Development of a GT4-Based Resource Broker Service: An Application to On-Demand Weather and Marine Forecasting

R. Montella

Dept. of Applied Science, University of Naples "Parthenope" – Italy

Abstract. The discovery and selection of needed resources, taking into account optimization criteria, local policies, computing and storage availability, resource reservations, and grid dynamics, is a technological challenge in the emerging technology of grid computing.

The Condor Project's ClassAd language is commonly adopted as a "*lingua franca*" for describing grid resources, but Condor itself does not make extensive use of Web Services. In contrast, the strongly service-oriented Globus Toolkit is implemented using the web services resource framework, and offers basic services for job submission, data replica and location, reliable file transfers and resource indexing, but does not provide a resource broker and matchmaking service.

In this paper we describe the development of a Resource Broker Service based on the Web Services technology offered by the Globus Toolkit version 4 (GT4). We implement a fully configurable and customizable matchmaking algorithm within a framework that allows users to direct complex queries to the GT4 index service and thus discover any published resource. The matchmaking algorithm supports both the native simple query form and the Condor ClassAd notation. We achieve this flexibility via a matchmaking API java class framework implemented on the extensible GT4 index service, which maps queries over ClassAds in a customizable fashion.

We show an example of the proposed grid application, namely an on demand weather and marine forecasting system. This system implements a Job Flow Scheduler and a Job Flow Description Language in order to access and exploit shared and distributed observations, model software, and 2D/3D graphical rendering resources. The system combines GT4 components and our Job Flow Scheduler and Resource Broker services to provide a fully grid-aware system.

1 Introduction

Our proposed grid infrastructure is based on the Globus Toolkit [1] version 4.x (GT4) middleware, developed within the Globus Alliance, a consortium of institutions from academia, government, and industry. We choose GT4 because it exposes its features, including service persistence, state and stateless behavior, event notification, data element management and index services, via the web services resource framework (WSRF).

The brokering service that we have developed is responsible for interpreting requests and enforcing virtual organization policies on resource access, hiding many

C. Cérin and K.-C. Li (Eds.): GPC 2007, LNCS 4459, pp. 204 – 217, 2007.

details involved in locating suitable resources. Resources register themselves to the resource broker, by performing an availability advertisement inside the virtual organization index [4]. These entities are classified as resource producers using many advertisement techniques, languages and interfaces. Resource are often discovered and collected by means of a performance monitor system and are mapped in a standard and well known description language [5] such as the Condor [8] ClassAd [9]. Ideally, the entire resource broking process can be divided into two parts. First, a matchmaking algorithm finds a set of matching resources using specific criteria such as "all submission services available on computing elements with at least 16 nodes using the PBS local scheduler and where the MM5 [3] weather forecast model is installed." Then, an optimization algorithm is used to select the best available resource among the elements [6]. Usually, the broker returns a match by pointing the consumer directly to the selected resource, after which the consumer contacts the resource producer. Alternatively, the client may still use the resource broker as an intermediary. When the resource broker selects a resource, the resource is tagged as claimed in order to prevent the selection of the same resource by another query with the same request. The resource will be unclaimed automatically when the resource broker catalogue is refreshed reflecting the resource state change [10].

In this scenario, the resource broker service is a key element of grid-aware applications development. Thus, users can totally ignore where their data are processed and stored, because the application workflow reacts to the dynamic nature of the grid, adapting automatically to the resource request and allocation according to grid health and status.

The allocation and scheduling of applications on a set of heterogeneous, dynamically changing resources is a complex problem without an efficient solution for every grid computing system. Actually, the application scenario and the involved resources influence the implemented scheduler and resource broker system while both the implicit complexity and the dynamic nature of the grid do not permit an efficient and effective static resource allocation.

Our demo applications are based on the use of software for the numerical simulation in environmental science, and are built and developed using a grid computing based virtual laboratory [11]. Weather and marine forecasts models need high performance parallel computing platforms, to ensure an effective solution and grid computing is a key technology, allowing the use of inhomogeneous and geographically-spread computational resources, shared across virtual organization. The resource broker service is the critical component to transform the grid computing environment in a naturally used operational reality. The buildup of grid-aware environmental science applications is a "grand challenge" for both computer and environmental scientists, hence on-demand weather forecast is used by domain experts, common people, amateur and enthusiasts sailing racers.

