Scalable Computing: Practice and Experience
Volume 11, Number 2, pp. 109–120. http://www.scpe.org
ISSN 1895-1767

MATCHING JOBS WITH RESOURCES: AN APPLICATION-DRIVEN APPROACH

A. CLEMATIS[†], A. CORANA[‡], D. D'AGOSTINO[†], A. GALIZIA[†], AND A. QUARATI[†]

Abstract. We present a distributed matchmaking methodology based on a two-level (low-level and application-level) benchmarking, that allows the specification of both syntactic and performance requirements. In particular, we point out how the use of application-level benchmarks gives a more accurate characterization of resources, so enabling a better exploitation of Grid power. The proposed methodology relies on the use of standard description languages at both application and resource sides, to foster interoperability. Moreover, the proposed tool is independent of the underlying middleware, and its distributed structure supports scalability.

Key words: grid platforms, benchmark-driven resource matchmaking, job submission languages extensions, interoperability

1. Introduction. Grid platforms supply users with a very large number of different resources to execute demanding applications. To exploit at best Grid power, efficient query and discovery tools are needed, able to provide a good matching of user requirements with resource characteristics. Unfortunately, Grid middleware offer only basic services for the retrieving of information about single resources, and thus they are often inadequate to describe more detailed and specific user requirements. So, usually, a matchmaking component (e.g. broker, matchmaker) manages over the middleware this supply-demand coupling process [1].

Some general criteria must be followed to provide a suitable and effective Grid matchmaker: a concise but as complete as possible description of application needs and resource properties, grounded on a common and shared basis, to assure interoperability; the management of both syntactic and performance requirements; the independence of the underlying middleware; a distributed structure, to allow scalability.

During past years we developed the tool GEDA (Grid Explorer for Distributed Applications), based on a distributed approach for Grid resource discovery, which combines a structured view of resources (single machines, homogeneous and heterogeneous clusters) at the Physical Organization (PO) level with an overlay network connecting the various POs [2, 3]. The GEDA architecture is modular and independent of the particular Grid middleware, although we worked with Globus Toolkit 4 [4]. The system is particularly suitable for discovering resources for structured parallel applications on large Grids.

To enhance the efficiency of the tool we develop a methodology to improve the matchmaking process based on information about performance of resources. Our aim is to supplement the basic information available via the Grid Information and Monitoring services by annotating resources with both low-level and application-specific performance metrics. These relevant aspects of resources could be examined by a broker to filter out the solutions that best fit application requirements.

Indeed, benchmarking is a widespread method to measure and evaluate performance of computer platforms [5]. Particularly, application-specific benchmarks are widely acknowledged tools in the High-Performance Computing (HPC) domain, to measure the performance of resources stressing simultaneously several aspects of the system. Notwithstanding, so far application benchmarks have not been extensively considered on the Grid, owing to various problems, such as very diversified types of applications, architectural complexity, dynamic Grid behavior, and heavy computational costs [6].

On this basis, we design GREEN (GRid Environment ENabler), a Grid service which represents an enhanced version of GEDA, whose main improvement is the management of benchmarks for a more precise characterization of resources. GREEN is a distributed matchmaker which complies with the above specifications, useful both for Grid administrators and users. It assists administrators in the insertion of benchmark information related to every PO composing the Grid, and provides users with features which a) facilitate the submission of job execution requests, by specifying both syntactic and performance requirements on resources; b) support the automatic discovery and selection of the most appropriate resources. The aim of GREEN is the discovery of the resources that satisfy user requirements and their ordering by performance ranking. The selection phase is left to a (meta)scheduler, allowing to apply the preferred scheduling policies to meet specific purposes.

An important point of our work is the use of both low-level and application-level benchmarks. Indeed, often it can occur that the rankings of resources based on low-level and application-level benchmarks are different,

[†]IMATI-CNR, Via De Marini 6, 16149 Genova, Italy
[‡]IEIIT-CNR, Via De Marini 6, 16149 Genova, Italy

with the second one usually closer to the effective performance obtainable by the user application. In this sense the use of application-level benchmarks allows a better exploitation of Grid resources.

Another important design goal of GREEN is interoperability. To this end, a unique standard language, namely JSDL (Job Submission Description Language) [7], is used to express job submission requirements, and an internal translation to the job submission languages used by the various middleware is performed. Middleware independence is pursued through an extension of JSDL in conformity with the GLUE (Grid Laboratory for a Uniform Environment) schema v. 2.0 [8]. Moreover, since we are interested in the execution of parallel applications, we borrowed from JSDL SPMD [9] some extensions to JSDL related to concurrency aspects.

This paper summarizes design principles, and provides an extended and modified version of the work [10]; main extensions regard the presentation of some experimental data on two different high performance platforms, and some preliminary results highlighting the usefulness of the proposed approach in a Grid environment.

The paper is organized as follow. Section 2 gives a brief overview on the state of the art about matchmaking and benchmarking on the Grid; Section 3 discusses the main contributions in the field of job and resource characterization languages. Section 4 briefly outlines the two-level benchmarking methodology, while Section 5 reports some preliminary experimental data collected on two parallel machines enlightening the usefulness of our approach. Section 6 gives a proof of concept of its adoption. Section 7 describes the design issues of GREEN, and an analysis of the extensions operated to existing languages. Section 8 gives some concluding remarks.

2. Related Works. The implementation of an efficient and automatic mechanism for the effective discovery of the resource that best suits a user job is one of the major problems in present Grids.

An important requirement is scalability, that is assured avoiding centralized structures; for example in [11] the Vigne tool is proposed, whose main features are a simple abstract view of resources, an application manager which selects resources using a resource allocator based on scalable and distributed discovery, and a decentralized overlay network. However, the tool does not support benchmark information.

