Broker Selection Strategies in Interoperable Grid Systems

Ivan Rodero 1, Francesc Guim 2, Julita Corbalan 1, Liana Fong 4, S. Masoud Sadjadi 5

† Barcelona Supercomputing Center (BSC-CNS), Spain
* Computer Architecture Department, Technical University of Catalonia (UPC), Spain
1 irodero@ac.upc.edu 2 fgui@ac.upc.edu 3 julia@ac.upc.edu
# IBM T.J. Watson Research Center, Hawthorne, New York, USA
4 llfong@us.ibm.com
5 saddadi@cs.fiu.edu

Abstract—The increasing demand for resources of the high performance computing systems has led to new forms of collaboration of distributed systems such as interoperable grid systems that contain and manage their own resources. While with a single grid domain one of the most important tasks is the selection of the most appropriate set of resources to dispatch a job, in an interoperable grid environment this problem shifts to selecting the most appropriate domain containing the requiring resources for the job. In this paper, we present and evaluate broker selection strategies for interoperable grid systems. They use aggregated resource information as well as dynamic performance information of the underlying scheduling layers. From our evaluations performed with simulation tools, we conclude that aggregation techniques do not penalize performance significantly, and that delegating part of the scheduling responsibilities to the underlying scheduling layers is a good way to balance the load among the different grid systems.

I. INTRODUCTION

The increasing demand for resources in the High Performance Computing (HPC) applications has led to new forms of collaboration of distributed systems. In these new distributed scenarios, such as grid systems, traditional scheduling techniques have evolved into more complex and sophisticated approaches where other factors, such as the heterogeneity of resources or geographical distribution, have been taken into account. Moreover, the need for interoperability among different grid systems has become increased in the last few years. Grid interoperability enables the interoperation among different grid systems through the use of common protocols and standard interfaces. We define an interoperable grid system as the system composed of multiple grids where the interoperability among them has been enabled. We will refer to each of these grid systems as a grid domain. A grid domain may be composed of distributed and potentially heterogeneous resources belonging to different Virtual Organizations [1]. While within a single grid domain one of the most important tasks is the selection of the most appropriate set of resources to dispatch a job, in an interoperable grid environment this problem shifts to selecting the most appropriate grid domain.

Our approach is part of the Latin American Grid (LAGrid) initiative [2], which provides a multi-party collaborative environment that we used to carry out our design and experiment. In our model each grid domain is managed by one grid resource broker that can be viewed as the meta-scheduling functional entity of an institution. This aspect of our model intends to reflect the reality of many organizations having multiple local schedulers for different lines of business or for various levels of services. In our grid model, all domains support a common data aggregation model that enables both encapsulation and sharing of its resources and scheduling details. A Peer-to-Peer relationship between domain brokers is dynamically established upon the agreement between peers. We use the meta-brokering approach [3] in order to perform the job scheduling on top of brokers (broker selection).

In this paper, we describe and evaluate different broker selection strategies. The basic broker selection policy is the bestBrokerRank policy that selects the best broker to submit a job based on available resources information. We also present two different variants of this policy. The first one uses the resources information in aggregated forms as input, and the second one also uses the brokers average bounded slowdown as a dynamic performance metric.

Based on the evaluations performed with our simulation tools, we observe that, although the aggregation algorithms lose resource information accuracy, the broker selection policies using aggregated resource data do not penalize their performance significantly. Moreover, we show that the best performance results are obtained with the coordinated policy using dynamic performance information, in addition to aggregated resource information. Finally, we conclude that delegating part of the scheduling responsibilities to the underlying scheduling levels promotes separation of concerns and is a good way to balance the load among the different grid systems.

The rest of the paper is organized as follows. Section II surveys related work. Sections III and IV present the aggregation algorithms and the broker selection policies respectively. Section V presents the evaluation methodology. Section VI presents the results and, finally, section VII concludes the paper and suggests some directions for future work.

