *Proceedings of the 2007 IEEE Symposium Series in Computational Intelligence*. IEEE Press, 2007. to appear.
11. Carsten Franke, Joachim Lepping, and Uwe Schwiegelshohn. On Advantages of Scheduling Using Genetic Fuzzy Systems. In E. Frachtenberg and U. Schwiegelshohn, editors, *Proceedings of the 12th Job Scheduling Strategies for Parallel Processing*, volume 4376 of *Lecture Notes in Computer Science (LNCS)*, pages 68–93. Springer, January 2007.
12. Ronald L. Graham. Bounds on Multiprocessing Timing Anomalies. *SIAM Journal of Applied Mathematics*, 17(2):416–429, 1969.
13. Christian Grimme, Tobias Langhammer, Alexander Papaspyrou, and Florian Schintke. Negotiation-based Choreography of Data-intensive Applications in the C3Grid Project. In *Proceedings of the German e-Science Conference (GES)*. to appear, 2007.
14. Volker Hamscher, Uwe Schwiegelshohn, Achim Streit, and Ramin Yahyapour. Evaluation of Job-Scheduling Strategies for Grid Computing. In R. Buyya and M. Baker, editors, *Proceedings of the 7th International Conference on High Performance Computing (HiPC00)*, volume 1971 of *Lecture Notes in Computer Science (LNCS)*, pages 191–202, Bangalore, Indien, 2000. Springer.
15. Katarzyna Keahey, Thomas W. Fredian, Qian Peng, David P. Schissel, Mary R. Thompson, Ian Foster, Martin J. Greenwald, and Douglas McCune. Computational Grids in Action: The National Fusion Collaboratory. *Future Generation Computer Systems*, 18(8):1005–1015, October 2002.
16. David A. Lifka. The ANL/IBM SP Scheduling System. In *Proceedings of Job Scheduling Strategies for Parallel Processing (JSSPP95)*, volume 949 of *Lecture Notes in Computer Science (LNCS)*, pages 295–303. Springer, 1995.
17. Kai Lu, Riky Subrata, and Albert Y. Zomaya. Towards Decentralized Load Balancing in a Computational Grid Environment. In *Advances in Grid and Pervasive Computing, First International Conference*, volume 3947 of *Lecture Notes in Computer Science (LNCS)*, pages 466–477. Springer, 2006.
18. Rajeev Motwani, Steven Phillips, and Eric Torng. Nonclairvoyant Scheduling. *Theoretical Computer Science*, 130(1):17–47, 1994.
19. Uwe Schwiegelshohn. Preemptive Weighted Completion Time Scheduling of Parallel Jobs. *SIAM Journal of Computing*, 33:1280–1308, 2004.
20. Uwe Schwiegelshohn and Ramin Yahyapour. Fairness in Parallel Job Scheduling. *Journal of Scheduling*, 3(5):297–320, 2000.
21. Larry Smarr and Charles E. Catlett. Metacomputing. *Commun. ACM*, 35(6):44–52, 1992.
22. Nicola Tonellotto, Philipp Wieder, and Ramin Yahyapour. A Proposal for a Generic Grid Scheduling Architecture. In S. Gorlach and M. Danelutto, editors, *Proceedings of the Integrated Research in Grid Computing Workshop*, pages 337–346, November 2005.

