# Evaluation of Broker Selection Strategies

Ivan Rodero [†1], Francesc Guim [†2], Julita Corbalan [†3]
Liana Fong [#4], S. Masoud Sadjadi [‡5]

[†] *Technical University of Catalonia (UPC), Spain*
[1] irodero@ac.upc.edu [2] fguim@ac.upc.edu [3] juli@ac.upc.edu

[#] *IBM T.J. Watson Research Center, Hawthorne, New York, USA*
[4] llfong@us.ibm.com

[‡] *Florida International University (FIU), Miami, Florida, USA*
[5] sadjadi@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 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 the Latin American Grid initiative, our model consists of multiple domains. Each domain has its domain broker, and the task of scheduling on top of brokers can be called meta-brokering or broker selection.

In this paper, we describe and evaluate our broker selection strategies. In particular, we present and evaluate the "bestBrokerRank" policy and two different variants. The first one uses the resource information in aggregated forms as input, and the second one also uses the brokers average bounded slowdown as a dynamic performance metric. From our evaluations performed with simulation tools, we firstly state that the interoperable grid scenario is better compared to the independent brokering one in terms of execution time and resource utilization. We also show that the proposed resource aggregation algorithms are scalable for an interoperable grid environment. Moreover, we show that the best performance results are obtained with our coordinated policy. Therefore, we conclude that delegating part of the scheduling responsibilities to the underlying scheduling layers promotes separation of concerns and is a good way to balance the performance among the different brokers and schedulers.

# 1 Introduction

Job scheduling strategies have been extensively studied in the last decades. The increasing demand for resources of the High Performance Computing (HPC) systems 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. These grid systems are composed of several Virtual Organizations (VO) [1], sharing resources that span one or more administration domain(s); in turn each domain is represented by one grid resource broker which acts as the gateway to that domain. Finally, each administration domain is composed of a set of different resources that are managed by their own job schedulers (e.g., PBS [2], Loadleveler [3], SGE [4], etc.), each of which may have its own local policies.

While within a single 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 computing environment this problem shifts to selecting the most appropriate domain. In fact, there are two possibilities: dispatching the job to the local resources of the originator domain or forwarding it to another domain. Since each domain is typically managed by one broker, the task of scheduling on top of brokers can be called *meta-brokering* or *broker selection*.

In [5], we presented a study of the requirements for the meta-brokering approach. We also introduced the Latin American Grid (LA Grid) initiative [6], which provides a multi-party collaborative environment that we used to carry out our design and experiment. Our grid model consists of multiple domains. Each domain has its domain broker and consists of a collection of local dispatchers, local schedulers, or even meta-schedulers. A domain 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 the encapsulation and sharing of its resources and scheduling details. A peer-to-peer (P2P) relationship between domain brokers is dynamically established upon the agreement between peers. Users of a domain would interact with that specific domain broker to access resources of collaborative partners. In [7], we presented our protocol, and the functionality results. Other initiatives have been proposed for interoperating meta-scheduling systems such as GRIP [8], HPC-Europa [9], Gridway [10], Koala [11], or Viola [12]. However, none of these approaches provide a comprehensive solution to broker selection policies.

In [13], we presented the eNANOS architecture that considers all the scheduling layers that can be involved in a job execution all the way from the local resource scheduling to brokering the interoperable grid systems. The key contribution of this paper is the coordination between the different scheduling entities. In this paper, we use the coordination philosophy of eNANOS to develop broker selection strategies.

In this paper, we describe and evaluate our broker selection strategies. The basic broker selection policy is the *bestBrokerRank* policy that selects the best broker to submit a job based on available resource information. In case the job is from the local domain, the resource match maker uses this policy and returns an appropriate resource(s) to execute the job. Otherwise, the resource match maker selects an appropriate broker and returns the broker ID instead to which the job will be forwarded. We also present two different variants of this policy. The first one uses the resource information as input to ranking. The resource information is in aggregated forms; details are presented in later sections. The second one is based on the coordination with the underlying scheduling level using the brokers average bounded slowdown as a dynamic performance metric.

To evaluate the performance of our policies, we extended the Alvio simulator [14] to suit our need. The Alvio simulator allows the researcher to evaluate from local data centers to interoperable grid environments simultaneously. To do this, we have extended the existing Alvio models to include the characteristics of interoperable grid scenarios.Based on our evaluations, we firstly state that the interoperable grid scenario is better compared to the independent brokering in terms of workloads execution performance and resource utilization. We also show that the resource aggregation algorithms are scalable in terms of resource information size, and their aggregation processing time are acceptable for an interoperable grid environment. As explained later, although the aggregation algorithms lose resource information accuracy, we show that the broker selection policies using aggregated resource data do not penalize the global performance significantly. Moreover, we show that the best performance results are obtained with the coordinated policy using the brokers average bounded slowdown, in addition to aggregated resource information. In fact, we state that delegating part of the scheduling responsibilities to the underlying scheduling layers is a good way to balance the performance among the different brokers and schedulers. We note that it is more difficult to balance the performance among the grid domains when the number of domains increases. Finally, we conclude that the interoperable grid scenario introduced in this paper complies with the requirements of the infrastructure for LA Grid project.

The rest of the paper is organized as follows. Section 2 surveys related work. Sections 3 and 4 present the aggregation algorithms and the broker selection policies, respectively. Section 5 presents the evaluation methodology. Section 6 presents the results and, finally, section 7 concludes the paper and suggests some directions for future work.

## 2 Related Work

The need for interoperability among different grid systems in different resource domains was observed and studied previously, and some projects have addressed this topic such as GRIP [8] and HPC-Europa SPA [9]. Lately, some initiatives have been started exploring grid interoperability with similar objectives but through different approaches. The two main approaches for grid interoperability are: 1) extending the existing schedulers to make them interoperable; and 2) using a meta-broker to represent each domain that can be connected to existing schedulers in its domain without modifying them.

GridWay has incorporated the support for multiple grids in its last release [15]. In the meta-scheduling layers, GridWay instances can communicate and interact *implicitly* through its grid gateways to access resources belonging to different domains. The basic idea is to forward user requests to another domain when the current one is overloaded, but there is no support for direct (or explicit) interaction among the GridWay instances to negotiative on global and local policies and to recieve feedback on delegated actions. The basic idea is to forward user requests to another domain when the current one is overloaded. GridWay is based on Globus, and they are experimenting with GT4 and gLite3. The Koala grid scheduler [11] 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 Grid5000. To inter-connect 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 [16] to obtain the matched resources from one of the peer Koala instances. VIOLA MetaScheduling Service [12] is implementing Grid interoperability via WS-Agreement and is providing co-allocation of multiple resources based on reservation. Although the previously introduced projects aim to enable communication among different broker instances, they are very specific. They only consider instances of the same brokering system and perform job forwarding among brokers only in very limited circumstances. Moreover, the policies for selecting the broker to forward a job are still preliminary.

The Grid Scheduling Architecture Research Group (GSA-RG) of Open Grid Forum (OGF) [17] is currently working on enabling the 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). However, 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.). Following a similar idea, in [18] [19] authors proposed a Broker Property Description Language (BPDL) to be used to perform the

broker selection. A preliminary evaluation of this approach can be found in [20]. Finally, the Grid Interoperation Now Community Group (GIN-CG) of the OGF [21] 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. Performance and optimization strategies are not the current focus.

There are different resource models for grid systems. One of the most well known models is the GLUE schema [22] 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 [23] (EU) and iVDGL [24] (US) and received contributions from DataGrid [25], Globus [26], PPDG [27] and GriPhyn [28]. 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). Another schema is provided by the UNICORE framework [29].

The grid interoperability initiatives are not numerous and, to the best of our knowledge, all of them 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) [30]. 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 [31]. Grid resource management systems have used aggregation mechanisms previously such as in Legion [32]. 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 [33] or Ganglia [34] 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 interoperable 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.

## 3 The Resource Aggregation Algorithms

In an interoperable grid system that can be composed of different domains the resource model is crucial. Our solution uses a common resource model among the different domains. As we have commented in the related work section, to the best of our knowledge, the other related projects to grid interoperability also use a common resource model for interchanging information between grids. Actually, the GSA-RG of the OGF that is currently working on establishing grid interoperability recommendations, pointed out this issue in its public working documents [17]. Moreover, since the interoperable grid systems can be composed of numerous domains, the amount of resource information exchanged between brokers is a scalability issue. Therefore, we interchange the resource information in an aggregated form to save the data transferred, the latency time, and communication bandwidth. The problem of aggregated data is the loss of details related to each resource description. However, this summarized information in the aggregated form is 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 [22]), but we also include relationships between the resources. 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 can be found in [35] and in the IBM Tivoli Dynamic Workload Broker [30]. 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 the aggregation algorithms. We also simplify the relationships considering only 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.

An example of a set of resources defined with the regular resource model is shown in listing 1. We will use the same set of resources for the aggregation algorithms examples in listings 3 and 5. In all examples the resource type of the relationships are abbreviated for readability (i.e., "CS" instead of "ComputingSystem").