II. RELATED WORK

The need for interoperability among different grid systems was observed and studied previously. On one hand, several projects have attempted the problem of unifying the access to a set of different HPC infrastructures such as HPC-Europa [4],
DEISA [5] and PRACE [6] that aims at forming the top level of the European HPC ecosystem. On the other hand, different initiatives have been started exploring grid interoperability with similar objectives but through different approaches: Inter-Grid [7] promotes interlinking different grid systems through peering agreements in an economic-based approach. Within InterGrid, Assuncao et al. [8] presented the performance evaluation of policies for resource provisioning across grids. GridWay incorporates the support for multiple grids [9] through its grid gateways [10] to access resources belonging to different domains. Within GridWay, Leal et al. [11] presented and evaluated a decentralized model for scheduling on federated grids focused on improving makespan and resource performance. gLite Workload Management Service (WMS) [12] supports large distributed computing environments such as the EGEE infrastructure. The gLite WMS acts as a resource broker and submits a job to a computational resource satisfying the client requirements. The Koala grid scheduler [13] is another grid initiative, which is focused on data and processor co-allocation. They use delegated matchmaking [14] to obtain the matched resources from one of the peer Koala instances. VIOLA MetaScheduling Service [15] implements grid interoperability via WS-Agreement and is providing co-allocation of multiple resources based on reservation. Other approaches have been proposed such as the Grid Interoperability Project (GRIP) [16], the Open Middleware Infrastructure Institute for Europe (OMI-Europe) project [17] or the work done within the P-GRADE portal [18]. Our approach has some similarities with these works in certain points. However, there are two main differences. On one hand, we use aggregated resource data to define ranking expressions to sort potential results for resource selection. On the other hand, we consider multiple grids with different resource management systems through the use of a higher level abstraction layer.

The meta-brokering approach was first proposed by Kertész et al. [19]. Compared to other solutions such as using one broker per user (group) [20] or cross-grid job submission without meta-brokering, it provides uniform interfaces and allows scheduling strategies at a higher scheduling level.

The Grid Scheduling Architecture Research Group (GSA-RG) of the OGF [21] is currently working on enabling the grid scheduler interaction. They are focused on defining a common protocol and interface among schedulers enabling inter-grid resource usage, using standard tools (JSDL, OGSA, and WS-Agreement). However, the group pays more attention to agreements. The OGF Grid Interoperation Now Community Group (GIN-CG) [22] also addresses the problem of grid interoperability driving and verifying interoperation strategies, but they are more focused on infrastructure. The OGF Production Grid Infrastructure Working Group (PGI-WG) [23] aims to formulate a well-defined set of profiles and additional specifications oriented to production grid infrastructures.

There are different resource models for grid systems such as the GLUE schema [24] and the schema provided by UNICORE [25]. In LA Grid, we use a resource model which is an extension of the one used in IBM Tivoli Dynamic Workload Broker (TDWB) [26]. We also consider the resource model in the aggregated form. Aggregation of the resource information is a usual way to save data transfers and has been widely used in different approaches such as in Legion [27], MDS [28], Ganglia [29] or the Virtual Grid Description Language (vGDL). However, the existing approaches to resource aggregation have not been applied to interoperable grid scenarios. Because of the peculiarities of these approaches, we decided to explore our own model and aggregation algorithms in order to optimize our scheduling strategies.

### III. The Resource Aggregation Algorithms

In an interoperable grid system that can be composed by different domains the resource model is crucial. Our solution uses a common resource model among the different domains. Since an interoperable grid systems can be composed by numerous grid domains, the amount of resource information exchanged between brokers is a scalability issue. Therefore, we exchange the resource information in an aggregated form to save the data transferred, the latency time, and communication bandwidth. The problem of using aggregated data is the loss of details related to each resource description. However, this summarized information in the aggregated form can be sufficient for the selection of the best broker to submit a job.

Our resource model is defined by a set of resources similar to other resource models (such as GLUE schema [24]). Moreover, we use a subset of resources that will be useful for the aggregation algorithms. For example, we have the ComputingSystem resource that includes attributes such as the processor vendor, the number of CPUs or the CPU load. The relationships are defined by a type (such as reference or contain) and they have the source resource type and name and the target ones. The details of the model and several examples can be found in [30] and in the IBM Tivoli Dynamic Workload Broker [26]. In our current implementation, we only use a subset of the resources and attributes of the model for simplicity. For example, we do not consider some resources such as Logical System or Agent and resource attributes such as the MinorVersion or SwapSpace of the OperatingSystem resource because they are not required for our aggregation algorithms. We also simplify the relationships considering only one type (the reference type) and we have restricted the order of the related resources. In particular, for defining a computer in aggregated form, it is sufficient to include “reference” relationships between ComputingSystem resources and OperatingSystem ones, and between ComputingSystem resources and FileSystem ones.

