The Gain of Resource Delegation in Distributed Computing Environments

Alexander Fölling, Christian Grimme,
Joachim Lepping, and Alexander Papaspyrou

Robotics Research Institute, TU Dortmund University, 44221 Dortmund, Germany
email: {*firstname.lastname*}@udo.edu

Abstract. In this paper, we address job scheduling in Distributed Computing Infrastructures, that is a loosely coupled network of autonomous acting High Performance Computing systems. In contrast to the common approach of mutual workload exchange, we consider the more intuitive operator's viewpoint of load-dependent resource reconfiguration. In case of a site's over-utilization, the scheduling system is able to lease resources from other sites to keep up service quality for its local user community. Contrary, the granting of idle resources can increase utilization in times of low local workload and thus ensure higher efficiency. The evaluation considers real workload data and is done with respect to common service quality indicators. For two simple resource exchange policies and three basic setups we show the possible gain of this approach and analyze the dynamics in workload-adaptive reconfiguration behavior.

1 Introduction

The use of High Performance Computing (HPC) in research, development, and production has become a typical part of day-to-day work since its emergence in the late 1980s; the operation of batch-oriented, large-scale Massively Parallel Processing systems is a commodity service for users in many universities, research centers, and medium-to-large enterprises.

Such systems are typically acquired with respect to the demand of the local user communities. Naturally, this demand is subject to constant change: the usage of HPC systems in research is typically bound to fixed publication dates, and industrial applications depend on the amount of orders or certain—internal and external—projects. Hence, the load of such systems fluctuates over time, a fact that obviously does not comply with the goal of its operator, namely permanent high utilization. In static environments, this leads to two undesirable situations: Either the system is underutilized, which harms the expected return on investment of the HPC system or, in case of over-utilization, the users are forced into unacceptable delays due to a large backlog of work.

From an operator's point of view, the natural way to cope with this tension would be a dynamic reconfiguration of their system in an on-demand fashion: For example, if the user-generated workload grows due to a conference deadline or a project deliverable, the operator would add additional resources to his local system until the backlog shrinks again. On the other hand, he could offer idle parts of the system to other parts of his organization, such as different departments in an enterprise or cooperating institutes within a network of universities. While such an approach ensures that the system is well-utilized—a fundamental performance benchmark for most operators and their institutions—over time, it also delivers a higher level of service quality to the users due to the adaptiveness to their workload.

The technical foundation for Distributed Computing Infrastructures (DCIs) that are capable of providing such service has been laid during the late 1990s with the emergence of Grid Computing[9]. In this area, a plethora of research has been conducted with respect to sensible workload distribution. Due to the architectural nature of Grid Computing, much effort has been put into mechanisms for the delegation of workload between participating compute centers. However, while being accepted as a basis for very large research projects such as the LHC, Grid Computing is not very wide-spread in the commercial domain and still—due to its stems in academic HPC infrastructures and its strong tailoring to their organizational architectures—comprises a high level of complexity.

Over the last two years, this technical foundation has been largely simplified and commoditized: With the widespread offering of Cloud Computing services and IaaS[1], system administrators can provision additional resources, e.g. compute, storage, and even networking, on-demand without having to make a permanent investment into extending the local facilities.

The availability of such technology in conjunction with the demand for adaptive reconfiguration of DCI environments open new challenges in the management of such systems. With respect to automated capacity planning, the efficient and situation-adaptive scheduling of incoming workload raises interesting questions:

- Is it beneficial for the system owner to invest into an expansion, or would it be sufficient to "lease" a certain amount of resources for a fixed period of time?
- Can the temporary give-away of local resources to befriend departments within a larger company provide both better overall utilization while at the same time ensuring user satisfaction?
- How does the meaning of classic utilization metrics change in such distributed, regularly self-reconfiguring systems?

In this paper, we attempt a first step towards addressing these issues: We assume a simplified DCI scenario with identical resources and investigate the performance of two algorithmic approaches to the leasing and granting of resources between autonomous HPC systems. Herein, we establish mechanisms for situation-based decision making on the distributed management level and evaluate the dynamics of system reconfiguration.

Although scheduling decision making happens mostly on the management level, it comprises to very different aspects in realization: a selection policy to find

[1] Infrastructure as a Service.

adequate partner sites for resource leasing in a distributed scenario as well as the development of decision policies for resource request and delegation respectively. While the former aspect is rather technically addressing balancing behavior on a global level, the latter emphasizes site performance for a local user community.

In order to investigate local behavior, we focus on minimum-sizes scenarios with only two sites and ignore the issue of load balancing on a global level. We evaluate these setups using workload data from three real-world HPC traces, analyze the behavior of resource leases and grants, and find improvements for both user- and provider-related metrics as basis and motivation for further research in resource delegation approaches. Nevertheless, the authors are aware of the fact that these first ideas have to be extended towards scalable heuristics that are capable to deal with mutable partner in a larger DCI.

The remainder of the paper is organized as follows: Section 2 gives an overview of existing approaches to scheduling in DCI environments. This is followed by a formal description of the DCI environment and the resulting scheduling problem in Section 3. Section 4 details our two-layered scheduling architecture while the proposed scheduling policies are then described in Section 5. We present a performance evaluation of our strategies in Section 6 and conclude the paper in Section 7.

