Inter-Operating Grids through Delegated MatchMaking

Alexandru Iosup[*]
Delft University of Technology,
2628CD, Delft, NL
A.Iosup@tudelft.nl

Dick H.J. Epema
Delft University of Technology,
2628CD, Delft, NL
D.H.J.Epema@tudelft.nl

Todd Tannenbaum
University of Wisconsin,
Madison, WI 53706, US
tannenba@cs.wisc.edu

Matthew Farrellee
University of Wisconsin,
Madison, WI 53706, US
matt@cs.wisc.edu

Miron Livny
University of Wisconsin,
Madison, WI 53706, US
miron@cs.wisc.edu

ABSTRACT

The grid vision of a single computing utility has yet to materialize: while many grids with thousands of processors each exist, most work in isolation. An important obstacle for the effective and efficient inter-operation of grids is the problem of resource selection. In this paper we propose a solution to this problem that combines the hierarchical and decentralized approaches for interconnecting grids. In our solution, a hierarchy of grid sites is augmented with peer-to-peer connections between sites under the same administrative control. To operate this architecture, we employ the key concept of delegated matchmaking, which temporarily binds resources from remote sites to the local environment. With trace-based simulations we evaluate our solution under various infrastructural and load conditions, and we show that it outperforms other approaches to inter-operating grids. Specifically, we show that delegated matchmaking achieves up to 60% more goodput and completes 26% more jobs than its best alternative.

1. INTRODUCTION

In the mid-1990s, the vision of the grid as a computing utility was formulated [14]. Since then, hundreds of grids have been built—in different countries, for different sciences, and both for production work and for computer-science research—but most of these grids work in isolation. So, the next natural step is to have multiple grids inter-operate in order to serve much larger and more diverse communities of scientists and to put the ensemble of resources of these grids to better use. However, grid inter-operation raises serious challenges in the areas of, among others, resource management and performance. In this paper we address these two challenges with the design and evaluation of a *delegated matchmaking protocol* for resource selection and load balancing in inter-operating grids.

Our work was motivated by the ongoing efforts for making two multi-cluster grids, the DAS [10] and Grid'5000 [7], inter-operate. Much like similar grid systems, e.g., CERN's LCG, their resources are in general under-utilized, yet in few occasions the demand exceeds the capacity of the individual systems. In such occasions, two (undesirable) alternatives are to queue the extra demand until it can be served, and to enlarge the individual systems. A third, and potentially more desirable option is to inter-operate grids, so that their collective demand will ideally incur a rather stable, medium-to-high utilization of the combined system.

The decision to inter-operate grids leads to non-trivial design choices with respect to resource selection and performance. If there is no common resource management system, jobs must be specifically submitted to one of the grids, which may lead to poor load balancing. If a central meta-scheduler is installed, it will quickly become a bottleneck leading to unnecessarily low system utilization, it will be a single point of failure leading to break-downs of the combined system, and it is unclear who will physically manage the centralized scheduler. Traditional decentralized solutions can also be impractical. Hierarchical mechanisms (still centralized, but arguably with less demand per hierarchy node) can be efficient and controllable, but still have single points of failure, and are administratively impractical (i.e., who administers the root of the hierarchy?). Completely decentralized systems can be scalable and fault-tolerant, but they can be much less efficient than their hierarchical alternatives. While many solutions have already been proposed [4, 6, 8, 21, 22, 23, 24], none has so far managed to achieve acceptance in the grid world, in part because they have yet to prove that they can yield significant benefits when managing typical grid workloads.

In this paper, we investigate a decentralized architecture for grid inter-operation that is based on two key ideas. First, we leverage a *hierarchical* architecture in which nodes represent computing sites, and in which we allow the nodes at the same hierarchical level and operating under the same
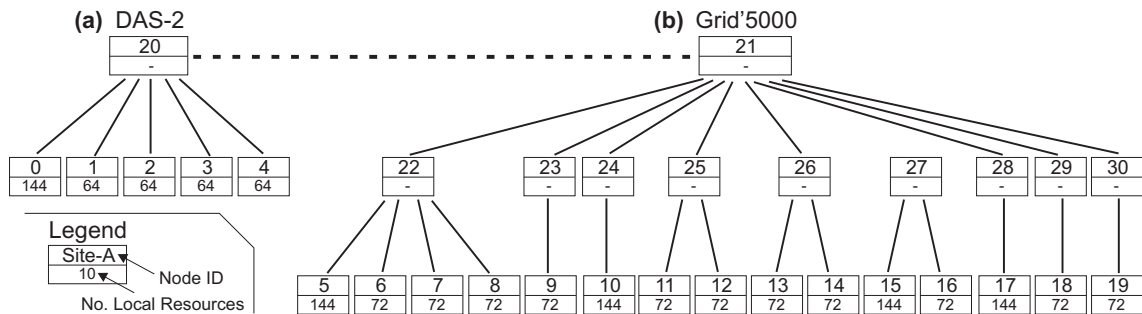
Figure 1: The logical structure of the dual-grid system composed of (a) the DAS, and (b) Grid'5000. Leaves in this structure represent actual clusters of resources. Nodes labelled 20 through 30 are administrative-only

authority (parent) to form a *completely decentralized* network. In this way, we attempt to combine the efficiency and the control of traditional hierarchical architectures with the scalability and the reliability of completely decentralized approaches. Second, we operate this architecture through *delegated matchmaking* in which requests for resources are delegated up-and-down the hierarchy, and within the completely decentralized networks. When resource request matches are found, the matched resources are delegated from the resource owner to the resource requester. By delegating resources to jobs instead of the traditional migration of jobs to resources, we lower the administrative overhead of managing user/group accounts on each site where they can use resources. Our architecture can be used as an addition to existing (local) resource managers.

We assess the performance of our architecture, and compare it against five architectural alternatives. Our experiments use a simulated system with 20 clusters and over 3000 resources. The workloads used throughout the experiments are either real long-term grid traces, or synthetic traces that reflect the properties of grid workloads. Our study shows that:

1. Our architecture achieves a good load balance for any system load, and in particular for high system loads.

2. Our architecture achieves a significant increase in goodput [5] and a reduction of the average job wait time, when compared to centralized and decentralized approaches. Furthermore, when facing severe imbalance between the loads of the system's composing grids, our architecture achieves a much better performance than its alternatives, while keeping most of the traffic in the originating grids.

3. The overhead of our architecture, expressed in number of messages, remains low, even for high system loads.

The remainder of the paper is structured as follows. In Section 2 we formulate the scenario that motivates this work: inter-operating the DAS and Grid'5000 grids. In Section 3 we survey briefly the architectural and the operational spectra of meta-scheduling systems. We illustrate our survey with a selection of real systems. In Section 4 we introduce our architecture for inter-operating grids. We assess the performance of our architecture, and that of five architectural alternatives, in Section 5. Last but not least, in Section 6 we present our conclusions, and hint towards future work.

2. THE MOTIVATING SCENARIO: INTER-OPERATING THE DAS AND GRID'5000

We consider as a motivating scenario the inter-operation of two grid environments, the DAS [10] and Grid'5000 [7].

2.1 The Dual-Grid System: Structure and Goals

The DAS environment (see Figure 1a) is a wide-area distributed system consisting of 400 processors located at five Dutch Universities (the cluster sizes range from 64 to 144). The users, a scientific community sized around 300, are associated with a home cluster, but a grid infrastructure grants DAS users access to any of the clusters. Each cluster is managed by an independent local cluster manager. The cluster owners may decide to partially or to completely take away the cluster resources, for limited periods of time. The DAS workload comprises a large variety of applications, from single-CPU jobs to parallel jobs that may span across clusters. Jobs can arrive directly at the local clusters managers, or to the KOALA meta-scheduler [21].

The Grid'5000 environment (see Figure 1b) is an experimental grid platform consisting of 9 sites, geographically distributed in France. Each site comprises one or several clusters, for a total of 15 clusters and over 2750 processors inside Grid'5000. The users, a community of over 600 scientists, are associated with a site, and have access to any of the Grid'5000 resources through a grid infrastructure. Each individual site is managed by an independent local cluster manager, the OAR [6], which has advance reservation capabilities. The other system characteristics, e.g., the cluster ownership and the workload, are similar to those of the DAS.

