FRAMEWORK FOR ADAPTIVE GRID COMPUTING SERVICE

Alexandru STANCIU[1], Marius GURAN[2]

This paper presents a framework for an adaptive Grid Computing Element service which can auto-scale its computational resources depending on the workload. To efficiently operate a Grid infrastructure the framework is designed to provide autonomic capabilities for the Grid service responsible for application execution. Self-management of the infrastructure is based on policy enforcement by a configuration management system. This capability is integrated with dynamic provisioning of the Grid site computational cluster on virtualized resources by a resource manager module using a feedback control loop. In order to efficiently allocate resources for the computing cluster we maximize the utility function for the job execution service. We have defined and experimented the dynamic allocation of resources so that the utility of the Grid Computing Element service is maximized.

Keywords: adaptive Grid computing service, autonomic element, utility based optimization, constraint programming

1. Introduction

Grid computing as a technology which enables distinct clusters operated by independent institutions to share resources in a transparent way, has its major benefit the provision of virtually unlimited computing capacity to execute both individual jobs and very complex scientific workflows. However, as there is the case that for some periods of time those resources are not utilized, it is useful to have mechanisms to be able to scale up and down the computing capacity according to the actual workload, in order to run the infrastructure in an efficient way.

Large distributed ICT infrastructures, like the Grid infrastructure, are known for their extremely high complexity which makes their management especially difficult and error prone [1]. Automatic configuration of the systems is designed to solve the problems which occur with systems that are administrated manually [2]. It is desirable to use a configuration management system, and to define exhaustive rules to achieve a specified system state.

[1] PhD student, The Faculty of Automatic Control and Computers, University POLITEHNICA of Bucharest, Romania, e-mail: stanciu@grid.ici.ro

[2] Prof., The Faculty of Automatic Control and Computers, University POLITEHNICA of Bucharest, Romania, e-mail: guranmarius89@yahoo.com

Because the Grid middleware is composed from a large number of various software components there is a great challenge to build it, but also to deploy and configure it. There are common configuration parameters used by a Grid site, as well as particular parameters which are specific to a Grid service.

Particularly for the Grid site administration, there were proposed various strategies that tried to address the need for a robust operation, like the separate and independent operation, collaboration on a local or regional area, or a centralized approach where the entire Grid infrastructure is managed from a central operation center [3].

This latter approach promoted the use of a configuration management system that can provide the functionality capable to address a number of specific requirements. For example, continuous maintenance of the system components must be performed during the life time of the system. In addition of the installation and post-installation configuration, the system should be updated and its operation should be catered for. All system components should be covered by the configuration tool in order to eliminate the need to perform manually configuration actions which can divert the state of the system.

By using a configuration management system to maintain the desired state of the infrastructure we can develop configuration templates for the Grid middleware components, which can be shared by the community and it can enable collaborative management of the Grid infrastructure by the means of sharing experience and using best practices.

This paper presents a framework for an adaptive Grid Computing Element (CE) service which can auto-scale its computational capacity depending on the workload. To efficiently operate a production Grid infrastructure the framework is designed to provide autonomic capabilities for a Grid CE service. Self-management of the infrastructure is based on policy enforcement by a configuration management system. This capability is integrated with dynamic provisioning of the computing cluster nodes on virtualized resources by a resource manager module. In order to efficiently allocate resources for the computing cluster we define a utility function for the Grid CE service, and then we proceed to maximize the utility value.

2. Management strategies for a Grid infrastructure

Our research was dedicated to the gLite Grid middleware stack which is widely used in various large scale Grid infrastructures. For example, it is successfully deployed on large international computing infrastructures such as the WLCG which supports LHC experiment at CERN, as well as by the National Grid Infrastructures (NGI) which are part of the European Grid Initiative (EGI).

One of the first tools for automatic installation and configurations of systems was the LCFG (Local ConFiGuration system) which was developed at Edinburgh University [4]. LCFG was the tool used to deploy and manage the Grid middleware developed by the EDG project, and then in the subsequent EGEE1 project until it was replaced by the YAIM (YAIM Ain't an Installation Manager) toolkit.