2 Background

Automated capacity planning and workload scheduling in DCI systems is a well-covered research topic and stems back to classic parallel machine scheduling problems.

In recent years, a remarkable amount of effort has been put into workload distribution among autonomously acting HPC centers within the broader context of Grid Computing: Scenarios that assume such federated environments often imply centralized scheduling services [10]. For example, Ernemann et al. [3] show advantages of hierarchical scheduling in general by considering the AWRT objective. Further, Kurowski et al. [15] identify multiple objectives for efficient job scheduling in Grids and propose a strategy based on prediction mechanisms and resource reservation. For decentralized environments, only few results that support the delegation of workload have been published. England and Weissman [1] give an estimation of costs and benefits of load sharing relying on synthetic workloads only. Grimme et al. [11] analyze the prospects of collaborative job sharing and compare their results to the non-cooperative scenario of the same machines. Recent works of Fölling et al. [7, 8] propose a fuzzy-based, evolutionary optimized exchange policy for a fully decentralized scenario which shows robustness even in changing environments and automatically adapts to the current local load.

With the elasticity of IaaS-supported DCI environments, a new kind of flexibility challenges current scheduling approaches due to the inherent reconfigurability of machines and the resulting changes in scheduling responsibilities. Up to now, this aspect—especially with respect to classic HPC workloads with parallel jobs—has only occasionally been discussed in research: Since the complexity of operating such systems in the large scale and the feasibility of provisioning and reconfiguring them on-demand hampered the realization for production environments, discussion focused on rather low-level computer hardware. For example, Kota et al. [14] consider the problem of scheduling and mapping of tasks onto reconfigurable logic units for a given application introducing a concept of parameterized modules. Their approach is a typical example of scheduling in the context of reconfigurable hardware that involves varying sizes of available hardware. Subramaniyan et al. [18] transferred similar ideas to the HPC context and analyzed the dynamic scheduling of large-scale HPC applications in parallel reconfigurable computing environments. They assess the performance of several common HPC scheduling heuristics that can be used by an automated job management service to schedule application tasks on parallel reconfigurable systems. However, their approach is limited to a single HPC system and does not involve the interaction of multiple autonomous partners in a DCI environment.

The reconfigurability of a HPC center within a larger DCI environment obviously provides inherent support of multi-site computing on the capacity planning and workload distribution level. In multi-site computing, jobs can be executed beyond site boundaries, effectively running parts of the job at distinct locations. Naturally, additional problems with respect to data availability and network performance arise here. Nevertheless, Ernemann et al. [2] identify improvements for the AWRT objective assuming hierarchical centralized scheduling structures in multi-site computing. Further, Zhang et al. [20] provide an overview of existing multi-site computing approaches and present an adaptive algorithm that incorporates also common local scheduling heuristics. Recently, Iosup et al. [13] proposed a delegated matchmaking method, which temporarily binds resources from remote sites to the local environment.

All approaches assume an additional scheduling layer on top of classic LRMS which coordinates the underlying resources in a hierarchical fashion but their architectures imply that local sites have to (partially) cede their autonomy for the benefit of coordinated DCI scheduling on a higher level.

Further, Weissmann and Grimshaw [19] presented an approach for decentralized DCI systems which introduces all basic policies to exchange jobs between autonomous sites. However, their policies are based on an unrestrictive information model between the sites which allows a local scheduler to query detailed information about the system states of potential delegation targets. This includes also queries on estimated start times at foreign sites for specific jobs. As in real DCI systems such information are usually treated confidential, it requires new heuristics that even yield acceptable scheduling performance when only local information is accessible for decision making. Moreover, their scheduling approaches are only considered theoretically without performance measurement on workload data. Thus, with respect to the work at hand, their results cannot be used for the matter of comparison.

3 Problem Formulation

In HPC systems, job scheduling is an online problem regardless of the assumed machine configurations. Users submit parallel jobs over time while neither their submission time nor the precise processing time are known in advance. We further consider independent[2] jobs that are neither malleable nor moldable. Each job j is characterized by its degree of parallelism m_j, its processor independent processing time p_j and its estimated processing time $\overline{p}_j$, see Feitelson et al. [5]. $\overline{p}_j$ is provided by the user at submit time and originally was intended to recognize erroneous jobs and abort them if they take longer than the user expected. Scheduling heuristics, however, also use $\overline{p}_j$ for making better decisions. The number of required processors m_j is available at the release time r_j of job j.

The DCI environment we consider in our work consists of K loosely coupled HPC sites. Each site k owns m_k identical processors such that every parallel job can be executed on each subset of local processors. Although existing studies for heterogeneous DCI environments show that the processing time of jobs depends on both the application structure and the target architecture, see for example Sabin et al. [16], the list of top-performing HPC installations[3] proves almost homogeneity in terms of processors families and architectures. Therefore, we additionally assume identical processors among all sites.

During its execution phase, each job requires exclusive access to $m_j \leq m_k$ processors. As users submit their jobs locally, the corresponding site has to guarantee that every submitted job can—regardless of the availability of remote systems—be executed. Therefore, jobs that require more than the total number of locally available processors ($m_j > m_k$) are rejected. Further, all jobs run to completion without the possibility of being preempted, since the majority of HPC applications and systems does not support this. As such, the completion time within the schedule S at site k is denoted by $C_j(S_k)$.