In this section, we present 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

```
HOST 1
Resource: Type="ComputingSystem", Name="CS_host1"        Resource: Type="OperatingSystem",
      Name="OS_host1"
          Attrs:  vendor="Intel"                                  Attrs: OSName="Linux"
                  clockSpeed="3,000"                                     RamSize="8,000"
                  CPUUtil="80" (20% avg)                                 RamAvailable="1,600" (20%)
                  totalCPUs="4"                          Resource: Type="FileSystem", Name="FS_host1"
                                                                  Attrs: RootPath="/"
                                                                         sizeMB="120,000"
                                                                         freeMB="96,000" (80%)
Relationship: SourceType="CS", SourceName="CS_host1", TargetType="OS", TargetName="OS_host1"
Relationship: SourceType="CS", SourceName="CS_host1", TargetType="FS", TargetName="FS_host1"
HOST 2
Resource: Type="ComputingSystem", Name="CS_host2"        Resource: Type="OperatingSystem",
      Name="OS_host2"
          Attrs:  vendor="Intel"                                  Attrs: OSName="Linux"
                  clockSpeed="2,600"                                     RamSize="4,000"
                  CPUUtil="100" (50% avg)                                RamAvailable="400" (10%)
                  totalCPUs="2"                          Resource: Type="FileSystem", Name="FS_host2"
                                                                  Attrs: RootPath="/"
                                                                         sizeMB="100,000"
                                                                         freeMB="60,000" (60%)
Relationship: SourceType="CS", SourceName="CS_host2", TargetType="OS", TargetName="OS_host2"
Relationship: SourceType="CS", SourceName="CS_host2", TargetType="FS", TargetName="FS_host2"
HOST 3
Resource: Type="ComputingSystem", Name="CS_host3"        Resource: Type="OperatingSystem",
      Name="OS_host3"
          Attrs:  vendor="Intel"                                  Attrs: OSName="AIX"
                  clockSpeed="2,800"                                     RamSize="16,000"
                  CPUUtil="720" (90% avg)                                RamAvailable="3,200" (20%)
                  totalCPUs="8"                          Resource: Type="FileSystem", Name="FS_host3"
                                                                  Attrs: RootPath="/"
                                                                         sizeMB="120,000"
                                                                         freeMB="48,000" (40%)
Relationship: SourceType="CS", SourceName="CS_host3", TargetType="OS", TargetName="OS_host3"
Relationship: SourceType="CS", SourceName="CS_host3", TargetType="FS", TargetName="FS_host3"
HOST 4
Resource: Type="ComputingSystem", Name="CS_host4"        Resource: Type="OperatingSystem",
      Name="OS_host4"
          Attrs:  vendor="Intel"                                  Attrs: OSName="AIX"
                  clockSpeed="2,600"                                     RamSize="64,000"
                  CPUUtil="1,280" (80% avg)                              RamAvailable="6,400" (10%)
                  totalCPUs="16"                         Resource: Type="FileSystem", Name="FS_host4"
                                                                  Attrs: RootPath="/"
                                                                         sizeMB="200,000"
                                                                         freeMB="100,000" (50%)
Relationship: SourceType="CS", SourceName="CS_host4", TargetType="OS", TargetName="OS_host4"
Relationship: SourceType="CS", SourceName="CS_host4", TargetType="FS", TargetName="FS_host4"
HOST 5
Resource: Type="ComputingSystem", Name="CS_host5"        Resource: Type="OperatingSystem",
      Name="OS_host5"
          Attrs:  vendor="AMD"                                    Attrs: OSName="Linux"
                  clockSpeed="2,200"                                     RamSize="16,000"
                  CPUUtil="160" (40% avg)                                RamAvailable="9,600" (60%)
                  totalCPUs="4"                          Resource: Type="FileSystem", Name="FS_host5"
                                                                  Attrs: RootPath="/"
                                                                         sizeMB="120,000"
                                                                         freeMB="60,000" (50%)
Relationship: SourceType="CS", SourceName="CS_host5", TargetType="OS", TargetName="OS_host5"
Relationship: SourceType="CS", SourceName="CS_host5", TargetType="FS", TargetName="FS_host5"
HOST 6
Resource: Type="ComputingSystem", Name="CS_host6"        Resource: Type="OperatingSystem",
      Name="OS_host6"
          Attrs:  vendor="AMD"                                    Attrs: OSName="Linux"
                  clockSpeed="2,400"                                     RamSize="128,000"
                  CPUUtil="800" (50% avg)                                RamAvailable="32,000" (25%)
                  totalCPUs="16"                         Resource: Type="FileSystem", Name="FS_host6"
                                                                  Attrs: RootPath="/"
                                                                         sizeMB="250,000"
                                                                         freeMB="100,000" (40%)
Relationship: SourceType="CS", SourceName="CS_host6", TargetType="OS", TargetName="OS_host6"
Relationship: SourceType="CS", SourceName="CS_host6", TargetType="FS", TargetName="FS_host6"

                                                TOTAL: 18 resources and 12 relationships
```

Listing 1: Set of computers described in the regular resource model

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 pseudo-code of the algorithm is shown in listing 2. The example of listing 3 shows the same resources of listing 1 but aggregated with *SIMPLE* algorithm.

**ComputeAggregatedInfo** computes the information that contains a resource in aggregated form. Examples are the number of resources that have been aggregated in the same category (*count*), the minimum and maximum values, and the sum of all values (*total*). The information contained in an aggregated resource may differ depending on the kind of resource. More statistical metrics can be included if it is required. **FindAggregatedResource** returns the aggregated resource that corresponds to the given resource in the original form (the one that matches the aggregation criteria).

```
FUNCTION:  getAggregatedData_SIMPLE

IN:

  CATEGORIES = {ProcType, OSType, FSType}
  RESOURCES = {r_1,...,r_n}
  RELS = {rel_1,...,rel_m}, ∀i=0,...,m : rel_i = {source,target} ∧ source,target ∈ RESOURCES

OUT:

  AGGR_RESOURCES = {ar_1,...,ar_k}
  AGGR_RELS = {arel_1,...,arel_p}, ∀i=0,...,p : arel_i = {source,target}
              ∧ source,target ∈ AGGR_RESOURCES

BEGIN:

  FOR k = 1 TO CATEGORIES.size() {
    FOREACH r_i ∈ RESOURCES ∧ category(r_i)==CATEGORIES_k {
      ar_k = computeAggregatedInfo (ar_k, r_i)
      FOREACH rel_j ∈ RELS ∧ rel_j.source==r_i {
        AGGR_RELS.insert(ar_k, rel_j.target) /* Avoiding repeated instances */
      }
    }
  }
  FOR i = 1 TO AGGR_RELS.size() {
    rel_i.target = findAggregatedResource (AGGR_RESOURCES, rel_i.target)
  }

RETURN AGGR_RESOURCES, AGGR_RELS
```

Listing 2: SIMPLE resource aggregation algorithm pseudo-code

```
Resource: Type="ComputingSystem", Name="CS_Intel"
          Attrs:  ProcessorType="{(Intel,<count=4>)}"
                  ProcessingSpeed="{(2600-3000,<count=4>,<total=11000>)}"
                  CPUUtilization="{(80-1280,<count=4>,<total=2180>)}"
                  NumOfProcessors="{(2-16,<count=4>,<total=30>)}"
Resource: Type="ComputingSystem", Name="CS_AMD"
          Attrs:  ProcessorType="{(AMD,<count=2>)}"
                  ProcessingSpeed="{(2200-2400,<count=2>,<total=4600>)}"
                  CPUUtilization="{(160-800,<count=2>,<total=960>)}"
                  NumOfProcessors="{(4-16,<count=2>,<total=20>)}"
Resource: Type="OperatingSystem", Name="OS_Linux"
          Attrs:  OperatingSystemType="{(Linux,<count=4>)}"
                  TotalPhysicalMemory="{(4000-128000,<count=4>,<total=156000>)}"
                  FreePhysicalMemory="{(400-32000,<count=4>,<total=43600>)}"
Resource: Type="OperatingSystem", Name="OS_AIX"
          Attrs:  OperatingSystemType="{(AIX,<count=2>)}"
                  TotalPhysicalMemory="{(16000-64000,<count=2>,<total=80000>)}"
                  FreePhysicalMemory="{(3200-6400,<count=2>,<total=9600>)}"
Resource: Type="FileSystem", Name="FS_unique"
          Attrs:  TotalStorageCapacity="{(100000-250000,<count=6>,<total=910000>)}"
                  FreeStorageCapacity="{(48000-100000,<count=6>,<total=464000>)}"

Relationship: SourceType="CS", SourceName="CS_Intel", TargetType="OS", TargetName="OS_Linux"
Relationship: SourceType="CS", SourceName="CS_Intel", TargetType="OS", TargetName="OS_AIX"
Relationship: SourceType="CS", SourceName="CS_AMD",   TargetType="OS", TargetName="OS_Linux"
Relationship: SourceType="CS", SourceName="CS_Intel", TargetType="FS", TargetName="FS_unique"
Relationship: SourceType="CS", SourceName="CS_AMD",   TargetType="FS", TargetName="FS_unique"

                                                TOTAL: 5 resources and 5 relationships
```

Listing 3: Same set of computers with the "SIMPLE" algorithm

The second algorithm (**CATEGORIZED**) tries to find a good balance between the accuracy of the resource data and the scalability. In addition to the input set of resources, relationships and fixed categories, it also considers different attributes and threshold values. These attributes and thresholds define subcategories inside the fixed categories. The subcategories increase the accuracy of the aggregated form. Actually, although in the algorithm shown in listing 4 we use three different attributes for subcategories, we can use more attributes until the detail information is sufficient. We can increase the level of detail by defining more threshold values. 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.

**GetAttributeValue** returns the discrete value of an attribute given a set of thresholds that define the discrete categories. **CleanVector** deletes the elements of a vector that are empty because there are not aggregated resources corresponding to certain categories and thresholds values. **ComputeAggregatedInfo** and **findAggregatedResource** functions were explained previously.

```
FUNCTION:  getAggregatedData_CATEGORIZED

IN:

  CATEGORIES = {ProcType, OSType, FSType}
  ATTRIBUTES = {%CPULoad, %UsedMEM, %UsedDISK}
  THRESHOLDS = {LOW=0.33, MEDIUM=0.66, HIGH=1.0}
  RESOURCES = {r_1,...,r_n}
  RELS = {rel_1,...,rel_m}, ∀i=0,...,m : rel_i = {source,target} ∧ source,target ∈ RESOURCES

OUT:

  AGGR_RESOURCES = {ar_1,...,ar_k}
  AGGR_RELS = {arel_1,...,arel_p}, ∀i=0,...,p : arel_i = {source,target}
              ∧ source,target ∈ AGGR_RESOURCES

BEGIN:

  FOR k = 1 TO CATEGORIES.size() {
    FOREACH r_i ∈ RESOURCES ∧ category(r_i)==CATEGORIES_k {
      FOREACH atr_j ∈ ATTRIBUTES {
        SWITCH getAttributeValue (atr_j, r_i, THRESHOLDS) {
          case LOW: c=1
          case MEDIUM: c=2
          case HIGH: c=3
        }
        ar_{3(k-1)+c} = computeAggregatedInfo (ar_{3(k-1)+c}, r_i)
        FOREACH rel_j ∈ RELS ∧ rel_j.source==r_i {
          AGGR_RELS.insert(ar_k, rel_j.target) /* Avoiding repeated instances */
        }
      }
    }
    FOR i = 1 TO AGGR_RELS.size()
      rel_i.target = findAggregatedResource (AGGR_RESOURCES, rel_i.target)
    }

RETURN cleanVector (AGGR_RESOURCES), AGGR_RELS
```

