Eighth IEEE International Symposium on Cluster Computing and the Grid

Enabling Interoperability among Meta-Schedulers

Norman Bobroff[1], Liana Fong[1], Selim Kalayci[2], Yanbin Liu[1], Juan Carlos Martinez[2], Ivan Rodero[3], S. Masoud Sadjadi[2], and David Villegas[2]

[1]IBM Watson Research Center, Hawthorne, NY, USA
{bobroff, llfong, ygliu}@us.ibm.com
[2]Florida International University, Miami, FL, USA
{skala001, jmart054, sadjadi, dvill013}@cs.fiu.edu
[3]Technical University of Catalonia and Barcelona Supercomputing Center, Barcelona, Spain
irodero@ac.upc.edu

Abstract

Grid computing supports shared access to computing resources from cooperating organizations or institutes in the form of virtual organizations. Resource brokering middleware, commonly known as a meta-scheduler or a resource broker, matches jobs to distributed resources. Recent advances in meta-scheduling capabilities are extended to enable resource matching across multiple virtual organizations. Several architectures have been proposed for interoperating meta-scheduling systems. This paper presents a hybrid approach, combining hierarchical and peer-to-peer architectures for flexibility and extensibility of these systems. A set of protocols are introduced to allow different meta-scheduler instances to communicate over Web Services. Interoperability between three heterogeneous and distributed organizations (namely, BSC, FIU, and IBM), each using different meta-scheduling technologies, is demonstrated under these protocols and resource models.

Keywords: meta-scheduler, resource broker, interoperable scheduling protocol.

1. Introduction

Grid computing supports the harnessing of computing resources from cooperating organizations or institutes in the form of a Virtual Organization (VO) [1] in order to meet the demand for computing power, increase resource utilization, and to share the cost of resource ownership. Recent advances in cooperating grids (or interoperating VOs) support fulfilling resource requests using resources across multiple grids. This vision of cooperating grids further enhances opportunity for global optimizations of resource usage, and reduces execution cost, especially for globally distributed partners.

Several obstacles exist to realizing such cooperating grids. Typically, each partner grid is independently developed, managed, and operated, and often does not adhere to common standards and specifications. Consequently, grid partners can vary widely in computing and storage capabilities, grid middleware, cluster managers, local schedulers, as well as organizational policies and accounting practices for accepting and executing jobs. The challenge is to define a management and scheduling structure that allows diverse partnering grids to interoperate by enabling global resource matching, job scheduling, and resource usage optimizations. Another structural barrier to interoperability is the lack of adoption of a common process and language for expressing job submission and resource requirements. Rodero et al. [2] articulated these challenges. Different architectures have been proposed for these interoperating MS systems. HPC-Europa's Single Point of Access (SPA) [3] provides uniform APIs for users to access multiple grids. GridWay [4] supports peer-to-peer intra-MS connections and job forwarding. Koala [5] supports resource co-allocation across multiple grids if a single grid can not fulfill incoming job requests. Unlike these approaches, our MS model has the focus on peer-to-peer and hierarchical intra-MS connections, matching with aggregated resource information, and job forwarding between partnering grids.