In this paper we describe the implementation of a GT4-based Resource Broker Service and the application of this component to a grid-aware dynamic application, developed using our grid based virtual laboratory tools. The resource broker architecture and design is described in the section 2, while in sections 3 and 4 we give a short description of the native matchmaking algorithm and of the interface to the Condor ClassAd querying features. In section 5, we show how all these components work together in an on-demand weather and marine forecasting application. The final section contains concluding remarks and information about plans for future work.

2 The Resource Broker Architecture and Design

Our resource brokering system, leveraging on a 2-phase commit approach, enables users to query a specified virtual organization index service for a specific resource, and then mediates between the resource consumer and the resource producer(s) that manage the resources of interest. Resources are represented by web services that provide access to grid features such as job submission, implemented by the Grid Resource Allocation Manager (GRAM) service and the file transfer feature, implemented by the Reliable File Transfer (RFT) service [2].

The sequence begins when the Resource Broker service is loaded into the GT4 Web Services container to create an instance of a Resource Home component. The Resource Home invokes the Resource initialization, triggering the creation of the matchmaking environment and collecting the grid-distributed published resources using the index service. The collector is a software component living inside the matchmaker environment managing the lifetime of the local resource index. The collector processes query results in order to evaluate and aggregate properties, map one or more properties to new ones, and store the result(s) in a local data structure ready to be interrogated by the requesting resource consumer.

The collector is a key component of the resource broker. Thus, we provide a fully documented API to extend and customize its behavior. In the implementation, a generic collector performs a query to the GT4 Monitor Discovery Service (MDS) [7] to identify all returned elements where the local name is "Entry." Element properties are parsed and stored in a format suitable for the resource brokering algorithm. The end point reference of each entry is retrieved to obtain the host name from which the resource is available. This step is needed because the collected properties are stored in a hostname-oriented form, more convenient for the matchmaking instead of the resource-oriented form published by the index service. In this way, each grid element is characterized by a collection of typed name/value properties.

Each entry has an aggregator content used to access the aggregator data. In the case of the ManagedJobFactorySystem, the aggregator data contains a reference to a GLUECE Useful Resource Property data type, where information about the grid element is stored by the MDS data provider interfaced to a monitor system such as Ganglia [13]. The collector navigates through the hierarchically organized properties performing aggregation in the case of clusters where master/nodes relationships are solved. A property builder helper component is used to perform this task, analyzing the stored data and producing numeric properties concerning hosts, clusters and nodes. In the collecting process new properties may be added to provide a better representation of resources available on grid elements.

A configurable property mapping component is used by the collector to perform some properties processing such as lookup: the value of a resource is extracted from a lookup table using another resource value as key; ranging: the resource value is evaluated using a step function defined using intervals; addition: a resource value is retrieved using and external component and added to the resource set; averaging: the value of a resource is calculated using the mean value of other resources.

The use of the property mapping component is needed in order to aggregate or better define resources from the semantic point of view: in the resource broker native representation, the available memory on a host is "MainMemory.RAMAvailable.Host,"

while usually the ClassAd uses the simple "Memory" notation, hence a copy operation between two properties is needed. A less trivial use of the property mapping tool is done by evaluating the Status property: there is no Status property definition in the GLUECE Useful Resource Property, while ProcessorLoad information are available. The property mapping algorithm averages the ProcessorLoad values storing this value in the LoadAvg property, then the LoadAvg property is range evaluated to assign the value to the Status property (Idle, Working) [14].

A principal resource broker service activity is to wait for index service values changing notification in order to perform an index service entity query and to collect data about the grid health represented by the availability of each VO resource. The resource brokering initialization phase ends when the collector's data structure is filled by the local resource index and the Resource component registers itself to the virtual organization main Index Services as a notification sink, and waits for index resource property data renewal events. In our resource broker, many users have to interact with the same stateful service querying resources that are tracked in order to be in coherence with the grid health status. Due to these requirements, we create the service using the singleton design pattern with the stateless web service side is interfaced with the stateful one via resource properties [12]. Due to the dynamic nature of grid resources, the resource property is not persistent and it is automatically renewed each time the index service notifies to the resource broker service that its entity status is changed. In this way the resource lifetime is automatically controlled by the effective update availability and not scheduled in a time dependent fashion [15]. Registered entity status changes are transferred upstream to the Index Service and then propagated to the Resource notification sink. Due to our application behavior, this approach could be inefficient because many events may be triggered with high frequency, degrading performance. We choose a threshold time interval value to trigger the data structure update.

From the resource consumer point of the view, the sequence starts when the user runs the resource broker client using one of the query notations that our system accepts.