YAIM is composed of shell scripts which perform the configuration actions required for each Grid service. It uses a configuration file where are declared specific configuration variables. It is possible to deploy multiple services on the same machine, as well as to configure only a some components for a Grid service, but these actions should be performed manually.

Another tool developed by the EDG project was Quattor, which was designed to provide the automated management of large Grid computing fabrics[5]. Quattor does not ensure the correct execution of configuration component, as this is followed up by the monitoring systems which should detect and alert on the eventual failures. Quattor can be used to manage the Grid middleware by two methods: one method is based on using the YAIM toolkit together with Quattor, and the second uses only special templates developed by the Grid community to manage the Grid services [6].

To dynamically adapt the capacity of the Grid CE service a resource management module is designed as a control loop with feedback, and should act as an independent component. There were some efforts in this direction, for example using Cloud computing to elastically extend Grid site resources [7], but they were not integrated with a configuration management system. Virtualized environments can bring benefits to data centers that want to consolidate smaller servers into higher capacity ones. However, an important challenge is the management of the physical resources so that each virtual environment receives an appropriate share of these resources as the workload varies with time [8].

By integrating the automated scaling of resources according to the current workload with automated configuration of the infrastructure which includes newly created resources, it is possible to have an autonomous infrastructure that is able to self-manage itself [9]. Self-configurations is achieved by using a configuration management system with a configuration policy which fully describes the infrastructure state. Self-optimization is performed by the feedback control loop of the resource manager component of the framework. Self-protection and self-healing properties are enforced as well by the configuration management system according to the computer immunology model [10].

Utility functions can be used to translate high level economic goals related to the provision of a service to resource allocation actions. For example, self-optimization of an autonomic system can be based on the maximization of the

utility function [11]. One approach to solve the optimization of the utility function is to use constraint programming [12].

3. Framework for an adaptive Grid Computing Element service

The problem that we need to solve is to optimize the usage of the computational resources. Grid CE service provides computational resources to run applications submitted by members of Virtual Organizations (VO). It uses underneath a Local Resource Management System (LRMS) which manages the applications on the local execution systems. These computing systems are part of a computational cluster which aims to maximize the number of applications executed in an amount of time which is specific to High Throughput Computing (HTC) paradigm.

Unfortunately, the workload on the cluster is not constant as it is used at a full capacity only for a fraction of time, which means that a number of systems are idle, when they can be used for other tasks.

For this reason we need an adaptive Grid CE service which can respond to workload fluctuations by reducing or expanding its computational resources as required. The aim is to maximize the number of applications which are executed in a period of time. We can use the queue waiting time for a job as a quality of service (QoS) parameter which can be defined by a Service Level Agreement (SLA) for the Grid CE service.

One approach to address these problems is to design a software framework that can provide elastic computational resources for the Grid CE service. For this we shall create elastic computing resources on demand as presented in Fig. 1.

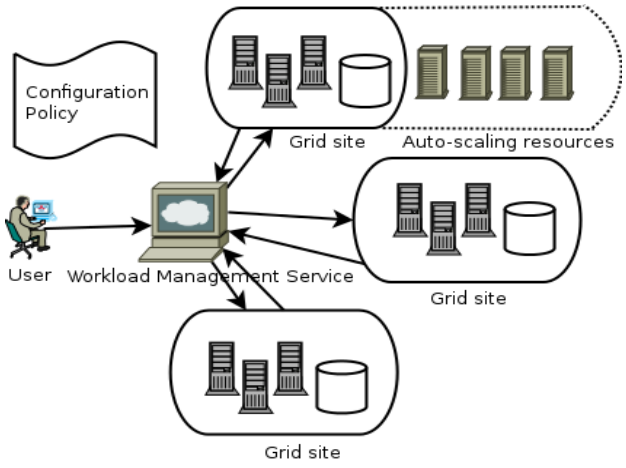

Fig. 1: Elastic computing resources described in a configuration policy

We use Cloud computing for new systems provisioning, and a resource manager developed with a feedback control loop which monitors the Grid CE service's workload, as presented in Fig. 2.