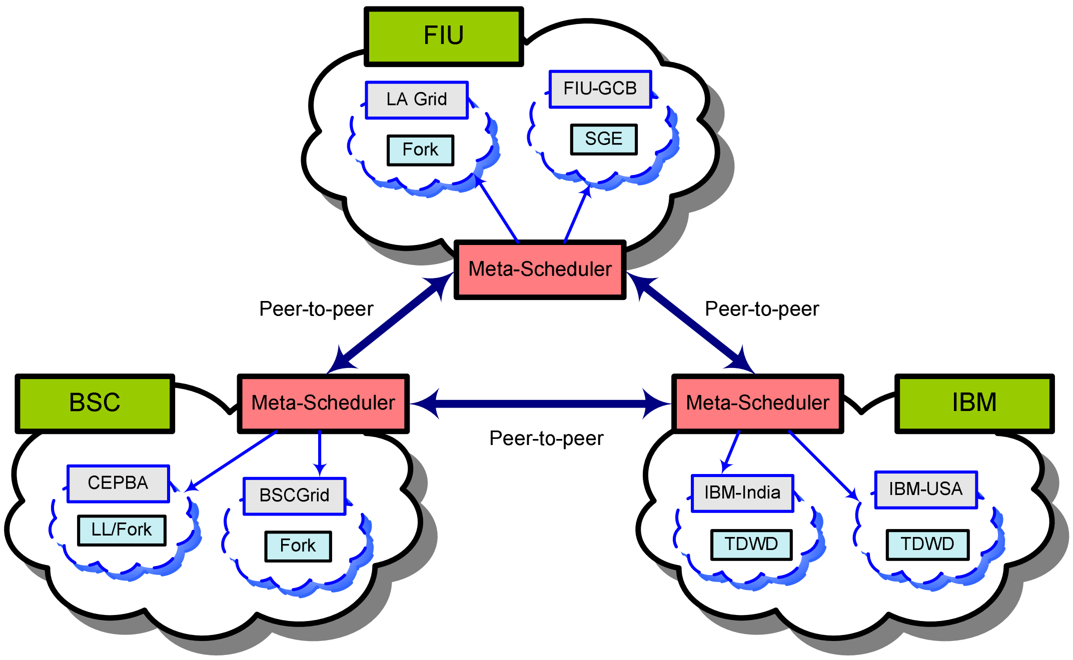

Figure 1: Cooperating meta-scheduling in LA Grid

This paper describes an approach to extending VOs across heterogeneous grids based on interoperating meta-schedulers (MSs). The interoperable MS model supports the autonomy of organizations. Each organization, namely a resource domain, has a MS that interfaces externally to a peer in the partner domains, and internally to local dispatchers, schedulers, and optionally other MSs. The resource information of each domain is aggregated and distributed among its peers. Specifically, we detail the design and implementation of a peer-to-peer collaborative meta-scheduling environment consisting of three VO domains – Florida International University (FIU), Barcelona Supercomputer Center (BSC), and IBM Research (IBM), three of the partnering institutions within the LA Grid Initiative [6].

The contributions presented here include the applicability of the meta-scheduling design to support the dynamic resource variability in a heterogeneous grid environment. In, particular with regard to MS connectivity protocols, aggregated resource information exchange, and aggregated resource matching. We present the implementations of protocols, resource models, and MSs based on three distinct MS architectures that are shown to interoperate, forming a federated grid environment.

The following section introduces the cooperating meta-scheduling model and describes the protocols and models for inter-connectivity, resource information exchange, and job management. Sections 3, 4, and 5 detail the architecture and implementations of the cooperative models at BSC, IBM Research and FIU, respectively. Section 4 presents our experimental platforms at these three different sites and some experimental data. Section 7 surveys related work. Section 8 concludes the paper and suggests future directions.

2. LA GRID Meta-Scheduling Interoperation

In our interoperable MS model, the MSs may have heterogeneous implementations, but they adhere to a common set of communication and information encapsulation protocols that allow them to interoperate and provide a homogeneous view of the interconnected grids to job submission users. These interoperation protocols are designed to achieve a peer-to-peer scheduling environment for forwarding jobs to execution domains without user involvement. Figure 1 shows interconnected MSs at three diverse institutions (BSC, FIU and IBM).

Administrators can restrict and shape the inter-connected structure of the grid by defining the connection paths between MSs, and by setting the role (consumer, provider, or peer) of the MS in the connection. In the provider role, an MS sends a description of its resources to connected consumer MSs. Provider MSs will accept jobs from connected

MSs in consumer roles. The consumer MS matches resources from connected providers and forwards jobs to the providers when necessary. Any MS can concurrently act in both roles on a single connection and this is called the peer role. Figure 1 shows the example of three cooperating organizations, which has peer-to-peer relationship between the organizations, while a hierarchy of providers and consumers has been defined within each organization.

Table 1: List of APIs in the LA Grid Meta-Scheduling

Connection API	**Resource Management API**	**Job Management API**
openConn()	*requestResourceData()*	*submitJob()*
notifyConn()	*sendResourceData()*	*queryJob()*
heartbeat()		*notifyJob()*
		cancelJobl()

The application programming interfaces (APIs) that support this interoperation are provided in Table 1 and Figure 2. Initially, an authenticated connection is negotiated between two MSs by invoking openConn(). Negotiated parameters include the heartbeat rate to monitor connection status, authentication protocol, and optionally the types of job submission languages that are understood by the MS. At connection initiation the parties also state their roles as resource provider, consumer, or both. These roles may be later changed at any time using the notifyConn() API.

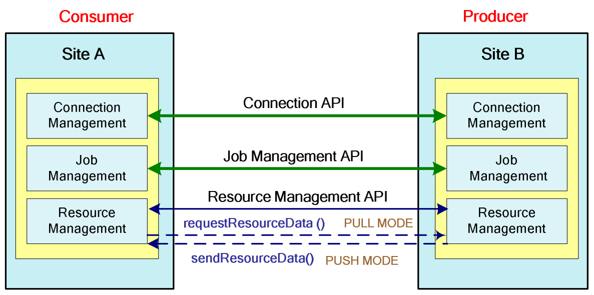

Figure 2: Protocols in the meta-scheduling implementation

Negotiation proceeds as follows: If the request parameters in openConn() are acceptable, the remote MS responds positively with the relevant information (e.g. heartbeat frequency, and web service endpoints at which to send resource information and jobs). The remote MS also starts sending heartbeats using heatbeat() API. If the request parameters are not acceptable the remote MS counters with a new openConn() call proposing alternative parameters. Negotiations continue until an agreement is reached or the number of rounds exceeds a threshold specified by the initiator, in which case the connection attempt fails. After a connection is established, the notifyConn() may be used to notify error conditions or to gracefully terminate the communication, when necessary.