Native notation: each selection criteria expression is separated by a space with the meaning of the logical and. Properties reflects the GLUECE Useful Resource Properties nomenclature with the dot symbol as property and sub property separator. The criteria are the same of the majority of query languages, plus special ones such as "max" and "min" to maximize or minimize a property and "dontcare" to ignore a pre-set condition.

```
Globus.Services.GRAM!="" Processor.InstructionSet.Host=="x86"
Cluster.WorkingNodes>=16 MainMemory.RAMAvailable.Average>=512
ComputingElement.PBS.WaitingJobs=min
```

This query looks for a PC cluster with at least 16 working nodes and 512 megabytes of available RAM using the PBS as local queue manager and where the GRAM Globus web service is up and running. Computing elements with the minimum number of waiting jobs are preferred.

ClassAd notation: the selection constraints are expressed as requirements using the well-known Condor classified advertisement notation for non structured data definition queries. In this notation, the query is enclosed in a brackets envelope and

each couple of property name/value is separated by a semicolon. Special mandatory fields are Rank and Requirements. The Requirements field contains the constraints criteria expressed using the standard C language notation.

```
[ Type="Job"; ImageSize=512; Rank=1/other.ComputingElement_PBS_WaitingJobs;
Requirements= other.Type=="Machine" && other.NumNodes>=16 && other.Arch=="x86" &&
other.Globus_Services_GRAM!="" ]
```

The shown classad performs the same query previously shown with the native notation. The NumNodes property is equivalent to the Cluster.WorkingNodes. The underscore substitutes the dot for the property/sub property access notation to avoid ambiguity with the dot meaning in ClassAd language. The ranking is mathematically computed using a simple expression involving the number of PBS waiting jobs [16].

The implementation of the matchmaking algorithm differs in relation to the chosen strategy, but can be formerly divided in two phases: the search and the selection.

In the search phase, some constraints are strictly satisfied, such as the number of nodes equal to or greater than a particular value, and the available memory being not less than a specified amount. If none suitable resource is available, the fail result is notified to the client applying the right strategy in order to prevent deadlock and starvation issues. After this step, resources satisfying the specified constraints are passed to the second phase, where the best matching resource is found using an optimization algorithm based on a ranking schema. The selected resource is tagged as claimed to prevent another resource broker query selecting the same resource causing a potential overload. At the end of the query process the resource broker client receives the End Point Reference (EPR) of the best matching resource and is ready to use it. The resource remains claimed until a new threshold filtered update is performed and the resource status reflects their actual behavior.

3 The Native Latent Semantic Indexing Based Matchmaking Algorithm

We implemented a matchmaking algorithm from scratch; it is based on an effective and efficient application of Latent Semantic Indexing (LSI) [17].

In the case of search engines, a singular-value decomposition (SVD) of the terms by document association matrix is computed producing a reduced dimensionality matrix to approximate the original as the model of "semantic" space for the collection. This simplification reflects the most important associative patterns in the data, while ignoring some smaller variations that may be due to idiosyncrasies in the word usage of individual documents [18]. The underlying "latent" semantic structure of the information is carried out by the LSI algorithm. In common LSI document search engine applications, this approach overcomes some of the problems of keyword matching based on the higher level semantic structure rather than just the surface level word choice [19].

In order to apply LSI to resource matchmaking, we have to map some concepts from the document classification and indexing to the grid resource discovery and selection field. As documents, in the web identified by URLs, are characterized by some keywords, resources, identified by EPRs in the grid, have name properties typed as string, integer, double and boolean values. A document may or may not contain a

particular word, so the matrix of occurrence document/words is large and sparse; in the same way each grid resource is not characterized by a value for each defined property, because not all properties are relevant to a specific grid resource description. Documents and grid resources share the same unstructured characterization, but while words and aggregated relations between words could have a special meaning because of the intrinsic semantic of the aggregation itself, grid resource properties are self descriptive, self contained and loosely coupled in the aggregation pattern. Under this condition, we have no need to apply the dimension reduction in grid resource properties indexing, while the application of the SVD is mandatory if dealing with documents. The grid resource description property values can be numeric, alphanumeric and boolean, but alphanumeric values have not hidden semantic mean build by aggregation, while a query can be performed specifying the exact value of one or more properties. Due to the deterministic behavior needed by the resource matchmaking process, a criteria based selection process is done before grid resources are threaded by our LSI based matchmaker algorithm. This kind of selection is performed in order to extract from all available resources the set of close matching requirements.

