Dynamic Resources Allocation in Grid Enviroments

Marius Ion, Florin Pop, Ciprian Dobre, Valentin Cristea

Faculty of Automatics and Computer Science, University "*Politehnica*" of Bucharest, Romania
Emails: marius.ion85@gmail.com, florin.pop@cs.pub.ro, ciprian.dobre@cs.pub.ro, valentin.cristea@cs.pub.ro

***Abstract*—This paper presents DyAG, an innovative solution for dynamic allocation of resources for services workflows in Grid enviroments. The proposed solution is responsible with the efficient mapping of the services which make up a Business Process Execution Language workflow onto resources, represented by Web Services, from the Grid enviroment. The presented solution is part of a framework that aims to allow the deployment of large scale workflow enabled scientific applications from a wide range of research fields onto the Grid. A series of allocation policies is considered, but the DyAG also allows the users to dynamically change the policy employed by the scheduler at runtime, through a class loading mechanism. This allows the employment of application profiling techniques in order to finely tune the scheduler in accordance with the characteristics of the environment it is running in, either by changing the various parameters of the policies proposed, or by loading completely new policies.**

Keywords

Resources Allocation, Workflow Applications, Grid Environment, SOA, Monitoring

I. Introduction

Grid Computing relies on a large number of individual computers interconnected through a private or public network such as the Internet. The loose coupling of the elements that form a computing grid make it highly scalable: computing grids range in dimension from only a few computers connected through a local area network (enterprise computing grids) to continental and intercontinental computing grids such as the EGEE Grid (Enabling Grids for E-SciencE Grid) [2] and the TeraGrid [15], which provide computing power in the range of hundreds of teraflops. Grid Computing facilitates the access to computing resources that are geographically dispersed, and allows users access to remote resources that are underutilized. Another important objective for Grid Computing is to allow users to easily share data between them, typically in the form of files that are being jointly produced and used by collaborators in disparate locations.

To this day, the Globus Toolkit [5] remains the de facto standard for building grid solutions and it is based on OGSA (Open Grid Service Architecture). In 2007 the term cloud computing came into popularity, which is conceptually similar to the canonical Foster definition of grid computing (in terms of computing resources being consumed as electricity is from the power grid).

In a distributed environment (e.g. Grids) discrete software agents must work together to perform some tasks. Furthermore, the agents in a distributed system do not operate in the same processing environment, so they must communicate by hardware/software protocol stacks over a network. This means that communications with a distributed system are intrinsically less fast and reliable than those using direct code invocation and shared memory. This has important architectural implications because distributed systems require that developers (of infrastructure and applications) consider the unpredictable latency of remote access, and take into account issues of concurrency and the possibility of partial failure [10].

A service oriented architecture (SOA) is a good solution to increas the abstraction level for communication, allowing applications to bind to services that evolve and improve over time without requiring modification to the applications that consume them. The advantages of SOA determined its adoption into Grid systems at a time when this concept was relatively new. This model was based on the existence of several layers, which were arranged in a way similar to an hourglass: the narrow part at the center corresponding to a small set of protocols and basic abstractions, on top of which a large number of components can be mapped, and underneath a large variety of low level technologies.

Grid Services can be aggregated in order to fulfill the needs of the Virtual Organizations. This new model, known as OGSA, aimed to align the Grid technologies with the web services standards. The advantages of this technology are the possibility to automatically generate code from the WSDL description, the possibility to discover services through the use of public catalogs, the association between the description of the services and inter operable network protocols, etc. The OGSA model has been implemented in the Globus Toolkit. The widely spread gLite middleware (especialy Workload Management System) is also based on a Service Oriented Architecture [11].

In this context, the dynamic resourses allocation improve the execution of workflow applications and allow users to define the adecvate policies. The most challenging issue is to allow users to dynamically change the policy employed by the scheduler at runtime, through a class loading mechanism. This allows the employment of application profiling techniques in order to finely tune the scheduler in accordance with the characteristics of the environment it is running in, either by changing the various parameters of the policies proposed, or by loading completely new policies.

The paper presents in Section 2 the Service Oriented Architecture paradigm and related work based on this paradigm for resource allocation. In the third section it is presented the

considered architecture for resource and services management, build on OGSA model. Section 4 presents the algorithm and policies used. In Section 5 it is presented the test scenarios and the analysis of experimental results. The final section contains the conclusions and possible development directions for the future.

II. Related Work

The Grid technologies that have been developed within the Grid community have produced protocols, services, and tools that address precisely the challenges that arise when we seek to build scalable VOs. These technologies include security solutions that support management of credentials and policies when computations span multiple institutions; resource management protocols and services that support secure remote access to computing and data resources and the co-allocation of multiple resources; information query protocols and services that provide configuration and status information about resources, organizations, and services; and data management services that locate and transport datasets between storage systems and applications.

Because of their focus on dynamic, cross-organizational sharing, Grid technologies complement rather than compete with existing distributed computing technologies. For example, enterprise distributed computing systems can use Grid technologies to achieve resource sharing across institutional boundaries; in the ASP/SSP space, Grid technologies can be used to establish dynamic markets for computing and storage resources, hence overcoming the limitations of current static configurations [6].