Once a connection is established, resource information is sent to the consumer MS either in pull (requestResourceData()) or push (sendResourceData()) mode. The push mode is also used when updates are triggered by dynamic changes in resource capacity, utilization, or availability. Resource updates may be full or incremental. Full updates are typically requested in pull mode by the consumer; incremental updates are generally pushed by the provider when the resource availability or/and utilization changes in the domain. If an MS has connections to multiple providers, it may share the resource information from them with other MSs that are consumers of this MS.

In any sizable grid, the exchange of detailed resource information presents a scalability issue. Hence we consolidate resource information exchanged among MSs by using aggregation. Details of the aggregated resource model are beyond the current scope of this paper, but a brief description and example is provided. Aggregated resources are derived from commonly known resources such as computer systems and operating systems. Each resource has a set of attribute value pairs such as ProcType='x86', ProcSpeed='3Ghz'. An aggregated resource is created from a resource by extending the attribute values to summarize the range of values in the aggregation. Resources also have relationships between them such as 'reference' or 'contain' The following is a partial example of an aggregated resource describing 100 'ComputerSystems', 20 with Linux OS.

Resource = {type = ComputerSystem, name = cs_x}
  ProcType = {(Intel, <count=100>)}
  ProcSpeed = {(3000, <count=100>*, <total=3000>*)}
  ProcNum = {(2, <count=100>*, <total=2>*)}
  CPUUtil = {(20, <count=100>, *<total=20>*)}
Resource = {type = OperatingSystem, name = os_y}
  OSType = {(Linux, <count=20>)}
  FreeMem = {(2000-4000, <count=20>, <total=6000>)}
  FreeVirMem = {(1000-2000, <count=20>, <total=3000>)}
  Relationship = {type = Reference
    SourceType=ComputingSystem SourceName=cs_x
    TargetType=OperatingSystem TargetName=os_y}

Aggregated resources suffice for job-to-resource matching as an MS only needs to provide a candidate resource, delegating exact matching to local schedulers.

Users can submit job requests to any consumer or provider MS using the synchronous submitJob() call. In our current implementation, we use JSDL (Job Submission Description Language [7]), an Open Grid Forum (OGF)[8] proposed recommendation, as the standard format for job submission. The receiving MS creates a local record of the submission and assigns a unique identifier to the job which is returned immediately to the submitter. Then the MS checks for a match against its resources and decides whether to schedule locally, or try to match against the potentially available resources of remote MSs with the provider roles. If the job is forwarded to a remote MS, the same submitJob() interface is used, as if it was a user submission with a different submission type. But, now the End Point Reference (EPR) of the forwarding scheduler is attached to the job forward information. This process is repeated until the job reaches the provider MS that will execute it locally.

As the job finally executes under the control of a local scheduler, the job state changes are propagated back to the job-originating MS using the notifyJob() service through intermediary forwarding MSs. By this mechanism, the client is informed asynchronously of job state changes. The client or a system administrator, may also query the state of the job using the synchronous queryJob() service.

Three LA Grid member sites have implemented this resource model and set of protocols each using their own specific implementations of the MS, local scheduler, and web service technology. Details of each implementation are explained in the following sections.

3. BSC Meta-Scheduler

At BSC, we have implemented the LA Grid MS functions using the eNANOS framework[9][10], which is based on GT4 services such that every component is a service. The implementation consists of 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 3 with the LA Grid extensions in dark shading.

Since other LA Grid MSs have implemented the protocols using regular web services, eNANOS extensions have been implemented as a set of Axis2 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 acts as a wrapper to support redirecting calls to GT4 and performing data transformations when necessary. To support interactions between the eNANOS and other LA Grid MSs, we implement a set of regular web services for the APIs as the client interface.

Leveraging the default persistency mechanism of GT4, the data relevant to LA Grid functions is stored using the GT4 Resource Properties. The data to be stored include MS connections and resource information from other MSs.

We have implemented a new set of resource management functions to support resource information exchange between partnering MSs. 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.