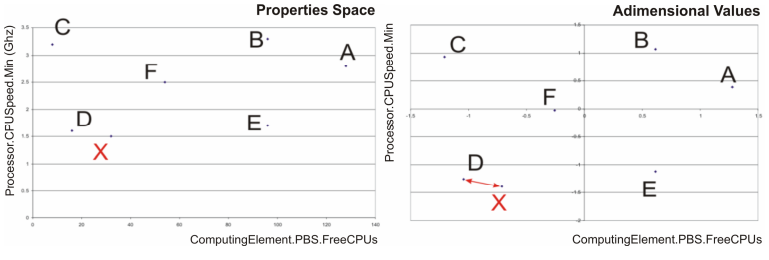

Fig. 1. The A ... F grid elements properties and the X query property. On the left in the dimensional space, on the right in the adimensional space.

Our LSI approach to matchmaking is based on the assumption that all boolean and alphanumeric query criteria are strictly satisfied in the selection phase, so the set of available grid resources comprises all suitable resources, from which we must extract the best one characterized by only numerical property values. After selection, the grid resources have a specific position in a hyperspace with a number of dimensions equal to those of the query: for example, after the ComputingElement.PBS.FreeCPUs>=25 Processor.ClockSpeed.Min==1500 Globus.Service.GRAM!="" query, the hyperspace is reduced to a Cartesian plane with the ComputingElement.PBS.FreeCPUs on the x axis and the Processor.ClockSpeed.Min on the y axis (Figure 2, left side). We assume, if the user asks for 25 CPUs or more, the best resource is the machine with 25 CPUs, while more CPUs are acceptable but something of better as in the case of ComputingElement.PBS.FreeCPUs=max. The best fitting resource could be considered to be the one that minimize the distance between the position of the requested resource and the offered one. This kind of ranking approach could be correct if all property values are in the same unit. If Processor.ClockSpeed.Min is expressed as GHz or MHz, and ComputingElement.PBS.FreeCPUs as an integer pure number the

computed distance is biased, because of the anisotropic space. An adimensionalization process is needed in order to map all offered and asked grid resources in an isotropic unitless n-dimensional space, with the goal of making distances comparable.

The goal of our adimensionalization process is to re-normalize property values so that they have a mean of zero and standard deviation equal to one. In order to achieve this result, we calculate the mean and standard deviation for each involved property. Then, using a lookup data structure, both the asked and offered grid resource, identified by their characteristics, are adimensionalized and projected in a isotropic space in which distance units on each axis are the same. Finally, a ranking table, ordered in ascending order of distance, is computed using the Euclidean distance; then the resource in the first position represents the best one fitting the querying criteria (Figure 2, right side).

4 The Condor ClassAd Based Matchmaking Algorithm

The world wide Condor open source ClassAds framework [20] is robust, scalable, flexible and evolvable as demonstrated by the production-quality distributed high throughput computing system developed at the University of Wisconsin-Madison. Classified Advertisements are stated as the "*lingua franca*" of Condor and are used for describing jobs, workstations, and other resources. In order to implement a GT4 resource oriented matchmaker algorithm using ClassAds framework, a mapping between Index Service entries and ClassAds component is needed. The component have to be flexible, full configurable, customizable and extensible in order to manage any kind of entries. In the GT4 Index Service each entry represents a resource of a specified type characterized by property values for which the ClassAd mapping process is trivial or straightforward. Resource properties, such as the GLUECE, are complex and data rich and the mapping process could be more tricky because some aggregation, synthesis and evaluation work is needed (as in the case of clusters computing elements).

Once the ClassAd representation of unclaimed GT4 grid element resources is available thanks to the developed mapping component, our matchmaker algorithm compares each ClassAd with the ClassAd form of the submitted query. The grid element vector is filled and each element each is characterized by the self and other Rank property (formerly the ClassAd Rank attribute computed from the query point of view, self, and the resource one, other). The Rank ClassAd parameter is used to perform a sort criteria in order to choose the best fitting resource represented by the one that maximize both self.Rank and other.Rank properties. Thanks to the native matchmaker algorithm, we have all tools needed to perform the best fitting resource selection, using a native query in the form "self.Rank=max other.Rank=max", that selects the grid element that maximize both properties.

