Enabling Grid Interoperability among Meta-Schedulers

Ivan Rodero[a,*], David Villegas[b], Norman Bobroff[c], Yanbin Liu[c], Liana Fong[c], S. Masoud Sadjadi[b]

[a]*Center for Autonomic Computing, Department of Electrical & Computer Engineering, Rutgers, the State University of New Jersey, USA*
[b]*School of Computing and Information Sciences, Florida International University (FIU), Miami, Florida, USA*
[c]*IBM T.J. Watson Research Center, Hawthorne, New York, USA*

Abstract

The goal of grid computing is to integrate the usage of computer resources from cooperating partners in the form of Virtual Organizations (VO). One of its keys functions is to match jobs to execution resources efficiently. For interoperability between VOs, this matching operation occurs in resource brokering middleware, commonly referred to as the meta-scheduler or meta-broker. In this paper, we present an approach to a meta-scheduler architecture, combining hierarchical and peer-to-peer models for flexibility and extensibility. Interoperability is further promoted through the introduction of a set of protocols, allowing meta-schedulers to maintain sessions and exchange job and resource state using Web Services. Our architecture also incorporates a resource model that enables an efficient resource matching across multiple Virtual Organizations, especially where the compute resources and state are dynamic. Experiments demonstrate these new functional features across three distributed organizations (BSC, FIU, and IBM), that internally use different job scheduling technologies, computing infrastructure and security mechanisms. Performance evaluations through actual system measurements and simulations provide the insights on the architecture's effectiveness and scalability.

Keywords: Meta-scheduler, resource broker, interoperable scheduling protocol, resource model.

1. Introduction

Grid computing leverages resources from multiple cooperating institutions to form Virtual Organizations (VOs) [1] that provide computing power while sharing the cost of resource ownership. Recent advances in cooperating grids, or interoperating VOs, include fulfilling job requests using data and computing resources distributed across multiple grids. This vision of cooperating grids further provides the opportunity for global optimizations of resource usage, reducing job execution cost and power consumption. The challenges to achieve this vision of interoperable, globally distributed VOs are well documented. They include providing common interface descriptions, system models, and levels of abstraction to hide heterogeneity in the computing resources, security mechanisms, and job management policies of the collaborating organizations, as articulated in the paper by Rodero, et al. [2]. Several architectures have been proposed to achieve these goals including SPA [3], GridWay [4] and Koala [5].

This paper describes an approach based on interoperating meta-schedulers. The interoperable meta-scheduler model supports the autonomy of organizations and presents a common external interface to partner domains as indicated in Figure 1. It shows a peer-to-peer collaborative environment constructed for the purpose of experimentation between three VO domains among three partnering institutions within the LA Grid Initiative [6]: Florida International University (FIU), Barcelona Supercomputing Center (BSC) and IBM Research (IBM). Meta-schedulers serve as the point of contact for end users submitting jobs to these sites as well as for cooperating VOs to exchange control messages.

To enable interoperation among meta-scheduler, interfaces and functions are defined for the operations of job lifecycle management and the distribution of current state and availability of system wide computing resources. When a user submits a job to a meta-scheduler, the meta-scheduler decides whether to execute the job on local resources under its control or forward it to another meta-scheduler either because there is not enough suitable local resources or because the remote meta-scheduler is better suited to execute the job. Because the grid is a dynamic environment, good forwarding decisions require global distribution of resource state data to each meta-scheduler domain. As the system of Figure 1 is scaled up the volume of data exchanged becomes an important issue. Forwarding based on inaccurate data leads to failures in job dispatching at the target domain, while real time exchanges of complete resource state incur significant network and storage management costs at the meta-scheduler endpoints.

A peer-to-peer —as opposed to centralized— model of resource data distribution is investigated to be consistent with our overall meta-scheduler topology of Figure 1. In order to reduce the amount of resource data sent and stored at each node, several aggregation models are developed in Section 2. Each model represents an increasing degree of data consolidation,

*This author was formerly at Barcelona Supercomputing Center
Email addresses: irodero@cac.rutgers.edu (Ivan Rodero), dvill013@cs.fiu.edu (David Villegas), bobroff@us.ibm.com (Norman Bobroff), ygliu@us.ibm.com (Yanbin Liu), llfong@us.ibm.com (Liana Fong), sadjadi@cs.fiu.edu (S. Masoud Sadjadi)

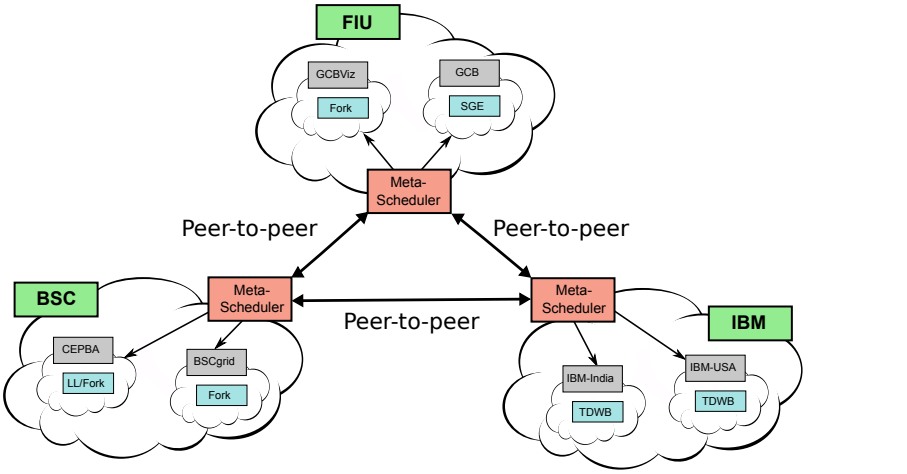


Figure 1: Cooperating meta-scheduling in LA Grid

with the trade-off of decreasing the accuracy of state information.

The accuracy of job forwarding decisions (i.e., the ability to avoid false positives when forwarding job requests among meta-schedulers) is also a strong function of the forwarding algorithm. The performance of several forwarding algorithms is investigated in combination with the resource models having different accuracy levels. These studies are based on the implementation of Figure 1 and are supplemented by simulations for very large distributed grid systems using peer-to-peer meta-schedulers. One result of the simulation is to show that resource models containing very compressed and less accurate data still provide useful hints to the meta-scheduler on where to forward jobs.

The contributions of this paper begin with the design of protocols for meta-scheduler interoperation and the peer oriented meta-scheduler topology (Section 2) and their implementation in three architecturally distinct meta-schedulers at BSC, IBM and FIU (Section 3). A key aspect of the design is the ability to scalably share static and dynamic resource information between meta-schedulers. To achieve scalability several models of resource data compression are developed that trade off data transmission volume against accuracy as covered in Section 2.2. These design choices are then evaluated and tested starting with a small scale experimental platform interconnecting the BSC, FIU, and IBM VOs to run high performance computing (HPC) jobs based on the Weather Research Forecasting (WRF) application. Section 4 demonstrates interoperability of these three sites. The small scale setup also provides operational performance data that is one input into a trace driven simulation of a large scale environment with up to 27 VOs. The simulation considers several matchmaking algorithms with increasing levels of resource data exchange and corresponding overhead. The most important conclusion from the experiments is that providing even highly aggregated resource information improves the execution times of jobs by orders of magnitude in a matchmaking system. We conclude that resource data greatly improves the accuracy of forwarding algorithms at the meta-scheduler. Poor decisions cause jobs to queue at the target site while waiting for resources. When resource availability information is known, even at a statistical level because it is aggregated, the forwarding decisions are much closer to optimal.

2. The Peer-to-Peer Model of Meta-Schedulers

The meta-scheduler is the primary contact point for grid users and other meta-schedulers. Internally, meta-schedulers may have heterogeneous implementations, but adhere to a common set of communication protocols and information models that allow them to interoperate and provide a consistent view of the interconnected grids. The interaction model adopted here is based on a peer-to-peer model where any meta-scheduler can interact with any other in a consistent way. There is no central or hierarchical structure such as a scheduling authority, global repository for resource information, or directory of meta-schedulers. Figure 1 shows such a model for the three interacting domains at BSC, FIU and IBM.

Flexibility is added to the peer-to-peer interactions through the introduction of *provider* and *consumer* roles. A meta-scheduler in the provider role offers its resources for the execution of other meta-schedulers' jobs, while a meta-scheduler with the consumer role requests other meta-schedulers' resources for the execution of its jobs. Providers advertise their sharable resources to connected consumers which in turn may propagate this information to their peers. When a job arrives to a meta-scheduler, it decides whether to run the job locally within its domain or forward it to one of its providers. These roles are properties of the meta-scheduler endpoints for a connection between meta-schedulers. A meta-scheduler can be a provider to a set of meta-schedulers and simultaneously be a consumer to another set of meta-schedulers.

Assignment of one or both of these roles to the connection between two meta-schedulers allows administrators to shape the interconnected structure of the grid. The flow of resource

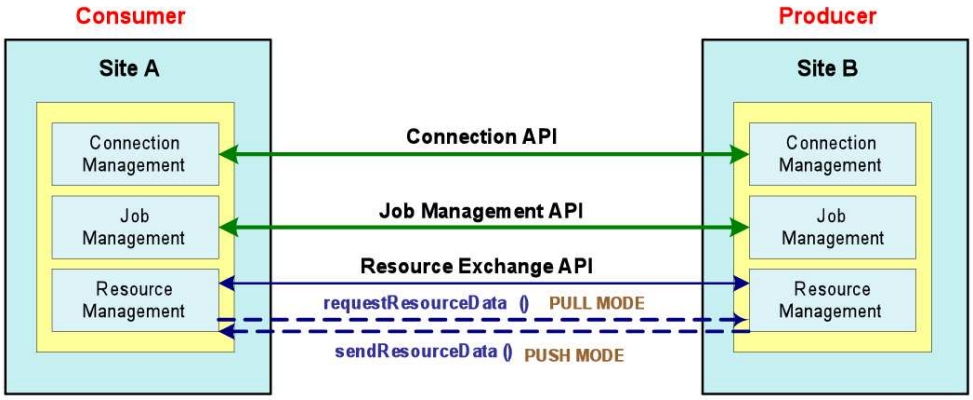


Figure 2: Cooperating meta-scheduling in LA Grid

information and forwarding of job submissions to providers is constrained by the provider and consumer roles of each connection. For example, in connecting the domains of BSC, FIU and IBM (Figure 1) each meta-scheduler is both a provider and a consumer to other meta-schedulers. Thus, resource information fully distributed between partners and jobs can be submitted or forwarded to every domain. In contrast, a centralized model is created using a 'star' topology where a central meta-scheduler is a consumer to all connected 'edge' meta-schedulers, and the edge meta-schedulers are both provider and consumer to the central meta-scheduler. Since the star has no connections between edge meta-schedulers, only central meta-scheduler will have the information of resources at every edge meta-scheduler and be able to forward jobs.

The meta-scheduler roles can be assigned and changed dynamically. For example, when a domain administrator makes available the domain's compute resources for sharing only during periods of low local activity. Here the domain's external facing meta-scheduler is a consumer most of the time, while taking on both consumer and provider roles off shift. In this example, meta-schedulers internal to the administrative domain may be providers at all times.

Multiple capabilities and operations are required on the part of the meta-scheduler to participate in the peer-to-peer model. Membership of the peer-to-peer network overlay requires each member to support the basic set of the inter-operation protocols described in the next sub-section.