The implementation will rely on the gLite middleware [16] developed within the EGEE project, due to its maturity and its large adoption. The Glite 3.1 middleware comes from a number of Grid projects, like DataGrid, DataTag, Globus, GriPhyN, iVDGL, EGEE and LCG. This middleware is currently installed in sites participating in EGEE. The Glite Grid services follow a Service Oriented Architecture which will facilitate interoperability among Grid services, using solution based on P-GRADE [8], and allow easier compliance with upcoming standards, such as OGSA, that are also based on these principles [14]. The architecture constituted by this set of services is not bound to specific implementations of the services and although the services are expected to work together in a concerted way in order to achieve the goals of the end-user they can be deployed and used independently, allowing their exploitation in different contexts.

The Information Service (IS) provides information about the WLCG/EGEE Grid resources and their status [4]. This information is essential for the operation of the whole Grid, as it is via the IS that resources are discovered. The published information is also used for monitoring and accounting purposes. Much of the data published to the IS conforms to the GLUE Schema [1], which defines a common conceptual data model to be used for Grid resource monitoring and discovery. These systems are of particular importance to this project. These will be used to obtain information about the available web services and their current state from the grid environment.

The full potential of Web Services as an integration platform will be achieved only when applications and business processes are able to integrate their complex interactions by using a standard process integration model. The interaction model that is directly supported by WSDL is essentially a stateless model of request-response or uncorrelated one-way interactions [7]. To define such interactions, a formal description of the message exchange protocols used by business processes in their interactions is needed. WS-BPEL defines a model and a grammar for describing the behavior of a business process based on interactions between the process and its partners. The interaction with each partner occurs through Web Service interfaces, and the structure of the relationship at the interface level is encapsulated in what is called a partnerLink. The WS-BPEL process defines how multiple service interactions with these partners are coordinated to achieve a business goal, as well as the state and the logic necessary for this coordination. WS-BPEL also introduces systematic mechanisms for dealing with business exceptions and processing faults. Moreover, WS-BPEL introduces a mechanism to define how individual or composite activities within a unit of work are to be compensated in cases where exceptions occur or a partner requests reversal. A WS-BPEL process is a reusable definition that can be deployed in different ways and in different scenarios, while maintaining a uniform application-level behavior across all of them. In Grids, application workflow could be developed using Globus Toolkit [12]. The WS-BPEL could be integrated with Globus.

Resources allocation process in the Grid can be done into three stages: resource discovering and filtering, resource selecting and scheduling according to certain objectives and policies, and job submission [17]. For submision process, within a domain, one or multiple local schedulers run with locally specified resource management policies [3]. Examples of such local schedulers include OpenPBS and Condor. An LRM also collects local resource information by tools such as Network Weather Service, Hawkeye, MDS2, R-GMA and Ganglia, and report the resource status information to GIS.

III. Experimental Grid Platform with the Dynamic Allocation of Workflows

The proposed experimental platform (called PEGAF) aims to supply a solution to assemble and run workflows, with benefits in the development of applications and the reaction speed to exceptional functioning conditions. Scheduling and dynamic allocation algorithms are used, as well as declarative programming methods and an implementation based on Grid services. Software tools for the assistance in fast development or application reconfiguration are going to be provided, which make use of Grid resources in order to identify and find suitable Grid services instances.

From an architectural point of view, the project can be divided into two large components: the workflow engine and the Dynamic Allocation module (DyAG). Based on the results of previous analysis [13], a solution based on an open

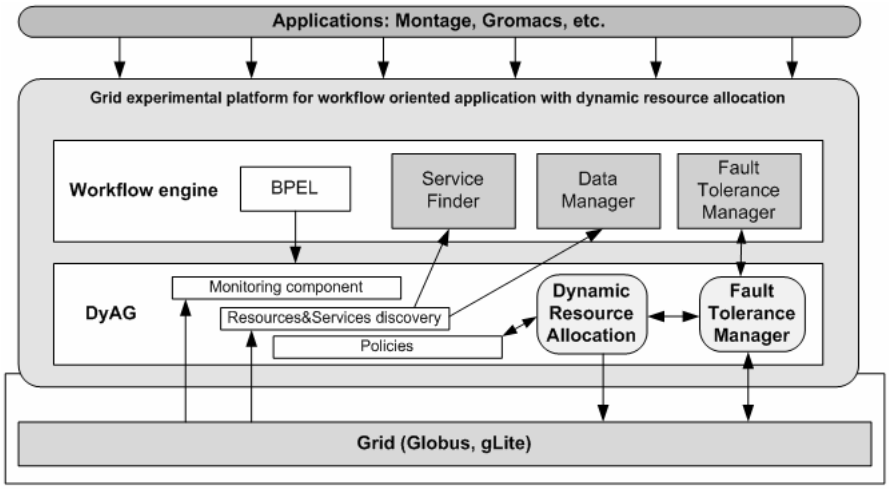

Fig. 1. PEGAF components. The architecture is organized in three layers: application layer, workflow engine layer and dynamic allocation layer

source workflow engine which can be further extended with additional modules was adopted. These ensure the scalability of the solution, while meeting the demands of the project. WS-BPEL was chosen to be the language which describes the workflows, and ActiveBPEL the engine, because it is the most widely employed open source workflow engine in academic and scientific environments and it is integrated in a large number of research projects. The ActiveBPEL engine runs over the Apache Tomcat servlet engine and uses an embedded Apache AXIS web service engine for message exchange. Figure 1 presents the components and extensions ot the PEGAF Platform.