We have implemented two different aggregation algorithms. The first one (SIMPLE) aggregates the resource data as much as possible looking for maximum compression for scalability; this algorithm loses more detailed information. It has as input a set of resources and relationships that define computers, and three fixed attributes for aggregating the information: the processor type for ComputingSystem resource, the operating system type for OperatingSystem resource, and the file system type for FileSystem resource. As output it returns a set of
resources in aggregated form and a set of relationships that describe the original resources.

The second algorithm (CATEGORIZED) tries to find a good balance between the accuracy of the resource data and the scalability of the solution. In addition to the list of resources, relationships and fixed categories provided as input it also considers different attributes and threshold values for defining subcategories inside the fixed categories. For example, RamAvailable and CpuUtil are attributes. An example of threshold values is defined as follows: “threshold < 33%”, “33% < threshold < 66%”, and “threshold > 66%”. The subcategories increase the accuracy of the aggregated form. Actually, although in the algorithm used in our evaluation we use three different attributes for subcategories (“CPUUtil”, “FreeMem”, “FreeStorage”), we can use more attributes until the detail information is sufficient. We can increase the level of detail by defining more threshold values (in our experiments we used the three values shown in the previous example). Therefore, as we have commented previously, the main purpose of this algorithm is to avoid the loss of important resource characteristics but maintaining the benefits of aggregation. For example, when we select a broker that contains an aggregated resource of Intel processor vendor and CPULoad attribute of subcategory LOW, we can be sure that some Intel-based computers with low CPU load will be available. Although the information is not as accurate as the regular model, the number of aggregated resources and relationships is higher than the one obtained with the SIMPLE algorithm. Therefore, the precision of resource information is better in the CATEGORIZED algorithm. Moreover, the level of accuracy can be improved by increasing the number of categories and/or subcategories. More details of the aggregation algorithms and some examples can be found in [31].

IV. THE BROKER SELECTION POLICIES

In this section, we present the bestBrokerRank policy that selects the best broker to submit a job in an interoperable grid scenario. In particular, given a set of job requirements and the resource information from different brokers, it returns the broker that best matches with these requirements. We first consider the accumulated rank value of a regular matching algorithm on the resources of each broker domain. Then, we also incorporate additional considerations such as promoting the job originator domain or giving dynamic priorities to the different brokers depending on the performance that they are achieving. The most important input parameters are the job requirements and the brokers resources. Job requirements are, for example, the processor vendor (i.e., Intel, AMD) or the operating system (i.e., Linux, AIX). Our optimization function also considers soft requirements (also known as recommendations), such as the recommended percentage of free virtual memory. Eventually, we define a set of weighting factors to be applied to different resource characteristics (i.e., CPU speed factor=4, free CPUs factor=50, running jobs factor=100, etc.).

The general bestBrokerRank policy is briefly described as follows.

- Given a job, its resource requirements, the list of available brokers, the brokers priorities, the resources and relationships that describe the brokers computers, and a set of attribute weighting factors:
- For each broker:
  - Matches the resource requirements
  - If the current broker does not match the resource requirements, it is not considered
  - Otherwise, computes the current broker rank value (based on matchmaking) given the broker resources/relationships, attributes weighting factors and broker priority
- Selects the broker with the maximum rank value

After selecting a broker, the selection of the local resources to dispatch the job is the responsibility of the broker following the policies established under its domain. However, we note that part of the bestBrokerRank function implementation can be re-used to select the local resource(s) to dispatch the job because it computes the rank values of the resources in the local domain.