In our system model, we further allow multi-site execution, that is each job can be executed on any subset of processors within the whole DCI environment. This is typically possible[4] for embarrassingly parallel jobs that comprise many sequential, independent invocations of the same application. Examples for this application class are parameter sweeps—tools that repeatedly process the same input data, with varying parameter settings—or SPMD[5]-style programs. Iosup et al. [12] have shown that this class is the most widely spread kind of jobs in productive grids and DCI environments. Although distributed filesystem access and network latency may impair the execution speed of such applications in a multi-site execution scenario, Ernemann et al.[2] have shown that the significant improvements in schedule quality often compensate for the inferior performance. Formally, such a multi-site job j is scheduled on $m_{j|k}$ own resources at the

[2] With no dependencies among them, that is.
[3] www.top500.org, January 2010.
[4] Provided that data availability (for example, via a shared file system) is guaranteed.
[5] Single Process, Multiple Data.

submission site $k \in K$ and $m_{j\nmid k}$ foreign resources using altogether $m_j = m_{j|k} + m_{j\nmid k}$ resources as defined before.

4 System Model

In our system model, we establish a two-layered architecture at every site, see Figure 1: The Local Resource Management System (LRMS) is responsible for

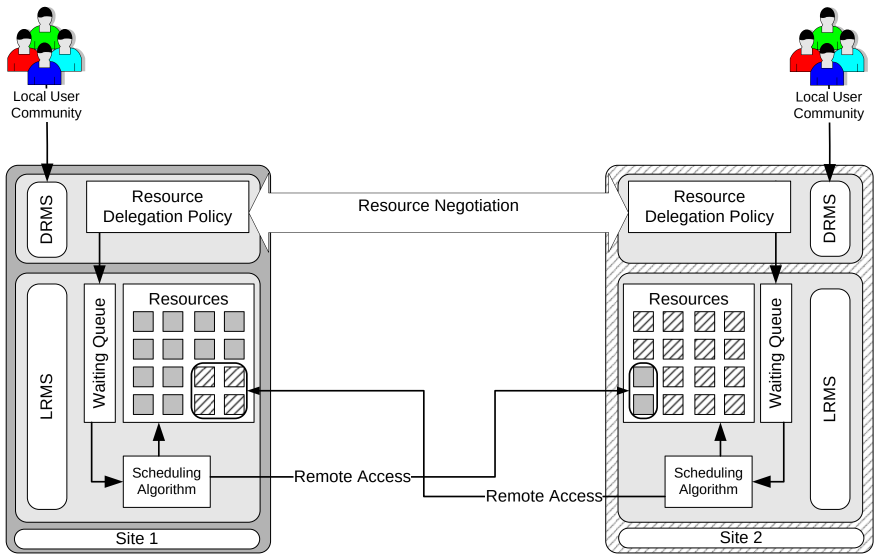


Fig. 1. Resource Brokering within a Computational Grid scenario with independent sites.

the local allocation of jobs onto resources, while the Distributed Resource Management System (DRMS) layer realizes the the lease-and-grant mechanism and policy. Within the latter layer, resource requests are formulated and negotiated in order to adapt the local system to the current load situation.

4.1 LRMS Layer

The Local Resource Management System (LRMS) layer consists of a waiting queue and a scheduling algorithm that assigns jobs to processors in its domain. This *local scheduling domain* comprises all processors that are exclusively controlled by the LRMS. In contrast to classic settings, this domain is subject to changes over time: While foreign resources can be logically integrated into the LRMS and used by the local scheduling algorithm, it is also possible to delegate own resources to *foreign scheduling domains*, putting them under exclusive control of the remote RMS. Further, jobs can be prioritized.

Among the variety of LRMS scheduling algorithms, we chose the Extensible Argonne Scheduling System (EASY) [6] for our analysis as it enjoys widespread application. On invocation, EASY tries to execute the job j at the head of the waiting queue, if—with respect to m_j—enough processors are currently available. Otherwise, it tries to "backfill" a subsequent job in the queue, ensuring that—based on $\overline{p}_j$—job j is not delayed. Note, however, that the overall methodology is not restricted to any kind of local scheduling algorithm.

4.2 DRMS Layer

The DRMS layer is able to extend the local scheduling domain by leasing resources from other sites. In this way, the site is able to gain exclusive control on foreign resources. We assume that submitted jobs have to pass through this layer before they can be handled by the underlying LRMS[6]. For each job, a *resource delegation policy* (RDP) decides whether additional resources should be leased to increase the scheduling performance at the local site or not. If yes, resource requests are send and negotiated with other partners within the DCI system. Each request contains the number of desired resources and a timespan for which the site wants to gain exclusive access to them. Granting sites also apply their RDP in order to determine whether to accept or decline the request. Decision making is based on multiple input features such as the users' job submission behavior, the current resource usage, and the local backlog. Finally, it is not allowed to grant already leased resources to a third party.

5 Resource Delegation Policy

At the DRMS level, the resource delegation policy steers the individual negotiation behavior of each participating site within the DCI. We here introduce two approaches that feature a very simple design, are minimally invasive to the LRMS, and still achieve good scheduling results. Both are triggered by the submission of a single job and can be applied under restrictive information policies, that is without exchange of information between the interacting partners.