The Globus toolkit does not provide a resource matchmaking/brokering as a core service, but the GridWay metascheduler [12, 13] was included as an optional high-level service since June 2007. GridWay provides dynamic scheduling, performance slowdown detection, opportunistic and on request migration, and fault recovery mechanisms. The main drawback of GridWay is that it allows users to specify only a fixed and limited set of resource requirements, most of them related to the queue policies of the underlying batch job systems. This choice limits the ranking of resources, and benchmarks are not considered at all.

On the contrary, gLite has a native matchmaking/brokering service that takes into account a richer set of requirements, including benchmark values [14]. However, this service is based on a semi-centralized approach, and may result in long waiting time in the job execution. Moreover, at the moment only the SPEC benchmark suite is considered, which mainly evaluates CPU performance; thus, the description of resources is partial, and can be inadequate to specific application requirements.

Work Binder [15] is a tool developed for the gLite middleware, based on the use of pilot jobs and aimed at assuring to incoming applications a fast access to computing resources; the tool is specifically designed to support interactive applications and on-demand computing, and can be adapted for different middleware.

A way to improve the efficiency of resource discovery, is to drive the search towards resources that shown good performance in the execution of jobs with similar or known behaviour. As explained in Section 3, the characterization of Grid resources based on pre-computed benchmarks seems a valid strategy to follow. The importance of benchmarking to evaluate resources in a Grid environment is largely acknowledged together with the criticalities that this task implies [16]. Actually, besides the set of interesting parameters to measure (e.g. CPU speed, memory size) different factors have to be taken into account when considering the execution of a benchmark suite on a Grid.

Several works proposed tools to manage and execute benchmarking on Grid. The Grid Assessment Probes [17] attempt to provide an insight into the stability, robustness, and performance of the Grid. The probes are designed to serve as simple Grid application exemplars and diagnostic tools. They test and measure performance of basic Grid functions, including file transfer, remote execution, and Grid Information Services (ISs) response.

The GridBench [18] is a modular tool aimed at exploring large-scale Grids in a interactive manner, taking into account performance aspects, adding new metrics to the basic ones supplied by middleware. It provides a graphical interface to define, execute and administrate benchmarks, also considering interconnection performance and resource workload. GridBench makes use of plug-ins to assure interoperability with the various middleware (currently Globus and gLite).

The NAS Grid Benchmark (NGB) suite [19] is defined by NASA, and represents typical activities of Computational Fluid Dynamics applications. It provides a set of computationally intensive benchmarks representative of scientific, post-processing and visualization workloads, and tests the Grid capabilities to manage and execute distributed applications. It uses four kinds of data-flow graphs according to parallel paradigms extracted from real applications in NASA.

All tools described above do not provide mechanisms for the submission of jobs and for their matching with resources. A brokering mechanism based on benchmarking of Grid resources is proposed in [20]. However, the scope of that broker is focused on the ARC middleware and the NorduGrid and SweGrid production environments, and it adopts xRSL, an extension of RSL (Resource Specification Language), to submit user's jobs. As a consequence, this approach lacks in generality and interoperability.

3. Resource and Job Characterization. To accomplish the matchmaking task, a proper description of resources is required at resource/owner and job/user side. To this end, different projects and research groups have proposed different languages.

At the resource side, adequate information is required to advertise resource's static (e.g. OS, number of processors) and dynamic (e.g. number of executing tasks, amount of free memory) properties. Actually, the main efforts in the direction of a standard resource description language come from the GLUE Working Group, which deployed the GLUE schema [8]. It is a conceptual model of Grid entities comprising a set of information specifications for Grid resources; an implementation through an XML Schema is given in [21]. As the schema has evolved during years, different versions have been used by various middleware, leading to the GLUE 2.0 specification. It allows the benchmarking characterization of resources by specifying the `Benchmark_t` complex type referencing benchmarks of type defined by `BenchmarkType_t`. Through the latter, declared as an open and extensible enumeration type, it is possible to specify a benchmark amongst a list of six values (e.g. specint2000, specfp2000, cint2006). However, other values compatible with the string type and with the recommended syntax are allowed.

At the user side, a job submission request expressed via a Job Submission Language (JSL), in addition to stating the application-related attributes (e.g. name and location of source code, input and output files), should express syntactic requirements (e.g. number of processors, main memory) and ranking preferences (if any) to guide and constraint the matching process on resources.

The Job Description Document (JDD) [22], introduced by Globus Alliance with the Web Services versions of the Globus Toolkit, defines an XML language closer to the XMLish dialects used in the Web Services Resource Framework (WSRF) family. The main purpose of a JDD document is to set the parameters for the correct execution of a job. The selection of the facilities to use has to be performed in advance by interacting with the WS MDS services of the available resources. In the JDD schema, it is possible to specify only few requirements, as the minimum amount of memory, or to set useful information as the expected maximum amount of CPU time. It is however possible to extend the schema with user-defined elements.

The European Data Grid Project proposed the Job Description Language (JDL), afterwards adopted by the EGEE project [23]. A JDL document contains a flat list of argument-value pairs, specifying two classes of job properties: job specific attributes and resources-related properties (e.g. `Requirements` and `Ranks`) used to guide the matching process towards the most appropriate resources. These values can be arbitrary expressions using the fields published by the resources in the MDS, and are not part of the predefined set of attributes for the JDL, as their naming and meaning depend on the adopted Information Service schema. In this way, JDL is independent of the resources information schema adopted.

The Job Submission Description Language (JSDL) developed by the JSDL- Working Group [7] of the Global Grid Forum, aims to synthesize consolidated and common features available in other JSLs, obtaining a standard language for the Grid. JSDL contains a vocabulary and normative XML Schema facilitating the declaration of job requirements as a set of XML elements. Likewise JDL, job attributes may be grouped in two classes. The `JobIdentification`, `Application` and `DataStaging` elements describe job-related properties. The `Resources` element lists some of the main attributes used to constraint the selection of the feasible resources (e.g. `CPUArchitecture`, `FileSystem`, `TotalCPUTime`). Since only a rather reduced set of these elements is stated by the JSDL schema, an extension mechanism is foreseen. Examples of JSDL extensions able to capture a more detailed description of the degree of parallelism of jobs are presented in [9, 24].