5 An Application to on Demand Weather and Marine Forecasting

In our grid computing based virtual laboratory we grid enabled several atmospheric, marine and air/water quality models such as MM5 (Mesoscale Model 5) [3], POM (Princeton Ocean Model) [21], the STdEM (Spatio-temporal distribution Emission

Model) [22], the PNAM (Parallel Naples Airsheld Model) [23], WRF (Weather and Research Forecasting model), sea-wave propagation models WW3 (WaveWatch III) and the CAMx (Comprehensive Air quality Model with eXtension) air quality model [26]. We made this models grid enabled using the black-box approach implementing a modular coupling system with the goal to perform several experiments and environmental science simulations without the need of a deep knowledge about grid computing. We are still working about the grid enabling of other environmental models developing other virtual laboratory components in order to deliver a comfortable environment for earth observation grid aware application deployment.

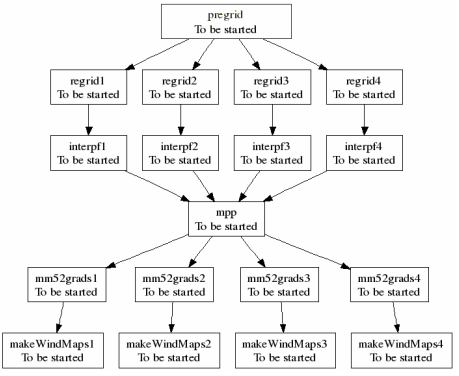

Fig. 3. The application workflow as represented by the JFDL file

We developed an on-demand weather and marine forecast, which is a full grid-aware application running in an effective ad efficient fashion on our department grid as test-bed for our resource broking service. The application environment in which the application runs is based on our virtual laboratory runtime grid software integrating our Job Flow Scheduler and the ResourceBroker Service. Using this tool, we develop the application using the Job Flow Description Language (JFDL), based on XML, with the needed extension for resource broking interfacing and late binding reference management [24].

The user need only specify the starting date and the number of hours for the simulation or the forecast. Then, all needed resources are requested from the resource broker and allocated at runtime. In the job elements of the JFDL application file, queries are coded to select resources using both the native and the ClassAd notation, while some design optimizations are made using the dynamic reference management syntax of the JFDL to run application components minimizing the data transfer time.

From the data point of view, the grid-aware application computes weather forecast and wind driven sea wave propagation on four nested domains ranging from the Mediterranean Europe (81 Km cell size) to the Bay of Naples (3 Km cell size), produces both thematic maps and GRIB data files ready for other processes and uses via standard, commercial or free software. This application is a smart and simplified version of the one we run operationally for regional weather and marine forecasts used by different local institutions.

The application workflow (Figure 3) begins with the starting event produced by the on-demand request coming, for example, from a multi access, mobile device enabled web portal. Then, the weather forecast model is initialized and the output data is rendered by a presentation software and concurrently consumed by the sea wave propagation model. Then each application branch proceeds on a separate thread.

The workflow could be represented as an acyclic direct graph into a JFDL file where each job to be submitted is described by an inner coded RSL [25] file while the launching scripts are stored in a separate repository (Figure 4). Our JFS component permits the grid application implementation using a single XML self describing file, while the RB service makes the application grid-aware.

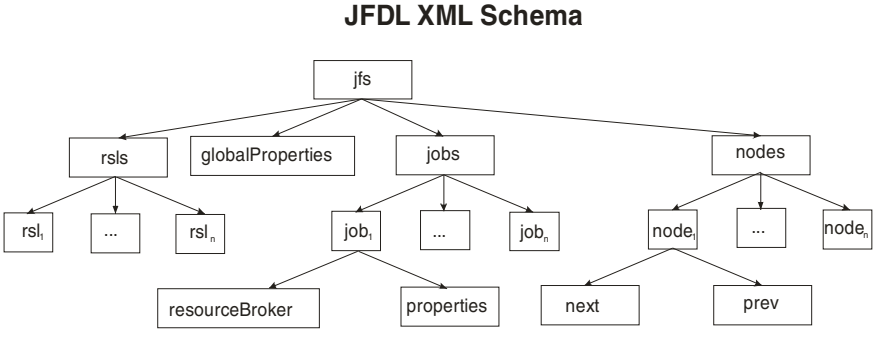

Fig. 4. The JFDL developed schema

In the element jfdl:globalProperties the developer can specify the values read in each job definition and substituted at runtime. The jfdl:rsls element contains a collection of jfdl:rsl named elements used to describe jobs with the Globus GRAM RSL file. In this files the use of environment variables place holding for scratch directory path and provided utility macros.

The file describing the grid aware application can be divided into two parts: inside the element <jfdl:jobs> each job belonging to the application is described specifying its symbolic name, the computing node where it will be submitted, and the name of the RSL file specifying all needed resources.