The Simple Submission Triggered Resource Delegation (S-STRD) policy tries to prioritize incoming jobs, if a resource lease for this particular submission can be acquired. Figure 2 presents the behavior of the policy in a flow chart (left side) and gives an example for a submitted job (right side). Resource leasing is attempted every time the currently available resources cannot meet the submitted job's processor demand. After the submission of a new job to the DRMS, it is automatically forwarded to and enqueued at the LRMS ①. Furthermore, the DRMS checks whether there are enough idle resources available to directly execute the job ②. In the positive case, the DRMS leaves further handling to the LRMS. Otherwise, the DRMS formulates a resource lease request with the

[6] This poses no restriction in terms of usability, since the DRMS can act as a proxy of the LRMS towards the user.

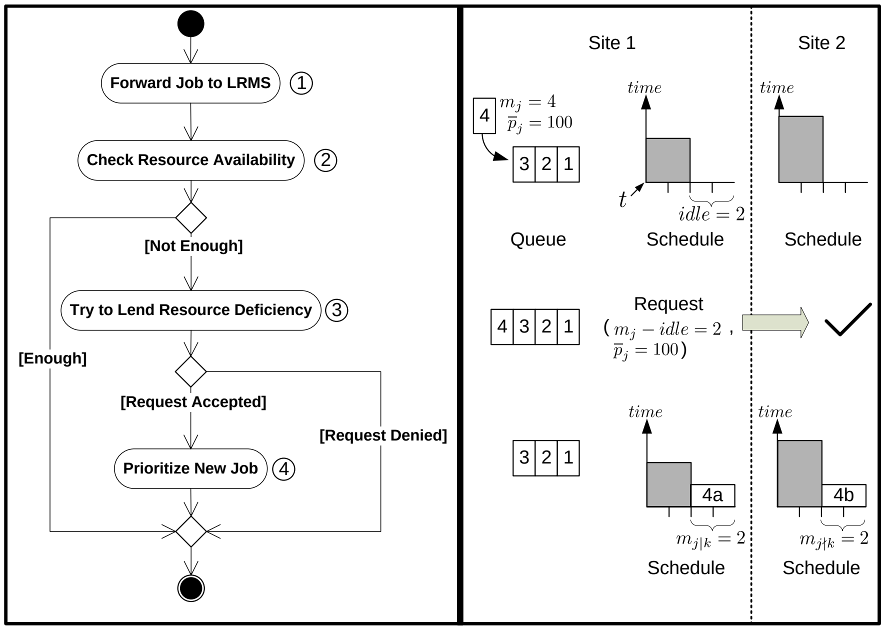


Fig. 2. Activity diagram and example for the Simple Submission Triggered Resource Delegation Policy.

number of requested, but locally unavailable resources and the user's runtime estimate for the job. This request is then posted to the delegation partners in the system ③. If the request is granted, the job is prioritized for direct execution ④. After completion of the job, which is not necessarily equal to the user estimate, leased resources are returned to the granting site.

In the example (cf. Figure 2), we assume a waiting queue with three jobs at Site 1. Further, both sites have scheduled occupations for different timespans. At time t a new job j with a demand of $m_j = 4$ and $\overline{p}_j = 100$ is submitted to Site 1. Since the job cannot be scheduled on local resources directly, Site 1 requests two additional resources for a timespan of $\overline{p}_j$ from Site 2, which in turn accepts the request and grants two resources to Site 1. Having a sufficient number of processors available for the immediate execution of j, S-STRD prioritizes the job and, on invocation of the LRMS's scheduling algorithm, it is started immediately on two local and two remote resources.

The extended version of the algorithm (X-STRD) differs in the repetition of steps ②–④ (compare Figure 2): These are applied for each job in the queue (including the new job), starting at the queue's head. Obviously, this approach less penalizes already waiting jobs, since they considered first. Still, this policy demands extensive inter-site communication due to many additional resource requests and makes the extended approach less practical for the use in real scheduling systems. We still evaluate this policy as an extremal case for an excessive use of resource delegation between the sites to assess the achievable performance.

6 Performance Evaluation

We estimate the quality of the proposed mechanisms by means of simulation. In order to quantify the performance, we apply common performance indicators for job scheduling in parallel machine and DCI environments and adapt them to reconfigurable machine environments accordingly. Moreover, we use recorded (non-synthetic, that is) workload traces as input to our simulations to ensure realistic results. Finally, we discuss the implications from the simulation results for three distinct scenarios.

6.1 Quality Measures

From the quantitative side, we look at three common metrics:

The *Average Weighted Response Time* (AWRT) basically denotes for all users how long they have to wait for their jobs to complete on the average.

$$\mathrm{AWRT}_k = \frac{\sum\limits_{j \in \tau_k} p_j \cdot m_j \cdot (C_j(S) - r_j)}{\sum\limits_{j \in \tau_k} p_j \cdot m_j} \tag{1}$$

It is computed for all jobs $j \in \tau_k$ that have been submitted to site k, see Equation 1. It is widely agreed that a short AWRT is the best way to describe the average performance a provider can offer users for job execution. Following Schwiegelshohn and Yahyapour [17], we weight the response time of each job with its resource consumption $(p_j \cdot m_j)$. This ensures that neither splitting nor combination of jobs can influence the objective function in a beneficial way. Note that we calculate $m_j = m_{j|k} + m_{j\nmid k}$ in order to incorporate the execution of jobs on remote resources.