We evaluate two different variants of the bestBrokerRank policy. In the first one (bestBrokerRank\_AGGR policy), the resources are defined in aggregated form. We also implemented the two different resource aggregation algorithms, namely, SIMPLE and CATEGORIZED. The main differences between these two variants are the input parameters and the implementation of the function that compute the rank values. Since the resource data is expressed in aggregated form, the actual information may differ significantly from the aggregated form. For example “NumOfProcessors=(1-4, <count=15>, <total=31>)” means 15 computers with a total of 31 CPUs having from 1 to 4 CPUs per computer, rather than, for example, “3 computers with single CPU, 10 computers with 2 CPUs, and two with 4 CPUs”. Thus, when we consider the aggregated form the resource information is significantly less accurate. Consequently, to compute the rank values from the aggregated data we take 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 to a non-optimal broker selection decisions. Therefore, the algorithm may unintentionally decide to submit a job to a broker with insufficient resources when another broker is able to dispatch the job immediately. In our evaluations we will assume that provided resource information is always updated and correct.

The second variant (bestBrokerRank\_SLOW policy) also uses the aggregated resource form but it coordinates with the brokering layer taking the broker average slowdown metric as the main characteristic in the matching optimization function. We define “broker average slowdown” as the mean of the average bounded slowdown of its resources. The average bounded slowdown of the resources is computed from its finished jobs. This is a way of using information from the
underlying scheduling levels instead of local scheduling policies to balance the load. Moreover, the resource matching is performed in a more relaxed manner. It means that the algorithm considers less job requirements attributes in the matching process. For example, the selected domain must contain at least a machine or a set of machines with enough CPUs to allocate the job but it is not mandatory to have these CPUs available at the submission moment. The best-BrokerRank\_SLOW policy uses the CATEGORIZED resource aggregation algorithm because, as we will show in later sections, this algorithm provides a good tradeoff between scalability and resource information accuracy. This policy can be seen as a subset of the bestBrokerRank\_AGGR policy but giving more priority to the resources that are achieving better performance. Actually, this is a way to balance the load among the different brokers so balancing the different grid systems performance. Since the broker slowdown is the main attribute for matching the brokers rather than the resource information, the underlying scheduling levels (local resource management systems) are more priority in the job scheduling process. Thus, jobs can be potentially queued in the local systems for longer time. In fact, with the previous policy the average broker slowdown is better because the jobs are only submitted to local systems when they can be allocated immediately; if not, they are queued at the brokering level. Consequently, from the point of view of the local resources this policy may obtain worse performance results but it may improve the global system performance.

V. EVALUATION METHODOLOGY

We have used simulation mechanisms for our policy evaluations. We used the Alvio Simulator [32], which is an event driven simulator, that has been designed and developed to evaluate scheduling policies in HPC architectures. It supports evaluation of schedulers in a large range of facilities from local centers to interoperable grid environments simultaneously. More information concerning the Alvio simulation framework can be found in [32]. The Alvio interoperable grid models can be also found in [33].

In our evaluation, we have used traces of well-known real grid systems (DAS-2, Grid5000, and Sharcnet) from the Grid Workloads Archive [34]. We have selected two weeks of job submissions for each workload trace. We have avoided the initial start up period of the systems (around the first 50,000 jobs) to skip unrepresentative data. The selected trace fragments are sized with 11,318 jobs for DAS-2, 12,719 jobs for Grid5000, and 13,283 for Sharcnet. Thereby, when we evaluate the interoperable system with up to 18 domains, we are considering around a quarter million jobs, 180 clusters, and more than 30,000 available processors. Since in our approach we assume grids composed of HPC resources that can accommodate both local and grid jobs, the load of these systems may be very high. Therefore, in order to evaluate our strategies in more loaded systems we have reduced the inter arrivals times of the jobs (by a factor of less than two) allowing us to increase the pressure on the system load. More details regarding these traces can be found in [34]. We have defined three different domain types, one for each workload system. The resources of each domain are based on real testbeds in terms of number of clusters, CPU architectures, and OSs. For the DAS-2 system, we have modeled 6 resources with a total of 400 CPUs, 12 resources with a total of 985 CPUs for the Grid5000 system, and 12 resources with a total of 3,712 CPUs for the Sharcnet system. We have chosen a subset of the CPU available architectures (Intel, AMD, and PowerPC) and Operating Systems (Linux, AIX, and Solaris). While DAS-2 is a homogeneous multi-cluster (only has Intel and Linux), the other two systems are more heterogeneous in term of number of CPUs, architectures and OSs (for example, Grid5000 has 50% Intel, 40% AMD and 10% PowerPC, and 60% Linux, 30% AIX and 10% Solaris). Since the original traces do not include resource requirements for each job, we generated a synthetic trace with the jobs requirements based on the original trace, the resources characteristics, and a combination of input parameters. In particular, we have used the CPU and memory demand, and for the disk utilization we have used a combination of the job duration with the CPU and memory usage, with a randomized factor. For the remaining attributes we have used percentages for each CPU architecture and OS type, applying a random distribution by bursts (sized from 3 to 6).