In Section 7 we present our proposal aimed at extending GLUE and JSDL with elements capable of accounting for the association of benchmarks data at both user and resource sides.

4. A Two-Level Benchmarking Methodology. To describe Grid resources, we propose a two-level methodology aimed at giving a useful enriched description of resources, and at facilitating the matchmaking process. Our methodology considers two approaches: I) the use of micro-benchmarks to supply a basic description of resource performance; II) the deployment of application-driven benchmarks to get a closer insight into the behavior of resources under more realistic conditions of a class of applications. Application-driven benchmarks can consist of:

a) the application itself, often in a light version obtained choosing a reference input data set and specific parameters to avoid long executions, obtaining in the meantime a representative run of the real application;
b) a suitable benchmark or benchmark suite belonging to the same class of the application of interest (e.g. the LINPACK benchmark for the class of linear algebra applications).

Through application-driven benchmarks, it is possible to add an evaluation of the resources on the basis of the system indicators that are more stressed by an application. Our present aim is to provide a proper description of each Grid resource in isolation, i. e. without considering complexity aspects of Grid environments. Future developments of our work foreseen to address more complex scenarios.

4.1. Micro-Benchmarks. In order to supply a basic resource characterization, mainly based on low-level performance capacity, we consider the use of traditional micro-benchmarks. To this aim, a reasonable assumption is that the performance of a machine mainly depends on the CPU, the memory and the cache, and on the interconnection network [25]; therefore, we choose a concise number of parameters to evaluate in order to provide an easy-to-use description of the various nodes. Table 4.1 shows resource properties and related metrics measured by the micro-benchmarks we employed.

TABLE 4.1
Low-level benchmarks and related metrics.

Resource Capability	CPU	Memory	Memory-Cache	Interconnection	I/O
Metric	MFLOPS	MB/sec	MB/sec	MB/sec	MB/sec
Benchmark	Flops	Stream	CacheBench	Mpptest	Bonnie

Flops provides an estimate of peak floating-point performance (MFLOPS); Stream is the standard benchmark for measuring sustained memory bandwidth, as it works with datasets much larger than the available cache; CacheBench is designed to characterize the performance of possibly multiple levels of cache present on the processor; Mpptest measures the performance of some basic MPI communication routines; Bonnie performs a series of tests on a file of known size (default 100 MB).

The micro-benchmarks used in this phase generally return more than one value; so, to obtain results easily usable in the matchmaking process, we considered for each benchmark synthetic parameters or the most significant value. They are used to characterize resources by populating the benchmark description managed by GREEN.

4.2. Application-Specific Benchmarks. Micro benchmarks are a good solution when the user has little information about the job she/he is submitting, and for applications that are not frequently executed. Indeed, very often the participants to a Virtual Organization have similar aims, and therefore it is possible to identify a set of the most used applications. In these cases the most suitable approach is to evaluate system performance through application-specific benchmarks that approximate at best the real application workload. This kind of benchmarks represent the second level of our methodology.

As case studies for this level we considered some applications of our interest, i. e. image processing, isosurface extraction, and linear algebra. For the first two classes of applications, we choose a light version code aiming to emphasize precise aspects of the considered metrics. With respect to image processing, we selected a compute intensive elaboration applied to a reference image of about 1 MB; in this way CPU metrics are mainly stressed. The isosurface extraction application provides a more exhaustive performance evaluation of the system, as also I/O operations are heavily involved. In this case, we considered the processing of a small 3D data set of 16 MB, producing a mesh made by 4 million triangles. On the contrary, to represent the class of applications based on linear algebra, we used the well known Linpack benchmark [26]. For application-driven benchmarks, the metric considered to characterize resources is execution time. Similarly to the micro-benchmarks case, the results are stored in the internal data structures of GREEN.

5. Benchmarking resources. To evaluate the effectiveness of our methodology in resource characterization, we performed some experiments on several resources normally used to deploy and execute our applications. For sake of simplicity, here we focus on two specific high-performance resources: 1) the Michelangelo system,

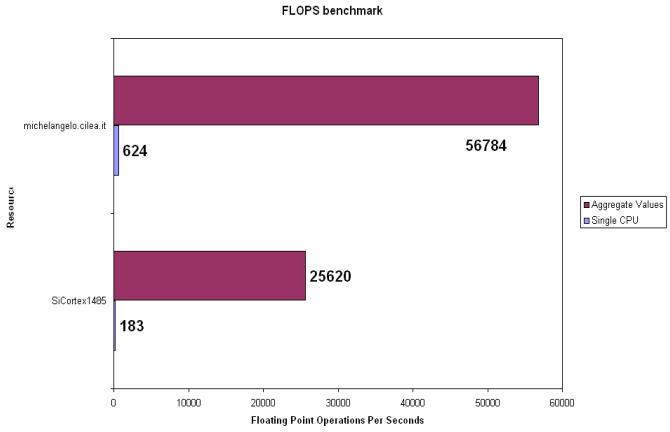

FIG. 5.1. *Comparison between resources according to FLOPS benchmark.*

made up of 53 nodes interconnected by a Gigabit switched Ethernet. As a whole, the system provides 212 AMD Opteron 275 dual core with a clock rate of 2.2 GHz. Each CPU is equipped with 2 GB RAM, and the total shared storage amounts to 30 TB [27]. 2) the SiCortex SC1458 system with 243 SiCortex node chips, each equipped with six cores, and linked by a proprietary interconnection network supporting a large message bandwidth of 4 GBytes/sec. This system pursues the Green Computing guidelines, through extremely low energy consumption [28].

By a quick comparison clearly emerges that the two resources greatly differ both in terms of the total number of CPUs and in terms of single CPU performance. In fact, SC1458 has a greater number of CPUs than the Michelangelo cluster, but the latter has faster CPUs. Despite these technical differences from which one may infer consequent performance results, this expectation is contradicted by our experiments as shown by the following discussion.