Listing 4: CATEGORIZED resource aggregation algorithm pseudo-code

The example of listing 5 shows the same resources of the two previous examples but using the *CATEGORIZED* algorithm. 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.

```
Resource: Type="ComputingSystem", Name="CS_Intel_LOW"
          Attrs: ProcessorType="{(Intel,<count=1>)}"
                 ProcessingSpeed="{(3000,<count=1>,<total=3000>)}"
                 CPUUtilization="{(80,<count=1>,<total=80>)}"
                 NumOfProcessors="{(4,<count=1>,<total=4>)}"
Resource: Type="ComputingSystem", Name="CS_Intel_MED"
          Attrs: ProcessorType="{(Intel,<count=1>)}"
                 ProcessingSpeed="{(2600,<count=1>,<total=2600>)}"
                 CPUUtilization="{(100,<count=1>,<total=100>)}"
                 NumOfProcessors="{(2,<count=1>,<total=2>)}"
Resource: Type="ComputingSystem", Name="CS_Intel_HIGH"
          Attrs: ProcessorType="{(Intel,<count=2>)}"
                 ProcessingSpeed="{(2600-2800,<count=2>,<total=5400>)}"
                 CPUUtilization="{(720-1280,<count=2>,<total=2000>)}"
                 NumOfProcessors="{(8-16,<count=2>,<total=24>)}"
Resource: Type="ComputingSystem", Name="CS_AMD_MED"
          Attrs: ProcessorType="{(AMD,<count=2>)}"
                 ProcessingSpeed="{(2200-2400,<count=2>,<total=4600>)}"
                 CPUUtilization="{(160-800,<count=2>,<total=960>)}"
                 NumOfProcessors="{(4-16,<count=2>,<total=20>)}"
Resource: Type="OperatingSystem", Name="OS_Linux_LOW"
          Attrs: OperatingSystemType="{(Linux,<count=3>)}"
                 TotalPhysicalMemory="{(4000-128000,<count=3>,<total=140000>)}"
                 FreePhysicalMemory="{(400-32000,<count=3>,<total=34000>)}"
Resource: Type="OperatingSystem", Name="OS_Linux_MED"
          Attrs: OperatingSystemType="{(Linux,<count=1>)}"
                 TotalPhysicalMemory="{(16000,<count=1>,<total=16000>)}"
                 FreePhysicalMemory="{(9600,<count=1>,<total=9600>)}"
Resource: Type="OperatingSystem", Name="OS_AIX_LOW"
          Attrs: OperatingSystemType="{(AIX,<count=2>)}"
                 TotalPhysicalMemory="{(16000-64000,<count=2>,<total=80000>)}"
                 FreePhysicalMemory="{(3200-6400,<count=2>,<total=9600>)}"
Resource: Type="FileSystem", Name="FS_MED"
          Attrs: TotalStorageCapacity="{(100000-250000,<count=5>,<total=790000>)}"
                 FreeStorageCapacity="{(48000-100000,<count=5>,<total=368000>)}"
Resource: Type="FileSystem", Name="FS_HIGH"
          Attrs: TotalStorageCapacity="{(120000,<count=1>,<total=120000>)}"
                 FreeStorageCapacity="{(96000,<count=1>,<total=96000>)}"

Relationship: SourceType="CS", SourceName="CS_Intel_LOW",  TargetType="OS", TargetName="OS_Linux_LOW"
Relationship: SourceType="CS", SourceName="CS_Intel_MED",  TargetType="OS", TargetName="OS_Linux_LOW"
Relationship: SourceType="CS", SourceName="CS_Intel_HIGH", TargetType="OS", TargetName="OS_AIX_LOW"
Relationship: SourceType="CS", SourceName="CS_AMD_MED",    TargetType="OS", TargetName="OS_Linux_LOW"
Relationship: SourceType="CS", SourceName="CS_AMD_MED",    TargetType="OS", TargetName="OS_Linux_MED"
Relationship: SourceType="CS", SourceName="CS_Intel_LOW",  TargetType="FS", TargetName="FS_HIGH"
Relationship: SourceType="CS", SourceName="CS_Intel_MED",  TargetType="FS", TargetName="FS_MED"
Relationship: SourceType="CS", SourceName="CS_Intel_HIGH", TargetType="FS", TargetName="FS_MED"
Relationship: SourceType="CS", SourceName="CS_AMD_MED",    TargetType="FS", TargetName="FS_MED"

                                                TOTAL: 9 resources and 9 relationships
```

Listing 5: Same set of computers with the "CATEGORIZED" algorithm

# 4 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 matches optimally 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 algorithm of this policy is described in the pseudo-code of listing 6. 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*). Moreover, 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. The brokers are also represented as resources that have the containment relationships with resources such as computers. They follow the resource model that was discussed in the previous section.

```
FUNCTION:  bestBrokerRank

IN:

  JR = {jr_1,...,jr_m} job requirements
  B = {b_1,...,b_n} brokers
  PRIORITIES = {p_1,...,p_n} priorities of brokers
  RESOURCES = {res_1,...,res_n}, ∀i = 0,...,n : res_i defines computers managed by broker b_i
  RELS = {rel_1,...,rel_n}, ∀i = 0,...,n : rel_i defines relationships between resources res_i
  FACTORS = {CpuSpeed_FACTOR, NumCpus_FACTOR, FreeMem_FACTOR, ...}

OUT:

  Rank value

BEGIN:

  initialize (RANKS, B.size())
  FOR i=1 TO B.size() {
    IF matchRequirements (JR, RESOURCES_i, RELS_i){
      RANKS_i = computeBrokerRank (RESOURCES_i, RELS_i, FACTORS) * PRIORITIES_i
    }
    ELSE {
      RANKS_i = -1
    }
  }

RETURN getMaxValueIndex(RANKS)
```

Listing 6: BestBrokerRank policy pseudo-code

**Initialize** creates a new vector of the given size and initializes its elements to 0. The *RANKS* vector is used to store the obtained rank values of each broker. The **matchRequirements** function returns *true* if the given job requirements are matched

by any of the given resources. When the job requirements are not matched by a set of resources in a broker domain, it is not considered in the subsequent steps. The **computeBrokerRank** function returns the accumulation of the rank values obtained from the requirements matching in each broker resource. Internally, it uses the *computeResourceRank* function that, given a particular broker resource, returns the rank obtained from its main attributes matching (i.e., ProcType, OSType, ProcSpeed) with an impact factor. Finally, the **getMaxValueIndex** function returns the index of the given vector that contains its maximum value. Thus, it returns the index of the broker with the best rank relative to the job requirements.

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 *computeResourceRank* function implementation. The new resource input parameters are defined in listing 7 that replace *RESOURCES* and *RELS* of listing 6. 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>)"* that 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 (see other examples in the following section). 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,

```
IN:

  AGGR_RESOURCES = {ar_1,...,ar_k}
  AGGR_RELS = {arel_1,...,arel_p}, ∀i = 0,...,p : arel_i = {source,target}
              ∧ source,target ∈ AGGR_RESOURCES
  ...
```

Listing 7: Aggregated resources definition as input

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.

The second variant (**bestBrokerRank_SLOW** policy) also uses the aggregated resource form but it coordinates with the brokering layer. In particular, it takes the broker average slowdown metric as the main characteristic in the matching optimization function. We define *"a 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. 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. Figure 1 depicts the bestBrokerRank_SLOW policy in a simplified scenario.

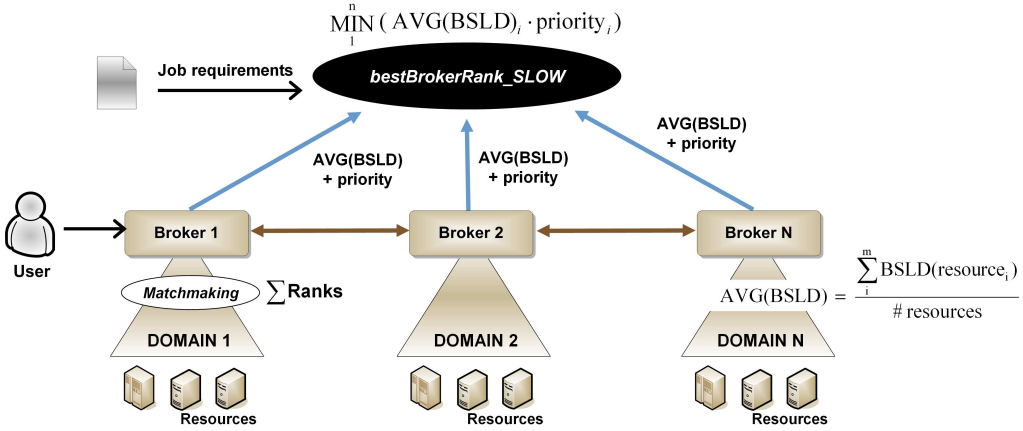



Figure 1: BestBrokerRank_SLOW policy schema

The bestBrokerRank_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 bestBrokerRank_AGGR but giving more priority to the resources which are achieving better performance. Actually, this is a way to balance the performance among the different brokers. Since the broker slowdown is the main attribute for matching the brokers rather than the resource information, it gives more priority to the underlying scheduling levels (local queuing and resource management systems) in the job scheduling process. Thus, the jobs can be potentially queued in the local systems

for longer time. In fact, with the previous policy the average broker slowdown is always 1 because the jobs are only submitted to a center when it has enough free resources to 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 reduce their performance metrics but it may improve the global system performance.

# 5 Evaluation Methodology

We have used simulation mechanisms for our policy evaluations. These simulations allow us to research policies for large and complex configurations with numerous jobs and high demand of resources and to easily include modifications and refinements in the policies.

## 5.1 The Alvio simulation framework

The Alvio Simulator [14] is a C++ 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. It allows research on job scheduling strategies in very different scenarios that may be composed of different VOs. It has been designed in order to provide an easy mechanism to extend its functionalities. Thus, extending this simulator with our models (i.e., adding new scheduling strategies or new resource models) required only a reasonable amount of effort in terms of development and design.

### 5.1.1 Local systems model

The simulator models different components which interact in local and distributed architectures. Conceptually, it is divided into three main parts: the *Simulator Engine*, the *Scheduling Polices* and the *Computational Resource Model* (CRM). A simulation of a local scheduling scenario allows us to simulate a given policy with a given architecture. Currently, the following local policies have been modeled:

- Local Resource Selection Policies (RSP): First Fit, First Continuous Fit, and Less Consume policies.
- Local Job Scheduling Polices (LJSP): the First Come First Serve policy, the backfilling policy, and finally, the Resource Usage Aware backfilling (RUA-Backfilling [36]). For backfilling policies, the different properties of the wait queue and backfilling queue are modeled (Sort Job First, LXWF and First Come First Serve) and different numbers of reservations can also be specified.

Like other simulators, given a workload and an architecture definition, Alvio is able to simulate how the jobs would be scheduled using a specific job scheduling policy (such as First Come First Serve, or the backfilling policies). The main contribution of this simulator at this level is that it not only allows the modeling of the jobflow in the system, but also the simulation of different resource allocation policies. To do this, it uses a reservation table that models how each job is allocated to each node of the architecture. As a result, the researcher is able to validate how different combinations of scheduling policies and resource selection policies impact on the performance of the system.