The *Squashed Area*[7] (SA_k) reflects the overall resource usage of all submitted jobs per participating site k. In a scenario where jobs are partially executed on remote sites, we have to refine the original metric as follows:

$$\mathrm{SA}_k = \sum_{j \in \tau_k} p_j \cdot m_{j|k} + \sum_{l \notin \tau_k} p_l \cdot m_{l|k} \tag{2}$$

SA_k is determined as the sum both local $(j \in \tau_k)$ and foreign $(l \notin \tau_k)$ jobs' resource consumption fractions $(p_j \cdot \ldots$ and $p_l \cdot \ldots)$ that are executed on resources belonging to site k $(m_{j|k}$ and $m_{l|k})$, see Equation 2.

$$\mathrm{SA}_k^{\lambda} = \sum_{j \in \tau_k} p_j \cdot m_{j\nmid k} \tag{3}$$

[7] This metric sometime also called *Total Work*.

To further measure the amount of work running on leased processors from within the DCI environment, we define the "leased" Squashed Area SA_k^λ as the sum of local ($j \in \tau_k$) jobs' resource consumption fractions ($p_j \cdot \ldots$) that are executed on resources not belonging to site k ($m_{j \nmid k}$), see Equation 3.

The *Utilization* (U_k) describes the ratio between overall resource usage available resources after the completion of all and measures how efficiently the processors of site k are used over time.

$$st(S_k) = \min\left\{\min_{j \in \tau_k}\{C_j(S_k) - p_j\}, \min_{l \notin \tau_k}\{C_l(S_k) - p_l\}\right\}, \text{ and} \tag{4}$$

$$C_{max,k} = \max\left\{\max_{j \in \tau_k}\{C_j(S_k)\}, \max_{l \notin \tau_k}\{C_l(S_k)\}\right\}. \tag{5}$$

It refers to the timespan relevant from the schedule's point of view, delimited by the start time of the first job, see Equation 4, to the end time of the last job, see Equation 5, in schedule S_k. Note that both points in time consider local jobs ($j \in \tau_k$) and fractions of delegated jobs ($l \notin \tau_k$).

$$\mathrm{U}_k = \frac{\mathrm{SA}_k}{m_k \cdot (C_{max,k} - st(S_k))} \tag{6}$$

U_k, formally defined in Equation 6, often serves as a quality measure from the site provider's point of view.

6.2 Input Data

The Parallel Workloads Archive[8] provides job submission and execution recordings on real-world HPC system site, each of which containing information on relevant job characteristics like estimated and real processing time, release date, resource demand, and others. We applied pre-filtering steps to the original data in order to remove partially erroneous information: we discard jobs with invalid release dates ($r_j < 0$), processing times ($p_j \leq 0$), resource requests ($m_j \leq 0$) as well as unsatisfiable resource demands on the submitted site ($m_j > m_k$).

Table 1. Workload characteristics of the used input data, including AWRT in seconds, U in %, and C_{max} in seconds for single site execution with EASY.

Identifier	#Jobs	m_k	AWRT	U	C_{max}	Setup 1	Setup 2	Setup 3
KTH-11	28479	100	75157.63	68.72	29363626	X	X	
CTC-11	77199	430	52937.96	65.70	29306682	X		X
SDSC05-11	74903	1664	54953.84	60.17	29357277		X	X

We select three traces for our evaluation: The KTH trace which contains records from a 100 processor IBM RS/6000 SP system at the Swedish Royal

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

Institute of Technology in Stockholm, the CTC trace from a 430 processor IBM RS/6000 SP system at the Cornell Theory Center in Ithaca, NY, and a log recorded 2005 at the San Diego Supercomputer Center in La Jolla, CA (SDSC05).

Since the original workloads cover unequal periods, we shorten all original workloads to the largest common lenght, namely eleven months. Additionally, we assume identical timezones and therefore similarize the diurnal rhythm of job submission: In geographically dispersed DCI scenarios, different timezones may induce positive scheduling effects as idle machines can be used by jobs from peak loaded sites in accordance with day-time differences, see Ernemann at al. [4]. Here, we cannot benefit from timezone shifts in our scenario. As such, the results will likely improve in time-shifted environments.

Finally, we simulate the workload on their original machine configuration with an non DCI-aware LRMS that uses the EASY algorithm and take the results as reference for local-only scheduling, see Section 4.1. Relevant characteristics of the examined traces and the corresponding results for AWRT, Utilization, and C_{max} are listed in Table 1. During the course of this paper, we will refer to this non-cooperative case for the matter of comparison.

6.3 Performance Results

We investigate three scenarios (cf. Table 1): The first scenario comprises the small KTH machine with 100 processors and the mid-sized CTC machine with 430 processors. Further, we evaluate a scenario with the small KTH machine and the large-scale HPC system SDSC05 with 1664 processors and combine further CTC and SDSC05. For all scenarios, we apply the two discussed strategies. The results of all evaluations are shown in Table 2.

Almost all results show an improvement in AWRT compared to local execution, which indicates that both partners benefit from their cooperation. Figure 3 depicts the improvements obtained in three scenarios for both policies. The S-STRD strategy yields good results, improving the AWRT of the smaller partner for at least 15% in all scenarios. As expected, small partners benefits from the enormous resource potential provided by large partners. However, simulations show that large partners also profit from cooperation with small partners. Although this improvement is marginal for the SDSC05 site, the increase of utilization, see Table 2, indicates a compact schedule and thus better resource usage.