The statically assigned grid element unique identifying name, specified in the job element host attribute, could be omitted, in which case a resource broker jfdl:resourceBroker element would have to be used. In this element could be specified the classAlgorithm attribute to select the matchmaker implementation class identifying the matchmaking algorithm using the native one if this parameter is omitted as shown in the following example:

```
<jfdl:resourceBroker
  classAlgorithm="it.uniparthenope.dsa.grid.ClassAdMatchmakingAlgorithm">
  [Type="Job"; ImageSize=512;
  Rank=1/other.ComputingElement.PBS.WaitingJobs;
  Requirements= other.Type=="Machine" &&
  other.Software_MM5_Regrid==true &&
  other.Disk>=64 && other.NumNodes==0 ]
</jfdl:resourceBroker>
```

Where the application is looking for a non cluster machine, such as a workstation or a dedicated server, on which the Regrid component is installed. Moreover, the job needs at least 64 MB of available space on disk, and the best fitting resource is the one that minimizes the number of waiting jobs in the PBS queue manager (implicitly the PBS local queue manager is needed as requirement).

In each job definition the user can specify local properties using the jfdl:propeties element. Properties are runtime accessible using the conventional name $propertyname; global properties referred to a particular job are referred by $jobname.propertyname. This is really useful if a sort of optimization is needed using an integrated grid-enabled/aware approach. In our application we want to assign grid elements dynamically but some components have data strictly related as the case of Regrid/Interpf pairs or mm52grads/makeWindMaps pairs, so it is better to execute Regrid and Interpf, as well mm52grads and makeWindMaps, on the same computing element to achieve best performances avoiding heavy data transfers.

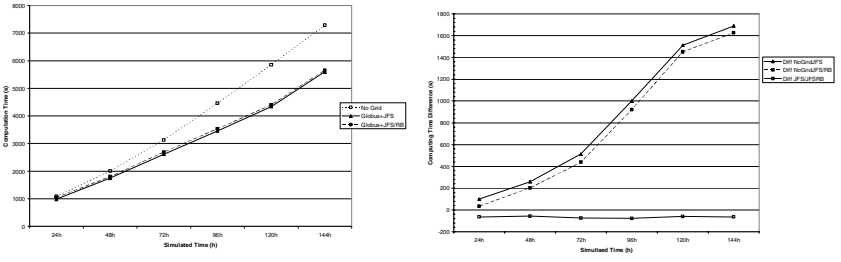

Fig. 5. Simulated Time versus Computing Time under several configurations. On the left absolute times, on the right relative times.

In order to evaluate the grid-aware application performance, we repeated the experiment 10 times and then averaged total computing time for 24, 48, 72, 96, 120, and 144 simulated hours.

We evaluated three different grid behavior configuration scenarios:

No grid technology use: The application runs as a common Bash shell script on the master node (Pentium IV at 2.8 GHz Hyper Threading equipped with 4 GByte of RAM and 2 160 GB hard disk and running Fedora Core 3 Linux) of the computing element named dgbeodi.uniparthenope.it formed by a cluster of 25 workstation powered by a hyper heading PentiumIV at 2.8 GHz, each with 1GByte of RAM and 80 GB hard disk, running Fedora Core 3 Linux. The local network is a copper gigabit using a high performance switch. This workstations are used also for student learning activities running concurrently Windows XP Professional operating system hosted by virtual environment. In this case no kind of explicit parallelism is performed and there is no need to use an external network for data transfer.

Grid-enabled mode: Globus+JFS, the application is developed using JFDL and runs under our virtual laboratory tools. Computational loads are distributed statically over available grid elements, with a design optimization performed regarding computing power and data file transfer needs. In this approach the Job Flow Scheduler

component is used, but the Resource Broker Service is switched off. The application takes advantage of the explicit parallelism carried out by parallel execution of regrid/intepf and mm52grads/makeWindMaps software modules pairs. As in the previous case of not us of grid technology, MM5 and WW3 models run on the same 25 CPUs computing element dgbeodi.uniparthenope.it.

Grid-aware mode: Globus+JFS/RB, the application is developed using JFDL and runs under our tools as in the previous case, but resources are assigned dynamically using our resource broker service performing queries each time it is needed. To achieve better performance and to avoid unnecessary data file transfers, Regrid and Interpf jobs and mm52grads/makeWindMaps are submitted to the same computing element using the Job Flow Scheduler late binding capabilities: the resource broker is invoked to choose the computing element for the Regrid job and then the same CE is used for the Interpf job. The query for parallel computing intensive load characterized jobs MM5 and WW3 is performed, but dgbeodi.uniparthenope.it is always used because the constraints.