The combined environment that is formed by inter-operating the DAS and Grid'5000 comprises 20 clusters, and over 3000 processors. The goal of this combined environment is to increase the performance–*reduce the job slowdown, even in a highly utilized system*. The performance should be higher than that of the individual systems, taken separately. However, in achieving this goal we have to ensure that:

1. The load is kept local as much as possible, that is, jobs submitted in one grid should not burden the other if this can be avoided (the "keep the load local" policy).

2. The inter-connection should not require that each user, or even that each group, should have an account on each cluster they wish to use.

3. The clusters should continue running their existing resource management systems.

2.2 Load Imbalance in Grids

A fundamental premise of our delegated matchmaking architecture is that there exists load imbalance between different parts of the dual-grid system. We show in this section that this imbalance actually exists.

We want to assess the imbalance between the loads of individual clusters. To this end, we analyze two long-term and complete traces of the DAS and of the Grid'5000 systems,
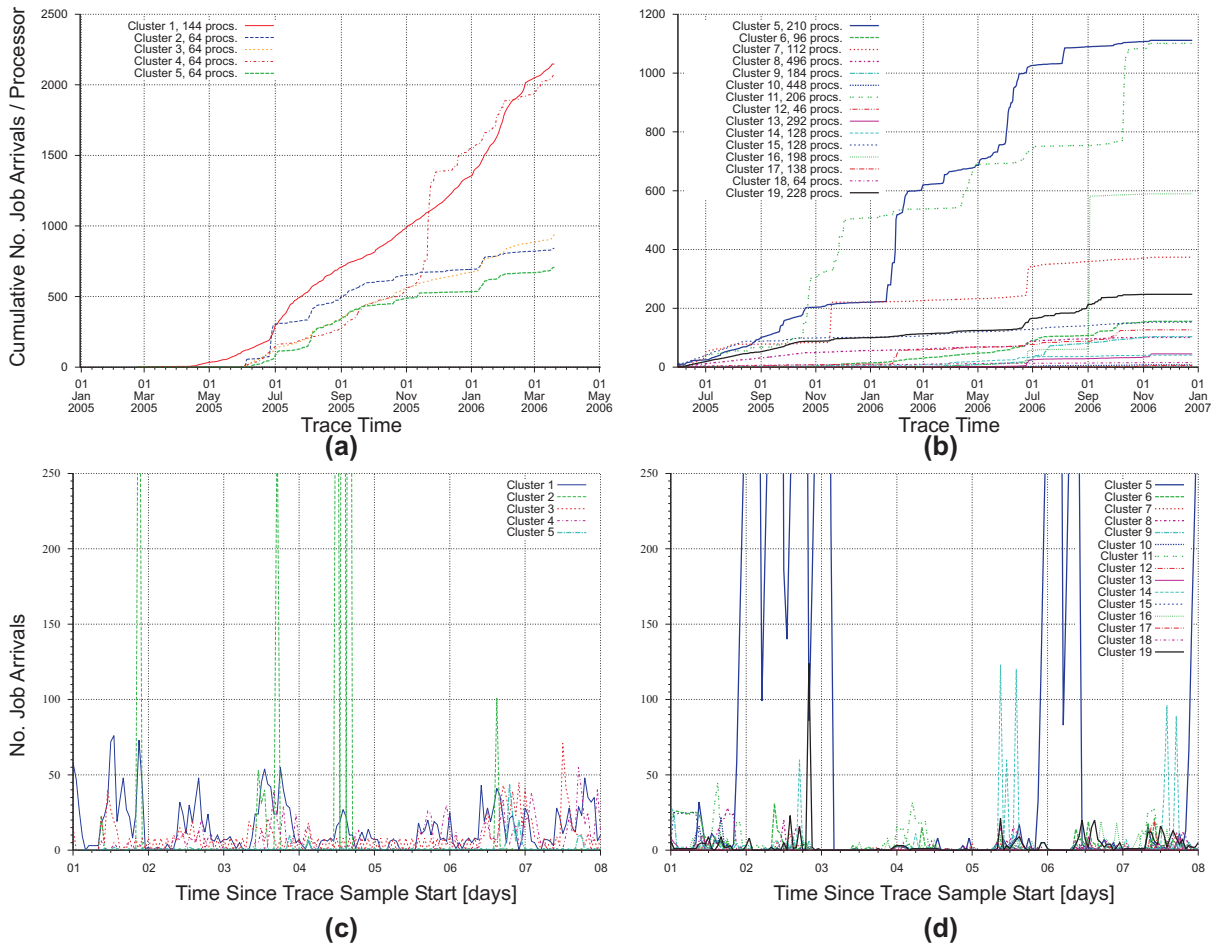
Figure 2: Load imbalance between clusters of the same grid. (a) and (b) the cumulative normalized daily load of the clusters in the DAS and Grid'5000 systems over time.; (c) and (d) the hourly load of the clusters in the DAS and Grid'5000 systems over time. Higher imbalance is denoted by more space between curves.

taken from the Grid Workloads Archive (GWA) [1]: traces GWA-T-1 and GWA-T-2, respectively. The traces, sized respectively over 1,000,000 and over 750,000 jobs, contain for each job information about the cluster of arrival, the arrival time, the duration, the number of processors, etc.

We define the *normalized daily load* of a cluster as the number of job arrivals over a day divided by the number of processors in the cluster during that period. We define the *hourly load* of a cluster as the number of job arrivals during hourly intervals. We distinguish two types of imbalance between the cluster loads, overall and temporary. We define the *overall imbalance* between two clusters over a period of time as the ratio between their normalized daily loads cumulated until the end of the period. We define the *temporary imbalance* between two clusters over a period of time as the maximum value of the ratio between the hourly loads of the two clusters, computed for each hour in the time period. The overall imbalance characterizes the load imbalance over a large period of time, while accounting for the differences in cluster size. The temporary imbalance characterizes the load imbalance over relatively short periods of time, regardless of the cluster sizes.

Figure 2(a) shows the cumulative normalized daily load of the DAS system, over a year, from 2005-03-20 to 2006-03-21. The right-most value indicates the average number of jobs served by each single processor during this period.

The maximum overall load imbalance between the clusters of the DAS system is above 3:1. Figure 2(c) shows the hourly load of the DAS system, over a week, starting from 2005-06-01. During the interval 2pm-3pm, 2005-06-04, there are over a thousand jobs arriving at cluster 2 and only one at cluster 5. The maximum temporary load imbalance between the clusters of the DAS system is over 1000:1. We have obtained similar results for the Grid'5000 traces, as shown by Figures 2(b) and 2(d).

We conclude that there exists a great potential to reduce the delays through load balancing across DAS and Grid'5000.

3. A BRIEF REVIEW OF METASCHEDULING SYSTEMS

In this section we review several meta-scheduling systems, from an architectural and from an operational point of view, and for each we give a concise description and a reference to a real system.

3.1 Architectural Spectrum

We consider a multi-cluster grid. Below we briefly present our taxonomy of architectures that can be used as grid resource management systems (GRMS). We illustrate this taxonomy in Figure 3.
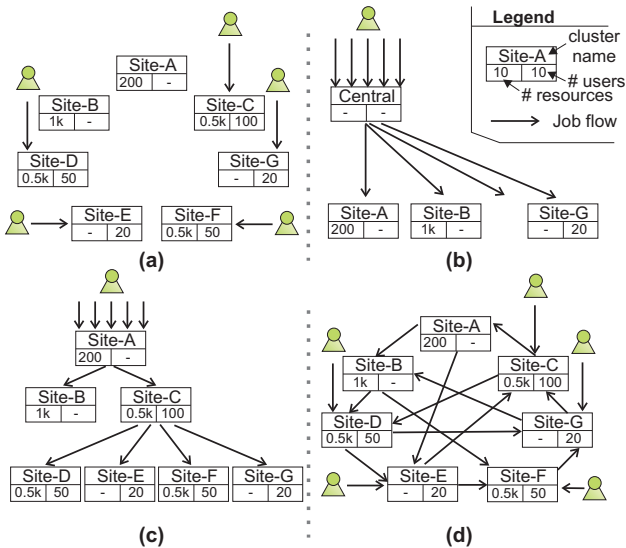
Figure 3: The meta-scheduling architectures: (a) independent clusters; (b) centralized meta-scheduler, (c) hierarchical K-level meta-scheduler; (d) distributed meta-scheduler with static links.