We have used the following metrics for evaluating our strategies: total workload execution time, average job waiting time, average bounded slowdown (BSLD) where $BSLD_{job} = max \left( \frac{\text{runtime}_{job} + \text{waittime}_{job}}{\max(\text{runtime}_{job}, \text{threshold})}, \text{threshold} \right)$ with threshold = 60 seconds, and average CPU utilization. Although a threshold of 10 seconds is used for BSLD in many works in the context of parallel job scheduling to limit the influence of very short jobs on the average BSLD, we define a threshold of 60 seconds because in grid scenarios jobs may take longer. The units for the first two metrics are seconds, for average CPU utilization are percentages, and the slowdown has no units. In our experiments, we try to minimize all the metrics except the average CPU utilization that should be maximized.

VI. RESULTS

In this section, we show the results obtained from experiments described in the previous section. In the simulations, we have evaluated the algorithms and scheduling strategies that we have presented in the first part of the paper. Firstly, we study if the aggregation algorithms can scale to large interoperable grid systems with several domains and thousands of resources. Afterwards, we present the performance evaluation of the different variants of the best BrokerRank broker selection policy, using the aggregation algorithms and being coordinated with the underlying scheduling level.

A. Scalability of the resource data aggregation algorithms

In order to evaluate the scalability of the two proposed aggregation algorithms, we have performed the experiments with different number of resources, from 10 to 10,000 HPC computing systems (that may be composed of several nodes).
The experiments were conducted by executing a Java program that implements the aggregation algorithms on a commodity computer. As input the program receives a file that includes a set of attribute values that define the computers. Firstly, the program transforms these attributes to our regular resource model. Afterwards, it applies an aggregation algorithm and returns the aggregated resource data and computes the processing time. For each experiment, we repeat this process with 5 different input files and we compute the average value. The input files with the computers definition are generated by a perl script that generates the computers definition randomly from a set of parameters.

Figure 1 shows the obtained results in logarithmic scale. Figure 1a shows the number of entries (of our resource model) used for describing the resources. It shows the results with regular resource model (Original) and the two aggregation algorithms (Simple and Categorized). With the number of relationships used for the same resources, the results follow the same pattern. In general, while the number of entries increases in a linear manner in the regular resource form, in both aggregation algorithms the number of entries is almost constant. In particular, for up to 100 HPC systems the number of entries for the aggregated forms is around 10 times lower than the regular one. For more than 100 HPC systems the number of entries in aggregated form is up to 1,000 times lower than the number of entries in the regular one. Although both aggregation algorithms follow the same pattern, the number of entries in the Categorized algorithm is around 10 times larger than in the Simple algorithm. Figure 1b shows the size of the resource information, including the attributes and their values. The pattern that it follows is similar to those described previously for the number of resources. This is explained due to the fact that the size of the resource information is proportional to the number of resource and relationship entries that it contains. However, in this case the difference between the two algorithms is smaller. The size with the regular form is around 100 times larger than with the aggregated forms, and the size with the Categorized algorithm is around six times larger than with the Simple algorithm. Figure 1c shows the processing time required for both aggregation algorithms. With 100 HPC systems or less the execution time of both algorithms is very similar. For more than 100 HPC systems the execution time of the Categorized algorithm is longer than the execution time of the Simple algorithm and grows almost linearly. However, the execution time difference between the two aggregation algorithms is not very large.