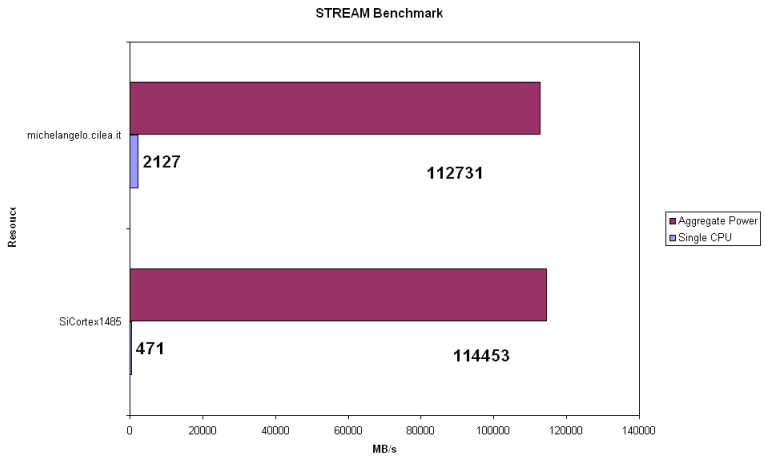

FIG. 5.2. *Comparison between resources according to STREAM benchmark.*

Starting from micro-benchmark results, the SC1458 achieves better performance in almost all cases and parameters evaluated, when considering aggregate computing power. However, its single cores have relatively low performance compared with the single CPU of the Michelangelo cluster, and the actual power of the resource

derives from the high number of provided cores and the native fast connection among processes. To outline CPU performance we depicted in Figures 5.1 and 5.2 the results obtained with FLOPS and STREAM benchmarks.

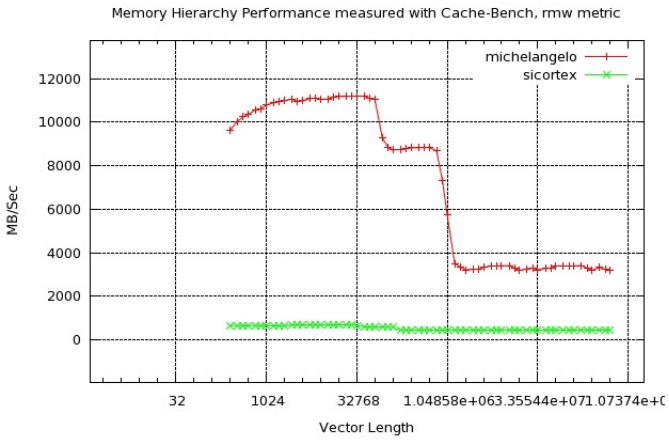

FIG. 5.3. *Comparison between resources according to CacheBench.*

Both benchmarks have been run on a CPU/core independently, and then the aggregated results are gathered to represent the performance of the whole parallel resources [16].

With respect to the cache evaluation, Figure 5.3 shows that Michelangelo performs better for all vector lengths. On the contrary, with respect to interconnection evaluation, the SC1458 achieved definitely better performance, as reported in Figure 5.4. We tested point-to-point communication performance, through the MPPTest benchmark; results are expressed in MB/Sec. As mentioned above, the Michelangelo Cluster employs a Gigabit switched Ethernet, while SC1458 has a proprietary interconnection network that performs significantly better.

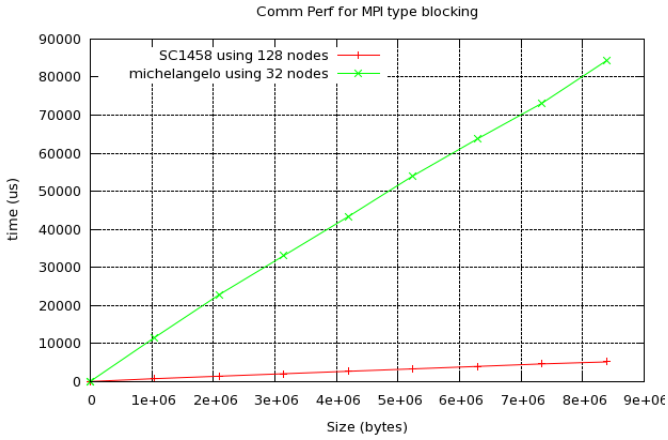

FIG. 5.4. Comparison between resources according to MPPTest benchmark.

Considering the second level of benchmark, the situation is quite different. In fact, depending on the application domain, best results are achieved alternatively by the two resources.

We conducted our tests considering the execution times (Wall Clock Time) as metric to evaluate performance. The results are normalized according to a base value; to this end, we adopted the values returned from the Michelangelo cluster. Tables 5.1 and 5.2 report the values obtained for Image Processing (IP), Isosurface Extraction (IE) and High-Performance Linpack (HPL) benchmarks. In the latter case, we examined separately the use of different sets of processors (32, 64 and 128 for both). Due to the chosen metric, lower values correspond to better execution times.

TABLE 5.1
Application-level benchmarks, execution time normalized with respect to Michelangelo.

	Michelangelo	**SC1458**
Isosurface Extraction	1	6.1
Image Processing	1	3.2

TABLE 5.2
Application-level benchmarks, execution time normalized with respect to Michelangelo.

	Michelangelo			**SC1458**		
	32p	**64p**	**128p**	**32p**	**64p**	**128p**
HPL	1	0.3	0.2	0.44	0.13	0.08

Table 5.1 shows that Michelangelo cluster performed significantly better considering the Image Processing and the Isosurface Extraction applications. Instead, Table 5.2 reporting the results related to the HPL benchmark highlights that SC1458 outperforms Michelangelo up to a factor 3, when increasing the number of processes. This behaviour depends on the different requirements of the various applications. As to Image Processing and Isosurface Extraction resulted that they benefit from fast single CPU and cache memory, while HPL tests the entire system and benefits from high number of processes linked with fast connections. Starting from these remarks, it is quite evident that the Michelangelo cluster is faster in the execution of IP and IE, while it poorly performs with respect to HPL. On the contrary, with respect to HPL, SC1458 outdoes the Michelangelo Cluster, but it does not achieve good results on the considered image processing operations and isosurface extraction.