2.1. Protocols and Interfaces

There are three types of operations between meta-schedulers as indicated in Figure 2 and Table 1: connection and communication, resource exchange, and job life cycle management. The connection protocol initiates membership in the grid and negotiates parameters including the authentication method and parameters, the provider/consumer roles, the heartbeat rate to monitor connection status, and the resource and job description language understood by the parties. *OpenConn()* messages are exchanged, multiple rounds if necessary, between two meta-schedulers during the negotiation until an agreement is reached, or the number of rounds exceeds a threshold specified by the initiator, in which case the connection attempt fails. Once a connection is established, *heartbeat()* messages are exchanged using the negotiated intervals and the *notifyConn()* message is used to notify partners of error conditions or gracefully terminate the connection.

Connection Messages	Resource Information Messages	Job Execution Messages
openConn()	*requestResourceData()*	*submitJob()*
notifyConn()	*sendResourceData()*	*queryJob()*
heartbeat()		*notifyJob()*
		cancelJob()

Table 1: List of messages in the LA Grid meta-scheduling protocol

Resource exchange occurs between connected meta-schedulers using either pull mode (*requestResourceData()*) by a consumer, or push mode by a provider (*sendResourceData()*). The provider triggers sendResourceData() when the dynamic changes in resource capacity, utilization, or availability are over a threshold. Resource updates may be full or incremental. The consumer typically requests full updates in pull mode, while the provider generally pushes incremental updates.

Job requests are submitted using the *submitJob()* message. Our implementation uses JSDL (Job Submission Description Language [7]), an OGF[1] proposed recommendation, as the standard format for job submission. The receiving meta-scheduler creates a local record of the submission and assigns a unique identifier to the job, which is returned to the submitter. Then it checks for a match against its resources and decides whether to schedule the job locally, or try to match the requirements against the available resources of other meta-schedulers. A job is forwarded between meta-schedulers using the same *submitJob()* interface and procedures. The job is tagged with a web service end point reference (EPR) of the forwarding meta-scheduler. This process is repeated until the job

[1] http://www.ogf.org

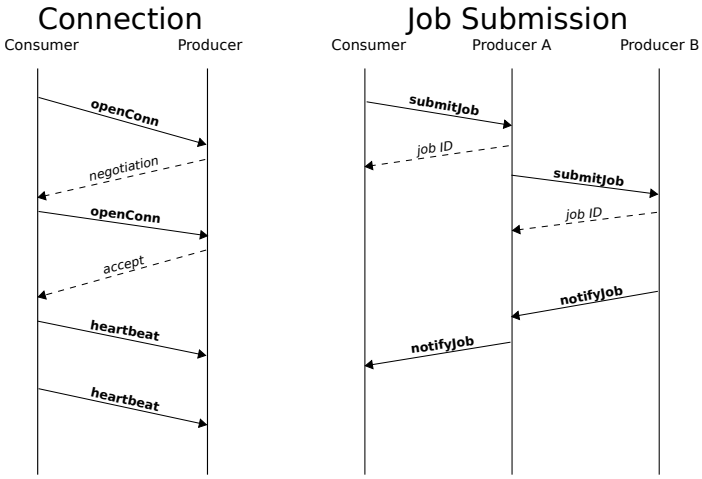


Figure 3: Meta-scheduler protocol example

reaches a meta-scheduler that executes the job using its local resources. A map of forwarding EPRs is retained at each intermediate meta-scheduler to allow job status updates to be sent back from the executing domain to the original submitter. The *notifyJob()* message is used to asynchronously inform the client of job status using this chain of forwarding meta-schedulers. The submitting client or a system administrator queries the job state or cancels the job through the synchronous *queryJob()* or asynchronous *cancelJob()* messages.

Figure 3 shows an example of two protocol operations. First, a consumer and a producer negotiate the connection parameters until they reach an agreement. After that, they start exchanging *heartbeat()* messages. The second part of the figure illustrates a job submission request with forwarding. In it, the consumer makes an initial *jobSubmit()* call to Producer A, which returns a job identifier. Then, Producer A decides to forward the call to Producer B, and after sending the request it receives a new job ID. Finally, when Producer B notifies Producer A, the later can also forward the notification message to the initial requester.

2.2. Resource Model

Efficient discovery of compute resources in a large-scale grid is challenging. In the cooperating meta-scheduler model each member maintains a local copy of resource information obtained from its peers. This allows each meta-scheduler to locate suitable resources for a job request without constantly consulting other meta-schedulers, reducing communication costs. However maintaining currency of resource information in a peer model requires more data exchanges than systems that use a central resource manager. This issue is addressed by combining a relational resource model with aggregation to reduce the size of the transmitted resource descriptions. The resulting compromise between aggregation level and accuracy of scheduling decisions is further studied in Section 4 for the bounding cases of full resource information and the most compact model presented here.

Table 2 provides a sample inventory of five computer system resource records including CPU architecture, operating system and file system type. The available CPU cycles field is normalized to the processing power of a standard system to take into account the heterogeneity of the systems. For each resource type such as *ComputerSystem*, an aggregated type is created, in this case *AggregateComputerSystem*. The attribute set of the aggregate resource type is a subset of the individual resource type plus the attributes that describe the total, mean, median, count and other statistical information. The data type of each attribute value in the aggregate resource is a string that is either a single value, a range value, or a list of values.

The process of creating an aggregated resource model using the sample inventory is illustrated in Figure 4. On the left are the computer system types differentiated by architecture (powerPC or X86) with aggregated attributes of ProcSpeed, ProcNum, and CPUUtil. In *ComputingSystemResource1* the range data type is used for the ProcSpeed attribute to show that there are 2 Power systems whose processor speed falls in the range 2.0–3.2GHz. On the right are combined resource types for the file and operating systems. The aggregate resource types are interconnected using relationships. Relationships link differing types and enhance the information content of the model allowing better matching of jobs to resources. The figure shows that *ComputingSystemResource1* "contains" *OperatingSystemResource1*, while it "uses" *FileSystemResource1*.

Clearly, this model provides considerable flexibility in selecting aggregating attributes. This allows construction of resource models with configurable accuracy levels based on the patterns of individual (detailed) resources. The greater the accuracy, the more data is transmitted. Three archetype models, *1-N*, *unique*, and *N-N*, that decrease incrementally in steps of accuracy, are presented below.

Other approaches taken to group together similar resources are described in the literature. For example, the algorithm implemented by Ganglia [8] adopts a hierarchical structure using a tree of connections among cluster nodes to federate clusters and aggregate their state. Other algorithms [9, 10, 8] focus on grouping by type and attributes, but do not consider the relationship among different types of resources. The model of Figure 4 maintains multiple resource types connected through relationships.

Because of the aggregation model's flexibility and the freedom for peer meta-schedulers to choose the details about how their resources are aggregated, it is necessary to have a schema language that describes the encoded data. The schemas are also input to the meta-scheduler allowing an administrator to define the types of models to use. The schema corresponding to Figure 4 is provided in Listing 1. It states that *ComputerSystem* resources are aggregated based on their *ProcType* values, while attributes *ProcNum*, *ProcSpeed* and *CPUUtil* are included in the aggregated resources. *OperatingSystem* resources are also aggregated based on their *OSType* values. Finally, the relationship between aggregated computer and operating systems is maintained using the unique model which is one of the archetype resource relationship models described next.

The relations between aggregated types introduced above are used to define four resource models representing a range of information consolidation, from the complete resource descrip-

ProcType	ProcSpeed	ProcNum	CPUUtil	**OSType**	FreeMem	FreeVirMem	**FSType**	SizeMB	FreeMB
PowerPC	3200	2	60	Linux	2000	1000	swap	120000	90000
PowerPC	2000	4	180	Windows	8000	4000	-	-	-
Intel x86	3000	1	50	Linux	4000	2000	-	-	-
Intel x86	2600	6	240	Windows	1000	500	-	-	-
Intel x86	3200	2	120	Windows	1000	1000	-	-	-

Table 2: Example resource inventory

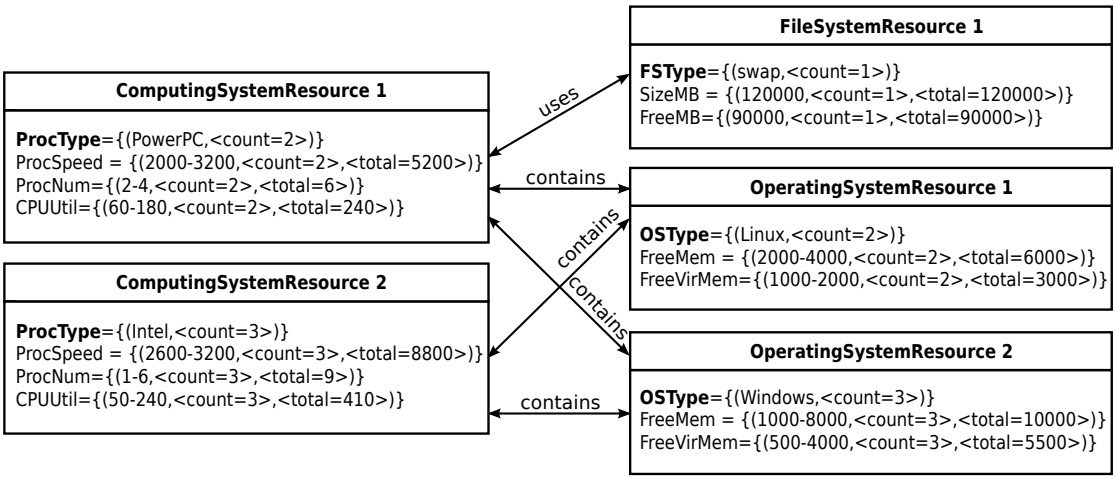


Figure 4: Aggregate resource model example

```
<root-resource>ComputerSystem
      <key-attribute>ProcType</key-attribute>
      <attribute>ProcNum</attribute>
      <attribute>ProcSpeed</attribute>
      <attribute>CPUUtil</attribute>
</root-resource>
<resource>OperatingSystem
      <key-attribute>OSType</key-attribute>
      <attribute>version</attribute>
      <attribute>FreeMem</attribute>
      <attribute>FreeVirMem</attribute>
</resource>
<relationship-model>
      unique
</relationship-model>
```

Listing 1: Example of resources

tion of the flat model to the compact N-N model, as illustrated in Figure 5 (as in the previous figure the data corresponds to that in Table 2).

- **Flat**: The flat model faithfully reproduces the system information, such as in Table 2. In that example, aggregated resource types are not used and there are 5 compute resources, 5 OS resources, and 1 file system interconnected by *contains* and *uses* relations.

- **N-to-N model**: This model represents the most compression and least accuracy. It aggregates resources of the same type, then retains multiple relations to other aggregated types, which may sometimes result in ambiguity. Notably, the parent of the contains relation does not distinguish by the type of each child of the relation. This model is useful as a limit case on what accuracy is needed for matching job requests to resources. Section 4 shows the model works surprisingly well in large systems.

- **Unique model**: This model maintains information in parent aggregated resource types about all unique relationships with children. It is created by first identifying unique relationships based on some standards. The example of Figure 5 shows unique relationship between *ComputerSystem* and *OperatingSystem* resources based on the former's architecture and the latter's OS types. Then, an aggregation is performed on the resources of each unique relationship.

- **1-to-N model**: This model aggregates resources in two stages. The first stage aggregates resources at the top (parents) of the relationship map. The example of the figure aggregates *ComputerSystem* first. Then, the second stage aggregates additional resources that have relationships with the same aggregate resource from first stage. The example of Figure 5 aggregates *OperatingSystem* resources that are contained in the same aggregated *ComputerSystem*. This maintains the "contains" relationship as 1-to-N such that one aggregate *ComputerSystem* contains multiple aggregated *OperatingSystem* while one *OperatingSystem* is contained at most by one aggregated *ComputerSystem*.