The integration of new systems into the Grid infrastructure, as well as the decommissioning of unused resources should be managed in an automated way, therefore we need to define a configuration policy for the Grid infrastructure which must describe the desired state of the systems and which should be enforced whenever there is deviation from it.

This approach is used by configuration management systems which are able to automatically configure systems according to a configuration description and then to maintain that configuration in a proper way.

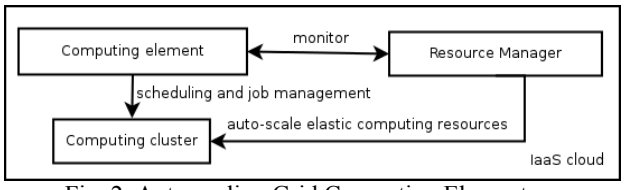

Fig. 2. Auto-scaling Grid Computing Element

The software framework is therefore split in two components which address two distinct requirements:

- For the static part which is related to the specific configuration of the Grid infrastructure, we propose a configuration policy which should be maintained by a configuration management system, such as Puppet [13], [14].
- For the dynamic part which is related to the actual utilization of the Grid site, and which should be able to react to the variable load of the Grid computing service, we devise a resource manager based on a feedback control loop.

In order to have auto-scaling capabilities of the elastic computing service, we need to use a resource management module which monitors the load of the computing cluster. According to defined policies, the resource management module is able to create virtual machines which are automatically configured as worker nodes for the Grid CE service. After the jobs are executed, the resource management module should remove unused worker nodes and automatically reconfigure the computing cluster and the Grid middleware.

To dynamically scale the computational resources of the Grid CE service we use virtualization technology. This brings important advantages such as flexibility for systems configuration, scalability and resource consolidation which provide cost savings. On the other hand, using virtualized resources has an impact on the computing system performance, which should be taken into account as there is a requirement for the Grid CE service to maximize the number of applications executed in a period of time.

In order to optimize the resource allocation for the application's execution we define the utility function for the Grid CE service. Utility represents a way to quantify the fulfillment of QoS requirements which can be defined by a SLA. For example, we can use as QoS parameter the waiting time in queue for a job.

We can consider that the QoS is respected, and a benefit is obtained, if the applications can be started in a period less than the limit defined by the SLA, and for the applications that cannot be started in due time as specified by the SLA, that the SLA is breached and there is a penalty to be paid.

4. Definition and evaluation of the utility function

We can consider the utility function of a LRMS for running applications for a Grid CE service to be an instrument to evaluate the fulfillment of requirements for the jobs execution as specified in a SLA. The utility function should represent an economic value related with a certain level of quality of the service, and it can include both the revenues and the costs related with the jobs execution.

For example, we can use as a parameter to evaluate the performance of an execution queue, the waiting time until the job is executed by the LRMS. As the time required for the job to complete is not known a priori, the waiting time in queue depends on different factors such as the advanced reservations, queue capacity for simultaneous running jobs and the total number of jobs, and the performance of the execution systems.

Therefore, the utility function of a LRMS can be used to evaluate the benefits obtained for providing the computational resources for jobs execution with a QoS defined in a SLA, where for the QoS requirements we have proposed the queue waiting time. The utility function evaluates the profits which can be obtained by the Grid site owner which provides computational resources at a certain QoS (max queue waiting time) bound by a SLA.

In order to calculate the utility function we should take into account the costs for providing the resources for jobs execution which should include the costs for running the servers, cooling and electricity, etc. We can assume that the queue waiting time depends on the following factors:

- Number of slots allocated for the queue (the number of concurrent running jobs or the capacity of the queue). It can be changed in order to maximize the utility function.
- Performance of the computing systems. This can be changed in order to maximize the utility function. For example, we can allocate more CPU capacity for a virtual machine.
- The number of jobs which are in the queue (the workload).