The other new capability of this simulator is the ability to model the local resource usage on the jobs that are running in the system. For each job, the researcher can specify the different fields that are specified in the Feitelson Standard Workload Format (SWF) [37]. However, at the local level, in addition to the SWF fields, for each job, the user can specify the memory, ethernet and network bandwidths. Consequently, depending on the configuration of the simulation, the impact of considering the penalty introduced in the job runtime due to resource sharing can be evaluated, as it implements a job runtime model and resource model that try to estimate the penalty introduced in the job runtime when sharing resources.

### 5.1.2 Multi-site systems model

The simulator allows the local scenario to be extended by having several instances of this scenario. As depicted in figure 2, different machines (e.g., clusters, single box servers, etc.) can be specified (including their architecture definition, local job scheduling policy, and local resource selection policy) and the different meta-scheduling policies that can be specified to schedule the jobs. When the simulation starts, all the different layers of the model are instantiated, from the local reservation tables (which model how the jobs are mapped to the processors) to the brokering component that manages the jobs submitted to the system.

In grid scenarios, a grid resource broker usually requires the specification of the job requirements from the user. In many cases, the meta-scheduling policies use these requirements to carry out the matchmaking with the local resources. To allow this, we have extended the Standard Workload Format (SWF) to specify the job requirements in the workload to be simulated. Each requirement is composed of an identifier (e.g., *Vendor*), an operator (e.g., *EQUAL*) and a value (e.g., *Intel*). In the current version of the simulator, the following requirements can be specified for each grid job: memory in MB (e.g., *1024 MB*), processor vendor (e.g., *Intel*, *AMD*), processor clock speed in MHZ (e.g., *1200 MHZ*), operating system (e.g., *Linux*, *AIX*), number of processors (e.g., *4 processors*) and disk size in MB (e.g., *1000 MB*). Concerning the meta-scheduling policies, two different kinds of policies can be used:

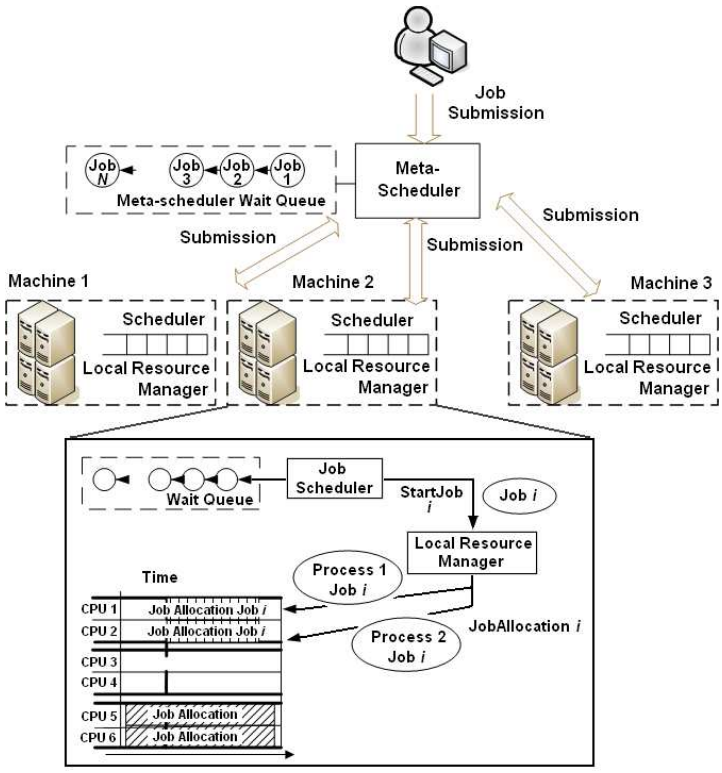



Figure 2: Brokering system

- Multi-site scheduling policies: the non-centralized ISIS-Dispatcher policy [38]. In this scenario, jobs are scheduled by their own dispatcher and there are no centralized scheduling decisions.

- Brokering scheduling policies: First Come First Serve, RealTime, Earliest Deadline First, or Job Rank (JR)-backfilling policies for the job scheduling, and resource selection based on the matchmaking approach.

### 5.1.3 Interoperable grid systems model

In our recent research work, we have focused on evaluating meta-brokering policies [5] based on P2P approaches. To do this, we have extended the Alvio components to model the different components that are included in such systems.

In a meta-brokering model, many different components are instantiated inside the simulator. Firstly, the *meta-system* entity is created. This is a conceptual component

that models the different elements that are included in the meta-environment (such as the different domains, generic prediction services, etc.). It can contain a centralized meta-broker scheduling entity which potentially can implement centralized based meta-brokering policies. It also allows P2P meta-brokering strategies to be evaluated (see figure 3). This meta-system component contains a set of different domain elements. Each of them contains a meta-brokering entity that is responsible for managing the jobs submitted to that domain. Furthermore, it contains a set of machines that model typical HPC local resources. Thus, the different centers of the local-scenarios of the simulation model are also instantiated. For each of the machine components, a local scheduling policy (such as the First Come First Serve or the EASY-Backfilling), a resource selection policy (such as the First Fit or the First Continuous Fit) and a reservation table are created.

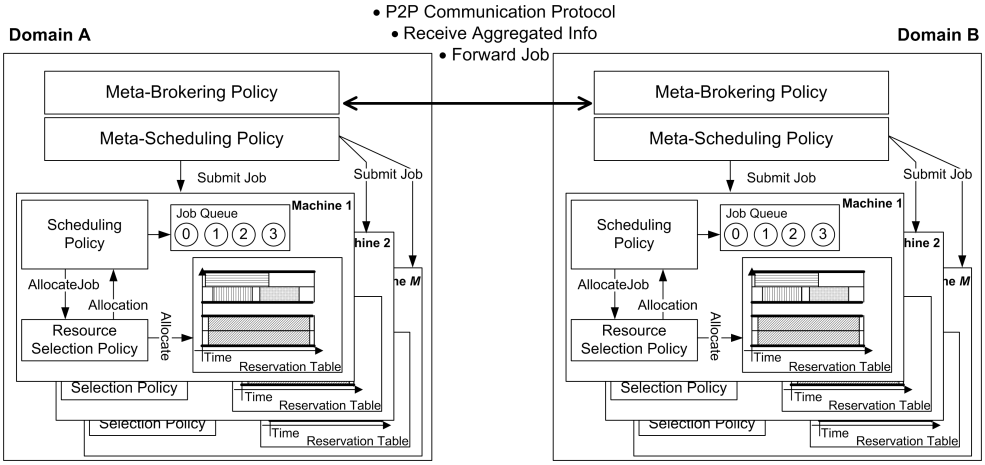



Figure 3: Meta-brokering model with a P2P approach

We have included in the simulator a prototype of the previously introduced *BestBrokerRank* policy and the resource aggregation algorithms. As we have pointed out, this basically selects the most appropriate broker to submit a job based on a set of ranks values corresponding to the different brokers, rather than using the local information directly. However, since we consider forwarding jobs between brokers, in our coordinated approach the different scheduling layers are important (for example, when we use the average slowdown of the brokers as a QoS metric in the one variant of our policies).

## 5.2 The workloads

In our evaluation, we have used traces of the DAS-2, Grid5000, and Sharcnet systems from the Grid Workloads Archive [39][40]. We have selected two weeks of job submissions for each workload trace. Thus, we have avoided the initial warm up period of the systems (around the first 50,000 jobs) to skip the 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 full interoperable system with up to 18 domains, we are considering around a quarter million jobs, 180 clusters, and more than 30,000 available processors.

We have analyzed the traces in order to select representative fragments. We have manually reduced the inter arrivals times of the jobs. Increasing the stream of jobs allows us to increase the pressure on the system incrementing the load. Moreover, we have adjusted the execution times and we have limited the memory and CPU demand (up to 512 CPUs) to scale the experiments according to a reduced scenario which simplifies the analysis of the results.

Figure 4 shows the distribution of the workloads job arrivals. The DAS-2 distribution is approximately linear and the Sharcnet one is by chunks. In the latter one, we have pairs of intervals where many jobs arrive and where there are no job arrivals. In the figure, we can also find two different Grid5000 workloads. The one as GRID5000_OUTLIER is an unrepresentative portion of the trace because the majority of jobs arrive at the first 30% of the workload duration and the rest of time there are only a few job arrivals. In fact, the simulation with the outlier trace gave us irregular and incoherent results compared to the other workloads. However, the other Grid5000 workload follows a more uniform distribution and it will be used in our experiments. More details regarding these traces can be found in [39].

In our experiments, we have defined three different domain types, one for each workload system: *"DOM_small"* for DAS-2, *"DOM_medium"* for Grid5000, and *"DOM_big"* for Sharcnet. The resources of each domain are based on real testbeds in terms of number of clusters, CPU architecture, and OS. Moreover, they are scaled in terms of memory and disk demand. 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. However, to simplify the experiments, 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 quite heterogeneous in term of number of CPUs, architectures and OS (for example, Grid5000 has 50% *Intel*, 40% *AMD* and 10% *PowerPC*, and 60% *Linux*, 30% *AIX* and 10% *Solaris*).

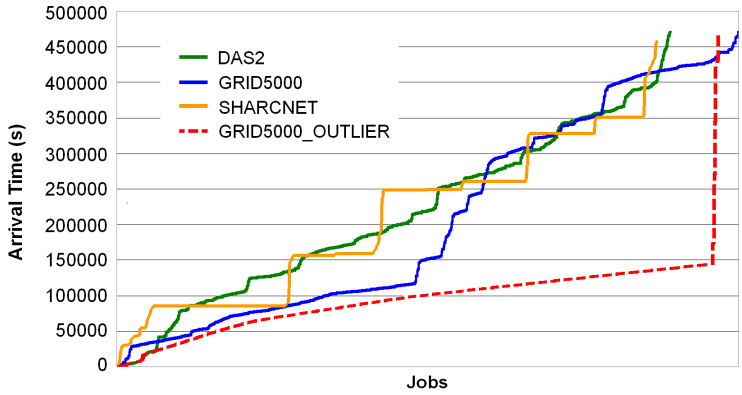



Figure 4: Workloads job arrivals

The original traces do not include resource requirements for each job. To fix this, we have generated a requirements trace per workload to be used by the simulator as explained in the previous sub-section. To generate these requirements, we have used a perl script that defines 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).

### 5.3 Metrics

We use the following metrics for evaluating our strategies:

- Total workload execution time
- Average job waiting time
- Average bounded slowdown (BSLD). We define BSLD for a given job:

$$BSLD_{job} = max\left(1, \frac{runtime_{job} + waittime_{job}}{max(runtime_{job},\ threshold)}\right)\ ,\ threshold = 60\ seconds$$

- Average CPUs and nodes utilization

The units for the first two metrics are seconds and hours, for average CPUs and nodes utilization are percentages, and the slowdown has no units. In our experiments, we try to minimize all the metrics except the average CPU and nodes utilization that should be maximized.