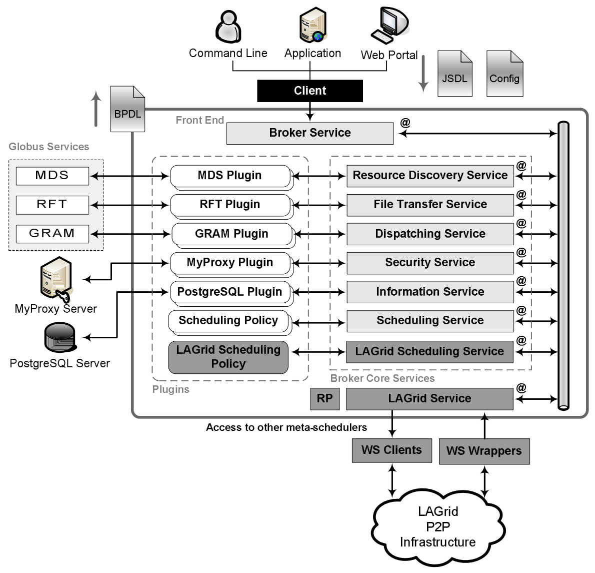

Figure 3: Architecture of eNANOS with the LA Grid

In addition to receiving jobs routed from other LA Grid MSs 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 3. The modifications in the job submission interface support the additional LA Grid parameters (such as the connection ID or the notification EPR). To allow the eNANOS services the ability to manage the LA Grid forwarded jobs, we have modified the job schema used in eNANOS. In particular, we have added a new element that contains a set of LA Grid information, and a new job status value (FORWARDED) for the jobs that have been forwarded to or from another MS. The extension of the schema has been done with the XML type in Figure 4, where ConnID is the connection ID, the OriginatorEPR is the end-point reference of the forwarding MS or the MS that has to receive notifications, and the job realStatus is the status from execution environment.

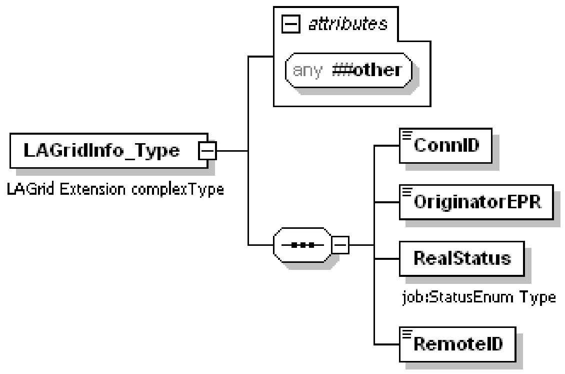

Figure 4: Extended job schema for eNANOS

Significant modifications in the scheduling service are necessary to implement the scheduling policies based on aggregated resource information and to allow job forwarding. Thus, we implement a separate scheduling service for LA Grid because the changes are structural upgrades that break the consistency with other existing services. Moreover, we implement a scheduling policy for LA Grid using a new scheduling policy plug-in called "LAGridPolicyService". Since both services and plug-ins are implemented as modules with abstract interfaces, the upgrading of the broker has been done by defining those new services and updating the configuration file.

"BestBrokerRankPolicy" selects the best broker to submit a job given a set of aggregated resource information. In case that the job is in the eNANOS domain, the policy returns the resource to execute the job. Otherwise, it returns the MS for forwarding the job to. The algorithm is summarized as follows:

BestBrokerRANK Policy = MAX (BrokerRANK($broker_i$), i=1...n)

where

BrokerRANK($broker_i$) = MAX (ResourceRANK$_i$($resource_j$), j=1...m), n is the number of brokers, m is the number of resources, and ResourceRANK(resource) is the accumulation of the RANK obtained from the requirements matching applied to each of the main resource attributes (e.g. ProcType, OSType, ProcSpeed) with an impact factor.

With respect to the notification functionality, we have modified the monitoring of eNANOS to notify other MSs when the status of a forwarded job has changed. It has been done as an extension of the current monitoring code but using the LA Grid information incorporated in the job schema.

4. IBM Meta-Scheduler

The IBM Research MS is implemented by extending an IBM scheduling product: IBM Tivoli Dynamic Workload Broker (ITDWB) [11].

An ITDWB server collects resource information from agents on resources, accepts job requests from users, and matches jobs to resources that have dynamic availability and utilization. The combined ITDWB architecture and MS extensions are shown in Figure 5. The base product components[12] (shown as boxes on the right side of Figure 5) are a job dispatcher, a resource advisor, a repository to persist job state and resource information, and a set of workload agents (shown as boxes at the bottom of Figure 5) for collecting resource information and for job execution and monitoring on computing platforms.

Our MS includes interface and functionality extensions to ITDWB. We add new web services components (the left side of Figure 5) to realize the interoperability with other MSs, as described in previous sections. Through these extensions the broker server interacts with other MSs, in addition to its own agents, for resource information retrieval and job dispatching.