A Complete Results for Three Sites

Site Setup	4			5		
Workload	LANL96-13	SDSC03-13	SDSC05-13	KTH-11	SDSC00-11	LANL96-11
m	1024	1152	1664	100	128	1024
k	1	2	3	1	2	3
FCFS-JS						
AWRT_k	19916.00	45870.87	52242.80	52851.95	41162.32	16021.66
U_k	60.74	64.68	62.03	55.11	57.15	59.15
$C_{max,k}$	33799198	33861823	33884574	29363626	29333220	29332006
SA_k	2122884048	25230465816	34972683920	1618211181	2116457552	17764967064
Δ AWRT_k	-41.61	44.16	28.13	87.36	84.56	-15.38
Δ U_k	8.00	-7.49	1.55	-19.75	-21.16	6.27
Δ SA_k	7.99	-7.95	1.82	-19.80	-23.87	6.37
Δ $C_{max,k}$	0.01	0.50	-0.27	0.06	3.39	-0.09
Δ_b AWRT_k	-59.74	15.91	8.33	29.68	44.08	-28.77
Δ_b U_k	8.00	-7.79	1.55	-19.80	-23.76	6.27
Δ_b SA_k	7.99	-7.95	1.82	-19.80	-23.87	6.37
Δ_b $C_{max,k}$	0.01	0.18	-0.27	0.00	0.14	-0.09
$M =$	83.82	7.19	8.98	85.22	4.51	10.27
	5.89	88.07	6.04	4.71	84.22	11.07
	6.28	5.64	88.08	4.52	1.64	93.85
EASY-JS						
AWRT_k	12790.98	41567.04	46570.09	53582.19	42083.57	14797.40
U_k	61.17	66.43	60.55	54.36	59.87	58.88
$C_{max,k}$	33802986	33861823	33884573	29363626	29333220	29332215
SA_k	21171896400	25914567656	34139569728	1596217383	2217375861	17686039823
Δ AWRT_k	-2.59	23.80	18.28	28.71	42.83	-18.93
Δ U_k	8.75	-5.29	-0.87	-20.71	-20.12	5.80
Δ SA_k	8.75	-5.46	-0.61	-20.89	-20.24	5.89
Δ $C_{max,k}$	0.00	0.18	-0.27	-20.89	0.14	-0.09
Δ_b AWRT_k	-2.59	23.80	18.28	28.71	42.83	-18.93
Δ_b U_k	8.75	-5.29	-0.87	-20.89	-20.12	5.80
Δ_b SA_k	8.75	-5.46	-0.61	-20.89	-20.24	5.89
Δ_b $C_{max,k}$	0.00	0.18	-0.27	0.00	0.14	-0.09
$M =$	80.85	8.50	10.65	85.09	4.39	10.52
	5.45	86.75	7.79	4.33	84.10	11.57
	7.03	7.24	85.73	2.04	3.12	94.83
LIST-JS						
AWRT_k	12308.17	39608.98	45554.56	51470.82	39959.20	15031.02
U_k	61.03	65.61	61.21	51.97	54.61	59.79
$C_{max,k}$	33796660	33861823	33884573	29363626	29333220	29320703
SA_k	21120338112	25591989520	34513706152	1526159515	2022449972	17951026280
Δ AWRT_k	6.61	23.85	17.55	36.03	46.38	-14.94
Δ U_k	8.51	-6.47	0.21	-24.36	-27.17	7.43
Δ SA_k	8.49	-6.63	0.48	-24.36	-27.25	7.48
Δ $C_{max,k}$	0.02	0.17	-0.27	0.00	0.11	-0.05
Δ_b AWRT_k	1.28	27.39	20.07	31.52	45.71	-20.86
Δ_b U_k	8.51	-6.46	0.21	-24.36	-27.15	7.43
Δ_b SA_k	8.49	-6.63	0.48	-24.36	-27.25	7.48
Δ_b $C_{max,k}$	0.02	0.18	-0.27	0.00	0.14	-0.05
$M =$	78.13	10.37	11.51	85.20	4.59	10.21
	6.44	85.51	8.06	4.05	84.76	11.19
	7.99	8.31	83.71	2.63	3.39	93.98

Table 4. Objectives for site setup 4 and 5 each with three participants ($K = 3$). AWRT_k and $C_{max,k}$ values are given in seconds while the U_k, SA_k, and all comparative objectives (Δ, Δ_b, and matrix M) are given in %.