The most important service on workflow engine layer is **BPEL** that has a few subcomponents: Process Manager, Queue Manager, Work Manager, Time Manager and Transaction Manager. *Process Manager* is responsible for the instantiation and execution of the processes and activities. It also associates them with states (inactive, running, finished, error, etc.) for the duration of their life cycle. The *Queue Manager* takes care of the messages and events which are addressed to the activities inside the processes, by maintaining a queue which contains the activities which are expecting messages. The *Work Manager* plans asynchronous operations based on "'work"' objects. The *Time Manager* offers support for time-based operations, and the *Transaction Manager* implements methods for the consistent use of transactions. The Concrete Workflow Generator (present in the workflow engine layer) transforms abstract workflows into concrete ones. The inputs of this component are the abstract workflows which are introduced by the users with the help of a workflow specification tool. It performs the mapping between the abstract functional components and web service port types or executable programs.

The **Service Finder** maps port types with sets of corresponding available web services. It contacts the scheduling component in order to discover the endpoints of the web services which correspond to the port types which are specified in the workflow by using a find-in-bind approach similar to the one present in [9].

The **Fault Tolerance Manager** has the role to attempt the recovery after an activity has crashed, by applying an available policy. Possible policies include rerunning the activity, finding an alternative service to be invoked, saving the partial results, light or heavy checkpointing, and replication of the activities. The user will be able to choose from the set of available policies the one which best suits the type of application that he intends to execute, and the platform will run the activities considering the chosen policy.

The **Data Manager** is responsible with the implementation of efficient data management mechanisms. It contacts the underlying middleware in order to find mappings between logical and physical file names. It generates meta-data which allow the association of files with the applications which have produced/modified them.

The DyAG level is a critical component in the PEGAF architecture, being responsible with the efficient mapping of abstract invocations with concrete web services which are running on grid. It receives requests from the upper layer, and returns recommendations for the web service which should be invoked, based on the information it gathers from the underlying Grid middleware (the Monitoring and Discovery System in Glite), from the Fault Tolerance Module, based on the service's history and considering the scheduling policy employed. The next section presents the DyAG componets, the scheduling algorithm and considered policies.

IV. DyAG: Dynamic resource Allocation in Grid environmets

The DyAG level contains the following components: a monitoring component, a resources and services descovery component, a plicies componente, a fault management service and a dynamic resuouce allocation service.

The monitoring component of the DyAG connects to one or more site BDIIs (Berkeley Database Information Index) which are running on the Grid and performs regular queries to the subsequent LDAP servers in order to discover the web services

which are deployed and to update their status. The information it extracts in this way is in Glue Schema 1.3 format and it is used to maintain a list of available services. This list can be restricted by using a white-list, because the monitoring component need not follow the status of web services which are of no relevance to the application which is running on the PEGAF platform, thus freeing vital network and CPU resources.

The Glue Schema information for the web services which is extracted by the monitoring component from the underlying grid middleware is further complemented by the information the DyAG receives from the Fault Management Service and from the Scheduling component. This additional data is used in order to maintain, for each web service, a history of events - either faults or moments when the web service was submitted a job. Based on the Glue Schema information, the scheduling component selects those services which match the type that was requested in the scheduling request, while the service history is used in order to compute the best service in this set. The method by which the services are ranked is implemented by the policies, and varies subsequently. The policies can be changed dynamically while the scheduler is running.

The DyAG dynamic resuouce allocation service exposes a set of methods through a RMI interface, and a subset of these through three Web Service interfaces. One of these interfaces is used for communication with the overlying Workflow Engine component and contains a method through which the DyAG is submitted jobs for scheduling. The second interface is used for communication with the Fault Tolerance Module, and contains a method through which it reports which services are being monitored and are thus of relevance to the FTM. The third interface is used to publish the available services and their relevant information to a web based interface which runs at another site. The remaining methods which are only present in the RMI interface are for administrative purposes and have not been exposed as web services for security considerations. These include methods to start, stop the service, to add or remove sites from the monitoring list and to modify the white-list which determines what services are to be monitored.

A. Architecture of the DyAG service

The DyAG service is situated between the overlying workflow engine layer and the grid middleware layer below. It communicates with the workflow engine by exposing some of its methods through a web service interface, while extracting information from the grid middleware layer by performing regular queries on one or more BDII servers through the LDAP protocol. The DyAG can be further divided into several components, which can be seen in Figure 2.