We conclude that the two aggregation algorithms are scalable in terms of resource information size, the aggregation algorithms processing time is acceptable for an interoperable grid environment (up to 10,000 HPC systems). With the Categorized algorithm the execution time is longer than with the Simple algorithm, and the size of the resource information with the Categorized algorithm is also larger than with the Simple algorithm. However, the accuracy of the resource data is much better with the Categorized algorithm. Actually, we observed tradeoffs between the regular and the aggregated resource models. On one hand, with the regular resource model the resource data is more accurate and it is provided faster. However, the scheduling algorithm processing time is longer. The network delay required to transfer the resource data between meta-schedulers is longer. On the other hand, the resource data in aggregated form is less accurate and the resource aggregation algorithm requires additional processing time. However, since the size of the data is smaller, both the delay to transfer the data and the delay to evaluate the scheduling algorithm are shorter.

---

![Figure 1: Aggregation algorithms evaluation results](image-url)
B. Performance results

We have defined different grid interoperable scenarios with different number of grid domains (DOM in figures), from 3 to 18, in order to evaluate the performance of the different broker selection strategies. We have taken as a reference the original bestBrokerRank policy (REGULAR), and we have compared it to three different variants: AGGR SIMP and AGGR CAT that use the Simple and Categorized aggregation algorithms respectively. SLOW is the bestBrokerRank SLOW policy. Figure 2 shows the performance and resource utilization results obtained with the different policies in the different scenarios using different combinations of the three workloads (DAS-2, Grid5000, and Sharcnet): 3 DOMs (one instance of each workload), 6 DOMs (two instances of each workload), etc. They are normalized to the policy that shows lower performance. Figure 2a shows the relative workload execution time. Figure 2b shows the average job bounded slowdown. Figures 2c and 2d show the percentage of forwarded jobs to another grid domain and re-scheduled jobs within a grid domain respectively. A re-scheduling is performed when there is no resource in the system that matches the job requirements. It is worth noting that in the first two figures the maximum difference between two policies are less than 14%. We note that in each figure the worst result is obtained by the AGGR SIMP policy (value equal to 1). We consider the processing time of the matchmaking algorithms and an estimation of the resource information transfer delays between brokers. They introduce...
around 1-2% of difference in the results. Although some of the obtained results are very close, all of the policies have been evaluated using the same framework and we have performed different executions in order to prevent variances in the results that, eventually, were not detected. Figure 2e shows the average resource utilization and Figure 2f shows the standard deviation that gives us some hints about the load balancing of the different brokers: smaller results show better load balance. With the SLOW policy the load balancing is better and more constant with the different configurations.

The results show that the performance metrics results are degraded when the number of domains increases. However, this degradation is not drastic, and the differences between the REGULAR policy and the rest of policies are not very large. In general, in Figure 2 the performance results with the AGGR_CAT policy are better than the results with the AGGR_SIMP policy (for example, 5% concerning the bounded slowdown). As with the rest of policies, the workload execution time and average bounded slowdown increase when the number of domains increases, especially with 18 domains. The execution time increases up to 5% with the AGGR_SIMP policy and the average bounded slowdown increases up to 13% with the AGGR_SIMP policy. AGGR_SIMP and AGGR_CAT policies also show less performance than the REGULAR policy with every metric (for example up to 22% of forwarded jobs). The AGGR_SIMP, AGGR_CAT, and REGULAR policies follow similar patterns in the execution time, average bounded slowdown, and percentage of forwarded jobs. However, while the percentage of re-scheduled jobs with the REGULAR policy is almost constant, with AGGR_SIMP and AGGR_CAT policies, it increases significantly with nine and 18 domains. The performance results with the SLOW policy are better than the results with the REGULAR policy in most of the cases. The execution time and the average bounded slowdown are better on average, the percentage of forwarded jobs is similar with both policies, and the percentage of re-scheduled jobs is somewhat better (less than 2% on average). As the previously discussed AGGR_SIMP and AGGR_CAT policies, the workload execution time and average bounded slowdown increase when the number of domains increases. However, the execution time and average bounded slowdown increase is not very large (less than 1%). Moreover, the SLOW policy follows a more constant pattern than the REGULAR policy in terms of execution time and average bounded slowdown. With 18 domains the amount of resource information that the REGULAR policy manages penalizes the performance results. The percentage of re-scheduled jobs pattern with the SLOW policy is different to the other policies. The percentage slightly decreases when the number of domains increases. This behavior is due to the fact that, with the SLOW policy, part of the responsibility of the job scheduling is delegated to the local schedulers. Thus, the jobs are queued in the local schedulers rather than being re-scheduled at the broker layer.