The Job Dispatcher (JD) component manages the lifecycle of jobs including storing and updating job information in the repository, dispatching jobs to the Job Executors of the allocated resources, and communicating with job submitters. The JD has been enhanced to process jobs routed from and to other MSs. For jobs routed to other schedulers, job state and forwarding information is stored in the repository. Thus, the MS receiving a job query knows how to forward the query and how to send back the status report and execution result in the reverse direction.

Since the current ITDWB supports only a dialect of JSDL as the job description language, we extend it to work with OGF's JDSL[1]. We implement a JSDL converter that can convert formats between different definitions represented in xml files. With the ongoing

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

evolution of JSDL, vendor specific extensions and dialects, such a converter is a necessity.

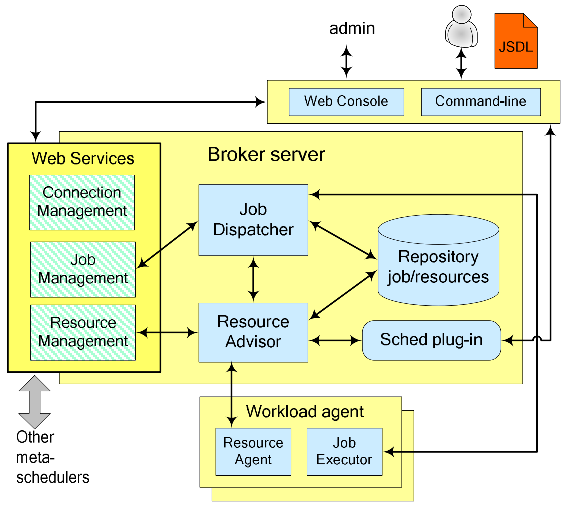

Figure 5: Extended ITDWB for meta-scheduling functions

The Resource Advisor (RA) has also been enhanced to process aggregated resource information supplied by remote MSs that have established a connection, in addition to resource information from the local agents. The RA also retrieves resource information from the database and aggregates them in response to requestResourceData() calls. Local and remote resource data, in either detailed or aggregated form, is maintained currently in a common repository. Using retrieved information from the repository, the RA performs resource matching and makes decisions about whether to dispatch a job locally, or to forward it another MS. The algorithms for matching jobs to resources can be implemented as a scheduling plug-in to the RA. In summary, the job brokering flow is:

1. JD receives job description and contacts RA for resources
2. RA searches database for sets of candidate resources matching the job's requirements; a set of resources can be local resources to the MS or an MS with potential resources with matched properties
3. RA calls the scheduling plug-in, which chooses the most suitable set of resources from the candidate sets
4. RA returns the chosen resources to the JD
5. JD sends the job to
   - Execution agents of the selected resources, or
   - A selected MS
6. JD listens to the status updates and forwards updates to job's submitter and/or other registered listeners.

5. FIU Meta-Scheduler

The design of FIU's MS was started with the selection of a grid scheduling software using the following self-imposed criteria: *open standards, open source software, and ease of use and integration*

The first criterion is that Grid computing needs to be based on well established standards in order for new development to easily integrate new products and technologies. Second, the use of open source standards was to allow the modification of the code in case of future needs. Finally, the software used to build our MS must be easy to integrate with, to minimize the impact of existing functions.

After considering different products that met our needs, such as LSF[10], we finally chose GridWay, which largely fulfills our requirements:

- *Open standards:* GridWay is a Globus incubator project that is supported by the Globus Consortium. This means that all advances in GridWay will be aligned with the Globus project direction.
- *Open source software:* GridWay is a community project with an open source license, allowing any party to contribute to it with new improvements.
- *Ease of use and integration:* The GridWay MS has a modular implementation, allowing new plug-in managers to handler different set of tasks. Additionally, GridWay supports Distributed Resource Management Application API (DRMAA) [14] [15], OGF's recommended specification for job submission and status query.

The FIU MS is implemented as a set of functional modules that support the MS protocols, inter-site scheduling functions and resource management. We rely on GridWay as a scheduler for our site in managing multiple clusters at FIU domain. The MS modules and the relation with GridWay are shown in Figure 6.

In the following, we describe briefly each functional module of the FIU MS. The User-Client module is in charge of receiving external users' requests such as job submission or resource management. Users can interact with the MS using a command line interface, and submit jobs using OGF's JSDL files.