The dyag is made up of two main components, the Grid Monitor which extracts information from the glite middleware, and the Scheduler which processes requests from the workflow engine. Together these two objects are encapsulated into a dyag object which exposes a set of methods through a RMI interface. The stub of the dyag object is put into a RMI registry, from where the administrative command line interface client and the web services, which are running inside an Apache AXIS2 container, can access it. The web services publish only a subset of the methods in the RMI interface, the ones which are used for administrative reasons being accessible only through the CLI, for security reasons. This was done because the access to the RMI registry can be easily restricted through firewall rules, the other option being to use another web service for these methods, but a secure one this time. All the modules are presented in greater detail in the following section.

B. DyAG Scheduling Algorithm

The resource allocation algorithm used by the DyAG service is a global algorithm, because it deals with resources that are web services, and are distributed in the Grid environment. The algorithm falls into the category of dynamic scheduling algorithms, because it receives requests one at a time, without having any knowledge about the dependencies between them. That information belongs to the overlying workflow engine layer, which breaks up the DAGs that make up the workflows and sends the resource allocation requests to the DyAG service one at a time.

The DyAG module has access to limited information about the state of the resources, which is in fact estimated based on the history of the web services. Because of this, the scheduling algorithm falls into the category of suboptimal algorithms.

The DyAG scheduler uses a heuristic algorithm based on a series of scheduling policies which will be discussed in detail in the next subsection. Some of the policies used by the DyAG scheduler employ the technique of prediction based on historical record, by considering the past events which a service has suffered, faults and submits.

The DyAG module only uses resource centric objective functions, because it only has information about the resources which will receive the tasks. The scheduling algorithm used is a variant of the Opportunistic Load Balancing algorithm.

Because the DyAG module does not submit jobs or deploy services to the resources, but returns invocation recommendations, the rescheduling policies fall into the responsibility of the upper layer.

The scheduling algorithm receives as input requests which contain information about what type of web service the workflow engine needs to invoke, and returns a recommendation consisting of the endpoint of the most suitable web service which is available on the Grid at the time of the request.

The scheduling algorithm does not have access to information about the duration of the jobs it is scheduling, nor does it have knowledge about the status of the resources in terms of the status of the tasks they are running.

Thus, the scheduler does not know whether a particular service is currently running a job, it has multiple jobs queued, or is idle. Furthermore, because of the web services abstraction, it can not know what the performances of the machine(s) which run the web service are. Its only sources of information are the Monitoring module, which reports what services are available and their types, and the service's history, which contains events

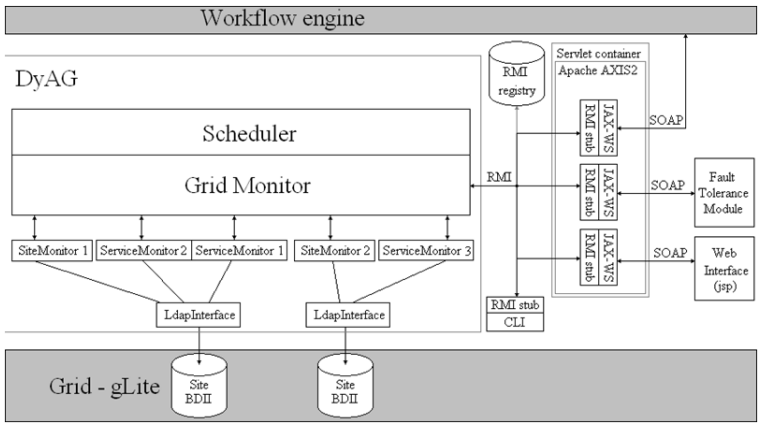

Fig. 2. DyAG service Architecture

Algorithm 1 Dinamic Resource Allocation Algorithm

```
services ← Monitor.GETSERVICES(request)
if services = NULL then
    "'No matching services"'.
end if
SORT(services, policy)
service ← services.GET(0)
Monitor.REPORTSUBMIT(service.getUniqueID())
```

reported by the Fault Tolerance Module and submission events. Because of these aspects, the choice of scheduling algorithms is limited.

The scheduler first extracts the list of compatible services from the monitoring module based on the request it has received from the workflow engine and then it proceeds to sorting them according to the policy specified by the user. The best option is chosen; the solution proposed being a variant of an Opportunistic Load Balancing algorithm.

C. DyAG Scheduling Policies

The scheduling policies are functions which take as input the information about a resource and return an evaluation of the web service in a numerical form. The information about the resource is made of the Glue Schema of the service and the history of the web service, which contains the known faults and the job submission events. Because of this, the policies of the DyAG scheduling module fall into the category of Resource Centric Objective Functions, and are aimed at optimizing Resource Utilization.

One of the most significant difficulties for task scheduling in the Grid is the dynamic nature of Grid resources. From the point of view of performance dynamism, the scheduling policies fall into the category of Performance Prediction Based on GIS, Historical Record and Workload Monitoring objective functions. We have chosen this because an On-Time-Information from the GIS strategy would be unable to efficiently map the jobs onto web services, because the MDS does not provide sufficient relevant information about the web services which are running, being oriented more towards Computing Elements and Storage Elements. Also, Rescheduling strategies were not an option, since the DyAG scheduler does not submit the jobs itself to the underlying Grid middleware; rather it makes recommendations to the upper workflow engine layer, which will submit them through the use of its ActiveBPEL engine. Thus, we have chosen a strategy using prediction based on historical modeling.