Independent clusters: (included for completeness) each cluster has its local resource management system (LRMS), i.e., there is no meta-scheduler. Users have accounts on each of the clusters they want to submit jobs to. For each job, users are faced with the task of selecting the destination cluster, typically a cumbersome and error-prone process.

Centralized meta-scheduler: there exists one (central) system queue, where all grid jobs arrive. From the central queue, jobs are routed towards the clusters where they are dispatched. The clusters may optionally employ an LRMS, in which case jobs may also arrive locally. It may be possible for the manager of the central queue to migrate load from one LRMS to another.

Hierarchical K-Level meta-scheduler: there exists a hierarchy of schedulers. Typically, grid jobs arrive either at the root of the hierarchy, or at the clusters' LRMSs. In both cases, jobs are routed (migrated) towards the clusters' LRMSs. The Hierarchical 2-Level metascheduler is the most encountered variant [22, 8] .

Distributed meta-scheduler: similarly to the independent clusters architecture, each cluster has its LRMS, and jobs arrive at the individual clusters' LRMSs. In addition, cluster schedulers can share jobs between each other. This forms in effect a distributed meta-scheduler. We distinguish between two ways of establishing links between clusters (sharing): static (fixed by administrator, e.g., at system start-up), and dynamic (automatically selected). We also call a distributed meta-scheduler architecture with static link establishment a *federated clusters* architecture.

Hybrid distributed/hierarchical meta-scheduler: each grid site, which may contain one or several clusters, is managed by a hierarchical K-Level meta-scheduler. In addition, the root meta-schedulers can share the load between each other. Other load-sharing links can also be established.

System	Architecture	Operation
Condor [26]	Independent	Matchmaking
Globus GRAM [9]	Independent	Job routing
Alien [23]	Centralized	Job pull
Koala [21]	Centralized	Job routing
OAR(Grid) [6]	Centralized	Job routing
CCS [22]	Hierarchical 2-Level	Job routing
Moab/Torque [8]	Hierarchical 2/3-Level	Job routing
NorduGrid ARC [11]	Indep./Federated	Job routing
NWIRE [24]	Federated	Job routing
Condor flocking [12]	Federated	Matchmaking
OurGrid [4]	Distributed, dynamic	Job routing
Askalon [25]	Distributed, dynamic	Job routing

Table 1: Currently deployed meta-scheduling systems. This work proposes a hybrid distributed/hierarchical architecture, operated through MatchMaking.

3.2 Operational Spectrum

We define an operational model as the mechanism that ensures that jobs entering the system arrive at the place where they can be run. We identify below three operational models employed by today's resource management systems.

Job routing: jobs are routed by the schedulers from the arrival point to the resources where they can run through a push operation (scheduler-initiated routing).

Job pulling: jobs are acquired by (unoccupied) resources from a higher-level scheduler through a pull operation (resource-initiated routing).

MatchMaking: jobs and resources are connected to each other by the resource manager, which thus acts as a broker responding to requests from both sides (job- and resource-initiated routing).

3.3 Real Systems

There exist several resource management systems that can operate a multi-cluster grid. Below we present a selection, which we summarize in Table 1, including references.

The Globus GRAM is a well-known middleware for managing independent clusters environments. It is operated through job routing. Globus GRAM is used in research and in industrial grids.

The Alien, Koala, and OARGrid architectures are all centralized. Alien is used in (a part of) CERN's production grid, and is operated through job pull. Koala and OARGrid are used in research grids, and are operated through job push. They are some of the first meta-schedulers which can co-allocate jobs, that is, they can simultaneously allocate resources located in different clusters for the same job.

CCS and Moab/Torque are both hierarchical meta-schedulers. CCS is one of the first hierarchical meta-schedulers that can operate clusters and super-computers together; it was used mainly in research environments. The commercial package Moab/Torque is currently one of the most used resource management systems.

The NorduGrid ARC implements an independent clusters architecture operated through job routing. However, the job submission process contacts cluster information systems from a fixed list, and routes jobs to the site where they could be started the fastest. This effectively makes NorduGrid a federated clusters architecture.

NWIRE, OurGrid, and Askalon are all distributed clusters architectures operated through job routing. NWIRE and OurGrid implement a federated clusters architecture.

NWIRE is the first such architecture to explore economic, negotiation-based interaction between clusters. OurGrid is the first to use a "tit-for-tat" job migration protocol, in which a destination site prioritizes migrated load by the number of jobs that it has migrated in the reverse direction in the past. Finally, Askalon is the first to build a negotiation-based distributed clusters architecture with dynamic link establishment.

Condor is a cluster management system. As such, it can straightforwardly be used in an independent clusters environment. However, through its flocking mechanism, Condor can be used in a federated clusters environment. In both cases, Condor operates through MatchMaking. Condor is widely used in research and production clusters.

4. THE DELEGATED MATCHMAKING ARCHITECTURE

In this section we present our resource management architecture for inter-operating multi-cluster grids: the delegated matchmaking architecture (DMM). We first build a hybrid distributed/hierarchical meta-scheduler architecture (see Section 3.1). Then, we operate it using (delegated) matchmaking (see Section 3.2).

4.1 Overview

We now define how grid clusters and other administrative units, from hereon *sites*, are connected. We aim to create a *network of sites* that manage the available resources, on top of and independently of the local cluster resource managers.First, sites are added according to administrative and political agreements, and *parent-child* (hierarchical) *links* are established. Thus, a hierarchy of sites is formed, in which the individual grid clusters are leaves of the hierarchical tree. Then, supplementary to the hierarchical links, *sibling links* can be formed between sites at the same hierarchical level and operating under the same authority (parent site).

Each site administers directly its local resources and the workloads of its local users. However, a site may have no local resources, or no local users, or both. A site with no local resources can be installed for a research laboratory with no local computing resources. A site without local users can be installed for an on-demand computing center. A site without users or resources serves only administrative purposes.

For our motivating scenario, described in Section 2, we create the hierarchical links between the sites as in Figure 1. Additionally, the sites 0-4, 5-8, 11-12, 13-14, 15-16, 22-30, and 20-21 are also inter-connected with sibling links, respectively. Sites 20 through 30 have been installed for administrative purposes. To avoid ownership and maintenance problems, there is in fact no root of the hierarchical tree. Instead, sites 20 and 21 serve as roots for each of the two grids, and are connected through a sibling link.

We operate our architecture through (delegated) matchmaking. The main idea of our delegated matchmaking mechanism is to *delegate resources' ownership to the user that requested them through a chain of sites (and of resource leases), and by adding the resource transparently for the user to the local user's site*. Binding the resource to the local user's site stands in contrast to the typical practice in today's systems based on either job routing or job pull, where jobs are sent to (or acquired from) the remote resources, where they are executed. This major change can be beneficial in practice: the resources are added to the trusted pool of resources of a neighboring site (simplifies security issues), and current systems already provide adequate mechanisms (e.g., the Condor glide-in [26]) that allow resources to be dynamically and temporarily added to a site without the need of root access to the resource (simplifies technical issues). On the contrary, when delegating jobs to resources, the resource management system needs to understand the job's semantics, and in particular the file dependencies, the job's structure, and the job-to-resource mapping strategy.

4.2 Local Operation

We assume that each of the grid's clusters uses a Condor-like resource management system. This assumption allows us to consider in our architecture only the mechanisms by which the clusters are inter-operated, while benefiting from the local resource management features of Condor [26]: complete administrative control over owned resources (resources can reject jobs), high tolerance to resource failures, the ability to dynamically add/remove computing resources (through matchmaking and glide-in). This also ensures that the administrators of the grid clusters will understand easily our architecture as it uses concepts from the Condor world, such as matchmaking.

Similarly to Condor, in our architecture each cluster is managed by a *site manager* (SM), which is responsible for gathering information about the local resources and jobs, informing resources about their match with a job and vice-versa, and maintaining the resource leasing information. According to this definition, our SM is equivalent to Condor's Negotiator, Collector, and Accountant components combined. Each resource is managed by a *resource manager* (RM), which will mainly be occupied with starting and running user's jobs. Each user has (at least) one permanent *job manager* (JM), which acts as an application-centric scheduler that obtains resources from the local SM. Our RM and JM definitions correspond to those of Condor's Start and Sched daemons, respectively. In addition to the Condor-specific functions, in our architecture a site manager is also responsible for communicating with other site managers.