The WS-Client module implements the support of MS APIs using Apache Axis2 as the web service container, and offers a set of classes to issue requests to other peers

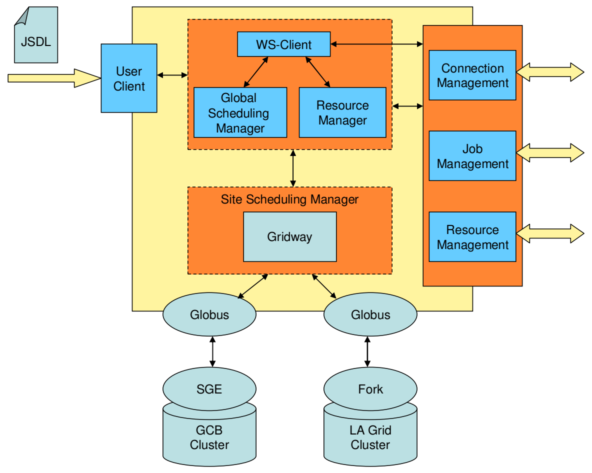

Figure 6: Architecture of FIU Meta-Scheduler

The Site Scheduling Manager module provides GridWay functionality for intra-site scheduling. It uses DRMAA for job submission and monitoring.

The Global Scheduling Manager module has the responsibility of performing inter-site scheduling. It implements the resource matching algorithm and decides whether a job will be scheduled and processed using resources in the local domain, or decides to forward the job to another connected provider MS. Additionally, this module keeps track of status of jobs submitted from other domains.

The Resource Manager module stores information about the resources in local domain and from the remote MSs. It also pushes incremental resource data to the connected MS in case of changes in local resource availability and utilization**.**

6. Experimental Results

6.1. Experimental Setup

We have tested the implemented MS APIs on the IBM, FIU and BSC implementations. We collected two sets of experimental data. It is important to note that the experimental data presented here serve as functional validation of the interoperability of the MSs. The data are intended for qualitative comparison among our MSs and not for quantitative performance evaluations.

For the resource information exchange protocols, each MS would gather and aggregate 100 resources to send back when it receives the requestResourceData() calls from the driver program. The driver program would then turn around and use the same resource information in the next set of sendResourceData() calls to MSs. While we have specified the number of resources used for the tests, the type of resources and the aggregation algorithms used vary in different MSs.

The job in our tests is a simple "sleep" job of 10 seconds. It is expressed in JSDL as follow:

```
<?xml version="1.0" encoding="UTF-8" ?>
<jsdl:JobDefinition xmlns:jsdl=
   "http://schemas.ggf.org/jsdl/2005/11/jsdl
 " xmlns:jsdl-posix=
   "http://schemas.ggf.org/jsdl/2005/11/jsdl
 posix">
 <jsdl:JobDescription>
   <jsdl:Application>
    <jsdl-posix:POSIXApplication>
    <jsdl-posix:Executable>/bin/sleep
     </jsdlposix:Executable>
    <jsdl-posix:Argument>120</jsdl-
   posix:Argument>
    <jsdl-posix:Input>/dev/null</jsdl-
   posix:Input>
    <jsdl-posix:Output>stdout</jsdl-
   posix:Output>
    <jsdl-posix:Error>stderr</jsdl-
   posix:Error>
     </jsdl-posix:POSIXApplication>
   </jsdl:Application>
 </jsdl:JobDescription>
  </jsdl:JobDefinition>
```

Table 2: Delay Time for meta-scheduling protocols

Operation	**Delay Time (milliseconds)**		
	BSC	IBM	FIU
openConn()	41	11	7
notifyConn()	35	11	8
requestResourceData()	121	70	25
sendResourceData()	22	137	14
submitJob()	147	125	62

In Table 2, we tabulate some experimental results for one run, showing the interaction of the testing driver program and each MS implemented. In each run, the driver program issues the set of protocol calls to each MS for 20 iterations and obtains the averages.

6.2. BSC Meta-Scheduler Functionality and Validation Test

We have installed 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 2 shows the results obtained from the tests using the driver program.

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

6.3. IBM Meta-Scheduler Functionality and Validation Test

We have installed an instance of the IBM MS 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, is also running on the same machine. The IBM column in Table 2 shows the data collected on interactions between the driver program and IBM MS. We note that the operations submitJob() and 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 show that the sendResourceData() operation consistently takes longer than requestResourceData(). We suspect that 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 show reduction in timing by 50% for sendResourceData() calls while timings for openConnect() and others remained similar.

6.4. FIU Meta-Scheduler Functionality and Validation Test

An instance of FIU MS has been 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 MS installation. The same machine is also installed with GridWay and GT4 services. The FIU column in Table 2 shows the experimental results.

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

6.5. IBM-BSC-FIU Meta-Scheduler Interoperability Test

To test interoperability between different MSs, the "sleep" job was submitted to one MS locally and then forwarded another MS. The validation tests for some combinations of MSs and the time delays are tabulated in Table 3.