RandomPolicy. This is the simplest policy available, the evaluation function returning a random double number in the $[0, 1]$ interval. It can be used when the number of jobs arriving online at the scheduler is a large one, because the time necessary to evaluate each job request is very small. It is not suitable in an environment where the number of errors is high, because it does not take into consideration the faults.

RoundRobinPolicy. As the name suggests, this policy evaluates the web services such that they will receive jobs in a round robin fashion. This is done by considering the last time when the service was submitted a job. Effectively, the evaluation function returns the difference between the current system time and the last submission time. Thus, the service which received a job last will have the highest mark, and therefore the job will be submitted to it. It is not suitable in an environment where the number of errors is high, because it does not take into consideration the faults.

RoundRobinExtPolicy. This policy is an adaptation of the Round Robin Policy so that it will also take into consideration the faults in the history of the web service. The evaluation function presented offers similar results to the Round Robin policy in an environment without errors, but it also takes into consideration the faults which the web service has suffered in normal operating environments. The objective function takes into consideration all the events in the history of the web service. Let $T = \{t_n | n \in N \quad \text{a service event has occured at } t_n\}$. Then we can define the objective function f as follows:

$$f(t) = \begin{cases} 1, & \text{if} \quad t < t_1 \\ g(t_1), & \text{if} \quad t = t_1 \\ min\{f_M + p\tau_r, 1\}, & \text{if} \quad t \notin T \\ max\{min\{f_M + p\tau_r, 1\} - Ev(t), 0\}, & \text{if} \quad t \in T \end{cases}$$

where t is the moment in time when the evaluation is done, $\tau_r = t' - t$, $f_M = f(t_M)$, $T_M = max\{t_n\}$, p is a constant which represents how long it takes a web service to complete a job, and

$$Ev(t) = \begin{cases} 0.5, & t \text{ is a submission} \\ 0.2L, & t \text{ is a fault of level L} \end{cases}$$

This policy depends on a series of parameters such as the slope of the line segment which represents the recovery time of a web service, and the values which are deducted when faults or submission events occur. These parameters can be recompiled into the class implementing the policy, which can be dynamically loaded into the scheduling engine. In order to find the suitable values for these parameters, a profiling must be made based on the characteristics of the jobs which are going to be run in the application.

AveragePolicy. This policy takes into consideration only the recent history of the web service, in a time interval which is defined as a parameter in the class implementing the policy. This time interval is split into multiple sub intervals, the number of sub intervals also being a parameter for the policy, and for each sub interval, a mark is given according to the number of events which have occurred inside that particular time frame. Thus, for an interval which contains no events, a mark of 0 is given; for each submission event, 0.5 is subtracted and for each fault a value of 0.2 times the level of the fault is subtracted. The negative values which are obtained ensure that the sorting will still be made such that the service with the largest mark will be chosen to receive the next job.
Let $[t_0, t_n]$ be the time window which is to be evaluated, and $t_0, t_1, \ldots, t_{n-1}, t_n$ its division into n subintervals. Let S_H be the history of the web service, where $S_H = \{E_i | E_i \text{ is a service event }\}$. Then the objective function f can be computed as follows:

$$f = -\frac{\sum_{i=0}^{n-1} \left(\sum_{E_j \in S_H} g(i, E_j) \right)}{n}$$

where

$$g(i, E_j) = \begin{cases} Ev(E_j), & \text{if } E_j \in [t_i, t_{i+1}] \\ 0, & \text{if } E_j \notin [t_i, t_{i+1}] \end{cases}$$

This policy considers the events which occurred at the beginning of the time interval equally important with those which occurred more recently. Because of this, there are leaps in the value of the function not only because of recent events, but also when the relevant time frame passes the events which occurred in the more distant past. This aspect will be corrected by the following policies which will perform a weighted average on the values computed for the subintervals in the considered time frame. Thus, the formula for calculating the value of the objective functions for a given time interval becomes:

$$f = -\frac{\sum_{i=0}^{n-1} \left(w\left(\frac{i}{n}\right) \sum_{E_j \in S_H} g(i, E_j) \right)}{\sum_{i=0}^{n-1} w\left(\frac{i}{n}\right)}$$

where $w : [0, 1] \to R$ is the weighing function, and will be further discussed in the following.

PwrWeightsAvgPolicy. As the name suggests, this policy uses the power function as its weighing function, which have $w : [0, 1] \to [0, 1]$, $w(x) = x^p$, $p > 0$. Only when p becomes greater than 1 does the transition begin to occur more smoothly, because $w'(x) = 0$. However, the values towards the end of the time frame start to grow too rapidly for some use cases. This issue is handled by the policies which use a spline interpolated weighing function.