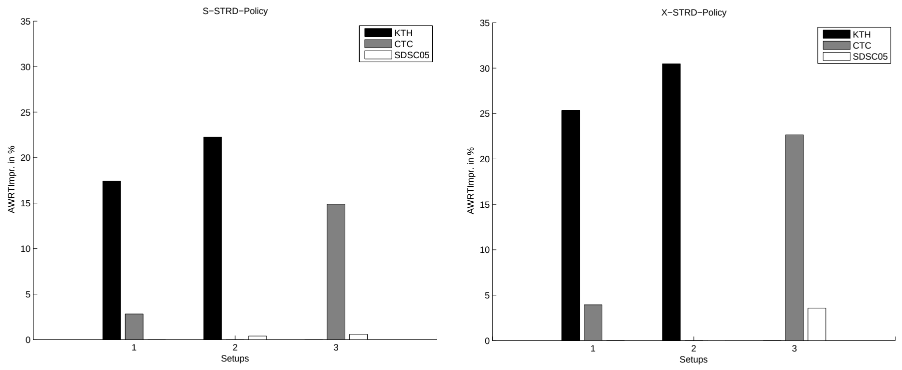
Besides AWRT comparison, we analyze the reconfiguration behavior at both sites. Figure 4 exemplary shows the dynamics of both systems for Setup 1 and the S-STRD policy. Obviously, the local resource configurations are subject to continuous changes while—on the average—they nearly keep their size, see Figure 5. During the simulated workload period, KTH occasionally grants all its resources to the larger site, but also quadruples its original size through leases. In the latter case, the reconfiguration almost switches the original sizes of the setup. This impressively demonstrates the potential of a workload-triggered re-

Table 2. Evaluation results for both strategies S-STRD and X-STRD for the given three scenarios. Values for AWRT, U, and C_{max} are shown as well as their improvements in %, the absolute amount of mutually exchanged resources as leased Squashed Area SA_k^{λ}, and the average queue length $\bar{Q}$.

S-STRD-Broker						
Metrics	Setup 1		Setup 2		Setup 3	
Workload	KTH-11	CTC-11	KTH-11	SDSC05-11	CTC-11	SDSC-11
AWRT_k	62055.37	51444.91	58432.61	54738.92	45062.71	54635.66
U_k	65.22	66.46	62.63	60.54	63.37	60.80
$\mathrm{C}_{max,k}$	29363626	29332185	29363626	29353826	29328089	29335555
$\Delta\mathrm{AWRT}_k$	17.43	2.82	22.25	0.39	14.88	0.58
$\Delta\mathrm{U}_k$	-5.09	1.15	-8.86	0.62	-3.54	1.05
SA_k^{λ}	573141728	413432502	630674342	383713739	1768336330	1493028875
$\bar{Q}$	4.08	8.06	2.31	17.07	3.07	13.22
X-STRD-Broker						
Metrics	Setup 1		Setup 2		Setup 3	
Workload	KTH-11	CTC-11	KTH-11	SDSC05-11	CTC-11	SDSC-11
AWRT_k	56115.99	50851.43	52253.12	55729.64	40940.23	52990.40
U_k	64.40	66.65	56.88	60.86	62.71	60.98
$\mathrm{C}_{max,k}$	29363626	29332185	29363626	29364647	29306682	29339742
$\Delta\mathrm{AWRT}_k$	25.34	3.94	30.48	-1.41	22.66	3.57
$\Delta\mathrm{U}_k$	-6.28	1.44	-17.22	1.16	-4.55	1.34
SA_k^{λ}	640012760	442455083	682003341	250461072	2105350162	1722880873
$\bar{Q}$	4.48	10.17	1.82	18.62	2.93	14.53

configuration where the user visible provider domains remain stable: resources adapt to submitted workload but offer an accustomed environment to users.

Finally, we investigate X-STRD and identify larger benefits for all smaller sites. Compared to the application of S-STRD we can also show AWRT improvements for larger sites. However, we observe a deterioration in AWRT for the Setup 2 compared to uncooperative processing, see Figure 3(b). This behavior is due to unbalanced exchange of resources indicated by SA_k^{λ} for the second setup in Table 2: While the small KTH site is able to increase its resource capacity, the larger site's requests are frequently rejected for S-STRD leading to higher utilization and increased AWRT. Thus, we conclude that the extended strategy can yield better results for all participating sites but is less robust against large discrepancies in machine size: In X-STRD, continuous workload submission results in frequent traversals of the complete queue. As a consequence, this gives small sites more opportunities to gain additional resources from the larger site to execute long waiting jobs. The opposite is not necessarily true, as the resource capacity of a small site restricts the larger site's chances.



(a) Improvements in AWRT for all setups and S-STRD-Policy.

(b) Improvements in AWRT for all setups and X-STRD-Policy.

Fig. 3. Improvements in AWRT for all setups and both policies.