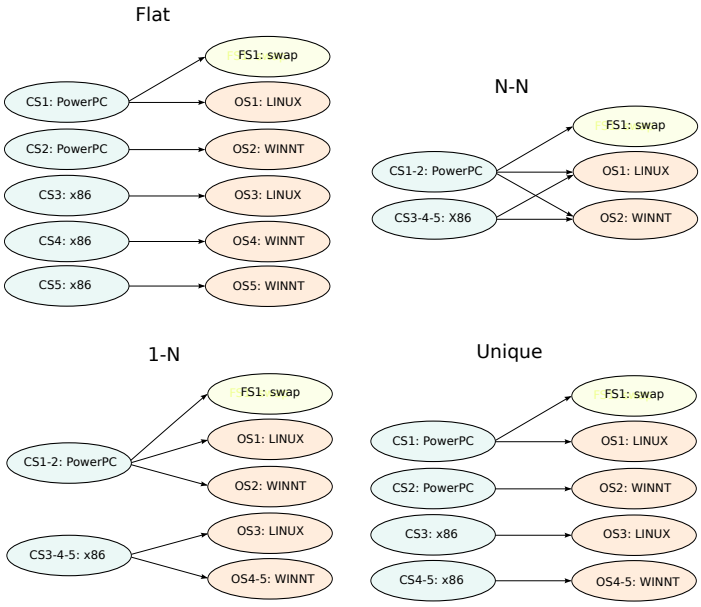


Figure 5: Examples for the resource aggregation models

These models keep different levels of relationship details and impose different complexity to aggregation algorithms, which results in different information density. The efficacy of the most condensed N-N model in job to resource matching is studied and compared to the full resource model (flat) in Section 4.

2.3. Job Broker and Scheduling Criteria

The function of a meta-scheduler is to optimally forward job submissions to a Local Resource Management System (LRMS) or connected meta-scheduler in the provider role. A job's requirements for resources and execution environments are expressed by a job submission description document (i.e., JSDL). In our meta-scheduler design, resource matching for a job is based on three considerations: capacity, capability, and utilization.

The resource models of meta-schedulers are described in the schema of Section 2.2. The capacity of a meta-scheduler corresponds to the aggregated capacities of the attached LRMSs plus peer meta-schedulers. Capabilities describe what LRMSs and meta-schedulers can do: a LRMS capability might include the ability to schedule parallel jobs, or make advance resource reservations. Capabilities at the meta-scheduler reflect high-level scheduling considerations, domain access, and membership in virtual organizations. Meta-schedulers in a corporate data center can have attributes stating that it will not accept forwarded jobs during times when important local work must complete. Utilization of resources is considered by a meta-scheduler when making scheduling decisions. Each meta-scheduler monitors the utilization of its associated LRMSs, as well as those reported by its peer meta-schedulers providing computing power. It is a challenge to measure utilization at the required granularity to make good scheduling decisions. For example, reporting the most recent short-term average CPU utilization of each of thousands of nodes in a cluster may not give the most appropriate information to a scheduler. Less granular methods of utilization reporting are often based on the notion of workload classes. A *class* defines the requirements of a job, such as small, medium, and large; or interactive and batch jobs. For example, a provider of computer resources can report how many jobs of each class it presently supports, and the occupancy of each class. Additional utilization information can be provided by giving the average waiting time or queue length for each class.

Since we consider aggregated resource information, data involved in the scheduling decision may be less accurate when certain aggregation schemas are applied (e.g., the N–N model). Consequently, to perform the matchmaking between job requests and resources from the aggregated data, we take statistical information such as maximum and minimum values contained in the resources for the requirements and a combination of average values for refining the selection. Furthermore, since the resource matching is performed at the broker level, the information loss can result in non-optimal broker selection decisions. For example, the algorithm may select a broker with insufficient resources when another broker is able to dispatch the job immediately. Therefore, the level of aggregation of the resource information is crucial.

2.4. Security and Data Transfer Mechanisms

Security protocols and processes vary considerably across virtual organizations and administrative domains. This variability is manifested in the operations of data staging and job dispatching. Data staging is typically an inter-domain operation in which either input data is transferred to the execution domain or job results are sent to a target destination. Typical protocols for data staging are SCP, FTP, or GridFTP. The job execution domain requires security credentials of the data domains in order to initiate file transfers.

Job dispatching is performed within a domain by a local scheduler and resource manager whose authentication scheme may require yet another security protocol and set of credentials. The diversity of security mechanisms include SSH private/public keys, x509 certificates for Grid Security Infrastructure (GSI) or simple user id/password systems for some schedulers.

Rather than requiring interoperating meta-schedulers to support APIs for exchanging all popular security protocols for data staging and job execution, in our design the meta-scheduler supports a limited number of common security options. One of these is selected by negotiation at connection time and is active for the meta-scheduler to meta-scheduler session. This approach allows each meta-scheduler to implement the most appropriate interfaces for a given virtual organization, translating external models to local ones when necessary. An internal mapping of submitted user/job credentials from the negotiated security mechanism to the possible different one required for data staging or job execution is maintained by the meta-scheduler. For example, a user submits a job to meta-scheduler 'A' using an SSH public/private key credential. If that meta-scheduler decides to forward a job to meta-scheduler 'B' that is using an x509 certificate it must maintain an internal repository that maps the SSH credentials to the user's x509 certificate

which is then sent to meta-scheduler 'B'. This exchange may be achieved with the MyProxy credential repository [11], which is populated *out of band* with the security mapping.

An important aspect of security is how to access supercomputing facilities, which usually have very restrictive access mechanisms. At IBM, a firewall has been enabled to hide both meta-scheduling services and supercomputing resources. To access LA Grid meta-scheduling services secure web service calls are required, with a public-private key security mechanism used for authentication. At BSC the eNANOS Broker acts as a gateway to BSC resources. To access the Marenostrum supercomputing facility, a secure channel is established. For the rest of resources, it uses GSI as security mechanism. Actually, the eNANOS Broker translates regular GSI calls to Marenostrum when required. At FIU, GridWay is configured with GSI security mechanisms.

3. Meta-Scheduler Implementations

The architecture of the meta-scheduling communication protocol allows for different sites to coexist while maintaining site-specific policies, internal security mechanisms or implementation details hidden from other peers. As long as all meta-schedulers implement the required interfaces, the proposed protocol allows them to communicate and share workloads transparently.

As a proof of concept, we have enabled three sites under different administrative domains to communicate using our meta-scheduling protocol. Each of these sites has its own implementation in terms of the underlying resource management systems or security and scheduling policies, but all of them implement a common protocol for communication of resource and job information. The site at BSC extends an in-house developed meta-scheduler, eNANOS, which uses the Globus Toolkit (GT) internally; IBM implements the meta-scheduling protocol on top of one of their commercial products, IBM Tivoli Dynamic Workload Broker; FIU developed an extension to the GridWay open source meta-scheduler to enable interoperability.

Some of the internal details of these three implementations, as well as the steps taken to enable them to join the other sites, are discussed next.

3.1. The BSC Meta-Scheduler

At BSC, the LA Grid meta-scheduler functions are implemented using the eNANOS framework [12][13], which is based on GT4 services (i.e., every component is a service). The implementation includes several extensions to the eNANOS broker and a new dedicated scheduling policy plug-in, which is also a GT4 service. The extended architecture of eNANOS is shown in Figure 6 with the LA Grid extensions in dark shading.

Since other LA Grid meta-schedulers implement the interoperation protocols using regular web services, eNANOS extensions are developed as a set of Axis2 [2] services to avoid incompatibility problems with GT4 services. Some of the compatibility problems include the SOAP message formats and data types. The Axis2 services implemented on the server side act as a wrapper to support redirecting calls to GT4 and perform data transformations when necessary. To support interactions between the eNANOS and other LA Grid meta-schedulers, we implement a set of regular web services for the APIs as the client interface.

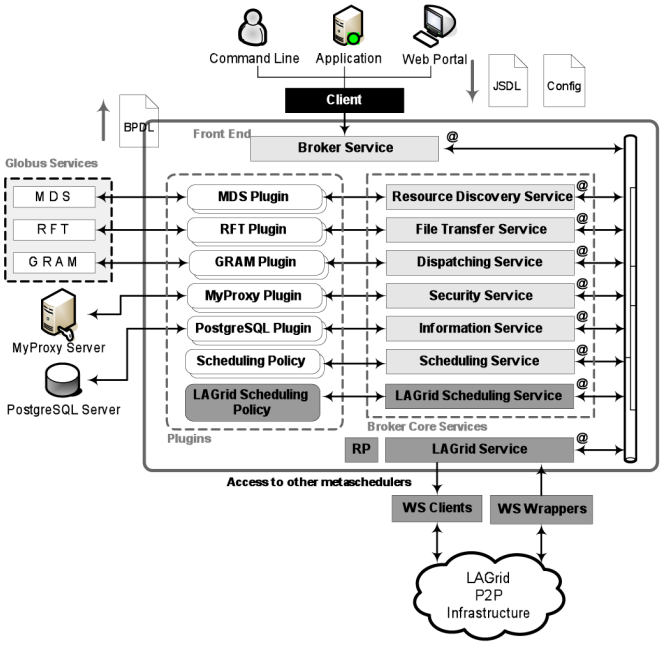


Figure 6: Architecture of eNANOS with the LA Grid specific components

Leveraging the default persistence mechanism of GT4, the data that is relevant to LA Grid functions, such as meta-scheduler connections and resource information from other meta-schedulers, is stored using GT4 *Resource Properties*.

We implement a new set of resource management functions to support resource information exchanged between partnering meta-schedulers. After obtaining the resource information within the domain, we create the aggregated form of resource information using the main attributes of resources and clustering the data by CPU and OS type, as shown in the resource example of section 2.2.

In addition to receiving jobs routed from other LA Grid meta-schedulers through the Job Management API, the eNANOS broker can receive job submissions and other requests from regular users through the eNANOS clients that may be a command-line or a Java API (which can be used, for example, by a web portal or an external application), as shown in the top part of Figure 6. We modify the job submission interface to support the LA Grid parameters (e.g., the connection ID or the notification End-Point Reference). To allow the eNANOS services manage the LA Grid forwarded jobs, we adapt the job schema used in eNANOS. In particular, we add a new element that contains a set of LA Grid information, and a new job status (FORWARDED) for the jobs that have been forwarded to or from another meta-scheduler.

Since we observe that significant modifications in the

[2] http://axis.apache.org/axis2/java/core/

scheduling service are required to implement the scheduling policies based on aggregated resource information and to allow job forwarding, we implement a separate scheduling service for LA Grid. Moreover, we implement a scheduling policy for LA Grid using a new scheduling policy plug-in. Since both services and plug-ins are implemented as modules with abstract interfaces, the broker is updated by defining those new services and updating the configuration file.

With respect to the notification functionality, we modify the monitoring of eNANOS to notify other meta-schedulers when the status of a forwarded job changes. This is done as an extension of the current monitoring module using the LA Grid information incorporated in the job schema.

3.2. The IBM Meta-Scheduler

The IBM Research meta-scheduler is implemented by extending an IBM scheduling product: IBM Tivoli Dynamic Workload Broker (ITDWB) [14].