ExpWeightsAvgPolicy. This policy uses an exponential weighing function, which gives greater importance to more recent events rather than the ones which occurred at the beginning of the time frame. As the p parameter decreases towards 0, the behavior of the function looks more like that of the linear function presented before. As p increases however, the function starts to resemble the power function for values of p greater than one, giving more importance to the recent events rather than to the ones which occurred at the beginning of the time frame. We have $w : [0, 1] \to [0, 1]$, $w(x) = \frac{e^{px}-1}{e^p-1}$.

CircWeightsPolicy. This policy uses as its objective function the equation of the circle with the center in $(0, 1)$, and a radius of 1. This function also has the property that its first derivate in 0 is nil, which translates into a smoother transition of the importance of the events in the considered time frame to those outside of it, which have an importance equal to zero. We have $w : [0, 1] \to [0, 1]$, $w(x) = 1 - \sqrt{1 - x^2}$.

SplWeightsPolicy. The weighing functions presented before have two issues in general: either the transition from the events which occurred before the beginning of the time frame and are not taken into consideration to those at the beginning of the time frame is not done smoothly, by having the first differential of the function equal to zero in the point of origin, or the function increases too abruptly towards the end of the time interval which is taken into consideration when evaluating the web service. Because of these aspects, a new weighing function was introduced, which has a point of inflexion $(x_0, y_0) \in [0, 1] \times [0, 1]$. This allows the weighing function to have the first derivate equal to 0 both in the point of origin, $(0, 0)$ as well as in $(1, 1)$. This translates into a smoother transition at the beginning of the time frame and into a more balanced increase of the weights towards the end of the interval. This was done using a second grade spline interpolation function, as follows. The weighing function, $w : [0, 1] \to [0, 1]$, is parameterized by x_0 the point of inflexion:

$$w(x) = \begin{cases} \frac{1}{x_0} x^2, & \text{if } x \leq x_0 \\ \frac{1}{x_0 - 1}(x^2 - 2x + x_0), & \text{if } x > x_0 \end{cases}$$

This category of weighing functions produces a more balanced output at both ends of the time frame, ensuring a smooth transition at the beginning and a more balanced increase towards the end.

All the previously presented functions can find their use in various scenarios, based on the application that is running on the PEGAF platform. Furthermore, application profiling should be performed in order to determine the best policy which should be used for each type of application, as well as to determine the values of the configuration parameters.

V. Experimental results

Figure 3 presents the following scenario: submit events at minutes one, three, five and seven, and faults at minutes two, three and four. The time interval is five minutes, and it is further divided into five one minute sub-intervals. The first graph represents the values for each subinterval for the first web service, and the second graph represents the global value for the first web service at each time sub interval.

In the beginning, both services' histories have no events, so the objective function evaluates to 0. The first service is submitted a job, and so its objective function will decrease to the value of 0.90 at the end of the first subinterval. Next, in the second, third, and fourth time intervals the second web service will be subject to a series of events which will decrease the value of its objective function: a level two fault followed by a submission event and another level two fault. This can be seen in the last graph, as the value decreases from 1.00 at minute two down to 0.74 in minutes five through seven. In minutes eight and nine, the value objective function starts to increase, because the events which caused the decrease, in minutes two, three and four, are beginning to exit the relevant time frame for the function.

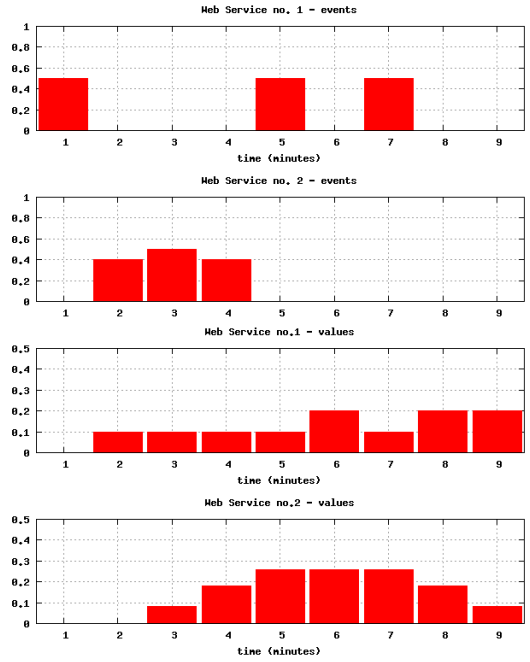

Fig. 3. AveragePolicy

For proposed policy (SplWeightsAvgPolicy) we used the same test scenario: four requests are made from the workflow engine layer to the DyAG service, all of which need the same type of service. The requests are made at the time moments of one minute, three minutes, five minutes and seven minutes, and the second web service suffers two level two faults at the time moments of four and six minutes. The scenario was run using the SplAvgWeightsPolicy, once with the parameter $x_0 = 0.25$ and once using $x_0 = 0.75$. The marks given to each web service are summarized Figure 4.