4.3 The Delegated MatchMaking Mechanism

We now present the operational mechanism of our architecture, the *delegated matchmaking mechanism*, for obtaining remote resources. Job manager JM-1 informs its SM about the need for resources, by sending a *resource request* (Step 1 in Figure 4). The resource request includes the type of and number of resources JM-1 requires. At its next delegation cycle, site manage SM-1 establishes that it cannot serve locally this request, and decides to delegate it. SM-1 selects then contacts SM-2 for this delegation (Step 2). To make the selection, SM-1 uses its target site ordering policy (see Section 4.4). During its next matchmaking cycle, SM-2 finds enough free resources, and delegates them to SM-1 through its local resource management protocol (Step 3). Then, SM-1 claims the resources, and adds them to its local environment (Step 4). At this point, a *delegation chain* has been created, with SM-2 being the *delegation source* and SM-1 the *delegation sink*. During its next delegation cycle, SM-1 handles JM-1's request using its own resource management protocol (Step 5). Upon receiving the resources, JM-1 starts the user's job(s) on RM-1 (Step 6). Finally, after the
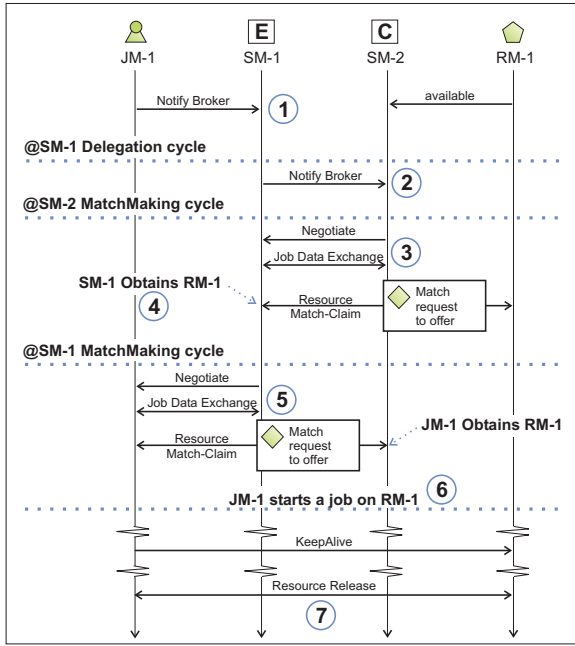
Figure 4: The delegated matchmaking mechanism, during a successful match.

Figure 5: A worst-case performance scenario for delegated matchmaking.

user job(s) have finished, JM-1 releases the resource, by informing both RM-1 and SM-1 (Step 7). The resource release information is transmitted backwards along the delegation chain: SM-1 informs SM-2 of the resource release. If SM-2's local resource management is Condor-based, RM-1 will also inform SM-2 of its release.

During Steps 1-7, several parties (e.g., JM-1, SM-1, SM-2, and RM-1) are involved in a string of negotiations. The main potential failures occurring in these multi-party negotiations are addressed as follows. First, an SM may not find suitable resources, both locally or through delegation. In this case, the SM sends back to the delegation requester (another SM) a DelegationReject message. Upon receiving a DelegationReject message, an SM will attempt to select and contact another SM for delegation (restart from Step 2 in Figure 4). Second, to prevent routing loops, and for efficiency reasons, the delegation chains are limited to a maximum length, which we call the delegation time-to-live (DTTL). Before delegating a resource request, the SM decreases its DTTL by 1. A resource request with a DTTL equal to 0 cannot be delegated. To ensure that routing loops do not occur, SMs retain a list of resource requests they have seen during the past hour. Third, we account for the case when the user changes his intentions, and cancels the resource request. In this case, JM-1 is still accountable for the time during which the resource management was delegated from SM-2 to SM-1, and charges the user for this time. To prevent being charged, the user can select a reduced DTTL, or even a DTTL of 0. However, requests with a DTTL of 0 will possibly wait more for available resources.

Delegated matchmaking promises to significantly improve the performance of the system, by occupying otherwise unused free resources with waiting jobs (load balancing). However, it can also worsen the performance of the system, by poor resource selection and by poor delegation routing. The resource selection is mostly influenced by the load management algorithm (discussed in Section 4.4). Figure 5 shows a worst-case performance scenario for delegation routing. Del-
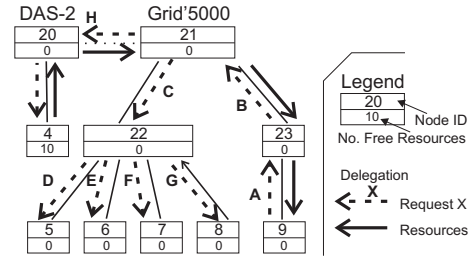
egation requests in the figure are ordered in time in their lexicographical order (i.e., delegation A occurs before delegation B). Site 9 issues a delegation request to site 23. The site schedulers base their decision only on local information. Due to the lack of information, sites are unable to find the appropriate candidate (here, site 4), and unnecessary delegation requests occur. This leads in turn to messaging overheads, and to increased waiting times, due to waiting for the delegation and matchmaking cycles of the target sites. Additionally, the decision to delegate resource requests can lead to a suboptimal number of delegations, either too few or too many. All these load management decisions influence decisively the way the delegated matchmaking mechanism is used. We dedicate therefore the next section to load management.

4.4 The Delegated MatchMaking Policies

To manage the load, we use two independent algorithms: the delegation algorithm, and the local requests dispatching algorithm. We describe them and their associated policies below.

The *delegation algorithm* selects what part of the load to delegate, and the site manager from which the resources necessary to serve the load can be obtained. This algorithm is executed whenever the current system load is over an administrator-specified *delegation threshold*, and at fixed intervals of time. First, requests are ordered according to a customizable *delegation policy*, e.g., FCFS. Then, the algorithm tries to delegate all the requests, in order, until the local load gets below the threshold that triggered the delegation alarm. The algorithm has to select for each request a possible target from which to bring resources locally. By design, the potential targets must be selected from the site's neighborhood. The neighbors are ordered according to a customizable *target site ordering policy*, which may take into account information about the current status of the target (e.g., its number of free resources), and an administrator selection of the request-to-target fitting (e.g., Best Fit). Upon finding the best target, the delegation protocol is initiated.

The *local requests dispatching policy* deals with the ordering of resource requests, both local and delegated. Similarly to the Condor's matchmaking cycle, we call this algorithm periodically, at intervals normally longer than those of the delegation algorithm cycle. The administrator may select the local request dispatching policy.

We argue that our architecture is operated with a generic load management mechanism. The three policies defined above allow for many traditional scheduling algorithms, and in particular gives our architecture the ability to leverage existing well-established on-line approximation algorithms [3].The target site ordering policy enables the use in our architecture

of many of the results in traditional networking/queuing theory [19]. However, we consider policy exploration and tuning outside the scope of this paper.

5. THE EXPERIMENTAL EVALUATION

In this section we present the experimental evaluation of our architecture for inter-operating grids. We first describe the typical grid workloads, which are significantly different from the workloads of traditional parallel production environments [15, 20]. Specifically, a grid workload comprises a high number of single-processor jobs, which are sent to the grid in batches (Section 5.1).

We then present our experimental setup (Section 5.2): a simulated environment encompassing both the DAS and Grid'5000 grids, for a total of 20 sites and over 3300 processors. The workloads used in our experiments are either year-long traces collected from the individual grids starting at identical moments in time, or synthetic traces that reflect the properties of grid workloads. Our experiments are aimed at characterizing:

- The behavior of the DMM architecture and of its alternatives, for a duration of one year (Section 5.3);
- The performance of the DMM architecture, and of its alternatives, under various load levels (Section 5.4);
- The effects of an imbalance between the loads of different grids on the performance of the DMM architecture and of its alternatives (Section 5.5);
- The influence of the DMM's delegation threshold parameter on the performance of the system (Section 5.6).