Following our methodology, the differences in the performance of both resources in each level of benchmark clearly emerge. SC1458 definitely outperforms Michelangelo with respect to almost all micro-benchmarks. However, considering the second level of benchmark, the Michelangelo cluster appears as the suitable choice for the execution of specific applications. This performance divergence also occurred in other similar comparisons we conducted for the other benchmarks executed against the resources normally used to deploy and execute our applications. This behaviour testifies the appropriateness of our approach.

6. A proof of concept. To exemplify the potentialities of our methodology, we introduce a simplified evaluation that highlights the benefits of adopting a benchmark aware matchmaker. Let us consider a simplified Grid scenario setting three jobs (J1, J2 and J3) belonging to the Image Processing and Linear Algebra classes, that are executed against SC1458 and Michelangelo, denoted R1 and R2 respectively.

Table 6.1 reports the relative spped of resources R1 and R2 with respect to the two classes of applications. As shown in the previous Section, Michelangelo performs three times better than SC1458 for Image Processing applications, while it is about two times slower than SC1458 for Linear Algebra applications.

TABLE 6.1
Relative speed of resources with respect to the applications.

Applications	**R1**	**R2**
Image Processing	3	1
Linear Algebra	0.43	1

Table 6.2 lists the jobs. In particular, J1 and J3 are two Image Processing jobs, and J3 is computationally heavier since it processes a larger image. J2 is a Linear Algebra job. For each job the computational time required on the two resources is reported, expressed in seconds; the first column also shows the temporal instant at which the job is submitted.

In Figure 6.1-(a) we depicted the case in which our benchmark-driven methodology does not apply, no matchmaker operates and jobs are scheduled on the first available resource in a FIFO order. In this case, when J1 arrives at time 0, it is mapped to R1 which employs 60 seconds to process it. At time 10, J2 arrives and is mapped to R2, which executes it in 80 seconds. When J3 arrives at time 20 no resources are available, hence J3 is queued until one is released. This happens at time 60, when J1 terminates and J3 is assigned to R1 which takes 120 seconds to run it, ending at time 180.

TABLE 6.2
Job characteristics.

Job	Comp. Time on R1 (sec)	Comp. Time on R2 (sec)	Application
J1(0)	60	20	Image Processing
J2(10)	35	80	Linear Algebra
J3(20)	120	40	Image Processing

Let us now consider the case in which our methodology is applied and benchmark information is used to describe resources, see Figure 6.1-(b). Now the matchmaker is at work and, when asked for a suitable resource to execute the Image Processing job J1, it returns R2, since it ranks better than R1 on that job class. Running three times faster, R2 employs just 20 seconds to execute J1. When J2 arrives at time 10, R1 is selected and it takes about 35 seconds to complete the Linear Algebra job. Finally, J3 arrives and it is assigned to R2 that in the meantime becomes free, and the job ends after 40 seconds. The overall computation in this case ends at time 60.

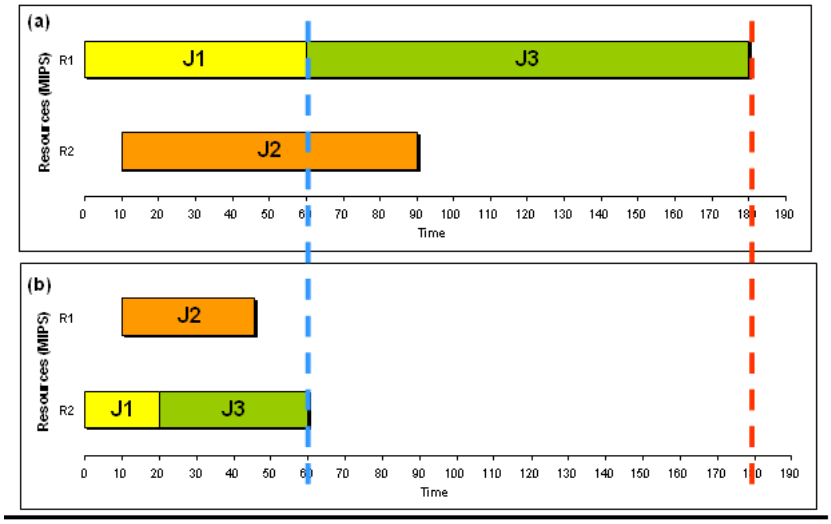

FIG. 6.1. *(a) scheduling of three jobs without any matchmaker, (b) improvement of execution times when a benchmark-driven matchmaker is applied.*

From our example it clearly emerges that the use of benchmark information could be adopted to improve scheduling strategy, raising the performance of the overall execution.

7. Benchmark-Driven Matchmaking. Due to the huge gap separating users and resources, tools that allow the two parts to better come to an agreement are highly useful. In [2, 3] we presented GEDA, a Grid service based on a distributed and cooperative approach for Grid resource discovery. It supplies users with a structured view of resources (single machines, homogeneous and heterogeneous clusters) at the PO level, and leverages on an overlay network infrastructure which connects the various POs constituting a Grid. For each PO, a GEDA instance is deployed to keep updated information about the state of all PO's resources, and to exchange them with other GEDA instances in the discovery phase.

In the present work, we describe an advanced version of GEDA, called GREEN, able to characterize Grid resources through benchmark evaluations. In this context, acting as a distributed matchmaker, GREEN manages and compares the enriched view of resources with user-submitted jobs, with the goal of selecting the most appropriate resource(s). Operating at intermediate level between applications (e.g. schedulers) and Grid middleware, GREEN aims to discover the whole set of resources satisfying user requirements ordered by ranks. The selection of a particular resource is left to a (meta)scheduler to which the resources set is forwarded; so it is possible to apply the preferred scheduling policies to optimize Grid throughput or other target functions

(e.g. response times, QoS). Once the "best" resource is chosen, GREEN will be re- invoked to carry-out the submission of the job on it, via the Execution Environment (EE).