Time	Service 1 x_0=0.25	Service 2 x_0=0.25	Events x_0=0.25	Service 1 x_0=.75	Service 2 x_0=.75	Events x_0=.75
1	0.0000	0.0000	Submit WS1	0.0000	0.0000	Submit WS1
2	-0.1689	0.0000	Fault WS2	-0.2308	0.0000	Fault WS2
3	-0.1507	-0.1352	Submit WS2	-0.1571	-0.1846	Submit WS1
4	-0.1142	-0.2895	Fault WS2	-0.3109	-0.1256	Fault WS2
5	-0.0594	-0.3772	Submit WS1	-0.1859	-0.2487	Submit WS1
6	-0.1758	-0.2822	None	-0.3141	-0.1487	None
7	-0.1507	-0.1562	Submit WS1	-0.1859	-0.0667	Submit WS2
8	-0.2831	-0.0543	None	-0.0833	-0.2538	None
9	-0.2100	-0.0055	None	-0.0288	-0.1596	None
10	-0.1210	0.0000	None	-0.0032	-0.0801	None

Fig. 4. Test scenario values for SplAvgWeightsPolicy

As it can be seen from the previous table, the variation of the x_0 parameter produces significantly different results, because of the different distribution of the weights after which the values are computed. Thus, at the time of the second submit, for $x_0 = 0.25$, the first web service has a lower value than the second, due to the fact that the policy gives greater importance to events that happened more recently than in the case of $x_0 = 0.75$ and because nothing happened to the first service in the previous time interval. The variation in time of the values of the two web services can be seen in the following graphs.

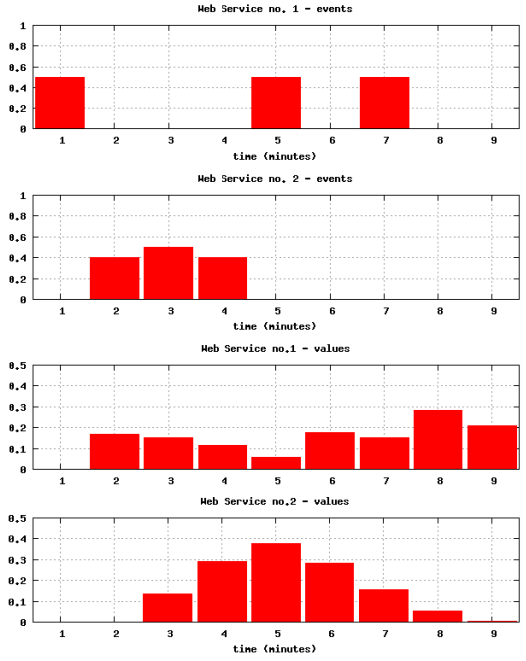

Fig. 5. SplWeightsAvgPolicy with $x_0 = 0.25$

In Figure 6 we can see how the evolution of the scheduling changes because of the modification of the x0 parameter. There is no generally suitable value for this parameter, but it can be obtained only through application specific profiling.

VI. Conclusion

The DyAG project improves the performance of applications running on the PEGAF platform by efficiently parallelizing the jobs which are invoked by the ActiveBPEL engine.

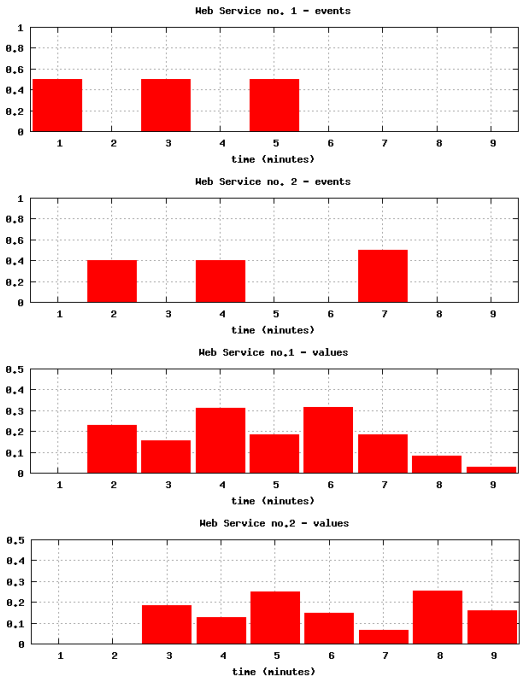

Fig. 6. SplWeightsAvgPolicy with $x_0 = 0.75$

Thus, instead of using a single service to invoke all the jobs, the scheduler allows for the transparent use of several web services, which can be located at the same site or can span a large geographical area. Furthermore, the DyAG scheduler takes into consideration other important aspects when performing the Opportunistic Load Balancing of the jobs onto the Grid resources. The most important one is the fact that it considers the previous history of the services, both the faults incurred and the jobs which were submitted, and uses this information to predict the future state of the service, thus increasing the chances of avoiding the need for the rescheduling of the jobs which could happen because of various errors.

The DyAG also allows the users to dynamically change the policy employed by the scheduler at runtime, through a class loading mechanism. This allows the employment of application profiling techniques in order to finely tune the scheduler in accordance with the characteristics of the environment it is running in, either by changing the various parameters of the policies proposed, or by loading completely new policies. The DyAG is designed to be an integrated part of a Systems Oriented Architecture, by exposing its interfaces through a series of Web Services, through which it interacts with the overlying workflow engine layer, with the Fault Tolerance Module and with the web application that shows the status of the resources which are being monitored.