5.1 Intermezzo: Typical Grid Workloads

Two main factors contributing to the reduction of the performance of large-scaled shared systems, such as grids, are the overhead imposed by the system architecture (i.e., the messages, the mechanisms for ensuring access to resources etc.), and the queuing effects due to the random nature of the demand. While the former is under the system designer's control, the latter is dependent on the workload. Despite a strong dependency of performance on the system workload, most of the research in grid resource management does not employ realistic workloads (i.e., trace-based, or based on a validated workload model with realistic parameter values). For the few reported research results that attempt to use realistic workloads, the traces considered have been taken from or modelled after the Parallel Workloads Archive (PWA) [2]. However, there exist significant differences between the parallel supercomputers workloads in the PWA and the workloads of real grid environments. In this section we present two important distinctions between them.

First, the percentage of "serial" (single-processor) jobs is much higher in grid traces than in the PWA traces. There exist 70%-100% single-processor jobs in grid traces (the percentage grows to 99-100% in most production grid environments), but only 20%-30% in the PWA traces [20, 16]. There are two potential consequences: on the one hand, the resource managers become much more loaded, due to the higher number of jobs. On the other hand, the resource managers can be much simpler, since individual single-processor jobs raise fewer scheduling issues.

Second, the grid single-processor jobs typically represent instances of conveniently parallel jobs, or batch submissions. A batch submission is a set of jobs ordered by the time when
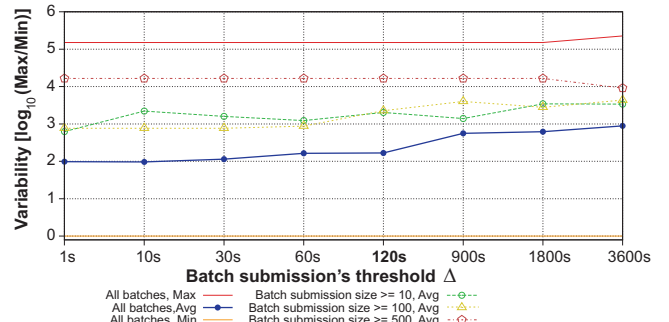

Figure 6: The variability of the runtimes of jobs in batch submissions in grids.

they arrive in the system, where each job was submitted at most Δ seconds after the first job ($\Delta = 120s$ is considered the most significant). In a recent study, Iosup et al. [17] show that 70% of the jobs, accounting for 80% of the consumed processor time, are part of batch submissions. The batch submissions are usually managed by batch engines, and the individual jobs arrive in the system independently. Figure 6 shows that the runtime of jobs belonging to the same batch submission varies on average by at least two orders of magnitude, and that the variability increases towards five orders of magnitude as the size of the batch reaches 500 or more jobs. The predominance of batch submissions and their jobs' high runtime variability have a high impact on the operation of a large number of today's cluster and grid schedulers. Indeed, the user must submit many jobs as a batch submission with a *single* runtime estimate. Hence, the user cannot estimate the runtime of individual jobs, other than specifying a large value, typically the largest value allowed by the system. As a result, the scheduling schemes relying on user estimates, e.g., all backfilling variants [27],are severely affected.

5.2 The Experimental Setup

5.2.1 The Simulator

We have developed a custom discrete event simulator to simulate the combined DAS-2 and Grid'5000 grid system (see Section 2). Each of the 20 clusters of the combined system receives an independent stream of jobs. Depending on each job's parallelism, one or several resources are assigned to it exclusively, from the time when the job starts until the time when the job finishes.

We attempt to evaluate the steady-state of the simulated system. To this end, unless otherwise specified the last simulated event in each simulation is the arrival of the last job, all job streams considered together. This ensures that our simulation does not include the cool-down phase of the system, in which no more jobs arrive while the system finishes the remaining load. The inclusion of the cool-down phase may bias the performance metrics, especially if the last jobs queue at only few of the clusters. We do not perform a similar elimination for system warm-up, as (a) we cannot distinguish reliably between the warm-up period and the normal system behavior, and (b) given the long duration of the jobs (see the workloads description in Section 5.2.2), the start-up period is small compared to the remainder of the simulation, especially for high load levels.

The simulator assesses the following performance metrics:

Utilization, Wait and Response Time, Slowdown We consider in this work the average values of the system

utilization (**U**), average job wait time (**AWT**), average job response time, and average job slowdown (**ASD**). For a review of these traditional performance metrics, we refer to [13].

Goodput, expressed as the total processing time of jobs that finish successfully, from the point of view of the grid resource manager (similar to the traditional definition [5], but taking into consideration that all grid jobs are executed "remotely" from the user's perspective). For the DMM architecture, we also measure the goodput of jobs running on resources obtained through delegated matchmaking. Furthermore, we account for goodput obtained on resources delegated from the same site (*intra-site goodput*), from the same grid (*intra-grid goodput*), and between the grids (*inter-grid goodput*).

Finished Jobs (JF%), expressed as the percentage of jobs that finish, from the jobs in the trace. Due to the cool-down period elimination, the maximum value for this metric is lower than 100%.

Overhead We consider the overhead of an architecture as the number of messages it employs to manage the workload. There are five types of messages: Notify Broker, Negotiate, Job Data Exchange, Resource Match-Claim-Release, and DMM (the last specific to our architecture). The *Overhead* is then expressed as a set of five values, one for the number of messages of each type. Additionally, we consider for our architecture the *number of delegations of a job*, which is defined as the length of its delegation chain.

5.2.2 The Workloads

We use two workload selection approaches: real grid traces, and synthetic workloads based on properties of real grid traces. To the best of our knowledge, ours is the first study that takes into account the difference between the parallel supercomputers workloads (comprising mostly parallel jobs), and the workloads in real grid environments (comprising almost only single-node jobs).

To validate our approach (Section 5.3), we use traces collected for each of the simulated clusters, starting at identical moments in time (the traces are described in Section 2). However, these traces raise two problems. First, they incur a load below 20% of the combined system [16]. Second, they represent the workload of research grid environments, which contains many more parallel jobs than in a production grid.

To address both these issues, we employ a model-based trace generation for the rest of our experiments. We use the Lublin and Feitelson model (LFM) [20], which has deservedly become the de-facto standard for the community that focuses on resource management in large-scale computing environments. Using this model, we generate streams of rigid jobs (that is, whose size is fixed at the job's arrival in the system) for each cluster. Unless otherwise specified, we use the default LFM parameter values. The job arrival times during the peak hours of the day are modelled in LFM using a Gamma distribution. To generate jobs for a longer period of time, the LFM uses daily cycle with a sinusoidal shape, where the highest value of the curve corresponds to the peak hours. The job parallelism is modelled for three classes: single-processor jobs, parallel jobs with a power-of-two number of nodes, and other parallel jobs. We change the default value of the probability of a new job to be single-processor, p, to reflect the values encountered

System	Architecture	Operation
condor	Independent	Matchmaking
sep-c	Independent	Job routing
cern	Centralized	Job pull
koala	Centralized	Job routing
fcondor	Federated	Matchmaking
DMM	Distributed	Matchmaking

Table 2: Simulated meta-scheduling architectures.

in grid systems: $p = 0.95$ [16]. The LFM divides the remaining jobs between the parallel jobs classes, with equal probability. The actual runtime time of a job is modelled with a hyper-Gamma distribution with two stages; for parallel jobs, the parameter that represents the probability of selecting the first hyper-Gamma stage over the second depends linearly on the number of nodes. Thus, the largest jobs have a high probability of also having a long runtime. With these parameters, the average job runtime is around one hour.

By modifying the parameters of the Lublin-Feitelson model that characterize the inter-arrival time between consecutive jobs during peak hours, we are able to generate a load of a given level (e.g., 70%), for a system of known size (e.g., 128 processors), during a specified period (e.g., 1 month). Using this approach, we generate 10 sets of 20 synthetic job streams (one per simulated cluster) for each of the following load levels: 10%, 30%, 50%-100% in increments of 10%, 95%, 98%, 120%, 150%, and 200%. We call the *default load levels* the following nine load levels: 10%, 30%, 50%, 60%, 70%, 80%, 90%, 95%, and 98%. The results reported in Sections 5.4, 5.5, and 5.6 are for workloads with a duration of 1 day, for a total of 953 to 39550 jobs per set (11827 jobs per set, on average). We have repeated some of the experiments in Sections 5.4 and 5.6 for traces with the duration of 1 week and 1 month, with similar results.

5.2.3 The Simulated Architectures

For the simulation of the DMM architecture, unless otherwise noted, we use a delegation threshold of 1.0 and a matchmaking cycle of 300s. Throughout the experiments, we employ a FCFS delegation policy, a target site ordering policy that considers only the directly connected neighbors of a site, and a FCFS local requests dispatching policy. We simulate five alternative architecture models, described below and summarized in Table 2.