# 6 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 earlier part of the paper.

In the evaluation, firstly, we compare the scenario of having independent grid systems with an interoperable scenario having the same grids and using our *bestBrokerRank* policy for the broker selection. Afterwards, the presented evaluation is focused on evaluating the performance of the different variants of the *bestBrokerRank* broker selection policy, using the aggregation algorithms and being coordinated with the underlying scheduling level. We also study if the aggregation algorithms can scale to large grid interoperable systems with several domains and thousands of resources.

## 6.1 Interoperable versus independent grids

In this sub-section, we present the evaluation of two different scenarios. In the first one, we have simulated the scenario where each domain schedules the jobs that users have submitted originally to the system. Here we evaluate the performance of each of the different domains without any connection among them. Based on the eNANOS resource selection strategy (it considers priorities in a matchmaking algorithm) for each broker (one per domain) we have evaluated four different configurations: balancing the priorities (*bal*), giving more priority to the CPUs (*cpu*), to memory (*mem*), and to disk (*disk*). For example, while in (*cpu*) configuration the factor that multiplies the number of CPUs in the matchmaking algorithm may be two or three times higher than factor that multiplies the free memory, in (*bal*) configuration the factors are similar.

In the second scenario, using the same workloads, we evaluate the effect of routing jobs between brokers in an interoperable grid environment. In particular, we have modeled a scenario where the brokers establish agreements with each other and share their resources in a P2P fashion. We have used the regular *bestBrokerRank* policy for the broker selection. In the evaluation we compare an interoperable system versus regular independent brokering systems. In this scenario, we also evaluate the impact of forwarding between brokers. To do this, we have evaluated two different configurations: *own* that considers local jobs with more priority, and *equal* that considers local and remote jobs having equal priorities.

Although we are interested in precise evaluation of the performance of our policies as well as the effect of having connections between the different domains, we do not include some P2P issues such as the connection time overhead, job routing overhead, or the loss of peering connections. These functionalities will be available in future implementations.

Table 1 shows the results of the different scenarios and configurations. We present for each experiment: the workload execution time, the $95^{th}$ percentile of the average waiting time and bounded slowdown (BSLD), the percentage of CPU, memory and disk conflicts[1], and the percentage of re-scheduled jobs[2]. The row labeled as *"overall"* is the combination of the independent grid workloads. This means that we take the average values of the different workloads and the maximum workload execution time to compare with the interoperable grid scenarios. The workloads labeled as "*interoperable"* are the combinations of the three different workloads (DAS-2, Grid5000 and Sharcnet), using an interoperable grid scenario with three different domains (one domain per workload type).

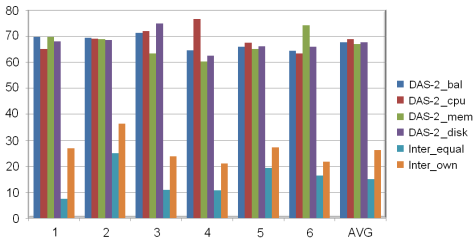
| Workload Name | Config | Exec Time (h) | Wait Time (s) | BSLD | % Conflicts | | | %Re-Scheduled |
|---|---|---|---|---|---|---|---|---|
| | | | | | *cpu* | *mem* | *disk* | |
| DAS-2 | *bal* | 159.38 | 998 | 9.053 | 78.10 | 10.40 | 11.50 | 37.32 |
| DAS-2 | *cpu* | 150.65 | 842 | 8.974 | 80.00 | 8.74 | 11.26 | 31.64 |
| DAS-2 | *mem* | 162.06 | 881 | 9.744 | 72.85 | 9.44 | 17.71 | 36.96 |
| DAS-2 | *disk* | 165.745 | 1,106 | 10.308 | 76.89 | 9.53 | 13.59 | 34.85 |
| Grid5000 | *bal* | 763.07 | 11,720 | 8.330 | 73.19 | 16.16 | 10.65 | 51.57 |
| Grid5000 | *cpu* | 744.79 | 11,914 | 8.342 | 75.73 | 13.99 | 10.29 | 51.20 |
| Grid5000 | *mem* | 769.48 | 11,876 | 8.332 | 74.56 | 16.29 | 9.14 | 51.64 |
| Grid5000 | *disk* | 778.30 | 11,266 | 8.315 | 74.00 | 16.24 | 9.76 | 51.81 |
| Sharcnet | *bal* | 344.62 | 795 | 2.754 | 82.69 | 8.47 | 8.85 | 11.61 |
| Sharcnet | *cpu* | 338.32 | 699 | 2.732 | 83.40 | 7.53 | 9.07 | 11.27 |
| Sharcnet | *mem* | 346.55 | 1,191 | 3.282 | 80.89 | 8.92 | 10.19 | 13.10 |
| Sharcnet | *disk* | 349.21 | 865 | 2.775 | 79.43 | 8.62 | 11.96 | 11.62 |
| Overall | | 778.30 | 4,513 | 6.91 | 77.64 | 11.19 | 11.16 | 29.35 |
| Interoperable | *equal* | 743,58 | 65 | 1.058 | 50.96 | 32.54 | 16.50 | 4.02 |
| Interoperable | *own* | 748.70 | 69 | 1.124 | 57.21 | 27.56 | 15.22 | 6.37 |

Table 1: Evaluation results with Alvio

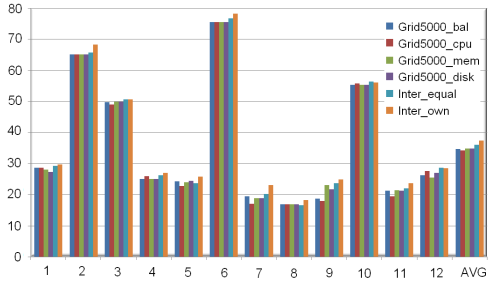
Figures 5a, 5b and 5c show the clusters utilization for each system using the different configurations. The X axis shows the different clusters and the average of them. The workloads and configurations used are the same as table 1. The utilization of the cluster

[1] A **conflict** is found for a given resource when it has insufficient CPUs/memory/disk to match the job requirements.

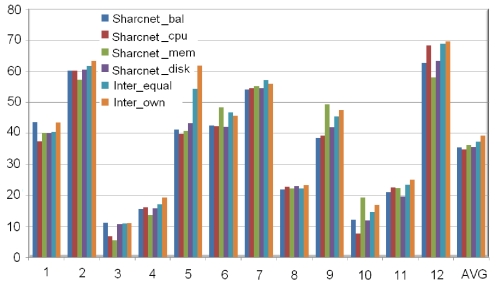
[2] A **re-scheduling** is performed when there is no resource in the system that matches the job requirements.




(a) DAS-2 clusters utilization




(b) Grid5000 clusters utilization



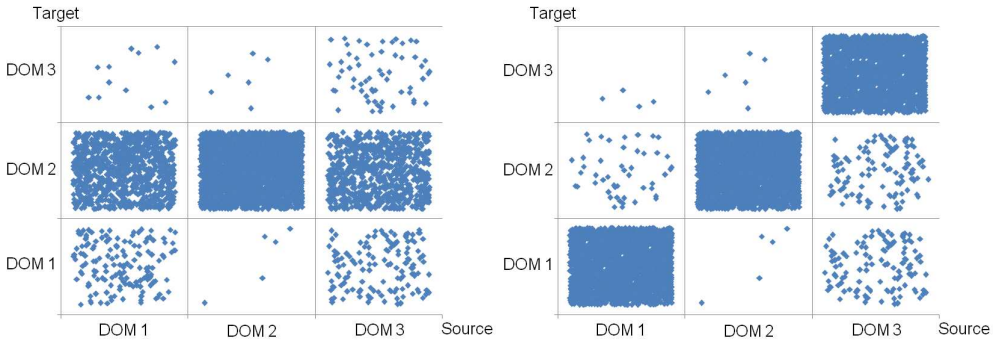

(c) Sharcnet clusters utilization

Figure 5: Resources utilization

is computed by the percentage of the processor time use by running jobs with respect to the maximum computing power available in the cluster in a given interval of time. The total computing power is calculated by multiplying the number of processors available in the system by the length of this interval. Figure 6 shows the job execution and forwarding distribution among the different domains.

The workload execution times are similar with the different configurations, especially with the Grid5000 and Sharcnet systems (the time difference is between 2-4%). However, with the DAS-2 system, the execution time difference between the configurations is up to 8%. Moreover, the execution time is shorter with the *cpu* configuration in the three systems.

In general, the waiting times and BSLD are lower with the *cpu* configuration, except with the Grid5000 system that shows better results with the *disk* configuration. However, the best global performance results are obtained with *cpu* and *bal* configurations. The CPU conflicts are around 78% on average and both memory and disk conflicts are only around 11% on average. It is also shown that for each configuration the conflicts percentage related to the configuration increases. Moreover, the resource utilization is increased in the clusters that contain more resources belonging to the given configuration.




(a) With *equal* configuration (b) With *own* configuration

Figure 6: Job execution among the domains

The percentage of re-scheduled jobs substantially differs depending on the systems. For example, the Grid5000 system has higher job re-scheduling because it is the most heterogeneous system. In general, we can see that with *cpu* configuration the percentage of re-scheduled jobs is lower and with the *mem* configuration is higher. Thus, it can be observed that the changes in the attributes priorities have a more substantial effect than when the systems are more homogeneous. This makes sense because the jobs can be distributed more easily among the resources as there are fewer restrictions in terms of architectures or OS requirements.

In the interoperable grid scenarios, the joint workloads execution time is similar to the maximum of the single workload executions. The main reason is the distribution type of the job inter-arrivals and the restrictions of job requirements. As can be observed in the figure 6, in both interoperable scenarios the Grid5000 is highly demanded. Again, this is caused by the fact that it has more heterogeneous resources. Therefore, the majority of the Grid5000 workload jobs can find resources that match its requirements only in this system.