7.1. Benchmarking Grid Resources. GREEN supplies Grid administrators with the facility of submitting, executing benchmarks (both micro and application-related) against the resources belonging to a certain administrative domain (PO), and storing results.

To reflect the underlying view of Grid resources offered by the GLUE 2.0 specification language, and to support the matching mechanism (i. e. the comparison with resources information contained in the previously acquired XML) the benchmark-value copies are directly represented as GLUE entities according to the XML reference realizations of GLUE 2.0. By employing the openness of `BenchmarkType_t` (as recalled in Section 3), the set of recognized benchmarks is extensible without any change in the document schema. An example of a benchmark document related to the execution of micro-benchmark Flops against the cluster identified by the IP 150.145.8.160, resulting in 480 MFlops is:

```
<Benchmark>
      <LocalID>150.145.8.160</LocalID>
      <Type>MFlops</Type>
      <Value>480</Value>
      <BenchLevel>micro</BenchLevel>
</Benchmark>
```

Through the use of the extension mechanism defined in GLUE specification, we enriched the `Benchmark_t` type by adding the element `BenchLevel` which specifies the benchmark level (by accepting the two string values `micro` and `application`) according to our two-level methodology.

Once a benchmark is executed and its results collected, an XML fragment, similar to the one reported above, is created for each resource and inserted in an XML document (namely *Benchmark image*), managed by GREEN, which collects all benchmark evaluations for the PO.

7.2. Extending JSDL. The counterpart of benchmarking resources is the ability for users submitting a job to express their preferences about the performance of target machines. Resources are then ordered according to performance values (ranks). As explained in Section 3, both JDD and JSDL do not provide any construct able to express some preferential ordering on selected resources. We add the element `Rank` (of complex type `Rank_Type`) devoted to this task, which embeds a sub-element `BenchmarkType_t` corresponding to the one contained in our extension of the GLUE schema. In the context of JSDL, the `Value` sub-element (see list below) is to be intended as a threshold to be satisfied by the corresponding `Value` (related to the benchmark stated by `Type`) contained in the `Benchmark` element of any resource to be selected by the matchmaker before the ranking takes place.

As we are interested in the execution of parallel applications, we borrowed from SPMD [9] an extension to JSDL that supports users with a rich description set of applications and resources related to concurrency aspects (e.g. number of processes, processes per host). The following is an example of an extended JSDL document, containing information related to parallel requirements, along with our extension to rank resources on benchmark specification. The document is requesting for nodes able to execute the application-level "IsoSurface_Benchmark" in no more than 300 time units. Note how the `Rank` element has been located inside the `Resource` one, according to the extension mechanism included by JSDL schema.

```
<?xml version="1.0" encoding="UTF-8"?>
<jsdl:JobDefinition
xmlns:jsdl="http://schemas.ggf.org/jsdl/2005/11/jsdl"
xmlns:jsdl-posix="http://schemas.ggf.org/jsdl/2005/11/jsdl-posix"
xmlns:jsdl-spmd="http://schemas.ogf.org/jsdl/2007/02/jsdl-spmd"
xmlns:jsdl-
rank="http://saturno.ima.ge.cnr.it/ima/PONG/jsdl/2009/01/jsdl-rank">
  <jsdl:JobDescription>
    <jsdl:Application>
      <jsdl:ApplicationName>ParIsoExtrctn</jsdl:ApplicationName>
      <jsdl-spmd:SPMDApplication>
        <jsdl-posix:Executable>parisoextraction</posix:Executable>
```

```
        <jsdl-posix:Argument> inputvolume.raw</posix:Argument>
        <jsdl-posix:Argument>200</posix:Argument>
        <jsdl-posix:Output>isosurface.raw</posix:Output>
        <jsdl-spmd:NumberOfProcesses>4</spmd:NumberOfProcesses>
        <jsdl-spmd:ProcessesPerHost>2</spmd:ProcessesPerHost>
        <jsdl-spmd:SPMDVariation>http://www.ogf.org/jsdl/2007/02/
                                  jsdl-spmd/MPICH2</>
      </jsdl-spmd:SPMDApplication>
    </jsdl:Application>
    <jsdl:Resources>
      <jsdl:OperatingSystemType>
        <jsdl:OperatingSystemName>LINUX</jsdl:OperatingSystemName>
      </jsdl:OperatingSystemType>
      <jsdl-rank:Rank>
        <jsdl-rank:Type>IsoSurface_Benchmark</rank:Type>
        <jsdl-rank:Value>300</rank:Value>
        <jsdl-rank:BenchLevel>application</rank:BenchLevel>
      </jsdl-rank:Rank>
    </jsdl:Resources>
  </jsdl:JobDescription>
</jsdl:JobDefinition>
```

7.3. Distributed Matchmaking Process. The main components of a GREEN instance along with some of their interactions with other middleware services, notably IS and EE, by considering a Grid composed of several POs are shown in Figure 7.1. In the following, we summarise their roles and the behaviours.