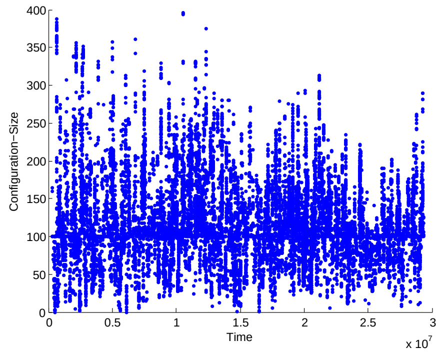
7 Conclusion and Future Work

In this work we approached the topic of collaboration in distributed computing infrastructures from a new and more operator-centric perspective: the delegation of resources between partner sites. In order to adapt to fluctuating local user demand while constantly offering high service quality, cooperating HPC providers are enabled to mutually lease resources from other partners or grant them to him. To this end, we devised a delegation layer above the local management layer of each site and two delegation policies which combine negotiation capabilities with scheduling decisions making. This ensures both independent acting sites in a decentralized scenario as well as a situation-aware delegation behavior while leaving the local management systems largely untouched.

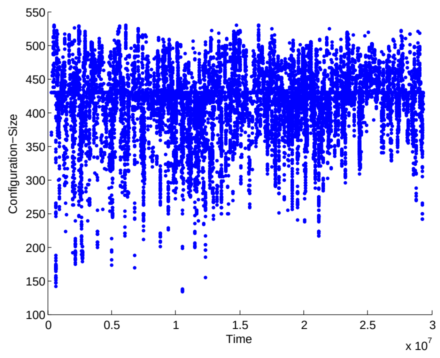
For evaluating the proposed collaboration scenario, we investigated several two-site setups consisting of different-sized installation by simulatively feeding them with real-world workload data. Both delegation policies demonstrated their potential realizing an enormous increase in service quality for almost all participating sites, with less robustness of second delegation approach against extremal differences in site size, leading to degradation of service quality in specific cases.

Moreover, we were able to show the dynamics of site reconfiguration in the proposed scenario: The sites frequently changed their configuration in order to fit their workload. In fact, the fluctuations in site configurations ranged from completely granting all resources to leases that multiply the site's own size. This impressively demonstrates the hidden potentials of collaboration in DCIs and should motivate operators to provide locally idle resources in order to benefit from cooperation in terms of service quality and effective resource usage.

Our next steps will be twofold: On the one hand, we will consider several restrictions in our current model: Since in practice multi-site execution of jobs might be prohibited, more powerful heuristics should yield good schedules with-



(a) KTH



(b) CTC

Fig. 4. Continuous reconfiguration of both KTH and CTC sites during workload processing.

out spreading jobs among site boundaries. This is, the have to decide between either local or remote execution.

On the other hand, advanced heuristics should be applicable under limited information exchange. To this end, they should favor a collaborative and/or partner-specific adaptive behavior. In this context—besides scalarization issues—global balancing effects and benefits of the second level partner site selection strategies have to be evaluated in larger scenarios with multiple participating sites. Those policies can possibly base on load balancing between multiple partners or cost models for resource delegation.

Finally, non job-specific leases have to be considered, allowing to "borrow" a certain amount of resources for a certain timeframe and thus to fully take care of capacity planning on these resources, as currently delivered by modern IaaS environments.

References

1. Darin England and Jon B. Weissman. Costs and Benefits of Load Sharing in the Computational Grid. In Dror G. Feitelson, Larry Rudolph, and Uwe 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.
2. Carsten Ernemann, Volker Hamscher, Uwe Schwiegelshohn, Achim Streit, and Ramin Yahyapour. Enhanced algorithms for multi-site scheduling. In *Proceedings of the 3rd International Workshop on Grid Computing, Grid 2002*, volume 2536 of *Lecture Notes in Computer Science (LNCS)*, pages 219–231. Springer, Nov 2002.
3. 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.

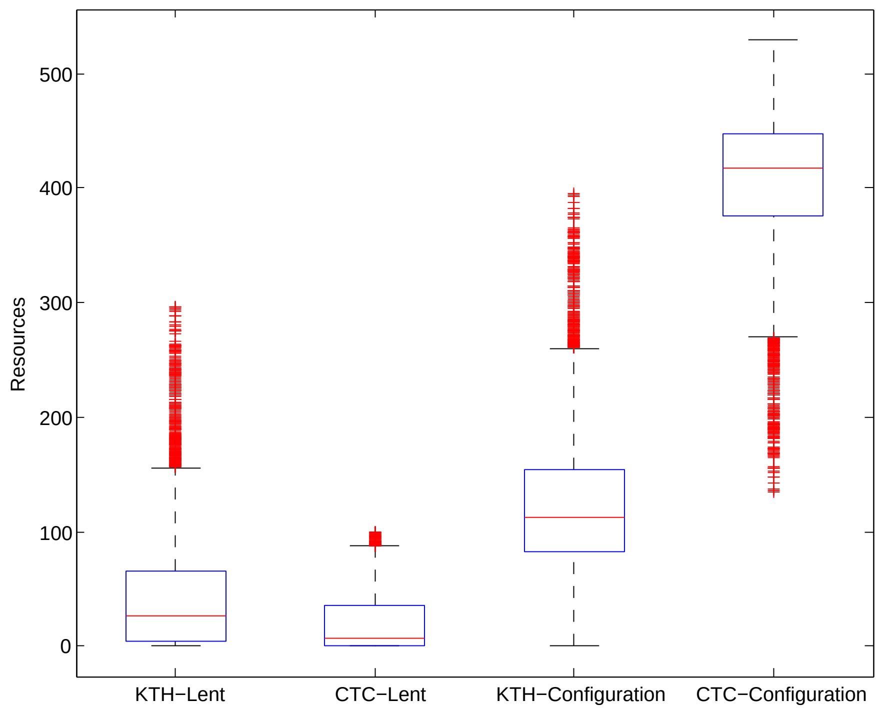


Fig. 5. Aggregated site reconfiguration properties for Setup 1. The two leftmost plots refer to leased resources by KTH and CTC to the respective partner site. The two rightmost plots state on each site's size configurations during cooperation.