1. **cern** This is a centralized meta-scheduler architecture with job pull operation, in which users submit all their jobs to a central queue. Whenever they have free resources, sites pull jobs from the central queue. Jobs are pulled in the order they arrive in the system.

2. **condor** This is an independent clusters architecture with matchmaking operation that simulates a Condor-like architecture. Unlike the real system, the emulation does not prioritize users through a fair-sharing mechanism [26]. Instead, at each matchmaking round jobs are considered in the order of arrival in the system. The matchmaking cycle occurs every 300s, the default value for Condor (see `NEGOTIATOR_INTERVAL` in the Condor manual).

3. **fcondor** This is a federated clusters architecture with matchmaking operation that simulates a Condor-like architecture with flocking capabilities [12]. The user's job manager will switch to a new site manager whenever the current site manager cannot solve all of its

resource demands. This simulation model also includes the concept of fair-sharing employed by Condor in practice [26]. At each matchmaking round, users are sorted by their past usage, which is reduced (decayed) with time. Then, users are served in order, and for each user all feasible demands are solved. Similarly to `condor`, the matchmaking cycle occurs every 300s. The performance of the `fcondor` simulator corresponds to an optimistic performance estimation of a real Condor system with flocking, for two reasons. First, in the `fcondor` simulator, we allow any job manager to connect to any site manager. This potentially reduces the average job wait time, especially when the grids receive imbalanced load, as JMs can use SMs otherwise unavailable. Second, jobs that cannot be temporarily served are bypassed by jobs that can. Given that 95% of the jobs are single-processor, this results in sequential jobs being executed before parallel jobs, when the system is highly loaded. Then, the resource fragmentation and the average wait time decrease, and the utilization increases.

4. `koala` This is a centralized meta-scheduler architecture with job push operation. Users submit all their jobs to a central queue. As soon as jobs arrive, the queue dispatches them on sites with free resources. Jobs stay in the queue until free resources are found. The information about the number of free resources is gathered periodically by a monitoring service. Note that the Koala implementation leverages non-conservative backfilling [21], and has therefore better performance than the simple model simulated here.

5. `sep-c` This is an independent clusters architecture with job push operation.

5.2.4 The Assumptions

In our simulations we make the following assumptions:

Assumption 1: No network overhead We assume a perfect communication network between the simulated systems, with 0-latency. Given the average job runtime of one hour, we argue that this assumption has little effect. However, we do present the number of messages used by our architecture to manage the workload.

Assumption 2: Identical processors To isolate the effects of the resource management solutions, we assume identical processors across all clusters. However, the system is heterogeneous in number of processors per cluster.

Assumption 3: FCFS scheduling policy at cluster-level We assume that each site employs a FCFS policy, *without* backfilling. Backfilling systems are effective when many parallel jobs exist in the system, and when accurate job runtime predictions are given by the users. This situation is uncommon in grids.

Assumption 4: Processors as scheduling unit In Condor, multi-processor machines can be viewed (and used) as several single-processor machines. We assume that this feature is available regardless of the modelled alternative architectures. Note that this increases the performance of the `cern`, `sep-c`, and `koala` architectures, and has no effect on the Condor-based `condor`, `fcondor`, and DMM.

Simulator	No.Jobs [kJobs]	AWT [s]	ASD	Goodput [CPUyr]	JF [%]
`cern`	455.4	44	6	117	100
`condor`	455.4	7,681	1,610	117	100
DMM	455.4	1,283	298	117	100
`fcondor`	455.4	2,570	255	117	100
`sep-c`	455.4	1,938	590	117	100

Table 3: Performance results when running real long-term traces.

Assumption 5: No background load In many grids, jobs may arrive directly at the local clusters' resource manager, i.e., bypassing the grid. However, there is little information on the load imposed by this additional workload in practice. Therefore, we assume that there exists no background load in the system.

5.3 Preliminary Real Trace-Based Evaluation

For this experiment, we first assess the behavior of the DMM architecture and of its alternatives when managing a contiguous subset of 4 months from the real grid traces described in Section 2, starting from 01/11/2005. Only for this experiment, we do not stop the simulation upon the arrival of the last job, and we do include the cool-down period. However, no jobs arrive after the 4-months limit.

Table 3 shows the performance results for running the real traces over the whole 4-months period. All architectures successfully manage all the load. However, the ASD and the AWT vary greatly across architectures. The `cern` has the smallest ASD and AWT, by a large margin. This is because, unlike the alternatives, it is centralized, and operates in a lightly loaded system, where little guidance is needed, but its speed is critical for good performance. The `fcondor` and the DMM have similar performance, and are both better than the independent clusters architectures (`condor` and `sep-c`). For the latter the job response time is dominated by the time spent waiting for resources. Independently of whether we use AWT or ASD, the `condor` has a poorer performance than `sep-c`: the time spent waiting for the next matchmaking cycle affects negatively the performance of `condor`.

We have repeated the experiments for a one-year sample with the same starting point, 01/11/2005. We have obtained similar results, with the notable exception of `fcondor`, whose AWT degraded significantly (it became the worst of all architectures !). We attribute this poor performance to the flocking target selection: if the target SM is also very loaded, `fcondor` wastes significant time, since the JM will have to wait for the target SM's next matchmaking cycle to discover that the latter cannot fulfill any demands. This gives further reasons for the development of a dynamic target selection mechanism such as the DMM.

5.4 Performance Assessment

In this section we assess the performance of the DMM architecture and of its alternatives under various load levels. We report the results for the default load levels (defined in Section 5.2.2). Figure 7 shows the performance of the DMM architecture and of its alternatives, under the default load levels. Starting with a medium system utilization (50%) and up, DMM offers better goodput than the alternatives. The largest difference is achieved for a load of 80%. At this load, DMM offers 32% more goodput than the `fcondor`. We attribute this difference to DMM's better target site selection policy, and quicker rejection of delegations by loaded sites that are
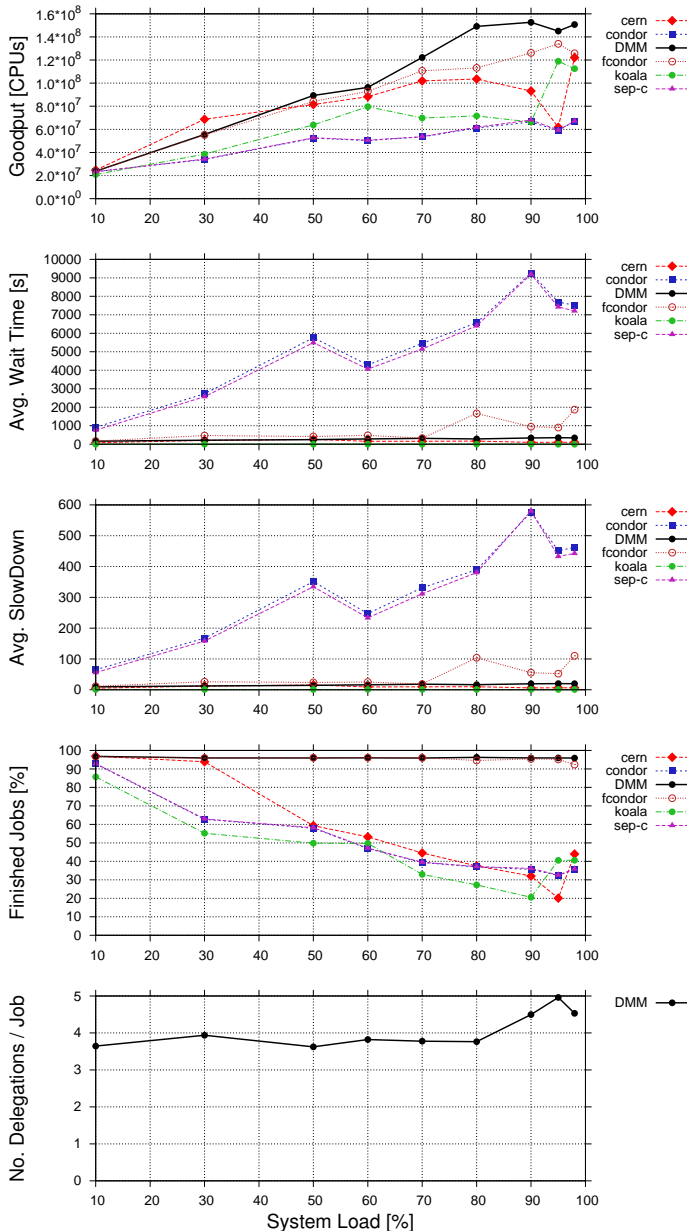
Figure 7: The performance of the DMM architecture compared with that of alternative architectures, for various system loads.

incorrectly targeted. The centralized meta-schedulers, `cern` and `koala`, offer in general lower goodput values with the notable exception of `cern`'s 24% improvement over the best architecture (DMM and `fcondor`, tied) for a load level of 30%.