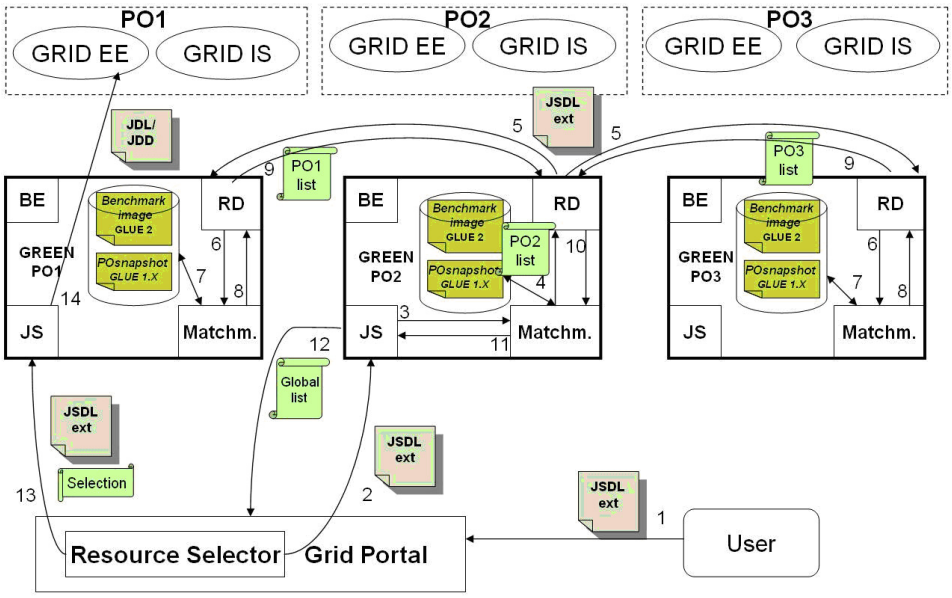

FIG. 7.1. Example of the matching phase with various GREEN instances.

The Job Submission (JS) component receives requests of jobs submission initiated by users; depending on the activation mode it behaves just like a messages dispatcher or as a translator of JSL documents, carrying out their subsequent submission to the EE. The Benchmark Evaluation (BE) supports administrators in the performance-based characterization of PO resources. The Resource Discovery (RD) is in charge of feeding GREEN with the state of Grid resources. RD operates both locally and globally by carrying out two tasks: 1) to discover the state of the PO resources; 2) to dispatch requests to other GREEN instances. As to the

first task, RD dialogues with the underlying IS (e.g. MDS, gLite IS) that periodically reports the state of the PO in the form of an XML file conformed to the GLUE version adopted by the underlying middleware. This document (namely the *PO snapshot*) is stored, as it is, and managed by GREEN to answer to external queries issued by various clients (e.g. other GREEN instances, meta-schedulers). To accomplish the dispatching task, RD handles the so-called neighbors view. Depending on the number of POs, i. e. GREEN instances running, their management could consider different strategies, whose description is beyond the scope of the paper.

To deal with different underlying middleware transparently to Grid users and applications, the syntactic differences among the various versions of GLUE are managed by GREEN through a conversion mapping at matching time. The Matchmaker performs the matching among resources in the Grid, and their subsequent ranking, with the requirements expressed by the users through the application submission document.

More in detail, let us consider the case in which a user submits an extended JSDL document through a Grid portal (1). The document is managed by the Resource Selector component, which initiates the distributed matchmaking by forwarding it to the JS component of a randomly selected GREEN instance (2) (e.g. PO2). JS activates the Matchmaker (3). This instance of matchmaker, namely the Master Matchmaker (MM), is responsible to provide the set of candidate resources to the Resource selector for this specific request. MM through RD forwards the document to all the other known GREEN instances and contemporaneously checks its local memory (4-5). All the matchmakers filter their *PO snapshot* selecting the set of PO resources satisfying the query. By analyzing the pre-computed *Benchmark image*, the satisfying resources with a `Value` element (for the chosen benchmark) that fulfils the threshold fixed in the corresponding `Rank` element of the JSDL document are extracted. The resources identifiers and their corresponding benchmark values are included in a list, called *PO list* which is returned to MM (6-10). MM merges these lists with its own PO list, producing a Global List ordered on the ranking values. The Global list is passed to JS (11) which returns it back to RS (12). Besides applying the selection policy to determine the resource to use, the Resource Selector calls the JS of the GREEN responsible of the PO owning the selected machine (GREEN PO1's instance in our case), by sending it the extended JSDL document along with the data identifying the selected resource (13). JS translates the information regarding the job execution of the original JSDL document in the format proper of the specific PO middleware, stating the resource on which the computation takes place. In particular, it will produce a JDD document for GT4 resources or a JDL document for the gLite ones. Finally, it activates the Execution Environment in charge of executing the job represented in the translated document (14).

8. Conclusions. The efficient matchmaking of application requirements with characteristics of resources is a very important issue in Grid computing, and developing a satisfactory solution may greatly improve the usefulness of Grid platforms for a large class of potential applications. However this is not an easy task, owing to the high abstraction level of Grid platforms which causes a semantic gap between application requirements and resource properties, the large number of possible Grid applications, the large number of available resources, the dynamicity of the Grid environment.

To reduce this gap we designed GREEN, a distributed matchmaker which provides Grid users with features to make easier the submission of job execution requests containing performance requirements, in order to support the automatic discovery and selection of the most suitable resource(s). GREEN relies on a two-level benchmarking methodology: resources are characterized by means of their performance evaluated through the execution of low-level and application-specific benchmarks. According to our methodology, every resource of a PO is tagged with the results obtained through the execution of the two levels of benchmarks and hence it is selectable, on a performance basis, during the matchmaking phase. A preliminary analysis outlined promising results; thus future efforts are planned in the direction of a deeper evaluation of our proposal in the context of a simulated Grid environment with particular emphasis on scheduling policies.

To ensure a good degree of independence from the underlying middleware, GREEN leverages on two standards such as JSDL and GLUE, that have been properly extended to manage the performance-based description of resources.

Acknowledgments. This work has been partially supported by project TECDOC-SIIT Under L.297 Miur Programme.

REFERENCES

[1] X. Bai, H. Yu, Y. Ji and D. C. Marinescu, *Resource matching and a matchmaking service for an intelligent Grid*, Int. Journal of Computational Intelligence, 1 (2004), pp. 163–171.

[2] A. Clematis, A. Corana, D. D'Agostino, V. Gianuzzi and A. Merlo, *Resource selection and application execution in a Grid: A migration experience from GT2 to GT4*, in Lecture Notes in Computer Science n. 4276 (OTM 2006), R. Meersman, Z. Tari et al., eds., Springer, Berlin, 2008, pp. 1132–1142.