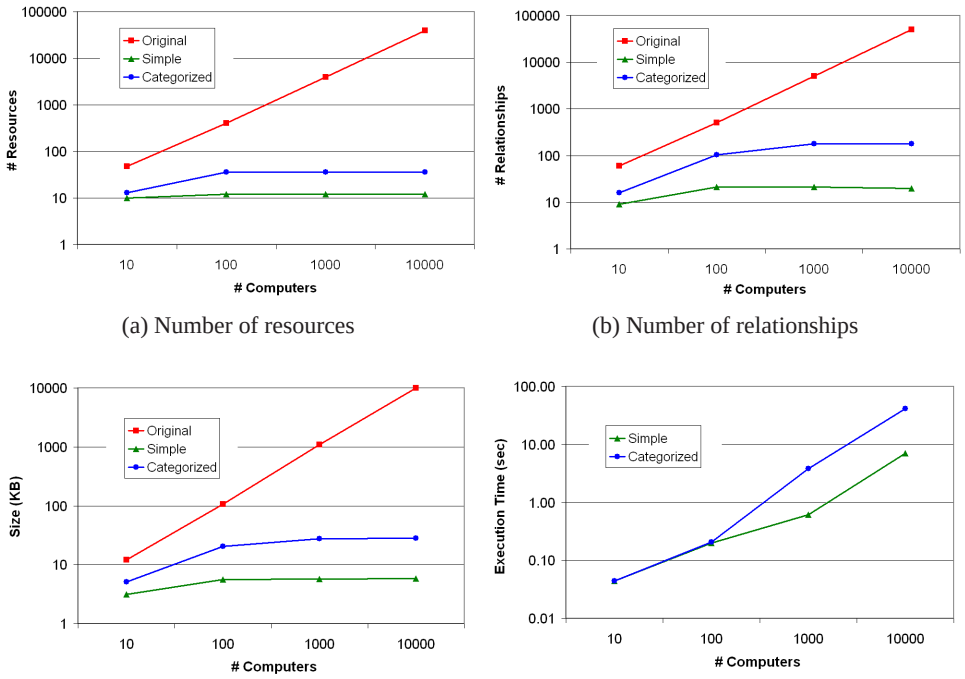
The results with the interoperable grid scenario are better compared to the independent brokering one in terms of waiting time and slowdown. The results are similar with both interoperable configurations but, in general, with the *equal* configuration they show better results. The results also show that the conflicts distribution is quite different compared to previous ones. Since there are more available resources to allocate the jobs, the number of conflicts and re-scheduling are significantly lower. In particular, the percentage of CPU conflicts is lower and, consequently, the percentage of memory and disk conflicts is higher. We can also appreciate that the percentage of job re-scheduling is a bit lower with the *equal* configuration. The resource

utilization has been increased around 5% in Grid5000 and Sharcnet systems, and has been reduced in the DAS-2 system. This can be explained by the fact that DAS-2 and Sharcnet jobs finish before the total workload and they receive just a few forwarded jobs from the Grid5000 system. Moreover, figure 6 states that with the *equal* configuration the majority of the job forwarding goes to the Grid5000 system. However, with the *own* configuration the resource utilization is substantially higher because the quantity of job forwarding is somewhat lower as can be seen in figure 6 as well.

## 6.2 Scalability of the resource data aggregation algorithms

Since the broker selection policies that we present in this paper depend on resource aggregation algorithms, in this sub-section, we evaluate the scalability of the two proposed aggregation algorithms. To this end, we have performed the experiments with different number of resources, from 10 to 10,000 computing systems. The experiments were conducted by executing a Java program that implements the aggregation algorithms on a commodity computer (an Intel core duo with 1Gb of memory). 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 spent time. For each experiment, we repeat this process with 5 different input files and we compute the average value. However, as we will show later, the variability of the resource characteristics does not affect significantly to the scalability results. The input files with the computers definition are generated by a perl script that we have developed. The script generates a file with the definition of the specified number of computers. The definition of these computers are obtained randomly from a set of parameters such as 6 processor vendor types, 8 processor speed types, 8 operating system types, etc.

In order to evaluate the scalability of the resource aggregation algorithms, we have measured the metrics that are shown in figure 7. All of them are in logarithmic scale. Figures 7a and 7b show the number of resources and relationships used for describing the same resources, respectively. They show the results with regular resource model (*Original*) and with the two aggregation algorithms (*Simple* and *Categorized*). We can appreciate that both figures follow a similar pattern. However, in figure 7a the difference between the number of resources with the aggregation algorithms and the regular form is larger. In general, while the number of resources and relationships increases in a linear manner with the regular resource form, with both aggregation algorithms the number of resources and relationships is almost constant. In particular, for up to 100 computers the number of resources and relationships for the aggregated forms is around 10 times lower than the regular one. For more than 100 computers the number of resources in




(a) Number of resources

(b) Number of relationships

(c) Size of aggregated data

(d) Execution time

Figure 7: Results from the evaluation of the aggregation algorithms

aggregated form is up to 1,000 times lower than the number of resources in the regular one. Although both aggregation algorithms follow the same pattern, the number of resources and relationships with the *Categorized* algorithm is around 10 times higher than with the *Simple* algorithm.

Figure 7c 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 and relationships. This is explained due to the fact that the size of the resource information is proportional to the number of resources and relationships that it contains. However, in this case the difference between the different algorithms is quite 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 6 times larger than with the *Simple* algorithm. Figure 7d shows the processing time required for both aggregation algorithms. With 100 computers or less the execution time of both algorithms is very similar. For more than 100 computers the execution time of the *Categorized* algorithm is longer than the execution time of the *Simple* algorithm.

However, the execution time difference between the two aggregation algorithms is less than 50 seconds in the worse case (with 10,000 computers).

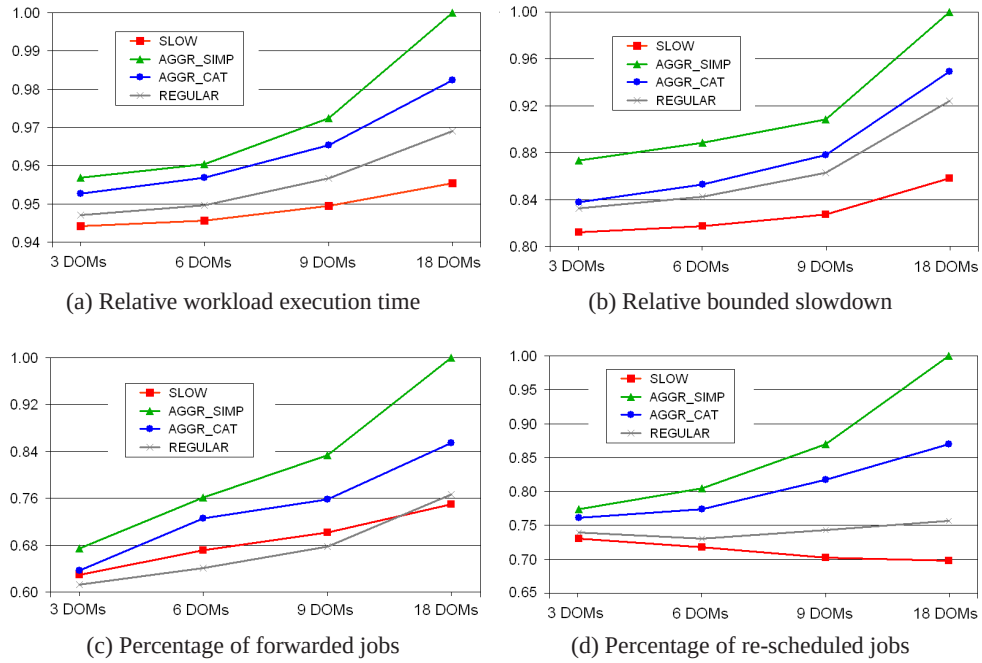
Therefore, we conclude that the two aggregation algorithms are scalable in terms of resource information size, and the execution time of the aggregation algorithms is acceptable for an interoperable grid environment. With the *Categorized* algorithm the execution time is longer than with the *Simple* 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.

### 6.3 Performance results

In this sub-section, we evaluate the different broker selection policies in different grid interoperable scenarios. We have defined them with different number of domains (from 3 to 18) in order to evaluate the performance and scalability of the different broker selection strategies. We have taken as a reference the original *bestBrokerRank* policy (*REGULAR* in the figures), and we have compared it to three different variants. One of them uses the *Simple* resource aggregation algorithm (*AGGR_SIMP* in the figures). The second one uses the *Categorized* resource aggregation algorithm (*AGGR_CAT* in the figures). The last one is the *bestBrokerRank_SLOW* policy (*SLOW* in figures).

In order to perform the evaluations presented below, we have done some modifications in the simulation environment. In particular, we have modified a little the Grid5000 domain in order to reduce its difference with the other two domains in terms of heterogeneity of the resources. We also have increased the workloads pressure in order to better compare the performance metrics results in more loaded systems with more demand of resources and with more requests per unit of time.

Figure 8 shows the performance results obtained with the different policies in the different interoperable 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 because our objective is to compare the policies. Figure 8a shows the total workload execution time. Figure 8b shows the average bounded slowdown of the brokers. Figures 8c and 8d shows the percentage of forwarded jobs to another broker and re-scheduled jobs in the grid domain, respectively. It is worth noting that in the first two figures, which are the most important ones, the maximum difference between two policies is less than 14%. We note that in each figure the worst result is obtained by the *AGGR_SIMP* policy (value equal to 1). It is also worth noting that we do not consider the processing time of the matchmaking algorithm. Thus, the difference between the *REGULAR* policy and the policies that uses aggregated resource information may be smaller.



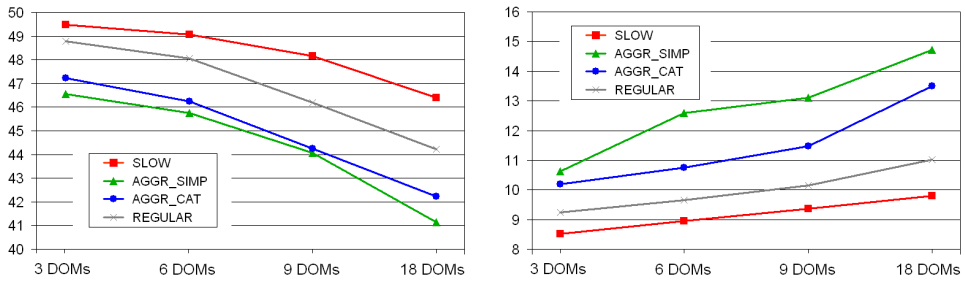

(a) Relative workload execution time

(b) Relative bounded slowdown

(c) Percentage of forwarded jobs

(d) Percentage of re-scheduled jobs

Figure 8: Normalized performance results




(a) Average resource utilization (%)

(b) Standard deviation

Figure 9: Resource utilization results

Figure 9 shows the resource utilization results of the same policies and scenarios of the previously commented figures. Figure 9a shows the average resource utilization and figure 9b shows the standard deviation that gives us some hints about the performance balancing of the different brokers.