From the performance analysis line graph (Figure 5, left side), we see that as simulated time increases from 24 to 144 hours, the grid-enabled application (filled line) performs well when compared to the no-grid (dotted line) technology use. This is because of the parallel execution of loosely coupled jobs and the optimized data high performance transfer. When resource broking capabilities are activated (outlined graph), the grid-aware system still performs better than the no-grid application version, but is slower than the grid-enabled version without resource brokering because of the latency introduced by the Web Services interactions, the adopted matchmaking technique related issues and the deadlock/starvation avoiding subsystem interactions. In the other graph (Figure 5, right side) are drown computing time differences between the no grid setup and the grid-enabled (filled line) and the grid-aware one (outlined graph). The dotted line represents the difference in computing time between the two approaches. The time consumed by the resource broker in all tests is quite constant because our grid was used in a exclusive manner (without other users). On the other hand, in production conditions (not exclusive grid use), the overall computing load of the department grid is better distributed using the grid-aware behavior, allowing for efficient and effective resource allocation optimization.

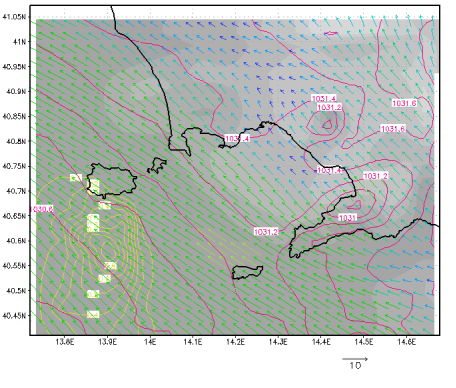

Fig. 6. Demo grid aware application results

6 Conclusions and Future Works

We have described some results in the field of grid computing research, with particular regard to the challenging issue of resource discovery and selection with matchmaking algorithms.

We developed a resource broker service, fully integrated with Globus Toolkit version 4, that is both modular and easy to expand. The plug-in architecture for both collector and matchmaking algorithm implementations we developed makes this tool an excellent environment for resource handling algorithms experiments and productions in the Globus Toolkit grid approach world. Our next goal develop an accurate testing suite, based on both real and simulated grid environment, in order to evaluate and compare native and ClassAd algorithm performances and effectiveness. In this scenario is our interest in developing a matchmaking algorithm based on the minimization of cost functions evaluated using resource characterization benchmarks in order to implement dynamic performance contracts. A better self registering approach to grid available application have to be followed to make the real use of our tools in a straightforward fashion.

In order to achieve a better, and more standard, application workflow environment, a Job Flow Scheduler refactoring is planned with the aim to be BPEL [27] compliant leveraging on open source workflow engines [28].

Our virtual laboratory for earth observation and computational environmental sciences based on the grid computing technology is enriched by the features provided by the Resource Broker Service, making possible the design and the implementation of truly grid-aware applications. The integration between the Job Flow Scheduler service and the Resource Broker service is a powerful tool that can be used both for research and application-oriented uses for running any kind of complex grid application (Figure 6).

Acknowledgments. I would like to thank Ian Foster for his suggestions and support in the revision of this paper.

References

1. I. Foster, “Globus Toolkit Version 4: Software for Service-Oriented Systems,” I. Foster, Journal of Computational Science and Technology, 21(4):523-530, 2006.
2. W. Allcock, J. Bresnahan, R. Kettimuthu, M. Link, C. Dumitrescu, I. Raicu, I. Foster. *“The Globus Striped GridFTP Framework and Server,”* .SC05, November 2005
3. The PSU/NCAR mesoscale model (MM5), Pennsylvania State University / National Center for Atmospheric Research, www.mmm.ucar.edu/mm5/mm5-home.html
4. I. Foster, C. Kesselman, *The Grid 2: Blueprint for a new Computing Infrastructure.* Morgan Kaufman, 2003
5. C. Liu, I. Foster. *A Constraint Language Approach to Matchmaking*. Proceedings of the 14th International Workshop on Research Issues on Data Engineering (RIDE 2004), Boston, 2004
6. I. Foster, C. Kesselman, S. Tuecke. *The Anatomy of the Grid: Enabling Scalable Virtual Organizations.* Intl. J. High Performance Computing Applications, 15(3):200-222, 2001.