Table 3: Delay across meta-scheduling sites

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**

For the the FIU→BSC and the BSC→FIU tests, the machines are situated at two different physical sites, namely Miami in the United States and Barcelona in Spain. Therefore, the calls have to travel through the Internet and the measured times are substantially higher and present an increased variability in the results. There is no MS selection logic used in this set of interoperability tests for the submitJob() process. The job submitted to one MS would automatically route to another MS.

For the FIU→IBM and the IBM→FIU tests, we use two machines of the same configuration on the same subnet. One machine has the IBM implementation and the other has the FIU implementation. Thus, the data collected would have negligible network delay and mainly represented the operation delay between two different MSs. As shown in Table 3, the FIU→IBM data showed significantly smaller delay than interactions between BSC and FIU. The openConn() and requestResourceData() delays are similar to those FIU→IBM test cases. However, the submitJob() delay is in the order of seconds, as a job submitted to the IBM MS would go through the full job processing cycle. The cycle consists many steps: 1) jobID created for arrived job, 2) persisted job information to database, 3) job waited on job queue for resource matching process, which wakes up on a configurable interval to scan queued jobs, 4) resource matching resulted in selecting FIU MS to route the job as the FIU has the required resource, and 5) received the job by the FIU MS and returned a unique jobID to the IBM MS.

7. Related Work

The need for interoperability among different grid systems in different resource domains was discussed in, and some projects have addressed this topic such as GRIP [16] and HPC-Europa SPA. Lately, some initiatives have been started exploring Grid interoperability following similar objectives but in different directions. The two main approaches for Grid interoperability are extending existing schedulers to make them interoperable, and using a meta-broker that can be connected to existing unmodified schedulers, as discussed by Kertesz, et al. [17]. GridWay has incorporated the support for multiple grids in the recent release [18]. In the MS layers, GridWay instances can communicate and interact through its grid gateways to access resources belonging to different domains. The basic idea is to forward user job requests to another domain when the current one is overloaded. The support of job forwarding to other domain in GridWay and our MS model are similar. However, the two models differ in inter-MS protocols and resource information management. The Koala grid scheduler is another initiative, which is focused on data and processor co-allocation. It was designed to work on DAS-2 multi-cluster and lately on DAS-3 and Grid'5000. To inter-connect these different grid domains, they use inter-broker communication between different Koala instances. Its policy is to use resources from a remote domain if the local one is saturated. It uses delegated matchmaking to obtain the matched resources from the peer Koala instances. Unlike Koala, our MS design supports job forwarding to the remote domains. VIOLA Meta-Scheduling Service implements grid interoperability via WS-Agreement [19] and provides co-allocation of multiple resources based on reservation. Our MS design consists openConnect() API that can be used as MS negotiation. However, our current implementations do not use WS-Agreement and do not yet support reservation.

The GSA-RG of OGF currently works on enabling grid scheduler interaction by defining some common protocols and interfaces among schedulers to enable inter-grid resource usage, using standard tools such as JSDL, OGSA and WS-Agreement. The group currently focuses on agreements. It proposes 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.). Following a similar idea, STAKI and BSC propose a Broker Property Description Language (BPDL) [20] [21]. The current focus of our project is not in broker description language. Instead, we focus on defining a set of protocols for connection and job management. In the future, we will investigate the possibility of using the above agreement or broker description languages.

There are two main activities of the OGF for the job management: SAGA [22] and DRMAA. 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 can use to interact with distributed resource management middleware. Both SAGA and DRMAA focus on client applications. Our LA Grid job management protocols focus on the interaction between scheduling middleware but not on the application programming model.

8. Conclusions and Future Work

This paper introduces an interoperating meta-scheduling model. It has been implemented by three partnering institutions: Barcelona Supercomputing Center's prototype (using eNANOS), IBM Research's prototype (using the IBM product ITDWB), and Florida International University's prototype (using the GridWay from the open source community). Our current work has focused on the meta-scheduling model and the mechanism to support the cooperation: a set of protocols in connecting the MSs, submitting jobs between MSs, and resource information exchanges. The data collected from the prototype implementations validate the interoperability between the three MSs. These prototypes serve as platforms for our current research activities in grid scheduling. We plan to expand our research in many directions: a richer set of meta-scheduling functions and protocols, optimization for job to resources and domains matching, aggregated resource data model, and scalability studies. Moreover, our platforms will also be used for LA Grid partners to explore applicability of grid computing in a few application areas such as hurricane migration, Bioinformatics, and healthcare [23].