### 6.3.1 Resource aggregation algorithms

In general, in figure 8 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). With these two policies, the workload execution time and average bounded slowdown increase when the number of domains increases, especially with 18 domains. The execution time increases 4.2% with the *AGGR_SIMP* policy and increases 3% with the *AGGR_CAT* policy. The average bounded slowdown increases around 13% with the *AGGR_SIMP* policy and increases 11% with the *AGGR_CAT* policy. Both policies show also less performance than the *REGULAR* policy with every metric. The execution time is around 1.5% worse on average, the average bounded slowdown is 2.8% worse on average, the percentage of forwarded jobs is 9.25% worse on average, and the percentage of re-scheduled jobs is 9.75% worse on average. The *AGGR_SIMP*, *AGGR_CAT* and *REGULAR* policies follow similar patterns as 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 9 and 18 domains.

The average resource utilization with *AGGR_SIMP* and *AGGR_CAT* policies is significantly lower than with the *REGULAR* policy (around 5% on average with respect to the *REGULAR* policy). 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 marked decrease with 18 domains. The standard deviation (SD) 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 SD 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, SD increases with every policy when the number of domains increases. With the *REGULAR* policy the increase is linear, and with the *AGGR_CAT* policy it is almost linear except for 18 domains that is significantly higher. With 3 domains both *AGGR_SIMP* and *AGGR_CAT* policies have similar values. However, with 6 and more domains the SD increases significantly with the *AGGR_SIMP* policy (up to 15% with respect to the *AGGR_CAT* policy). It indicates that with *AGGR_SIMP* and *AGGR_CAT* policies the performance balancing of the brokers is worse than with the *REGULAR* policy, especially with the *AGGR_SIMP* policy.

The results show that the performance metrics can be degraded when the number of domains increases. They also indicate that *AGGR_SIMP* and *AGGR_CAT* policies obtain worse results. However, this degradation is not drastic, and the differences between the *REGULAR* policy and the policies that use aggregated resource form are not very large.

### 6.3.2 Coordination with the underlying levels

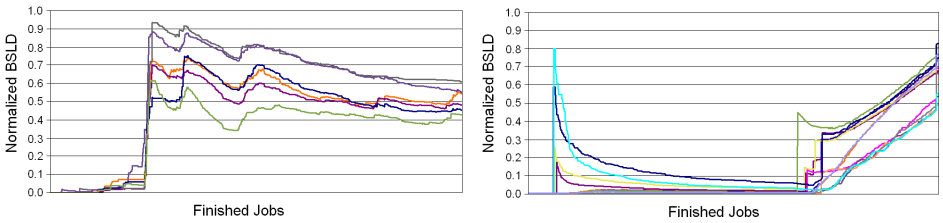
The performance results of figure 8 with the *SLOW* policy are better than the results with the *REGULAR* policy in most of the cases. The execution time is around 1% better on average, the average bounded slowdown is 3.9% better on average, the percentage of forwarded jobs is around 2% better on average, except with 18 domains that is a bit worse, and the percentage of re-scheduled jobs is 5% better 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 increases less than 1% and the average bounded slowdown increases less than 4%. Moreover, the *SLOW* and *REGULAR* policies follow similar patterns in terms of execution time and average bounded slowdown. The percentage of re-scheduled jobs pattern 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 the *SLOW* policy is 4% larger on average than with the *REGULAR* policy. However, they follow the same pattern: the resource utilization decreases when the number of domains increases. The total resource utilization increase is around 6% with the *SLOW* policy. The standard deviation with the *SLOW* policy is 10% smaller than with the *REGULAR* policy. However, they follow very similar patterns. It indicates that with the *SLOW* policy the performance balancing of the brokers is better than with the *REGULAR* policy.

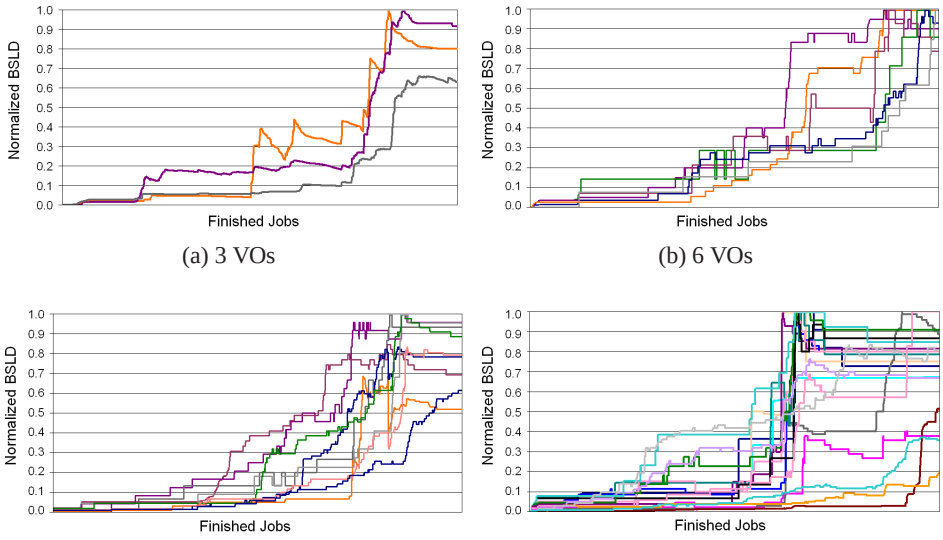
Figure 10 shows the evolution of the average bounded slowdown of the different clusters in two different domains (DAS-2 and Sharcnet) using the *SLOW* policy. Each data series of the figure shows the bounded slowdown of a cluster. They clearly converge to a similar value, which is very close to the total workload average bounded slowdown. This indicates that the coordinated strategy can perform a good balancing of performance among the different clusters. Moreover, it is shown that the different clusters follow similar patterns. Since in the DAS-2 system the resources are homogeneous, probably it is easier to balance the performance. However, in the Sharcnet case the performance balancing is a bit worse and there are some peaks during the workload execution. These peaks can be explained with the fact that the Sharcnet workload distribution has job submission peaks. Moreover, the performance balancing is worse probably because the Sharcnet resources are more heterogeneous.

Figure 11 shows the evolution of the average bounded slowdown of the brokers using the *SLOW* policy with different number of domains. In this case, each data series shows the bounded slowdown of a brokers rather than a cluster. We note that with three domains




(a) DAS-2 (b) Sharcnet

Figure 10: Evolution of the clusters normalized AVG slowdown



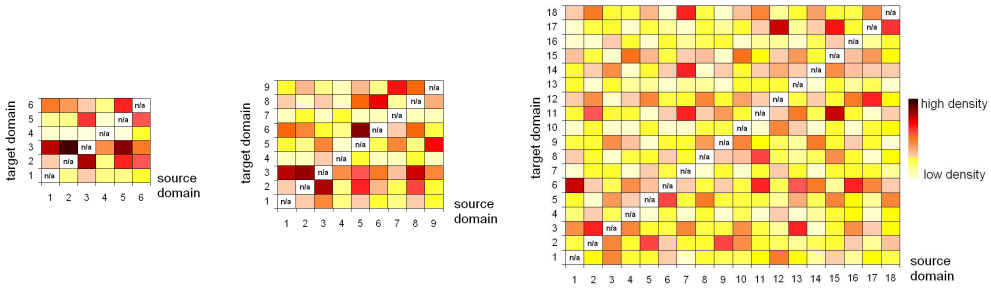

(a) 3 VOs (b) 6 VOs

(c) 9 VOs (d) 18 VOs

Figure 11: Evolution of the domains normalized AVG bounded slowdown

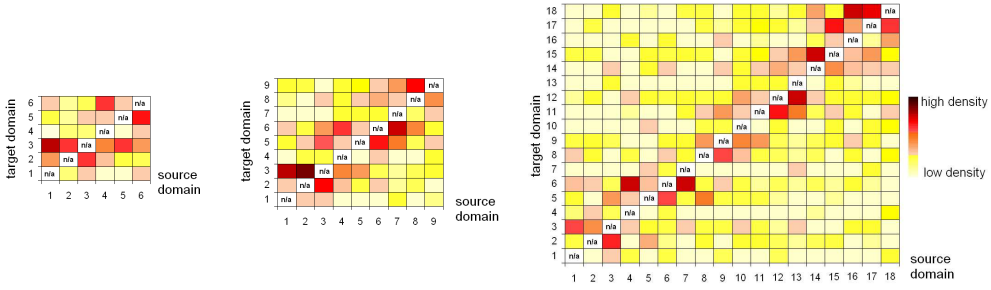
the bounded slowdown is quite well balanced. However, when the number of domains increases the balance of the different brokers slowdown is worse. This degradation fits with the performance results of figure 8. We saw that with the *SLOW* policy, as the rest of the policies, the average bounded slowdown increases when the number of domains increases. Consequently, a worse general performance results in a degraded broker performance balancing. Moreover, in contrast to balancing the clusters slowdown of a given broker, balancing the brokers slowdown is much more difficult, especially when the number of domains is large.

Figures 12 and 13 show the histogram of the job forwarding among different domains with the *AGGR_CAT* and *SLOW* policies, respectively. They do not consider the job execution inside a domain, for this reason the positions of the figures with same source and target domains are marked as "n/a". As it is shown in the figures legend, the dark regions indicate more density of forwarded jobs, and the light ones indicate less number of forwarded jobs. In fact, this is another way to analyze the performance balancing among the different brokers or domains. Moreover, it better visualizes which brokers are more demanded, which ones require forwarding more jobs, and what the patterns of the job forwarding are.




(a) With 6 VOs (b) With 9 VOs (c) With 18 VOs

Figure 12: Job forwarding among domains with AGGR_CAT configuration




(a) With 6 VOs (b) With 9 VOs (c) With 18 VOs

Figure 13: Job forwarding among domains with SLOW configuration

We note the *SLOW* policy can achieve better balancing of the job forwarding among domains. This observation fits with figure 8 where the *SLOW* policy has the best results. It is also shown that with the *AGGR_CAT* policy the forwarding is more than with *SLOW* policy because it has more dark regions. The balance is better with the *SLOW* policy than with the *AGGR_CAT* policy. On one hand the forwarding distribution is well defined with the *SLOW* policy: the target domains are usually near the source domains (the dark regions are found near the diagonal of the figures), and with the *AGGR_CAT* policy the forwarding distribution is not uniform. Furthermore, in figure 12 there are some rows with only light regions, and the other rows have lots of dark regions. This indicates that some domains receive lots of forwarded jobs and other domains receive just a few.

## 7 Conclusions and Future Work

In this paper, we have addressed the problem of broker selection in grid interoperable scenarios. We have described and evaluated the *bestBrokerRank* policy which selects the best broker to submit a job given a set of resource information. We also have described and evaluated two variant of this policy: using resource information in aggregated form, and coordinating the scheduling with the underlying layers based on the brokers average bounded slowdown, in addition to aggregated resource information. We also have presented two different resource aggregation algorithms that have been used by our broker selection policies.