An ITDWB server collects resource information from agents on resources, accepts job requests from users and is able to match jobs to resources that have dynamic availability and utilization. The combined ITDWB architecture and meta-scheduler extensions are shown in Figure 7. The base product components [15] (shown as boxes on the right hand side of Figure 7) are a job dispatcher, a resource advisor and a repository to persist job and resource information. Other components include a set of agents (shown as boxes at the bottom of the Figure 7) to collect resource information and to execute jobs on computing platforms.

Our extension comprises a set of changes to the interface and functionality. New web service components (shown as boxes on the left hand of Figure 7) are added to realize the interoperability with other meta-schedulers as described in previous sections. Thus, a broker server retrieves information from and dispatches jobs not only to its agents, but also to other meta-schedulers.

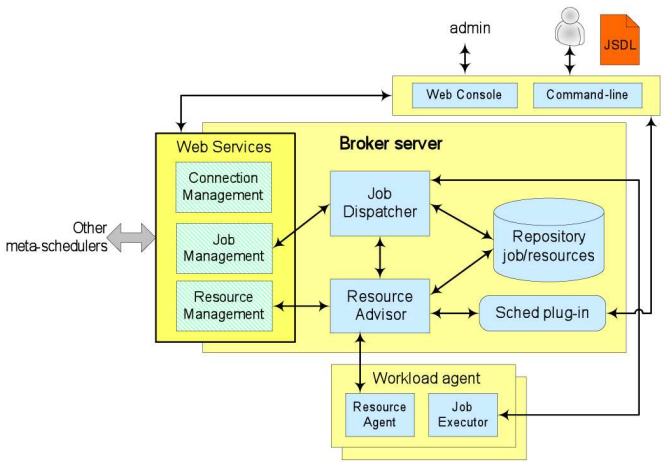


Figure 7: Extended ITDWB for meta-scheduling functions

Since jobs can be dispatched to other meta-schedulers, we need to extend the *Job Dispatcher* component, which manages the lifecycle of jobs including storing and updating jobs' information, dispatching them to *Job Executors*, and communicating with submitters. First, ITDWB uses a dialect of JSDL to describe jobs. However, OGF's JDSL[3] is a more accepted standard. Thus, we implement, using Axis2, a module that can convert formats between different JSDL formats that are represented in XML files. With the ongoing evolution of JSDL, vendor specific extensions and dialects, we believe that such a converter would be a necessity. Our converter is implemented in Java for its potential flexibility, though we debate different approaches such as using XSLT, with which transformation can be dynamically manipulated without recompilation.

Second, the *Job Dispatcher* is enhanced to process jobs routed from and to other resource domains. For jobs routed to other schedulers, job-forwarding information is stored in the repository. Thus, the meta-scheduler receiving a job query knows how to forward it and how to send back the status report and execution result in the reverse direction.

The *Resource Advisor* is also enhanced to process resource information supplied by remote meta-schedulers with whom connections are established. Note that in the extended version we not only have more information to deal with, but the information can have different formats and different security levels. We extend the resource model of ITDWB and include aggregated format of resources. For example, the *Resource Advisor* retrieves individual resource information from a database and aggregates it as part of the response to *requestResourceData()* calls.

Scheduling algorithms, which choose a set of resources that satisfies a job request, are implemented as plug-ins of *Resource Advisor*. We implement new algorithms that take resource location into consideration and that can match the resource requirement of jobs with aggregate information.

3.3. The FIU Meta-Scheduler

The implementation of FIU's meta-scheduler leverages an existing open source solution, the GridWay meta-scheduling platform [4]. GridWay is a community project with an open source license. We choose it for its use of open standards and its modular nature. GridWay is implemented as a group of managers that communicate using standard I/O. It offers facilities for job management, data transfers and resource information.

GridWay is a Globus incubator project, and therefore, it is supported by the Globus Consortium. This means that all advances in GridWay will be in line with the Globus project. It provides one of the first implementations of the DRMAA (Distributed Resource Management Application) API [4] for job submission and status querying.

The FIU meta-scheduler is built as a set of modules that deal with different aspects such as peer communication, site scheduling, and resource management. The details related to local resource management are delegated to GridWay, while the high-level scheduling decisions are made by our wrapper.

[3] For example, JSDL includes a schema describing an application that can be executed on a POSIX compliant system.

[4] http://www.ogf.org/documents/GFD.22.pdf

The core modules and their interaction is shown in Figure 8. Here is a brief explanation of each module's responsibilities:

- ***User Interface***: The user interface module is in charge of receiving external users' requests such as job submission or resource information. Users can interact with the meta-scheduler using a command-line interface and job submission is done through OGF's JSDL files.
- ***Site Scheduling Manager***: This module wraps GridWay functionality for intra-domain scheduling. The rest of the modules interact with it using DRMAA.
- ***Global Scheduling Manager***: This module deals with global scheduling policies. It is in charge of deciding whether a job will be scheduled and processed on the local domain or it will be forwarded to another domain. Additionally, this module keeps track of the jobs' status submitted from other domains.
- ***Resource Manager***: It stores information about the resources in local domain and the remote domains, which have exchanged their resource information through an open connection. It also pushes resource data to the connected meta-scheduler in case of changes in local resource availability.
- ***Web Service Communication***: We use Apache Axis2 as the container for the different web services used to communicate with other domains. Axis2 provides the points of entry for SOAP requests and stubs to initiate conversations with other meta-schedulers.

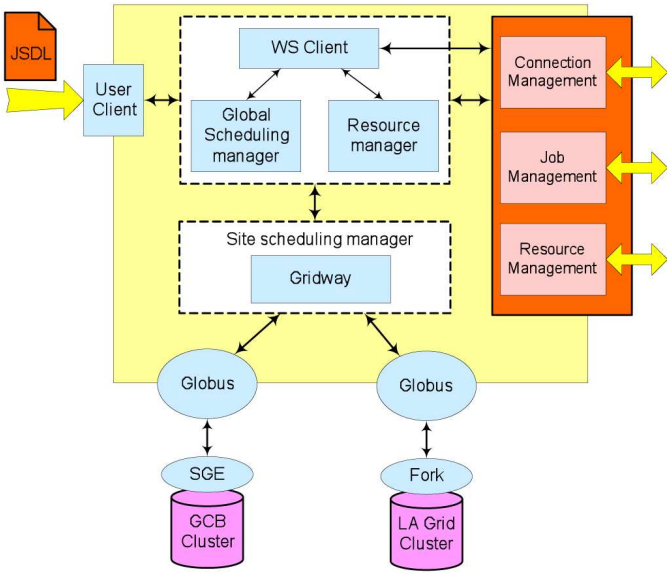


Figure 8: Architecture of FIU meta-scheduler

4. Experimental Evaluation

We perform three sets of experiments to validate our meta-scheduler implementations. In the first set, we measure the timings for different operations of the protocols for three sites with their own implementations. In the second set, we run a larger-scale scientific software simulating a realistic scenario in the domain of weather forecasting. In the final set, we perform a scalability study of global scheduling policies (i.e., job forwarding strategies) in different scenarios.

4.1. Protocol Measurements

We test the implemented APIs for the IBM, FIU and BSC versions of the meta-scheduler. The main operations supported by the protocol are tested to perform a functional validation of the meta-schedulers. In this experiment, we use a driver program that generates requests for each of the tested operations, and measure the timing.

For the resource information exchange protocol, each meta-scheduler aggregates 100 resources and sends them back when it receives the *requestResourceData()* call from the driver program. The driver program then sends the same resource information using the *sendResourceData()* call to meta-schedulers. While we have specified the number of resources used for the tests, the type of resources and the aggregation algorithms used vary in different meta-schedulers.

For the job execution protocol, we send a probe job defined by a JSDL document that runs the UNIX sleep command for 10 seconds. This allows us to measure the protocol itself, without considering the actual job load.

4.1.1. BSC protocol measurements

We install an instance of eNANOS on a machine with dual Intel(R) Pentium(R) 4 3.60GHz with 1024 KB of cache in each core and 1 GB of main memory. The BSC column in Table 3 shows the results obtained from the tests using the driver program.

Compared to two other meta-schedulers, BSC has longer delays in most of the operations. The critical factor is the additional delay produced by the WS wrapper between other meta-schedulers and the eNANOS LA Grid service. Eventually, we will remove the wrapping layer by implementing the LA Grid APIs directly within the eNANOS LA Grid GT4 service.

For the *requestResourceData()* operation, the delay time includes having resource information retrieved from the *Resource Properties* without actual resource discovery. There is a background eNANOS service responsible for resource discovery and populating the resource properties at a configurable interval. The retrieved resource information is then transformed into an aggregated form and packaged as part of the returned SOAP message back to the caller. The *sendResourceData()* operation involves depositing the resource information in the aggregated form to the Resource Properties.

4.1.2. IBM protocol measurements

We install an instance of the IBM meta-scheduler on a machine that has dual AMD Opteron processors of 2.6GHz, 1024 KB cache and 2GB core memory. A DB2 database, as the resource repository, also runs on the same machine. The IBM column in Table 3 shows the data collected on interactions between the driver program and IBM meta-scheduler. We

Operation	Delay Time (milliseconds)			
	FIU → BSC	BSC → FIU	FIU → IBM	IBM → FIU
openConn()	562	659	15	40
requestResourceData()	983	706	69	90
submitJob()	642	694	124	3162

Table 3: Delay across meta-scheduling sites

note that the operations *submitJob()*, *requestResourceData()* and *sendResourceData()* involve operations on multiple tables in the database such as for storing job information, and retrieving and storing resource data. The data shows that the *sendResourceData()* operation consistently takes longer than *requestResourceData()*. We suspect the delays are caused by the database update overhead, as we store resource data received by the *sendResourceData()* operation. To verify this, we clean the database table by removing old resource entries, and observe that the subsequent run shows reduction in timing by 50% for *sendResourceData()* calls while timings for *openConnect()* and others remained similar.

4.1.3. FIU protocol measurements

An instance of FIU meta-scheduler is installed on a machine with dual AMD Opteron processors of 2.6GHz, 1024 KB cache and 2GB core memory, same type of machine as IBM meta-scheduler installation. The same machine is also installed with GridWay and GT4 services. The FIU column in Table 3 shows FIU's corresponding experimental results.

For the FIU meta-scheduler implementation, the connection and job information are stored in memory without database access (as opposed to IBM meta-scheduler) or invoking other services (as is the case in the BSC meta-scheduler). Thus, the measured delays for FIU are shorter than those of the BSC and IBM meta-schedulers. The information for FIU meta-scheduler resources is stored in a file. For the *requestResourceData()* operation, resource information is read from the file and then aggregated. For the *sendRequestData()* operation, the FIU meta-scheduler keeps the aggregated data from other meta-schedulers in memory without file operations. For *submitJob()*, the FIU meta-scheduler routes the job to GridWay and obtains a job ID before returning to the caller.

4.2. Weather Research Scenario

Interoperability among the three meta-scheduler implementations is verified by running the Weather Research and Forecasting (WRF) model[5] [16]. WRF is developed by the National Center for Atmospheric Research and several other research institutes as a tool for meteorologists to do regional forecasting. The WRF model is an MPI application that follows the SPMD (Single Program, Multiple Data) paradigm. Blind scaling of WRF across the grid is generally counter-productive and in most cases degrades the total running time of the model [17]. This is due to a design which assumes a low latency underlying network and uses intensive communication among working processes.