However, there is room for significant improvements, because choice of scheduling algorithms that can be employed is severely limited by the particularities of the problem of dynamic allocation of web services. This is because, unlike in traditional scheduling algorithms which have access to important information such as the number of processors on a machine, their type, the memory capacity, CPU speed, etc., the information about web services is much more limited. This is because of the basic concept of the web service, which aims to completely separate the interface from the implementation and the underlying technologies.

Furthermore, the current scheduling algorithm could greatly benefit from the use of profiling techniques in order to better adapt the policies to the requirements of the workflow applications running on the PEGAF platform. In addition to these aspects, further scheduling policies could be implemented.

REFERENCES

[1] S. Andreozzi, S. Burke, L. Field, S. Fisher, B. Konya, M. Mambelli, J.M. Schopf, M. Viljoen, and A. Wilson. GLUE Schema Specification - Version 1.2, December 2005.

[2] Rüdiger Berlich, Marcus Hardt, Marcel Kunze, Malcolm Atkinson, and David Fergusson. Egee: building a pan-european grid training organisation. In *ACSW Frontiers '06: Proceedings of the 2006 Australasian workshops on Grid computing and e-research*, pages 105–111, Darlinghurst, Australia, Australia, 2006. Australian Computer Society.

[3] Fangpeng Dong and Selim G. Akl. An adaptive double-layer workflow scheduling approach for grid computing. In *HPCS '07: Proceedings of the 21st International Symposium on High Performance Computing Systems and Applications*, page 7, USA, 2007. IEEE Computer Society.

[4] A. Nobrega Duarte, Piotr Nyczyk, Antonio Retico, and Domenico Vicinanza. Global grid monitoring: the egee/wlcg case. In *GMW '07: Proceedings of the 2007 workshop on Grid monitoring*, pages 9–16, New York, NY, USA, 2007. ACM.

[5] Ian Foster and Carl Kesselman. Globus: A metacomputing infrastructure toolkit. *International Journal of Supercomputer Applications*, 2(11):115–128, 1997.

[6] Ian Foster, Carl Kesselman, and Steven Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *Int. J. High Perform. Comput. Appl.*, 15(3):200–222, 2001.

[7] Matjaz B. Juric. *Business Process Execution Language for Web Services BPEL and BPEL4WS 2nd Edition*. Packt Publishing, 2006.

[8] Peter Kacsuk, Tamas Kiss, and Gergely Sipos. Solving the grid interoperability problem by p-grade portal at workflow level. *Future Gener. Comput. Syst.*, 24(7):744–751, 2008.

[9] Dimka Karastoyanova, Alejandro Houspanossian, Mariano Cilia, Frank Leymann, and Alejandro Buchmann. Extending bpel for run time adaptability. In *EDOC '05: Proceedings of the Ninth IEEE International EDOC Enterprise Computing Conference*, pages 15–26, Washington, DC, USA, 2005. IEEE Computer Society.

[10] Samuel C. Kendall, Jim Waldo, Ann Wollrath, and Geoff Wyant. A note on distributed computing. Technical report, CA, USA, 1994.

[11] Cecchi Marco, Capannini Fabio, Dorigo Alvise, Ghiselli Antonia, Giacomini Francesco, Maraschini Alessandro, Marzolla Moreno, Monforte Salvatore, Pacini Fabrizio, Petronzio Luca, and Prelz Francesco. The glite workload management system. In *GPC '09: Proceedings of the 4th International Conference on Advances in Grid and Pervasive Computing*, pages 256–268, Berlin, Heidelberg, 2009. Springer-Verlag.

[12] Dana Petcu. A comprehensive development guide for the globus toolkit. *IEEE Distributed Systems Online*, 9(6):4, 2008.

[13] Florin Pop, Ciprian Dobre, and Valentin Cristea. Decentralized dynamic resource allocation for workflows in grid environments. In *Proceedings of 10th International Symposium on Symbolic and Numeric Algorithms, SYNASC08*, pages 557–563, Timisoara, Romania, 2008. IEEE Comp.

[14] Radu Prodan and Thomas Fahringer. From web services to ogsa: Experiences in implementing an ogsa-based grid application. In *GRID '03: Proceedings of the 4th International Workshop on Grid Computing*, page 2, Washington, DC, USA, 2003. IEEE Computer Society.

[15] Mohamed Sayeed, Kumar Mahinthakumar, and Nicholas T. Karonis. Grid-enabled solution of groundwater inverse problems on the teragrid network. *Simulation*, 83(6):437–448, 2007.

[16] Diego Scardaci and Giordano Scuderi. A secure storage service for the glite middleware. In *IAS '07: Proceedings of the Third International Symposium on Information Assurance and Security*, pages 261–266, Washington, DC, USA, 2007. IEEE Computer Society.

[17] Jennifer M. Schopf. Ten actions when grid scheduling: the user as a grid scheduler. pages 15–23, 2004.