Site Setup	6			7		
Workload	KTH-11	CTC-11	SDSC00-11	SDSC00-24	LANL96-24	SDSC03-24
m	100	430	128	128	1024	1152
k	1	2	3	1	2	3
FCFS-JS						
AWRT_k	62338.86	57201.08	50232.14	62101.21	34797.44	66909.13
U_k	63.89	69.93	64.48	60.36	66.13	66.56
$C_{max,k}$	29363626	29308287	29331912	63983290	62404360	64089508
SA_k	1875983688	8813151320	2387988478	4912160835	42260968454	49142843942
Δ AWRT_k	85.09	2.37	81.16	97.21	-183.69	59.74
Δ U_k	-6.97	6.44	-11.04	-20.41	19.08	-9.09
Δ SA_k	-7.03	6.45	-14.10	-26.20	19.29	-9.39
Δ $C_{max,k}$	0.06	-0.01	3.39	7.24	-0.18	0.33
Δ_b AWRT_k	17.06	-8.05	31.76	44.57	-213.35	5.46
Δ_b U_kL	-7.03	6.44	-13.98	-26.66	19.08	-9.94
Δ_b SA_k	-7.03	6.45	-14.10	-26.20	19.29	-9.39
Δ_b $C_{max,k}$	0.00	-0.01	0.15	-0.62	-0.18	-0.62
M =	80.01 3.94 5.17	14.59 92.15 14.50	5.39 3.91 80.34	80.69 2.60 2.32	9.70 86.54 8.43	9.61 10.86 89.25
EASY-JS						
AWRT_k	56824.14	52231.77	45829.72	40991.60	13975.35	54443.17
U_k	63.41	68.67	68.90	63.07	64.40	68.23
$C_{max,k}$	29363626	29333225	29349199	63588898	62404876	63694151
SA_k	1861958410	8662107095	2553057981	5100935288	41152934711	50062103232
Δ AWRT_k	24.39	1.33	37.74	63.41	-25.85	23.08
Δ U_k	-7.72	4.53	-8.08	-23.36	15.95	-7.69
Δ SA_k	-7.72	4.62	-8.16	-23.37	16.16	-7.69
Δ $C_{max,k}$	0.00	-0.09	0.09	0.00	-0.18	0.00
Δ_b AWRT_k	24.39	1.33	37.74	63.41	-25.85	23.08
Δ_b U_k	-7.72	4.53	-8.08	-23.36	15.95	-7.69
Δ_b SA_k	-7.72	4.62	-8.16	-23.37	16.16	-7.69
Δ_b $C_{max,k}$	0.00	-0.09	0.09	0.00	-0.18	0.00
M =	75.66 4.99 5.33	18.02 88.69 17.09	6.32 6.32 77.58	78.48 1.41 2.08	7.83 84.62 9.56	13.69 13.97 88.36
LIST-JS						
AWRT_k	53667.36	53030.53	42517.13	37582.73	14032.26	52078.06
U_k	59.96	70.54	65.54	56.22	64.86	68.66
$C_{max,k}$	29363626	29308287	29331912	63588898	62321145	63694095
SA_k	1760513685	8889577981	2427031820	4547218912	41391071039	50377683280
Δ AWRT_k	33.30	0.47	42.95	67.67	-20.42	27.89
Δ U_k	-12.75	7.36	-12.60	-31.65	16.78	-7.11
Δ SA_k	-12.75	7.37	-12.70	-31.69	16.84	-7.11
Δ $C_{max,k}$	0.00	-0.01	0.05	0.05	-0.05	0.00
Δ_b AWRT_k	28.59	-0.17	42.24	66.46	-26.36	26.42
Δ_b U_k	-12.75	7.36	-12.60	-31.68	16.78	-7.11
Δ_b SA_k	-12.75	7.37	-12.70	-31.69	16.84	-7.11
Δ_b $C_{max,k}$	0.00	-0.01	0.05	0.00	-0.05	0.00
M =	73.93 5.73 4.42	18.75 87.74 16.97	7.32 6.53 78.61	80.82 1.82 2.62	7.07 81.99 10.63	12.10 16.19 86.75

Table 5. Objectives for site setup 6 and 7 each with three participants ($K = 3$). AWRT_k and $C_{max,k}$ values are given in seconds while the U_k, SA_k, and all comparative objectives (Δ, Δ_b, and matrix M) are given in %.