The AWT and the ASD of DMM remain similar to that of central meta-scheduler architectures, regardless of the load level. However, DMM incurs lower AWT and ASD than `fcondor` at loads over 70%, and much better JF% than `cern` and `koala`. DMM and `fcondor` manage to finish many more jobs than their alternatives, in the same time: up to 93% more jobs finished, for load levels below 60%, and up to 378% more jobs finished, for loads up to 98%. We explain these large differences to the scheduling mechanism. The centralized architectures (`cern` and `koala`) operated with FCFS may keep many jobs blocked in the queue when the old-

est job cannot find its needed resources. The decentralized architectures (DMM and `fcondor`) act as *natural backfilling* algorithms, that is, they delegate the blocked jobs and dispatch the others. The large delegated jobs will not starve, due to the DTTL (see Section 4.3). Finally, there is a difference of 0%-4% in JF% between DMM and `fcondor`, in favor of DMM.

The independent cluster architectures, `sep-c` and `condor`, are outperformed by the other architectures for all load levels and for all performance metrics.

The additional performance comes at a cost: the additional messages sent by the DMM architecture for its delegations. Figure 7 also shows the number of delegations per job. Surprisingly, this overhead is relatively constant for all loads below 90%. This suggests that at medium to high load levels, the DMM manages to find suitable delegation targets in linear time, while using a completely decentralized routing algorithm. Above 80% load, the system is overloaded, and DMM struggles to find good delegation targets, which increases the number of delegations per job linearly with the load level increase.

5.5 Influence of Load Imbalance

In this section we present the effects of an imbalance between the loads of the two grids on the performance of the DMM architecture and of its alternatives. We simulate an imbalance between the load of the DAS, which we keep fixed at 60%, and that of Grid'5000, which we vary from 60% to 200%. Note that at load levels higher than 120% for Grid'5000, the two-grid system is overloaded.

Figure 8 shows the performance of the DMM for various imbalanced system loads. The figure uses a logarithmic scale for the average wait time and for the average slowdown. At 60%/100% load, the system starts to be overloaded, and all architectures but the DMM "suffocate", i.e., they are unable to start all jobs. At 60%/150%, when the system is truly saturated, only DMM can finish more than 80% of the load (it finishes over 95%). The DMM architecture is superior to its alternatives both in goodput and in percentage of finished jobs. Compared to its best alternative, `fcondor`, DMM achieves up to 60% more goodput, and finishes up to 26% more jobs. The `cern` architecture achieves lower ASD by *not* starting most of its incoming workload!

Similarly to the case of balanced load, the number of delegations per job is relatively constant for imbalanced loads of up to 60%/100%. Afterwards, the number of delegations per job increases linearly with the load level increase, but at a higher rate than for the balanced load case.

To better understand the cause for the performance of the DMM, we show in Figure 9 the breakdown of the goodput components for various imbalanced system loads. According to the "keep the load local" policy (defined in Section 2.1), the goodput on resources delegated between sites is low for per-grid loads below 100%. However, as soon as the Grid'5000 grid is overloaded, the inter-grid delegations become frequent, and the inter-grid goodput rises, to up to 37% from the goodput obtained on delegated resources. A similar effect can be observed for the intra-grid goodput, and for the intra-site goodput.

5.6 Influence of the Delegation Threshold

The moments when the DMM architecture issues delegations and the number of requested resources depend highly on the
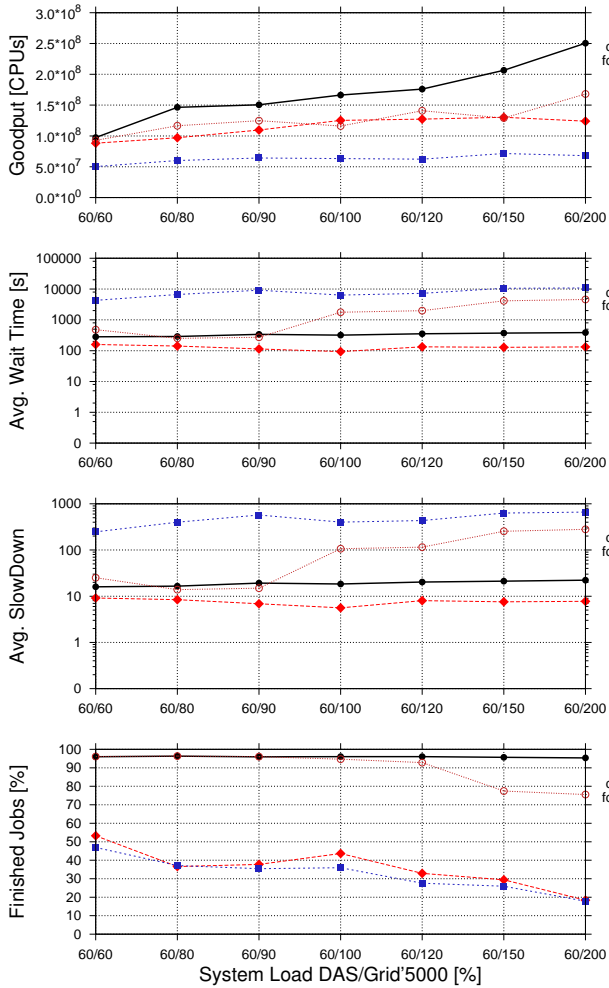
Figure 8: The performance of the DMM compared with that of alternative architectures, for various imbalanced system loads.
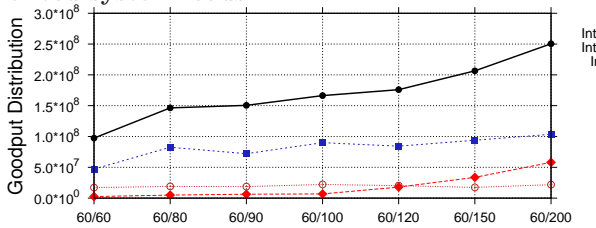

Figure 9: The components of goodput for various imbalanced system loads, with DMM: overall, inter-grid, intra-grid, and intra-site goodput.

delegation threshold. We therefore assess in this section the influence of the delegation threshold on the performance of the system.

Figure 10 shows the performance of the DMM architecture for values of the delegation threshold ranging from 0.60 to 1.25, and for six load levels ranging from 10% to 98%. A system administrator attempting to tune the system performance while keeping the overhead reduced, should select the best delegation threshold for the predicted system load. For a lightly loaded system, with a load of 30%, setting a delegation threshold higher than 1.0 leads to a quick degradation of the system performance. For a system load of 70%, con-
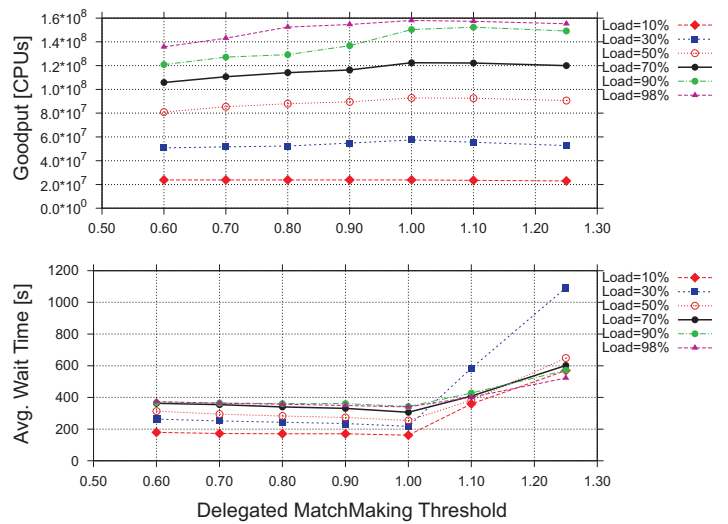

Figure 10: The performance of the DMM architecture for various values of the delegation threshold.
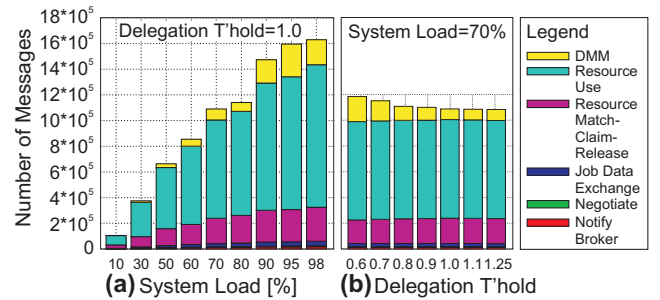

Figure 11: The distribution of the messages in the DMM architecture: (a) for various system load levels and a Delegation T'hold of 1.0; (b) for various Delegation T'hold values and a 70% system load level.

sidered high in systems that can run parallel jobs [18], the best delegation threshold is 1.0, as it offers both the best goodput, and the lowest AWT and ASD.