If we consider *R(t)* as the number of jobs executed according to the SLA, *Q(t)* as the number of jobs for which the SLA is not met, and *C(t)* as the capacity of the queue at the moment of time *t,* we can define the utility function as follows:

$$U(t)=a\cdot R(t)-b\cdot Q(t)-c\cdot C(t) \quad (1)$$

The jobs that are executed conform with the SLA requirements provide a revenue which is expressed by the parameter *„a"*. For the jobs that cannot be started in period of time specified by the SLA there is penalty which is defined by parameter *„b"*. Finally, computational resources which are allocated for the job execution incur an operational cost which is expressed by parameter *„c"*.

In order to validate the model for the utility based optimization of the capacity of the LRMS we have simulated the job execution for a Grid CE service.

For the experimental model we have made the following assumptions: each job is successfully executed in a specific amount of time, which is not known a priori, but it cannot be extended over a certain limit which is enforced by the LRMS. The jobs that are not finished until the queue time limit are terminated automatically and are lost.

We have considered discrete moments of time *t*, when we have made measurement of the LRMS state. We have recorded the number of new jobs, the number of running jobs, and the number of queued jobs.

For simplification we have considered that the interval between two consecutive moments is sufficient for finalizing all the running jobs in the queue at the start of the interval (the interval is bigger than the queue max running time limit) and that the SLA for all new jobs has not been breached.

We have also considered that the number of running jobs is equal with the capacity of the LRMS, so that in each moment of time there are no free resources that are not allocated for job execution.

For resource accounting reasons we have assumed that one job is executed in a computing system, and each computing system has allocated only one slot for job execution so that only one job is executed at a time. We have also assumed that the cost for using each computing system is the same for all systems. For the utility function optimization we have used constraint programming, and we modeled the problem in Minizinc language [15]. We have used a mixed integer programming (MIP) solver – G12-MIP, to calculate the solution for computational resource allocation (capacity of the queue) *C(t)*, in order to maximize the value of the utility *U(t),* at each moment of time *t*.

For solving the optimization problem we have defined the following variables and constraints:

$T = \{t_1, t_2, \ldots, t_{20}\}$, where *T* is a set of 20 moments of time.

$C = \{c_1, c_2, \ldots, c_{20}\}$, where c_i is the capacity of the queue at the moment t_i.

We must have $c_i > 0$ and $c_i < CapMax$, $i = 1..20$, where *CapMax* is the maximum capacity of the queue.

$J = \{j_1, j_2, \ldots, j_{20}\}$, where j_i is the number of new jobs in the queue at the moment t_i.

$Q = \{q_1, q_2, \ldots, q_{20}\}$, where q_i represents the number of jobs waiting in queue at the moment t_i, $q_i >= 0$, $i = 1..20$.

We have performed two experiments for computational resource allocation based on optimization of utility function, with a similar distribution of new jobs at the selected moments of time, but with different values for parameters *"a", "b" and "c"*, and for the maximum capacity of the queue *CapMax*, as presented in the Table 1.