Different approaches to scaling out WRF have been discussed in [18], of which meta-scheduling is a paramount component in the process. On this experiment, we show the advantages of running jobs through the meta-scheduler compared to running them directly on a local cluster. The meta-scheduler can extend available resources by connecting to other peers and delegating jobs to them based on different policies. Users don't need to do any additional work, since the interoperation protocol takes care of contacting other sites, keeping track of remote resources and forwarding the actual jobs.

We compare three cases to assess the run-time improvements of delegating jobs, and measure the overhead of the meta-scheduler layer. We performed 4 different runs of a small forecasting region for different number of processors.

All experiments include three execution sites with different resources. The local site consists of 8 Pentium 4 nodes at 3 GHz with 1 Gb of RAM located at FIU in Miami. There is one remote cluster at IBM TJ Watson center in New York, which has 4 IBM JS22 blades with two 4GHz dual core Power 6 processors and 16 Gb of memory. Finally, we also use the Marenostrum supercomputer at BSC, which is composed of 10,240 IBM Power PC 970MP processors at 2.3 GHz (2560 JS21 blades) with 20TB of main memory.

First, we measure the running time of the different WRF instances at the local site without using a meta-scheduler. The user executes WRF through mpirun (or equivalent command) to spawn parallel tasks in the cluster.

Then, the same jobs are executed, but this time through a meta-scheduler installed at the local site. Instead of running the job by logging in the node and issuing the mpirun command, now the user constructs a JSDL definition file with the command, and specifies how many nodes he/she wants to allocate for the job. We use the SPMD extension of JSDL [7] to describe the parallel environment to be used (MPI in this case) and the number of nodes to use. The meta-scheduler is configured to use the local cluster to run the job, obtaining similar results as in the first case.

Finally, the four WRF instances are sent to the same meta-scheduler at the local site, but in this case we change the scheduling policy to forward jobs to another peer. The user submits the job using the same method of submission and job description file as in the previous case, but this time the meta-scheduler contacts the remote instance running at IBM and forwards the jobs there. Then, the same steps are repeated with

[5] http://www.wrf-model.org/

# nodes	FIU	IBM	BSC	
	Run Time	Run Time	Run Time	AVG Queuing Time
1	15,355 s	N/A	N/A	N/A
2	8,595 s	2,271 s	3,640 s	47 s
4	5,059 s	1,280 s	2,571 s	45 s
8	2,942 s	775 s	1,285 s	41 s
16	N/A	N/A	786 s	85 s
32	N/A	N/A	390 s	101 s
64	N/A	N/A	269 s	144 s

Table 4: Execution time of WRF in different sites

Sites	Overheads	# nodes						
		1	2	4	8	16	32	64
FIU	Run Time	100%	100%	100%	100%	N/A	N/A	N/A
	Data Transfer	0%	0%	0%	0%			
	Protocol	0%	0%	0%	0%			
FIU ⟶ IBM	Run Time	99.51%	99.02%	98.43%	97.42%	N/A	N/A	N/A
	Data Transfer	0.46%	0.93%	1.49%	2.45%			
	Protocol	0.03%	0.05%	0.08%	0.13%			
FIU ⟶ BSC	Run Time	N/A	97.82%	96.96%	94.17%	91.39%	85.68%	83.43%
	Data Transfer		2.12%	2.96%	5.68%	8.39%	13.96%	16.16%
	Protocol		0.05%	0.07%	0.15%	0.22%	0.36%	0.41%

Table 5: Relative WRF run time and overheads on FIU and forwarding from FIU to IBM/BSC

BSC.

Table 4 shows the run-times for the execution of WRF at the different sites of our infrastructure and the average queuing time at BSC. The queuing time at BSC depends on many parameters such as the user, the system status or the queue used. However, we provide the average of the queuing times obtained for WRF with a specific user and queue. We believe that they are sufficient illustrative, as a matter of example.

Table 5 shows the relative overheads due to the experiment's execution, the meta-scheduler protocol, and the data transfer for three cases: when the workload is executed locally, when it is forwarded to IBM and when the same is done to BSC. As it can be noticed, the processing and network overheads produced by the protocol when using the forwarding policy –an average of 1.74 seconds– are negligible for a long running scientific job such as this one. Data transfer costs are relatively low for the experiments between FIU and IBM, specially due to the fast connections between these two sites, and they increase for the experiment between FIU and BSC due to a slower connection. The relative overhead of data transfer is specially notable when more nodes are used and therefore computation run-time is lower.

As already pointed out, the heterogeneous nature of our federated infrastructure results in different overheads depending on the meta-scheduler technology used. Also, the different infrastructures behind the meta-schedulers result in different execution times for a given job request. Furthermore, the different federated sites may use a local resource management system (queuing system) that can potentially add an additional overhead to execute a job request. This is the case of BSC, which uses SLURM [19] to manage Marenostrum supercomputer.

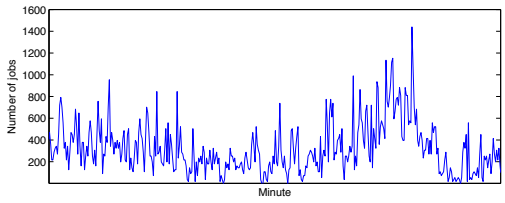
4.3. Scalability Study

In order to evaluate the scalability of the proposed approach for a large number of grid domains and job requests under different policies, we run simulation tools along with archived HPC workload traces from production systems. We use the Alvio simulator [20], a C++ event driven simulator that was designed to study scheduling policies in different scenarios, from HPC clusters to multi-grid systems. The simulator is extended to model the grid interoperability components discussed in this paper. They include, for example, P2P communication and aggregated resource information. Under each grid domain, a meta-scheduling policy is responsible for managing the jobs submitted to that domain. Furthermore, each site contains a set of machines that model typical HPC local resources.

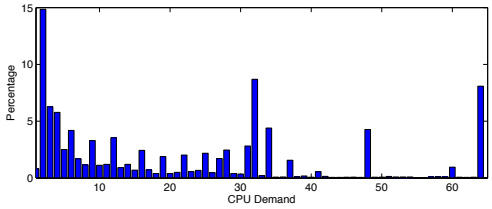
4.3.1. Workloads

In the present work, we use traces from the Grid Observatory[6], which collects, publishes, and analyzes data on the behavior of the EGEE[7] grid. Since it constitutes one of the most complex public grid traces, it allows us to study the scalability of our approach. The frequency of request arrivals is much

[6] http://www.grid-observatory.org
[7] http://www.eu-egee.org



(a) Job requests per minute



(b) CPU demand distribution (percentage)

Figure 9: Characteristics of the input trace

higher than other large grids such as Grid5000[8]. Figure 9 shows the job arrivals and CPU demand distribution of the trace, which consists in two weeks of job submissions. It has a high arrival rate of parallel jobs ranging from 1 to 64 CPUs.

Since jobs in Alvio are specified using Feitelson's Standard Workload Format (SWF)[9] and the traces are in different formats and include data that is not used, they are pre-processed before being entered in the simulation framework. First, we convert the input traces to SWF. We also combine the multiple files of which they are composed into a single file. Then, we clean the trace in SWF format in order to eliminate failed jobs, canceled jobs and incomplete records. Finally, we generate the additional requirements that are not included in the traces. For instance, traces from Grid Observatory do not include required memory, and the mapping between gLite (i.e., the EGEE Grid middleware) and the Local Resource Management Systems (LRMS) that contains such information is not available. Thus, we include memory requirements following the distribution of a trace from Los Alamos National Lab (LANL) CM-5 log, and we include the disk demand using a combination of the job duration with the CPU and memory usage, with a randomized factor. For the remaining attributes we use percentages for each CPU architecture and OS type, applying a random distribution by bursts (sized from 3 to 6). Although the memory requirements are obtained from a dated trace, we scale them to the characteristics of our system model. We estimate that similar results can be obtained using a resource requirement distribution from other traces or recent models.

4.3.2. Scenarios

In our experiments, we define three different domain types, FIU, IBM and BSC, that represent an approximation of the systems introduced previously. FIU and IBM domain types are composed of a similar number of resources (with 15 clusters and around 1,000 CPUs available each) while BSC is a bit more than two times larger. However, IBM resources are faster than the other systems' resources. In fact, we model the run-time and queuing times of FIU, IBM and BSC systems based on the data shown in Table 4. We also model the delay due to job forwarding among meta-schedulers in the P2P network based on the disk demand. We assume that the data associated to a job request is transmitted to another domain over the network when the job request is forwarded to another meta-scheduler.

In the experiments with multiple interoperating VOs we model the same number of each FIU, IBM and BSC systems (e.g., 9 VOs, which include 3 FIU systems, 3 IBM systems and 3 BSC systems) and the same relative number of jobs per VO (e.g., the number of jobs for 9 VOs is 3 times larger than the number of jobs for 3 VOs). In our experiments we simulate the interoperation of up to 27 domains. Thereby, when we evaluate the full interoperable system with up to 27 domains, we are considering more than a million and a half jobs, around 200 clusters, and more than 20,000 available processors.

4.3.3. Metrics

We use the following metrics for evaluating our strategies:

- Makespan (or workload execution time, which is the difference between the earliest time of submission of any of the workload tasks, and the latest time of completion of any of its tasks)

- Average bounded slowdown. We define bounded slowdown (BSLD) for a given job:

$$BSLD_{job} = max\left(1, \frac{runtime_{job} + waittime_{job}}{max(runtime_{job},\ threshold)}\right),$$

  $threshold = 60\ \ seconds$
  Although a threshold of 10 seconds is used in many works in the context of parallel job scheduling to limit the influence of very short jobs on the average bounded slowdown, we define a threshold of 60 seconds, because in grid scenarios jobs typically take longer. We define the average bounded slowdown, given the set of finished jobs:

$$AVG\ BSLD = \frac{\sum_{i=1}^{finished\_jobs} BSLD(job_i)}{\#finished\_jobs}$$

- Average utilization (CPU utilization)

- Average job forwarding between VOs, which is the ratio between the total number of job forwarded and the total number of job requests

4.3.4. Policies

We evaluate two different strategies for job submission in interoperable grid systems that are widely used in literature: "Round Robin" (which distributes equally the job requests among meta-schedulers) and "Random" (which distributes the

[8] http://www.grid5000.fr
[9] http://www.cs.huji.ac.il/labs/parallel/workload

job requests among meta-schedulers randomly). We evaluate two different types of job forwarding policies. In the first case, we implement the "bestBrokerRank" job forwarding policy ("Inter" in the figures), which is based on matchmaking and was proposed and evaluated in [21]. In particular, given a set of job requirements and the resource information from different meta-schedulers, it returns the meta-scheduler that matches these requirements optimally. It considers the accumulated rank value of a regular matchmaking algorithm [22] on the resources of each meta-scheduler domain. The main difference between "bestBrokerRank" and existing matchmaking approaches, such as the one used in Condor, is that we select a meta-scheduler based on statistical information rather than selecting a the resource that matches best the requirements to run a job request. In the second case, we consider a variant of the "bestBrokerRank" policy that manages the resource information in aggregated form ("Aggr" in the figures).