The average resource utilization with AGGR_SIMP and AGGR_CAT policies is significantly lower than with the REGULAR policy (around 5% on average). However, they follow the same pattern: the resource utilization decreases when the number of domains increases. Both AGGR_SIMP and AGGR_CAT policies have very similar results. However, with the AGGR_SIMP policy the resource utilization has a notorious decrease with 18 domains. The standard deviation (STDEV) with AGGR_SIMP and AGGR_CAT policies is larger than with the REGULAR policy (around 30% on average with respect to the REGULAR policy). The STDEV is especially larger with the AGGR_SIMP policy. The difference between the standard deviation with AGGR_SIMP and AGGR_CAT policies is around 10% on average. Moreover, STDEV increases with every policy when the number of domains increases. It indicates that with AGGR_SIMP and AGGR_CAT policies the load is worse balanced among the brokers than with the REGULAR policy, especially with the AGGR_SIMP policy. The average resource utilization with the SLOW policy is larger (4% on average) than with the REGULAR policy. However, they follow the same pattern: the resource utilization decreases when the number of domains increases. The STDEV with the SLOW policy is 10% smaller than with the REGULAR policy. However, they follow very similar patterns. This indicates that with the SLOW policy the load is better balanced among the brokers than with the REGULAR policy.

Therefore, we can state that the AGGR_CAT policy obtains better results than AGGR_SIMP. Moreover, with both policies that only use aggregated resource information the results are worse than with the REGULAR policy. However, the difference in the performance is not very large while aggregation provides a solution to the resource information exchange scalability problem. We also state that the SLOW policy obtains the best results. Furthermore, as well as providing a solution to the resource information exchange scalability problem, it improves the performance results coordinating the scheduling with the underlying scheduling layers in order to balance the performance of the different grid domains.

VII. CONCLUSIONS AND FUTURE WORK

In this paper, we have addressed the problem of broker selection in interoperable grid scenarios. We have described and evaluated the bestBrokerRank policy and two variants of this policy: one using resource information in aggregated form, and the other coordinating the scheduling with the underlying levels. We also have presented two different resource aggregation algorithms that have been used by our broker selection policies.

The experimental evaluation results show that the resource aggregation algorithms are scalable (up to 10,000 HPC Systems) and that their aggregation processing times are acceptable for interoperable grid systems. We also observed the tradeoffs between different possibilities. We have evaluated the performance of the proposed broker selection policies with different levels of resource aggregation. As an example, we have shown that the best results were obtained with the broker selection strategy that considers performance information from the underlying
scheduling levels in addition to resource aggregation techniques. Moreover, the other broker selection policies that use aggregated resource information obtained worse results than the reference policy. However, the difference between the different techniques results is not very large. Through the study of the resource utilization results we claim that using performance information from the underlying scheduling levels is a good way to balance the load among the brokers. Therefore, we conclude that delegating part of the scheduling responsibilities to the underlying scheduling levels is a good way to balance the load among different grid systems.

There are different lines of work that we plan to address in the near future. On one hand, we are targeted to include Peer-to-Peer details in our models to improve the simulations. We will also add new features to our negotiation protocol that was presented in [35]. On the other hand, we plan to validate the results of our broker selection strategies in a real scenario with real applications. We will use the LA Grid infrastructure with HPC applications such as the Weather Research and Forecasting (WRF) [36]. Actually, we are already working on emulation mechanisms that simulate the execution times of the local systems but consider the real infrastructure and protocols (with the network latencies, bandwidths, overhead, etc.).

VIII. ACKNOWLEDGEMENTS

This paper has been supported by the Spanish Ministry of Science and Education under contract TIN200760625C0201. This work is also part of the Latin American (LA Grid) project and in part was supported by IBM and NSF.

REFERENCES