5.7 The Message Overhead

We define the *workload management messages* as the `NotifyBroker`, the `Negotiate`, the `Job Data Exchange`, and the `Resource Match-Claim-Release` messages used by the delegated matchmaking mechanism (see Section 4.3). Figure 11(a) shows the distributions of the number of workload management and of the `Resource Use` messages, for various system loads. DMM adds only up to 19% more messages to the workload management messages for load levels below 60%, but up to 97% for higher load levels. However, the majority of messages in Condor-based systems are the `KeepAlive` messages exchanged between the JM and the RM while user's jobs are being run by the RM. When taking into consideration the KeepAlive messages, the messaging overhead incurred by DMM is at most 16%.

Figure 11(b) shows the distribution of the messages in the DMM architecture, for various values of the delegation threshold. The system's load level is set to 70%. The number of DMM messages accounts for 7% to 16% of the total number of messages, and decreases with the growth of the delegation threshold. The workload management overhead grows from 35% (threshold 1.0) to 86% (threshold 0.6).

6. CONCLUSION AND FUTURE WORK

The next step in the evolution of grids is to inter-operate several grids into a single computing infrastructure, to serve larger and more diverse communities of scientists. This raises additional challenges, e.g., load management between separate administrative entities. In this paper we have proposed DMM, a novel delegated matchmaking architecture for inter-operating grids. Our hybrid hierarchical/distributed architecture allows the interconnection of several grids, without requiring the operation of a central point of the hierarchy. In DMM, when a user's request cannot be satisfied locally, remote resources are transparently added to the user's site through delegated matchmaking.

We have evaluated with simulations the performance of our proposed architecture, and compared it against that of five alternative architectures. A key aspect of this research is that the workloads used throughout the experiments are either real long-term grid traces, or synthetic traces that reflect the properties of grid workloads. The analysis of system performance under balanced inter-grid load shows that our architecture can accommodate equally well low and high (up to 80%) system loads. In addition, the results show that starting from a system utilization of 50% and up to 98%, the DMM offers a better goodput and a lower average wait time than the considered alternatives. As a result, DMM can finish up to 378% more jobs than its alternatives. The difference increases when the inter-operated grids experience high and imbalanced loads. Our analysis of performance under imbalanced inter-grid load reveals that, compared to its best alternative, DMM achieves up to 60% more goodput, and finishes up to 26% more jobs.

These results demonstrate that the DMM architecture can result in significant performance and administrative advantages. We expect that this work will simplify the current efforts in inter-operating the DAS and Grid'5000 systems, which are currently under way. To this end, we expect to implement and to deploy our architecture in the following year. From the technical point of view, we also intend to extend our simulations to a more heterogeneous platform, to account for resource and job failures, and to investigate the impact of existing and unmovable load at the cluster level. Finally, we hope that this architecture will become a useful step for sharing resources across grids.

7. REFERENCES

[1] The Grid Workloads Archive. [Online] http://gwa.ewi.tudelft.nl., Jul 2007.

[2] The Parallel Workloads Archive. [Online] http://www.cs.huji.ac.il/labs/parallel/workload/., Jul 2007.

[3] S. Albers and S. Leonardi. On-line algorithms. *ACM Comput. Surv.*, 31(3es):4, 1999.

[4] N. Andrade et al. OurGrid: An approach to easily assemble grids with equitable resource sharing. In *JSSPP*, volume 2862 of *LNCS*, pages 61–86, 2003.

[5] J. Basney and M. Livny. Improving goodput by co-scheduling cpu and network capacity. *J. HPCA*, 13(3), 1999.

[6] N. Capit et al. A batch scheduler with high level components. In *CCGrid*, pages 776–783, 2005.

[7] F. Cappello et al. Grid'5000: a large scale and highly reconfigurable experimental grid testbed. *J. HPCA*, 20(4):481–494, Nov. 2006.

[8] Cluster Resources Inc. Moab workload manager administrator's guide. Tech. Doc. v.5.0, Jan 2007.

[9] K. Czajkowski et al. A resource management architecture for metacomputing systems. In *IPPS/SPDP*, pages 62–82, 1998.

[10] Dutch University Backbone. The distributed ASCI supercomputer 2 (DAS-2), 2006.

[11] M. Ellert et al. Advanced Resource Connector middleware for lightweight computational grids. *FGCS*, 23(2):219–240, 2007.

[12] D. Epema, M. Livny, R. Dantzig, X. Evers, and J. Pruyne. A worldwide flock of Condors: Load sharing among workstation clusters. *FGCS*, 12:53–65.

[13] D. G. Feitelson and L. Rudolph. Metrics and benchmarking for parallel job scheduling. In *IPPS/SPDP*, volume 1459 of *LNCS*, pages 1–24, 1998.

[14] I. Foster and C. Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure.* Morgan Kaufmann Pub., 1999.

[15] H. Franke, J. Jann, J. E. Moreira, P. Pattnaik, and M. A. Jette. An evaluation of parallel job scheduling for ASCI Blue-Pacific. In *SC*, 1999.

[16] A. Iosup, C. Dumitrescu, D. H. Epema, H. Li, and L. Wolters. How are real grids used? The analysis of four grid traces and its implications. In *GRID*, pages 262–270. IEEE CS, 2006.

[17] A. Iosup, M. Jan, O. Sonmez, and D. Epema. The characteristics and performance of groups of jobs in grids. In *Euro-Par*, LNCS, 2007.

[18] J. P. Jones and B. Nitzberg. Scheduling for parallel supercomputing: A historical perspective of achievable utilization. In *JSSPP*, volume 1659 of *LNCS*, pages 1–16, 1999.

[19] L. Kleinrock. *Queueing Systems, Vol. 1: Theory.* Wiley, New York, 1975.

[20] U. Lublin and D. G. Feitelson. The workload on parallel supercomputers: modeling the characteristics of rigid jobs. *J. PDC*, 63(11):1105–1122, 2003.

[21] H. H. Mohamed and D. H. J. Epema. Experiences with the koala co-allocating scheduler in multiclusters. In *CCGrid*, pages 784–791. IEEE CS, 2005.

[22] A. Reinefeld et al. Managing clusters of geographically distributed high-performance computers. *CP&E*, 11(15):887–911, 1999.

[23] P. Saiz, P. Buncic, and A. J. Peters. AliEn resource brokers. In *Computing in High Energy and Nuclear Physics*, 2003. Also available as CoRR cs.DC/0306068.

[24] U. Schwiegelshohn and R. Yahyapour. Resource allocation and scheduling in metasystems. In *DCM*, volume 1593 of *LNCS*, pages 851–860, 1999.

[25] M. Siddiqui, A. Villazón, and T. Fahringer. Grid allocation and reservation - grid capacity planning with negotiation-based advance reservation for optimized qos. In *SC*, page 103, 2006.

[26] D. Thain, T. Tannenbaum, and M. Livny. Distributed computing in practice: the condor experience. *CP&E*, 17(2-4):323–356, 2005.

[27] A. M. Weil and D. G. Feitelson. Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling. *IEEE Trans. Parallel Distrib. Syst.*, 12(6):529–543, 2001.