In addition to the forwarding policies, we also propose two policies that are not based on matchmaking and we incorporate an additional consideration to the "bestBrokerRank" policy, which is a factor (α) for promoting the job originator domain. For example, a factor α=1.25 promotes 25% the job originator domain by increasing 25% its rank. Specifically, we consider the following job forwarding policies:

- Inter: "bestBrokerRank" policy (with flat resource model). Although the flat resource model is not scalable (the size of the resources grows linearly as shown in [21]), we use this policy as reference. However, in our simulations we do not consider the overhead due to transferring and processing time of resource information
- Inter_RR: if a meta-scheduler does not have enough resources to run a job, the job is forwarded to another meta-scheduler following a "Round Robin" policy
- Inter_RM: if a meta-scheduler does not have enough resources to run a job, the job is forwarded to another meta-scheduler randomly
- Aggr_1.25: "bestBrokerRank" with aggregated resource information and α=1.25 (the job originator domain is promoted 25%)
- Aggr_1.10: "bestBrokerRank" with aggregated resource information and α=1.10 (the job originator domain is promoted 10%)
- Aggr_1.0: "bestBrokerRank" with aggregated resource information and α=1.00 (all brokers have the same priority)

Both Inter_RR and Inter_RM policies use a time to live (TTL) parameter to avoid starvation. If a job has not been scheduled after being forwarded the specified number of times, it is queued in the job originator domain to be re-scheduled. The Aggr policies use the N-N resource aggregation model since it is scalable in terms of resource information size and its aggregation processing time is acceptable for multiple grids while providing sufficient accuracy to implement job forwarding policies effectively as shown in [21].

4.3.5. System characteristics and job requirements evaluation

In this subsection, we study the different systems described previously (FIU, IBM, BSC) and their interoperation (Inter). To do this we run the same workload trace in each experiment. Figure 10 shows the obtained results for each of these experiments. Although in Section 4.2 we do not consider resource requirements, we evaluate the performance impact when they are included in job requests. In Figure 10, the *Reqs* series consider job requirements and the *NoReqs* series do not consider job requirements.

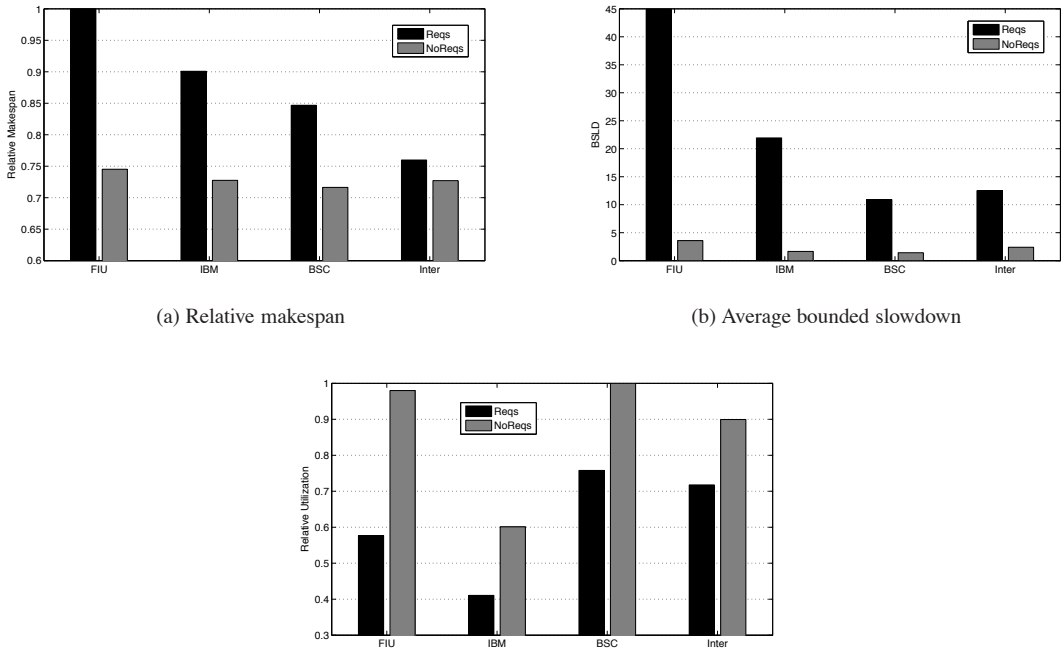
Figure 10a shows that the makespan is similar for all systems when there are no resource requirements; however, with resource requirements the makespan obtained for FIU, IBM and BSC systems is increased up to 15%. The results are consistent with the empirical execution of WRF on the different systems shown in Table 4. Since the Inter scenario has higher number of resources of each type, the obtained makespan is significantly shorter with resource requirements (almost 25% shorter than FIU); however, it is very similar to the makespan obtained for the other systems without resource requirements. In fact, the makespan for Inter is higher than the makespan for IBM because Inter may run jobs on slower resources (e.g., from the FIU system) in order to reduce the waiting time. Figure 10b shows that the average BSLD is several times lower without resource requirements because jobs have to wait for available resources with specific characteristics. Since FIU resources are slower than the other systems' resources, jobs' execution take longer, which result in longer queue waiting times and therefore the BSLD is especially high for FIU. Figure 10c shows that the average resource utilization is around 25% lower on average with resource requirements. Also it shows that the average resource utilization is especially lower for IBM with resource requirements due to faster resources result in resources being idle for longer time.

The obtained results state that the interoperation between VOs can perform better than the different systems individually if resource requirements are considered along with the job requests. Therefore, in order to study the scalability of our approach we consider resource requirements in the evaluation of job submission and job forwarding policies.

4.3.6. Job submission strategies evaluation

In this subsection we evaluate both "bestBrokerRank" and its variant that manages the resource information in aggregated form with 3, 9 and 27 VOs.

Figure 11a shows that the makespan with the Aggr policy is around 3.5% longer on average than the makespan with the Inter policy. It also shows that the makespan is similar with both Random and RoundRobin policies (<1% on average). Figure 11b shows that the average BSLD is significantly lower with the Inter policies with respect to the Aggr policy. Figure 11c shows that the average utilization is similar with both job submission policies (<2% on average). Figure 11d shows that the average job forwarding is higher with the Aggr policy due to accuracy of the resource information is lower with respect to the Inter policy; however, both Random and RoundRobin policies present similar results (<1% on average).



(a) Relative makespan

(b) Average bounded slowdown

(c) Relative average utilization

Figure 10: Performance results for different system configurations with and without resource requirements

Although the results are similar with both job submission policies, the Random policy shows slightly better results than the RoundRobin policy. One of the main reasons is that RoundRobin only balances the number of jobs among VOs but it considers neither the number of CPUs nor the load of the systems. It is worth mentioning that both policies follow the same trend when the number of VO increases. Therefore, in the following subsection we use the Random job submission policy.

4.3.7. Job forwarding policies evaluation

In this subsection we evaluate all the forwarding policies described previously with 3, 9 and 27 VOs.

Overall, Figure 12 shows that the Inter policy is much better than the other ones. We use it as a reference; however, it neither considers the processing time of the scheduling policies with accurate resource information nor the transmission of the resource information, which might be very big.

Figure 12a shows that the makespan with both Inter_RR and Inter_RM policies is almost 10% higher with respect to the Inter policy. Using the Aggr policy, a higher value of the factor α results in a shorter makespan. In fact, the makespan with the Aggr_1.25 policy is only 3% longer with respect to the Inter policy, which means that promoting the job originator VO is crucial for the matchmaking algorithm based on aggregated resource information. Furthermore, the relative makespan increases with higher number of VOs but the difference between the makespan obtained with 3, 9 and 27 VOs is <1.5% on average. Figure 12b shows that the relative makespan and relative BSLD follow a similar trend. Figure 12c shows that the average utilization decreases (around 3.5% on average) when the number of VOs increases. Also it shows that with the Aggr_1.25 policy the average utilization is close to the average utilization with the Inter policy. Figure 12d shows that the average forwarding varies significantly with the different policies. In general, the shorter makespan/BSLD the lower average job forwarding; however, its variation with different number of VOs is moderate.

Therefore, the obtained results state that our approach is scalable with up to 27 VOs with a large amount of jobs and resources.

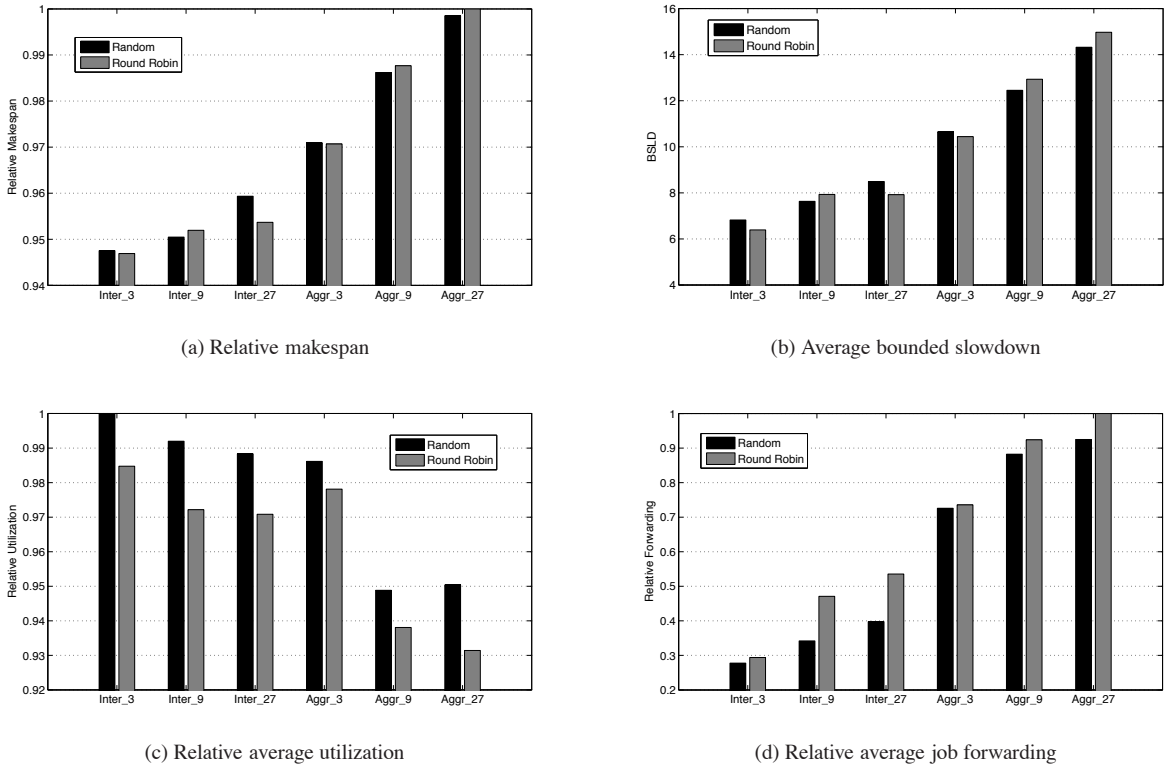
5. Related Work

In this section, we review existing work related to the major aspects of grid interoperability presented in this paper: interoperability architectural designs, their performance studies, and resource information models.