4. 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.
5. Dror G. Feitelson and Bill Nitzberg. Job characteristics of a production parallel scientific workload on the NASA ames iPSC/860. In Dror G. Feitelson and Larry Rudolph, editors, *Proceedings of the 1st Job Scheduling Strategies for Parallel Processing*, volume 949 of *Lecture Notes in Computer Science (LNCS)*, pages 337–360. Springer, 1995.
6. 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.
7. Alexander Fölling, Christian Grimme, Joachim Lepping, and Alexander Papaspyrou. Decentralized Grid Scheduling with Evolutionary Fuzzy Systems. In Eitan Frachtenberg and Uwe Schwiegelshohn, editors, *Proceedings of the 14th Job Scheduling Strategies for Parallel Processing*, volume 5798 of *Lecture Notes in Computer Science (LNCS)*, pages 16–36. Springer, 2009.
8. Alexander Fölling, Christian Grimme, Joachim Lepping, Alexander Papaspyrou, and Uwe Schwiegelshohn. Competitive co-evolutionary learning of fuzzy systems

for job exchange in computational grids. *Evolutionary Computation*, 17(4):545–560, 2009.

9. I. Foster and C. Kesselman. Globus: A Toolkit-Based Grid Architecture. In *The Grid: Blueprint for a Future Computing Infrastructure*, pages 259–278. Morgan Kaufman, San Mateo, 1st edition, 1998.
10. Fabrizio Gagliardi, Bob Jones, Francois Grey, Marc-Elian Begin, and Matti Heikkurinen. Building an infrastructure for scientific grid computing: status and goals of the egee project. *Philosophical transactions. Series A, Mathematical, physical, and engineering sciences*, 363(1833):1729–1742, 2005.
11. Christian Grimme, Joachim Lepping, and Alexander Papaspyrou. Prospects of Collaboration between Compute Providers by means of Job Interchange. In Eitan Frachtenberg and Uwe Schwiegelshohn, editors, *Proceedings of Job Scheduling Strategies for Parallel Processing*, volume 4942 of *Lecture Notes in Computer Science (LNCS)*, pages 132–151. Springer, June 2007.
12. Alexandru Iosup, Catalin Dumitrescu, Dick Epema, Hui Li, and Lex Wolters. How are Real Grids Used? The Analysis of Four Grid Traces and Its Implications. In Dennis Gannon, Rosa M. Badia, and Rajkumar Buyya, editors, *Proceedings of the 7th IEEE/ACM International Conference on Grid Computing*, pages 262–269. IEEE Press, September 2006.
13. Alexandru Iosup, Todd Tannenbaum, Matthew Farrellee, Dick Epema, and Miron Livny. Inter-operating grids through delegated matchmaking. *Scientific Programming*, 16(2-3):233–253, 2008.
14. Solomon Raju Kota, Chandra Shekhar, Archana Kokkula, Durga Toshniwal, M. V. Kartikeyan, and Ramesh Chandra Joshi. Parameterized module scheduling algorithm for reconfigurable computing systems. In *ADCOM '07: Proceedings of the 15th International Conference on Advanced Computing and Communications*, pages 473–478, Washington, DC, USA, 2007. IEEE Computer Society.
15. Krzysztof Kurowski, Jarek Nabrzyski, Ariel Oleksiak, and Jan Weglarz. Scheduling Jobs on the Grid - Multicriteria Approach. *Computational Methods in Science and Technology*, 12(2):123–138, 2006.
16. Gerald Sabin, Rajkumar Kettimuthu, Arun Rajan, and Ponnuswamy Sadayappan. Scheduling of parallel jobs in a heterogeneous multi-site environment. In *Proceedings of the 9th Job Scheduling Strategies for Parallel Processing*, pages 87–104. Springer, 2003.
17. Uwe Schwiegelshohn and Ramin Yahyapour. Fairness in parallel job scheduling. *Journal of Scheduling*, 3(5):297–320, 2000.
18. Rajagopal Subramaniyan, Ian Troxel, Alan D. George, and Melissa Smith. Simulative analysis of dynamic scheduling heuristics for reconfigurable computing of parallel applications. In *Proceedings of the 2006 ACM/SIGDA 14th international symposium on Field programmable gate arrays*, pages 230–230. ACM, 2006.
19. Jon B. Weissman and Andrew S. Grimshaw. A federated model for scheduling in wide-area systems. In *Fifths IEEE International Symposium on High-Performace Distributed Computing (HPDC-5 '96)*, pages 542–550. IEEE Computer Society, 1996.
20. Weizhe Zhang, Albert M.K. Cheng, and Mingzeng Hu. Multisite co-allocation algorithms for computational grid. In *International Parallel and Distributed Processing Symposium*, Los Alamitos, CA, USA, 2006. IEEE Computer Society.