7. J. M. Schopf, M. D'Arcy, N. Miller, L. Pearlman, I. Foster, and C. Kesselman. Monitoring and Discovery in a Web Services Framework: Functionality and Performance of the Globus Toolkit's MDS4. Argonne National Laboratory Tech Report ANL/MCS-P1248-0405, April 2005.
8. D. Thain, T. Tannenbaum, M. Livny. *Distributed Computing in Practice: The Condor Experience.* Concurrency and Computation: Practice and Experience, Vol. 17, No. 2-4, pages 323-356, February-April, 2005.
9. R. Raman. *Matchmaking Frameworks for Distributed Resource Management*. Ph.D. Dissertation, October 2000
10. R.Raman, M. Livny, M. Solomon. *Matchmaking: Distributed Resource Management for High Throughput Computing*. Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing, July 28-31, 1998, Chicago, IL
11. I. Ascione, G. Giunta, R. Montella, P. Mariani, A. Riccio. *A Grid Computing Based Virtual Laboratory for Environmental Simulations*. Proceedings of 12th International Euro-Par 2006, Dresden, Germany, August/September 2006. LNCS 4128, Springer 2006
12. B. Sotomayor, L. Childers. *Globus Toolkit 4: Programming Java Services*. Morgan Kaufman, 2005
13. M. L. Massie, B. N. Chunm D. E. Culler. *The Ganglia Distributed Monitoring System: Design, Implementation, and Experience*. Parallel Computing, Elsevier 2004
14. S. Andreozzi, S. Burke, L. Field, S. Fisher, B. K´onya, M. Mambelli, J. M. Schopf, M. Viljoen, and A. Wilson. Glue schema specification version 1.3 draft 1, INFN, 2006
15. R. Raman, M. Livny, M. Solomon. *Policy Driven Heterogeneous Resource Co-Allocation with Gangmatching*. Proceedings of the Twelfth IEEE International Symposium on High-Performance Distributed Computing, June, 2003, Seattle, WA.
16. S. Andreozzi, G. Garzoglio, S. Reddy, M Mambelli, A. Roy, S. Wang, T. Wenaus. *GLUE Schema v1.2 Mapping to Old ClassAd Format*, INFN, July 2006
17. S.Deerwester, S. T. Dumais, T. K. Landauer, G. W. Furnas and R. A. Harshman. *Indexing by latent semantic analysis.* Journal of the Society for Information Science, 41(6), 391-407, 1990
18. P. Drineas, A Frieze, R. Kannan, S. Vempala, V. Vinay.Clustering Large Graphs via the Singular Value Decomposition.Machine Learning, 56, 9–33, 2004
19. S. T. Dumais. Using LSI for Information Retrieval, Information Filtering, and Other Things". Cognitive Technology Workshop, April 4-5, 1997.
20. Condor High Throughput Computing. Classified Advertisements. Univeristy of Wisconsin, http://www.cs.wisc.edu/condor/classad
21. G. Giunta, P. Mariani, R. Montella, A. Riccio. *pPOM: A nested, scalable, parallel and Fortran 90 implementation of the Princeton Ocean Model.* Envirnonmental Modelling & Software 22 (2007) pp 117-122.
22. G. Barone, P. D'Ambra, D. di Serafino, G. Giunta, R. Montella, A. Murli, A. Riccio, *An Operational Mesoscale Air Quality Model for the Campania Region* – Proc. 3th GLOREAM Workshop, Annali Istituto Universitario Navale (special issue), 179-189, giugno 2000
23. G. Barone, P. D'Ambra, D. di Serafino, G. Giunta, A. Murli, A. Riccio, *Parallel software for air quality simulation in Naples area,* J. Eviron. Manag. and Health, 2000(10), pp. 209-215
24. G. Giunta, R. Montella, A. Riccio. *Globus GT4 based Job Flow Scheduler and Resource Broker development for a grid computing based environmental simulations laboratory.* Technical Report 2006/07 Dept. of Applied Sciences, University of Naples "Parthenope"

25. Resource Specification Language (RSL), Globus Alliance, www-unix.globus.org/developer/rsl-schema.html
26. G. Giunta, R. Montella, P. Mariani, A. Riccio. *Modeling and computational issues for air/water quality problems. A grid computing approach.* Il Nuovo Cimento, vol 28C, N.2, March-April 2005
27. T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, S. Weerawarana, IBM, *Business Process Execution Language for Web Services Version 1.1*, http://www.oasis-open.org, 2003
28. Active BPEL Engine Site. http://www.activebpel.org