5.1. Grid Interoperability

The core concept of grid computing defines an architecture to support shared access to resources provided by members of virtual organizations (VO) that are formed by collaborative data centers and institutions. Some examples of grids are TeraGrid in US [23], GridX1 in Canada [24], Naregi in Japan[10], APACGrid in Australia [25], Garuda in India [26], Grid5000

[10] http://www.naregi.org



(a) Relative makespan

(b) Average bounded slowdown

(c) Relative average utilization

(d) Relative average job forwarding

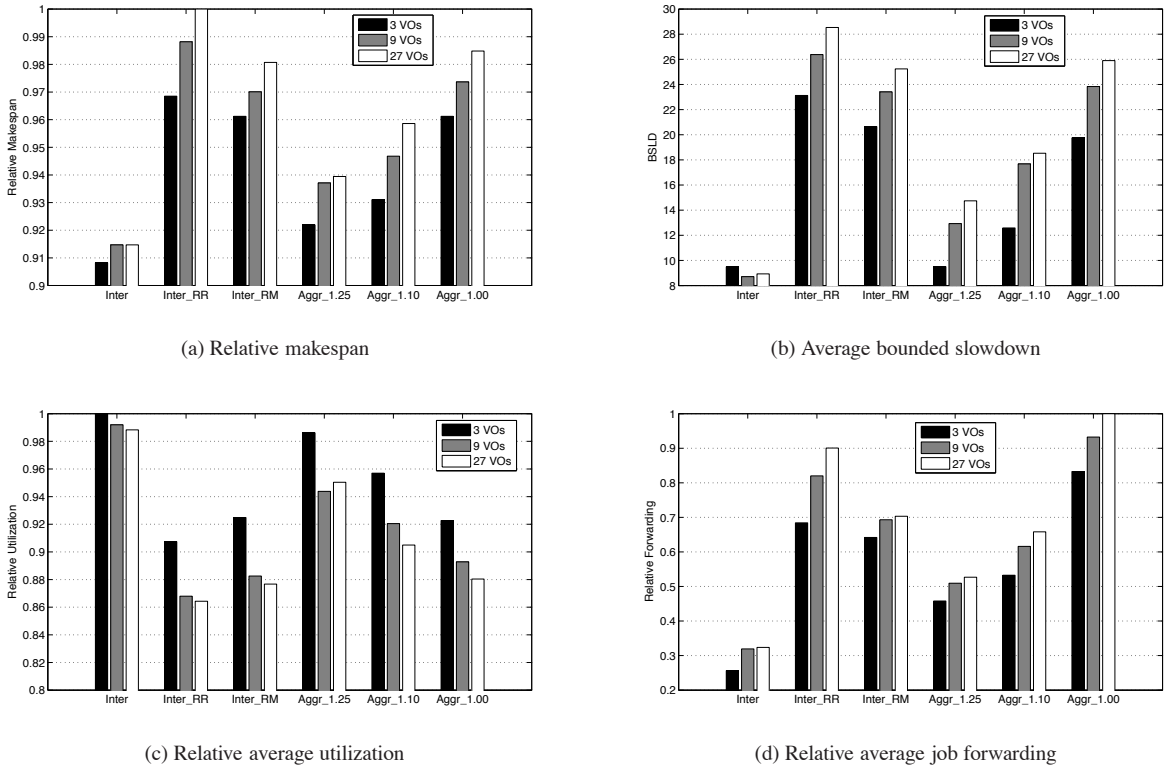
Figure 11: Performance results obtained with different submission strategies using "bestBrokerRank" (Inter) and using aggregated resource information (Aggr) for different number of VOs

in France [27], DAS-2 in the Netherlands[11], D-Grid in Germany[12], e-Science in UK [28], and EGEE in Europe[13]. Note that the majority of grids result from regional initiatives. There is a need for interoperability among different grid systems to form large grid environments such that users can access resources across VOs. In other words, grid users should deal with only one unique mechanism despite of the fact that they may be able to access many different grid systems. On one hand, several projects have attempted to unify user interfaces for accessing different supercomputing grids such as HPC-Europa [3], DEISA[14] and PRACE[15], achieving the formation in the top level of the European HPC ecosystem. On the other hand, different approaches have been started exploring the interoperability of grid systems beyond user interface. Some examples are briefly reviewed below:

- GridWay supports grid interoperability [29] through its grid gateways [30]. User requests from one grid are forwarded to others when the current one is overloaded. GridWay is based on Globus [31], and they are experimenting with GT4 and gLite3. Based on the GridWay approach, Leal *et al*. [32] presents a decentralized model for scheduling on federated grids to improve makespan and resource performance. They evaluate five different configurations using the GridSim toolkit [33] and conclude that using their proposed Dynamic Objective and Advance Scheduling (DO-AS) strategy on each grid of the federated grid reduces the makespan of the applications and increases the performance of the grid infrastructure.
- InterGrid [34] promotes interlinking different grid systems through economic-based peering agreements to enable inter-grid resource sharing. The *IntraGrid Resource Managers* (IRM) play the role of resource brokers. The *InterGrid Gateway* is responsible for establishing agreements with other grids through their IRMs. Within InterGrid, Assuncao *et al*. [35] presents some performance studies also using the GridSim simulator to evaluate some cost-aware policies to redirect requests to other grids at peak demand and using performance metric such as the average weighted response time [36].
- The gLite Workload Management Service (WMS) provides functions of a resource broker. It accepts job submission described in the gLite Description Language (JDL) that was originally developed for the EU DataGrid project[16] and is based on the Condor ClassAd lan-

[11] http://www.cs.vu.nl/das2/
[12] http://www.d-grid.de
[13] http://www.eu-egee.org/
[14] http://www.deisa.eu
[15] http://www.prace-project.eu

[16] http://www.eu-datagrid.org



(a) Relative makespan

(b) Average bounded slowdown

(c) Relative average utilization

(d) Relative average job forwarding

Figure 12: Performance results obtained with different job forwarding policies for different number of VOs

guage [22]. Within EGEE, there are efforts to enable interoperability between gLite and UNICORE [37] systems [38][39].

- The Koala grid scheduler [5] focuses on data and processor co-allocation. It is designed to work on DAS-2 multi-cluster and lately on DAS-3 and Grid5000. To interconnect these different grid domains, they use inter-broker communication between different Koala instances. Their policy is to use resources from a remote domain only if the local one is saturated. They use delegated matchmaking [40] to obtain the matched resources from one of the peer Koala instances.

- VIOLA MetaScheduling Service [41] implements grid interoperability via WS-Agreement [42] and provides coallocation of multiple resources based on reservations. The main goal of VIOLA is to achieve the necessary interoperability using Service Level Agreement (SLA) mechanisms.

Unlike Gridway and InterGrid, which support interoperability of instances of the same resource brokering and local resource management, our interoperability design assumes that there is common meta-brokering protocol and resource information model implemented by each partnering grid resource brokers, while other brokering and local resource management systems can have heterogeneous designs and implementations. We assume that JSDL is the common job description language among different grids. Similar to Gridway and InterGrid, user requests are forwarded to other grids if necessary. In this aspect, our approach differs from Koala and Viola, which acquire or coallocate resources from partnering grids when necessary. For performance studies, we use Alvio simulator and study WRF job performance in terms of makespan and bounded slowdown for different job forwarding policies.

In addition to the above projects, there are also efforts like the P-GRADE portal [43], Grid Interoperability Project (GRIP) [44], and the Open Middleware Infrastructure Institute for Europe (OMII-Europe) project[17]. The P-GRADE portal tries to bridge different grid and e-Science infrastructures by providing access to standard-based interoperable middleware. GRIP was one of the first proposals enabling interoperability between UNICORE and Globus Toolkit. OMII-Europe aims to significantly influence the adoption and development of open standards that facilitate interoperability between gLite and UNICORE such as OGSA BES [45] or JSDL [7].

There are many projects with the goal of establishing an open standard besides OMII-Europe. The Grid Scheduling Architecture Research Group (GSA-RG) of Open Grid Forum (OGF)[18] is currently working on enabling grid scheduler interaction. They are working to define a common protocol and interface among schedulers enabling inter-grid resource usage, using standard tools (JSDL, OGSA, WS-Agreement). How-

[17] http://www.omii-europe.org

[18] http://forge.ogf.org/sf/projects/gsa-rg

ever, the group is paying more attention to agreements. They proposed the Scheduling Description Language (SDL) to allow specification of scheduling policies based on broker scheduling objectives/capabilities (such as time constraints, job dependencies, scheduling objectives, preferences, etc.). The Grid Interoperation Now Community Group (GIN-CG) of the OGF[19] also addresses the problem of grid interoperability driving and verifying interoperation strategies. They are more focused on infrastructure with five sub-groups: information services, job submission, data movement, authorization, and applications. Aligned with GIN-CG, the OGF Production Grid Infrastructure Working Group (PGI-WG)[20] aims to formulate a well-defined set of profiles and additional specifications. LA Grid experiments with interoperability and supports shared resources to optimization job execution. We hope that our experiences and lessons learned can shed insights to the standard activities.

There are also two main activities of the OGF for job management: SAGA [46] and DRMAA [47]. SAGA provides a set of interfaces used as the application programming model for developing applications for execution in grid environments. DRMAA defines a set of generalized interfaces that applications used to interact with distributed resource management middleware. Both SAGA and DRMAA focus on applications. Our LA Grid meta-brokering protocols focus on the interaction between interoperability and not on the application management.

5.2. Resource Information Models

There are different models to represent resources for grid systems. One of the most well known models is the GLUE schema[21] used to provide a uniform description of resources and to facilitate interoperation between grid infrastructures. It was conceived as a collaboration effort focusing on interoperability between US and EU related projects. It was promoted by DataTAG [48] (EU) and iVDGL[22] (US) and received contributions from DataGrid, Globus [31], PPDG[23] and GriPhyn[24]. More recently it was included in OGF within the GLUE Working Group. GLUE has been widely used, for example by Globus Monitoring and Discovery Service (MDS) [10]. Another schema is provided by the UNICORE framework [37].

To the best of our knowledge, all of the grid interoperability initiatives use a common resource model for interchanging information between grid domains. In LA Grid, we use a resource model which is an extension of the one used in IBM Tivoli Dynamic Workload Broker (TDWB) [49]. We also consider the resource model in the aggregated form. The aggregation of resource information is a usual way to save data transfers. It has been widely used in different areas such as networking [50]. Grid resource management systems have used aggregation mechanisms previously such as in Legion [9]. Legion uses an object-based information store organization through the collection objects. Information about multiple objects is aggregated into these collection objects. Moreover, grid information systems such as MDS or Ganglia [8] provide resource data in aggregated form. MDS combines arbitrary GRIS (Grid Resource Information Service) services to provide aggregate view that can be explored or searched. Ganglia is based on a hierarchical design, relies on a multicast-based listen/announce protocol to monitor state within clusters and uses a tree of point-to-point connections amongst representative cluster nodes to federate clusters and aggregate their state. However, the existing approaches to resource aggregation have not been applied to scenarios where the interoperability between different grid systems is exploited.

[19] http://forge.ogf.org/sf/projects/gin
[20] http://forge.ogf.org/sf/projects/pgi-wg
[21] http://forge.ogf.org/sf/projects/glue-wg
[22] http://igoc.ivdgl.indiana.edu
[23] http://www.ppdg.net
[24] http://www.griphyn.org

6. Conclusions and Future Work

This paper introduces a cooperating meta-scheduling model that has been implemented by three partnering institutions: Barcelona Supercomputing Center's prototype using eNANOS, IBM Research prototype using the IBM product ITDWB, and Florida International University's prototype using the GridWay from the open source community. Our current work focuses on the meta-scheduling model and the mechanisms to support cooperation: a set of protocols to connect the meta-schedulers, submitting jobs between them, and resource information exchange. The data collected from the different implementations is intended to validate the operations between the three sites and the results from our simulations state that our solution is scalable with a large amount of jobs and resources.

The presented prototypes serve as a platform for our current research activities in the area of meta-scheduling, and will allow the exploration of new functions and protocols to optimize the matching of jobs to remote domain resources. Moreover, our platforms will also be used for LA Grid partners to explore applicability of grid computing in areas such as hurricane migration, bioinformatics, and healthcare [6].