[3] A. Clematis, A. Corana, D. D'Agostino, V. Gianuzzi, A. Merlo and A. Quarati, *A distributed approach for structured resource discovery on Grid*, in Proc. Int. Conference on Complex, Intelligent and Software Intensive Systems (CISIS 2008, Barcelona), IEEE Computer Society, 2008, pp. 117–125.

[4] I. Foster, *Globus Toolkit Version 4: Software for Service-Oriented Systems*, in Lecture Notes in Computer Science n. 3779 (Proc. IFIP Int. Conference on Network and Parallel Computing), Springer, Berlin, 2005, pp. 2–13.

[5] R. W. Hockney, *The science of computer benchmarking. Software, environments, tools*, SIAM, Philadelphia, 1996.

[6] F. Nadeem, R. Prodan, T. Fahringer and A. Iosup, *Benchmarking Grid Applications for Performance and Scalability Predictions*, in Grid Middleware and Services: Challenges and Solutions (CoreGRID series), D.Talia, R.Yahyapour and W.Ziegler, eds., Springer, 2008, pp. 19–37.

[7] A. Anjomshoaa, F. Brisard, M. Drescher, D. Fellows, A. Ly, S. McGough, D. Pulsipher and A. Savva, *Job Submission Description Language (JSDL) Specification v1.0*, Grid Forum Document GFD. 56, Open Grid Forum (OGF), 2005.

[8] S. Andreozzi et al., *GLUE Specification v. 2.0 (rev. 3)*, Open Grid Forum (OGF), 2009.

[9] A. Savva (Ed.), *JSDL SPMD Application Extension, Version 1.0*, Grid Forum Document GFD. 115, Open Grid Forum (OGF), 2007.

[10] A. Clematis, A. Corana, D. D'Agostino, A. Galizia and A. Quarati, *Performance based matchmaking on Grid*, in Lecture Notes in Computer Science n. 6068 (Proc. Int. Conf. on Parallel Processing and Applied Mathematics (PPAM 2009), Wroclaw (Poland), September 13-16, 2009), R. Wyrzykowski, ed., Springer, Berlin, 2010, pp. 174–183.

[11] E. Jeanvoine, L. Rilling, C. Morin and D. Leprince, *Using Overlay Networks to Build Operating System Services for Large Scale Grids*, Scalable Computing: Practice and Experience, 8 (2007), pp. 229-239.

[12] E. Huedo, R. S. Montero and I. M. Llorente, *A Framework for Adaptive Scheduling and Execution on Grids*, Software: Practice and Experience, 34 (2004), pp. 631–651.

[13] E. Huedo, R. S. Montero and I. M. Llorente, *The GridWay Framework for Adaptive Scheduling and Execution on Grids*, Scalable Computing: Practice and Experience, 6 (2005), pp. 1–8.

[14] *gLite 3.1 User Guide*, Doc. CERN-LCG-GDEIS-722398, January 2009 `https://edms.cern.ch/file/722398/1.2/gLite-3-UserGuide.html`

[15] B. Marović, M. Potočnik and B. Čukanović, *Multi-application bag of jobs for interactive and on-demand computing*, Scalable Computing: Practice and Experience, 10 (2009), pp. 413-418.

[16] M. D. Dikaiakos, *Grid benchmarking: vision, challenges, and current status*, Concurrency and Computation: Practice and Experience, 19 (2007), pp. 89–105.

[17] G. Chun, H. Dail, H. Casanova and A. Snavely, *Benchmark probes for Grid assessment*, in 18th Int. Parallel and Distributed Processing Symposium (IPDPS 2004), Santa Fe (USA), IEEE Computer Society, 2004, pp. 276.

[18] G. Tsouloupas and M. D. Dikaiakos, *GridBench: A Tool for the Interactive Performance Exploration of Grid Infrastructures*, Journal of Parallel and Distributed Computing, 67 (2007), pp. 1029–1045.

[19] M. Frumkin and R. F. Van der Wijngaart, *NAS Grid Benchmarks: A tool for Grid space exploration*, Cluster Computing, 5 (2002), pp. 315-324.

[20] E. Elmroth and J. Tordsson, *Grid resource brokering algorithms enabling advance reservations and resource selection based on performance predictions*, Future Generation Computer Systems, 24 (2008), pp. 585–593.

[21] S. Andreozzi (Ed.), *GLUE v. 2.0 Reference Realizations to Concrete Data Models*, Open Grid Forum (OGF), 2008.

[22] M. Feller, I. Foster, S. Martin, *GT4 GRAM: A Functionality and Performance Study*, in Proc. 2007 TeraGrid Conference, 2007, Madison (USA) `http://www.globus.org/toolkit/docs/4.2/4.2.0/user/gtuser-execution.html`

[23] *Job Description Language Attributes Specification for the gLite Middleware*, Doc. EGEE-JRA1-TEC-555796-JDL-Attributes-v0-8, May 2006 `https://edms.cern.ch/file/555796/1/EGEE-JRA1-TEC-555796-JDL-Attributes-v0-8.pdf`

[24] I. Rodero, F. Guim, J. Corbal and J. Labarta, *How the JSDL can Exploit the Parallelism?*, in Sixth IEEE Int. Symposium on Cluster Computing and the Grid (CCGRID'06), 2006, pp. 275–282.

[25] G. Tsouloupas and M. Dikaiakos, *Characterization of Computational Grid Resources Using Low-level Benchmarks*, in Proc. 2nd IEEE Int. Conference on e-Science and Grid Computing, IEEE Computer Society, 2006, pp. 70.

[26] J. J. Dongarra, P. Luszczek and A. Petitet, *The LINPACK benchmark: Past, present, and future*, Concurrency and Computation: Practice and Experience, 15 (2003), pp. 1–18.

[27] Michelangelo Hardware, LITBIO Project `http://www.supercomputing.it/At_Cilea/michelangelo_eng.htm`

[28] SiCortex Home page, `http://sicortex.com/`

Edited by: Ewa Deelman
Received: March 30, 2010
Accepted: May 25, 2010