9. Acknowledgement

This work was supported in part by IBM, and the National Science Foundation (grants OISE-0730065, OCI-0636031, REU-0552555, and HRD-0317692) and in part by the Spanish Ministry of Science and Technology under contract TIN2007-60625. 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 and Balaji Visanwanthan. Finally, we would like to thank Leonard Reisman for proofreading this paper.

10. References

[1] I. Foster, C. Kesselman, editors, "The Grid: Blueprint for a New Computing Infrastructure", Morgan Kaufmann Publishers 1999.

[2] I. Rodero, J. Corbalan, F. Guim, L.L. Fong, Y. G. Liu, S. M. Sadjadi,. "Looking for an evolution of grid scheduling: Meta-brokering". Proceedings of the Second CoreGRID Workshop on Middleware at ISC2007, Dresden, Germany, June 2007

[3] HPC-Europa Web Site. http://www.hpc-europa.org

[4] GridWay: http://www.gridway.org/

[5] A. Iosup, D.H.J. Epema, T. Tannenbaum, M. Farrelle, M. Livny, "Inter-Operable Grids thorugh Delegated MatchMaking", in proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC07), Reno, Nevada, November 2007.

[6] LA Grid Initiative: http://latinamericangrid.org/

[7] A. Anjomshoaa, M. Drescher, et. al. "Job Submission Description Language (JSDL) Specification"; Ver. 1.0, 2005.

[8] Open Grid Forum. www.gridforum.org

[9] I. Rodero, R.M. Badia, J. Corbalan, J. Labarta, "eNANOS Grid Resource Broker", European Grid Conference'05, LNCS 3470, pp. 111-121, Amsterdam, The Netherlands, February, 2005.

[10] I. Rodero, F. Guim, J. Corbalan, J. Labarta, "eNANOS: Coordinated Scheduling in Grid Environments", Parallel Computing: Current & Future". Issues of High-End Computing, G.R. Joubert et al. (Eds.), Parallel Computing (ParCo 2005), pp. 81-88, Málaga, Spain, September 2005.

[11] IBM Tivoli Dynamci Workload Broker: User's Guide; SG32-2281-01

[12] V. Gucer, J. Biggs-Finstad, et. al. "Getting Started with Tivoli Dynamic Workload Broker 1.1". www.redbooks.ibm.com, SG24-7442-00.

[13] Load Sharing Facility of Platform Computing: http://www.platform.com/Products/Platform.LSF.Family/Platform.LSF/

[14] Distributed Resource Management Application API Specifications http://www.ogf.org/documents/GFD.22.pdf

[15] P. Troger, H. Rajic, A. Haas, P. Domagalski, "Standardization of an API for Distributed Resource Management Systems", in proceedings of CCGrid'07, 2007.

[16] J. Brooke, D. Fellows, K. Garwood, C. Goble, "Semantic Matching of Grid Resource Descriptions", LNCS 3165, January 2004, pp. 240-249.

[17] A. Kertesz, P. Kacsuk, Z. Farkas, T. Kiss, "Grid Interoperability by Multiple Broker Utilization and Meta-Brokering", INGRID2007 - 2nd International Workshop on Distributed Cooperative Laboratories, Italy, April 16-18, 2007.

[18] T. Vazquez, E. Huedo, R.S. Montero, I.M. Lorente, "Evaluation of a Utility Computing Model Based on the Federation of Grid Infrastructures", pp. 372-381, Euro-Par 2007, August 28, 2007.

[19] J. Seidel, O. Waldrich, W. Ziegler, P. Wieder, R. Yahyapour, "Using SLA for resource management and scheduling - a survey", CoreGRID Technical Report TR-0096, Institute on Resource Management and Scheduling, 2007.

[20] A. Kertesz, I. Rodero, F. Guim, "BPDL: A Data Model for Grid Resource Broker Capabilities", CoreGRID TR-0074, Institute on Resource Management and Scheduling, 2007.

[21] A. Kertesz, P. Kacsuk, I. Rodero, F. Guim, J. Corbalan, "Meta-Brokering requirements and research directions in state-of-the-art Grid Resource Management", CoreGRID TR-0116, Institute on Resource Management and Scheduling, 2007.

[22] Simple API for Grid Application Research Group (SAGA-RG) http://forge.ogf.org/sf/projects/saga-rg

[23] R. Badia, G. Dasgupta, O. Ezenwoye, L. Fong, H. Ho, S. Khuri, Y. Liu, S. Luis, A. Praino, J. P. Prost, A. Radwan, S. M. Sadjadi, S. Shivaji, B. Viswanathan, P. Welsh, A. Younis, "Innovative Grid Technologies Applied to Bioinformatics and Hurricane Mitigation". High Performance Computing and Grids in Action, IOS Press - Amsterdam, Lucio Grandinetti editor. Dec 2007.