Acknowledgments

We would like to thank the joint collaboration of other team members that are working on the job flow management aspects of this meta-scheduling project: Gargi Dasgupta, Onyenka Ezenwoye, Selim Kalayci, Juan Carlos Martinez and Balaji Visanwanthan.

This work was supported in part by IBM (SUR and Student Support awards), the National Science Foundation (grants OISE-0730065, OCI-0636031, REU-0552555, and HRD-0317692), and the Spanish Ministry of Science and Technology under contract TIN2007-60625. This work is part of the Latin American Grid (LA Grid) project. We also thank the anonymous reviewers for their comments and recommendations, which have been crucial to improve the quality of this work.

References

1. Foster, I., Kesselman, C., Tuecke, S.. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International Journal of High Perfomance Computing Applications* 2001;15(3):200–222.
2. Rodero, I., Guim, F., Corbalan, J., Fong, L., Liu, Y., Sadjadi, S.. Looking for an Evolution of Grid Scheduling: Meta-brokering. *Grid Middleware and Services: Challenges and Solutions* 2008;:105–119.
3. Oleksiak, A., Tullo, A., Graham, P., Kuczynski, T., Nabrzyski, J., Szejnfeld, D., Sloan, T.. HPC-Europa:Towards Uniform Access to European HPC Infrastructures. In: *IEEE/ACM International Workshop on Grid Computing*. Seattle, WA, USA; 2005:308–311.
4. Huedo, E., Montero, R., Llorente, I.. A Framework for Adaptive Execution in Grids. *Software: Practice & Experience* 2004;34:631–651.
5. Mohamed, H., Epema, D.. KOALA: a Co-allocating Grid Scheduler. *Concurrency and Computation: Practice & Experience* 2008;20:1851–1876.
6. Badia, R., Dasgupta, G., Ezenwoye, O., Fong, L., et al. High Performance Computing and Grids in Action; chap. Innovative Grid Technologies Applied to Bioinformatics and Hurricane Mitigation. Amsterdam: IOS Press; 2007:436–462.
7. Anjomshoaa, A., Brisard, F., Drescher, M., Fellows, D., et al. Job Submission Description Language (JSDL) Specification Version 1.0, GFD-R.056. Tech. Rep.; Open Grid Forum (OGF); 2005.
8. Massie, M., Chun, B., Culler, D.. The Ganglia Distributed Monitoring System: Design, Implementation, and Experience. *Parallel Computing* 2004;30:817–840.
9. Chapin, S., Katramatos, D., Karpovich, J., Grimshaw, A.. The Legion Resource Management System. In: *Job Scheduling Strategies for Parallel Processing (JSSPP)*. Puerto Rico; 1999:162–178. LNCS 1659.
10. Czajkowski, K., Fitzgerald, S., Foster, I., Kesselman, C.. Grid Information Services for Distributed Resource Sharing. In: *IEEE International Symposium on High-Performance Distributed Computing (HPDC)*. San Francisco, CA, USA; 2001:181–194.
11. Basney, J., Humphrey, M., Welch, V.. The myproxy online credential repository. *Software: Practice and Experience* 2005;35:801–816.
12. Rodero, I., Corbalan, J., Badia, R., Labarta, J.. eNANOS Grid Resource Broker. In: *European Grid Conference 2005*. Amsterdam; 2005:111–121. LNCS 3470.
13. Rodero, I., Guim, F., Corbalan, J., Labarta, J.. eNANOS: Coordinated Scheduling in Grid Environments. In: *International Conference on Parallel Computing (ParCo)*. Malaga, Spain; 2005:81–88.
14. IBM. Tivoli Dynamic Workload Broker: User's Guide. 2007. URL `www.redbooks.ibm.com, SG32-2281-01`.
15. Gucer, V., Biggs-Finstad, J., Cappariello, A., Dufner, M., et al. Getting Started with Tivoli Dynamic Workload Broker 1.1. 2007. URL `www.redbooks.ibm.com, SG24-7442-00`.
16. Michalakes, J., Dudhia, J., Gill, D., Henderson, T., Klemp, J., Skamarock, W., Wang, W.. Reseach and Forecast Model: Software Architecture and Performance. In: *11th ECMWF Workshop on the Use of High Performance Computing In Meteorology*. Reading, UK; 2004:156–168.
17. Martinez, J.C., Wang, L., Zhao, M., Sadjadi, S.M.. Experimental study of large-scale computing on virtualized resources. In: *Proceedings of the 3rd international workshop on Virtualization technologies in distributed computing*. Barcelona, Spain; 2009:35–42.
18. Sadjadi, S., Fong, L., Badia, R., Figueroa, J., et al. Transparent Grid Enablement of Weather Research and Forecasting. In: *Proceedings of the Mardi Gras Conference 2008 - Workshop on Grid-Enabling Applications*. Baton Rouge, LA, USA; 2008: 8.
19. Jette, M.A., Yoo, A.B., Grondona, M.. Slurm: Simple linux utility for resource management. In: *Lecture Notes in Computer Science: Proceedings of Job Scheduling Strategies for Parallel Processing (JSSPP)*. 2002:44–60.
20. Guim, F., Labarta, J., Corbalan, J.. Modeling the impact of resource sharing in backfilling policies using the alvio simulator. In: *IEEE Intl. Symp. on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*. 2007:145–150.
21. Rodero, I., Guim, F., Corbalan, J., Fong, L., Sadjadi, S.. Broker Selection Strategies in Interoperable Grid Systems. *Future Generation Computer Systems* 2010;26(1):72–86.
22. Solomon, M., Raman, R., Livny, M.. Matchmaking: Distributed Resource Management for High Throughput Computing. In: *IEEE International Symposium on High Performnce Distributed Computing (HPDC)*. Chicago, USA; 1998:28–31.
23. Catlett, C., Beckman, P., Skow, D., Foster, I.. Creating and Operating National-Scale Cyberinfrastructure Services. *Cyberinfrastructure Technology Watch Quarterly* 2006;2:2–10.
24. Agarwal, A., Ahmed, M., Berman, A., Caron, B.L., et al. GridX1: A Canadian computational grid. *Future Generation Computer Systems* 2007;23:680–687.
25. Dunning, T., Nandkumar, R.. International cyberinfrastructure: activities around the globe. *Cyberinfrastructure Technology Watch Quarterly* 2006;2:2–4.
26. Ram, N., Ramakrishran, S.. International cyberinfrastructure: activities around the globe. *Cyberinfrastructure Technology Watch Quarterly* 2006;2:15–19.
27. Bolze, R., Cappello, F., Caron, E., Dayde, M., et al. Grid'5000: a large scale and highly reconfigurable experimental Grid testbed. *International Journal of High Performance Computing Applications* 2006;20:481–494.
28. Hey, T., Trefethen, A.. The UK e-Science Core Programme and the Grid. *Future Generation Computer Systems* 2002;18:1017–1031.
29. Vazquez, T., Huedo, E., Montero, R., Lorente, I.. Evaluation of a Utility Computing Model Based on the Federation of Grid Infrastructures. In: *International Euro-Par Conference on Parallel Processing*. Rennes, France; 2007:372–381.
30. Huedo, E., Montero, R., Llorente, I.. A recursive architecture for hierarchical grid resource management. *Future Generation Computer Systems* 2009;25:401–405.
31. Foster, I., Kesselman, C.. Globus: A Metacomputing Infrastructure Toolkit. *International Journal of Supercomputer Applications* 1997;11:115–128.
32. Leal, K., Huedo, E., Llorente, I.. A decentralized model for scheduling independent tasks in federated grids. *Future Generation Computer Systems* 2009;25:840–852.
33. Buyya, R., Murshed, M.. GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing. *Concurrency and Computation: Practice and Experience* 2002;14:1175–1220.
34. Assuncao, M.D., Buyya, R., Venugopal, S.. InterGrid: A Case for Internetworking Islands of Grids. *Concurrency and Computation: Practice and Experience* 2008;20:997–1024.
35. Assuncao, M.D., Buyya, R.. Performance analysis of allocation policies for interGrid resource provisioning. *Information and Software Technology* 2009;51:42–55.
36. Grimme, C., Lepping, J., Papaspyrou, A.. Prospects of collaboration between compute providers by means of job interchange. In: *Job Scheduling Strategies for Parallel Processing (JSSPP)*. 2008:132–151. LNCS 4942.
37. Erwin, D., Snelling, D.. UNICORE: A Grid Computing Environment. In: *International Euro-Par Conference on Parallel Processing*. Manchester, UK; 2001:825–834.
38. Marzolla, M., Andreetto, P., Venturi, V., Ferraro, A., et al. Open Standards-Based Interoperability of Job Submission and Management Interfaces across the Grid Middleware Platforms gLite and UNICORE. In: *IEEE International Conference on e-Science and Grid Computing*. Bangalore, India; 2007:592–601.
39. Riedel, M., Memon, A., Memon, M., Mallmann, D., et al. Improving e-Science with Interoperability of the e-Infrastructures EGEE and DEISA. In: *International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*. Opatija, Croatia; 2008:225–231.
40. Iosup, A., Epema, D., Tannenbaum, T., Farrelle, M., Livny, M.. Inter-Operable Grids through Delegated MatchMaking. In: *International Conference for High Performance Computing, Networking, Storage and Analysis (SC07)*. Reno, Nevada; 2007:.
41. Seidel, J., Waldrich, O., Ziegler, W., Wieder, P., Yahyapour, R.. Using SLA for Resource Management and Scheduling - a Survey, TR-0096. Tech. Rep.; Institute on Resource Management and Scheduling; 2007.
42. Andrieux, A., Czajkowski, K., Dan, A., Keahey, K., Ludwig, H., Nakata, T., Pruyne, J., Rofrano, J., Tuecke, S., Xu, M.. Web Services Agreement Specification (WS-Agreement), GFD-R-P.107. Tech. Rep.; Grid Resource Allocation Agreement Protocol (GRAAP) WG, Open Grid Forum; 2007.

43. Kacsuk, P., Kiss, T., Sipos, G.. Solving the Grid Interoperability Problem by P-GRADE Portal at Workflow Level. *Future Generation Computer Systems* 2008;24:744–751.
44. Brooke, J., Fellows, D., Garwood, K., Goble, C.. Semantic Matching of Grid Resource Descriptions. In: *European Acrossgrids Conference*. Nicosia, Greece; 2004:240–249. LNCS 3165.
45. Foster, I., Grimshaw, A., Lane, P., Lee, W., et al. OGSA Basic Execution Service Version 1.0, GFD-R.108. Tech. Rep.; Open Grid Forum (OGF); 2008.
46. Goodale, T., Jha, S., Kielmann, T., Merzky, A., Shalf, J., Smith, C.. A Simple API for Grid Applications (SAGA), GWD-R.72. Tech. Rep.; SAGA-CORE Working Group, Open Grid Forum; 2006.
47. Troger, P., Rajic, H., Haas, A., Domagalski, P.. Standardization of an API for Distributed Resource Management Systems. In: *Proceedings of the Seventh IEEE International Symposium on Cluster Computing and the Grid*. Washington, DC, USA. ISBN 0-7695-2833-3; 2007:619–626.
48. Martin, O., Martin-Flatin, J., Martelli, E., Moroni, P., Newman, H., Ravot, S., Nae, D.. The DataTAG transatlantic testbed. *Future Generation Computer Systems* 2005;21:443–456.
49. Workload automation and service execution: challenges and solutions for today. IBM White paper; 2007.
50. Helvaci, A., Cetinkaya, C., Yildirim, M.. Using Rerouting to Improve Aggregate Based Resource Allocation. *Journal of Networks* 2008;3:1–12.