Using our simulation platform, we have performed various evaluations. Firstly, we have compared independent grid systems to the interoperable scenario of these systems. We also have evaluated two different interoperable configurations: *own* that considers local jobs with more priority, and *equal* which consider local and remote jobs with equal priority. The results with the interoperable grid scenario are better compared to those of the independent brokering one in terms of waiting time and slowdown. Moreover, the number of conflicts and re-scheduling are significantly smaller and the resource utilization has been slightly increased. The results are similar in both interoperable configurations but, in general, with the *equal* configuration they are better. Thus, we conclude that an interoperable grid scenario can improve the global system performance compared to the independent grid systems. However, we have not taken into account some P2P issues, such as protocols overhead or the loss of peering connections.

Before evaluating broker selection policies, we have studied the scalability of *Simple* and *Categorized* resource aggregation algorithms. The results show that the algorithms are scalable in terms of resource information size, and their aggregation processing time is acceptable for an interoperable grid environment. Although with the *Categorized* algorithm the execution time is longer, and the resource information size is larger than

with the *Simple* algorithm, the accuracy of the resource data is much better with the *Categorized* algorithm. However, we did not address the gain in matching time with aggregated resource information.

We have evaluated the performance of broker selection policies comparing the regular *bestBrokerRank* policy (*REGULAR*) with its two variants: *bestBrokerRank_AGGR* and *bestBrokerRank_SLOW*. The first one considers the resource information in aggregated form using the two described resource aggregation algorithms (*AGGR_SIMP* and *AGGR_CAT*), and the second one uses the brokers average bounded slowdown as well as the aggregated resource form (*SLOW*). We have obtained the best results with the *SLOW* policy. The execution time is 1% better, and the average bounded slowdown is almost 4% better with respect to the *REGULAR* policy. The resource utilization is 4% higher. The worst results have been obtained with the *bestBrokerRank_AGGR* policies. On average, their execution time is 1,5% worse and their average bounded slowdown is almost 3% worse with respect to the *REGULAR* policy. The resource utilization is 5% lower on average. In general, with the *AGGR_CAT* policy we have obtained better results than with the *AGGR_SIMP* policy. The difference between both policies is up to 5% in the average bounded slowdown. Through the study of the standard deviation of broker resource utilization, the evolution of the brokers slowdown, and job forwarding, we claim that the *SLOW* policy balances the performance among the brokers better than the other policies. Therefore, the results obtained with our evaluation clearly support the argument that coordination with the underlying scheduling levels in interoperable grid scenarios can improve workloads execution as well as resource utilization.

There are different lines of work that we plan to address in the near future. On one hand, we are targeted to include the P2P details in our models to improve the simulations. We will also add these features to our negotiation protocol that was presented in [7]. 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) [41].

## 8 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

[1] I. Foster, C. Kesselman, and S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," *International Journal of High Perfomance Computing Applications*, vol. 15, no. 3, pp. 200–222, 2001.

[2] "Portable Batch System (PBS) Web Site," 1998. [Online]. Available: http://www.nas.nasa.gov/Software/PBS

[3] S. Kannan, M. Roberts, P. Mayes, D. Brelsford, and J. Skovira, *Workload Management with LoadLeveler*. IBM Redbooks, November 2001.

[4] "SGE execution scripts Web Site," 2006. [Online]. Available: http://www.sun.com/software/gridware/

[5] I. Rodero, F. Guim, J. Corbalan, L. Fong, Y. Liu, and S. Sadjadi, "Looking for an Evolution of Grid Scheduling: Meta-brokering," *Grid Middleware and Services: Challenges and Solutions*, pp. 105–119, August 2008.

[6] "Latin American Grid (LA Grid) Web Site," 2008. [Online]. Available: http://latinamericangrid.org/

[7] N. Bobroff, L. Fong, Y. Liu, J. Martinez, I. Rodero, S. Sadjadi, and D. Villegas, "Enabling Interoperability among Meta-Schedulers," in *IEEE International Symposium on Cluster Computing and the Grid (CCGrid)*, Lyon, France, May 2008, pp. 306–315.

[8] J. Brooke, D. Fellows, K. Garwood, and C. Goble, "Semantic Matching of Grid Resource Descriptions," in *European Acrossgrids Conference*, Nicosia, Greece, January 2004, pp. 240–249, LNCS 3165.

[9] "HPC-Europa Project Web Site." [Online]. Available: http://www.hpc-europa.org

[10] "GridWay Project Web Site," 2008. [Online]. Available: http://www.gridway.org

[11] "KOALA Co-Allocating Grid Scheduler Web Site." [Online]. Available: http://www.st.ewi.tudelft.nl/koala/

[12] J. Seidel, O. Waldrich, W. Ziegler, P. Wieder, and R. Yahyapour, "Using SLA for Resource Management and Scheduling - a Survey, TR-0096," Institute on Resource Management and Scheduling, Tech. Rep., 2007.

[13] I. Rodero, F. Guim, J. Corbalan, and J. Labarta, "eNANOS: Coordinated Scheduling in Grid Environments," in *International Conference on Parallel Computing (ParCo)*, Malaga, Spain, 2005, pp. 81–88.

[14] F. Guim, J. Corbalan, and J. Labarta, "Modeling the Impact of Resource Sharing in Backfilling Policies Using the Alvio Simulator," in *Annual Meeting of the IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, Istambul, November 2007.

[15] T. Vazquez, E. Huedo, R. Montero, and I. Lorente, "Evaluation of a Utility Computing Model Based on the Federation of Grid Infrastructures," in *International Euro-Par Conference on Parallel Processing*, Rennes, France, August 2007, pp. 372–381.

[16] A. Iosup, D. Epema, T. Tannenbaum, M. Farrelle, and M. Livny, "Inter-Operable Grids through Delegated MatchMaking," in *International Conference for High Performance Computing, Networking, Storage and Analysis (SC07)*, Reno, Nevada, November 2007.

[17] "OGF Grid Scheduling Architecture Research Group (GSA-RG) Web Site." [Online]. Available: https://forge.gridforum.org/sf/projects/gsa-rg

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

[19] A. Kertesz, P. Kacsuk, I. Rodero, F. Guim, and J. Corbalan, "Meta-Brokering Requirements and Research Directions in State-of-the-art Grid Resource Management, TR-0116," Institute on Resource Management and Scheduling, Tech. Rep., 2007.

[20] A. Kertesz, I. Rodero, and F. Guim, "Meta-Brokering Solutions for Expanding Grid Middleware Limitations," in *Workshop on Secure, Trusted, Manageable and Controllable Grid Services (SGS) in conjunction with International Euro-Par Conference on Parallel Processing*, Gran Canaria, Spain, July 2008.

[21] "OGF Grid Interoperation Now Community Group (GIN-CG) Web Site." [Online]. Available: http://forge.gridforum.org/sf/projects/gin

[22] "OGF GLUE Working Group (GLUE) Web Site." [Online]. Available: http://forge.ogf.org/sf/projects/glue-wg

[23] O. Martin, J. Martin-Flatin, E. Martelli, P. Moroni, H. Newman, S. Ravot, and D. Nae, "The DataTAG transatlantic testbed," *Future Generation Computer Systems*, vol. 21, pp. 443–456, April 2005.

[24] "International Virtual Data Grid Laboratory (iVDGL) Web Site," 2008. [Online]. Available: http://igoc.ivdgl.indiana.edu

[25] "DataGrid Project Web Site," 2008. [Online]. Available: http://www.eu-datagrid.org

[26] "The Globus Project (Globus Alliance) Web Site," 2008. [Online]. Available: http://www.globus.org

[27] "Particle Physics Data Grid (PPDG) Web Site," 2008. [Online]. Available: http://www.ppdg.net

[28] "Grid Physics Networks (GriPhyn) Web Site," 2008. [Online]. Available: http://www.griphyn.org

[29] D. Erwin and D. Snelling, "UNICORE: A Grid Computing Environment," in *International Euro-Par Conference on Parallel Processing*, Manchester, UK, August 2001, pp. 825–834.

[30] "IBM Tivoli Dynamic Workload Broker Web Site." [Online]. Available: http://www-306.ibm.com/software/tivoli/products/
dynamic-workload-broker

[31] A. Helvaci, C. Cetinkaya, and M. Yildirim, "Using Rerouting to Improve Aggregate Based Resource Allocation," *Journal of Networks*, vol. 3, pp. 1–12, June 2008.

[32] S. Chapin, D. Katramatos, J. Karpovich, and A. Grimshaw, "The Legion Resource Management System," in *Job Scheduling Strategies for Parallel Processing (JSSPP)*, Puerto Rico, April 1999, pp. 162–178, LNCS 1659.

[33] "Globus Monitoring and Discovery System (MDS) Web Site." [Online]. Available: http://www.globus.org/mds/

[34] M. Massie, B. Chun, and D. Culler, "The Ganglia Distributed Monitoring System: Design, Implementation, and Experience," *Parallel Computing*, vol. 30, pp. 817–840, July 2004.

[35] N. Bobroff, G. Dasgupta, L. Fong, Y. Liu, B. Viswanathan, F. Benedetti, and J. Wagner, "A Distributed Job Scheduling and Flow Management System," *ACM Operating Systems Review*, vol. 42, pp. 63–70, January 2008.

[36] F. Guim, J. Corbalan, and J. Labarta, "Resource Sharing Usage Aware Resource Selection Policies for Backfilling Strategies," in *High Performance Computing & Simulation Conference (HPCS)*, Cyprus, June 2008.

[37] S. Chapin, W. Cirne, D. Feitelson, J. Jones, S. Leutenegger, U. Schwiegelshohn, W. Smith, and D. Talby, "Benchmarks and Standards for the Evaluation of Parallel Job Schedulers," in *Job Scheduling Strategies for Parallel Processing (JSSPP)*, April 1999, pp. 66–89, LNCS 1659.

[38] F. Guim and J. Corbalan, "A Job Self-Scheduling Policy for HPC Infrastructures," in *Job Scheduling Strategies for Parallel Processing (JSSPP)*, 2008, pp. 51–75, LNCS 4942.

[39] A. Iosup, H. Li, M. Jan, S. Anoep, C. Dumitrescu, L. Wolters, and D. Epema, "The Grid Workloads Archive," *Future Generation Computer Systems*, vol. 24, pp. 672–686, May 2008.

[40] "Grid Workloads Archive Web Site." [Online]. Available: http://gwa.ewi.tudelft.nl/

[41] "The Weather Research and Forecasting Model Web Site." [Online]. Available: http://www.wrf-model.org/index.php