Table 1

Parameters used for experimental evaluation of utility based resource allocation

	a	b	c	CapMax
Experiment 1	50	1	30	8
Experiment 2	50	20	30	7

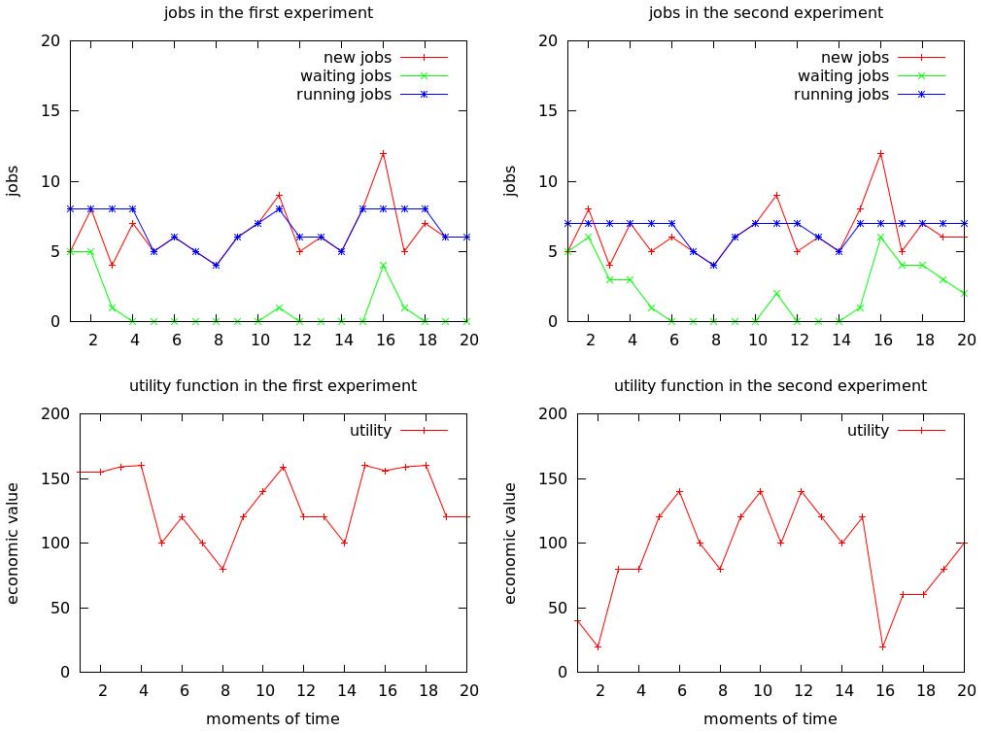

Fig. 3 Experimental results

The *"a", "b"* and *"c"* parameters provide the economic value of the utility function, and they are chosen with the assumption that the revenue and the associated costs and penalties have a comparable value. The simulation results are presented in the Fig. 3.

In the first experiment we have considered that the capacity of the LRMS was of 8 concurrent jobs, and the following parameters were used for the utility function: a=50, b=1, c=30 as we wanted to stimulate the executions of jobs with a bigger reward than the penalties for SLA breach and cost of resources used.

In the second experiment we have used the following parameters: a=50, b=20, c=30 and we have reduced the capacity of LRMS to 7 running jobs.

As the first experiment shows a similar shape of the utility function and the queue capacity, the second experiment stresses the influence of the waiting jobs which cannot be started in due time as expected by the QoS requirement. For these jobs the LRMS cannot meet the SLA, and therefore there is a penalty to be paid which affects the value of the utility function.

Both experiments have a similar distribution of new jobs, but their utility functions have very different shapes. This is due to the modification of parameter "b" which increases the penalty for the SLA breach, and it can be observed very clearly at the moments 0-4, 15-18 when the number of jobs in the waiting queue increases and the utility function value decreases significantly in the second experiment.

5. Conclusions and future work

The Grid CE service represents the interface for the local batch system which is implemented as a computing cluster. Using virtualization for creating new systems for the computing cluster on demand has the advantages of scalability and configuration flexibility. Virtualization provides another benefit such as reducing operating costs by systems consolidation. As this can affect the performance we need to consider it for the resource allocation optimization.

The contributions of the paper are:

1. In order to maximize the number of applications which are executed in a period of time, we devise a framework for an adaptive Grid CE service. As the cluster capacity is adapted in response to the workload, it is necessary to automate the reconfiguration process. This ca be achieved with the help of a configuration policy which is monitored and is implemented by a configuration management system that has the role to maintain the system configuration in a stable state. Any deviation from this reference state is corrected so that the system can self-repair itself.
2. For dynamic allocation of resources for the computing cluster we have defined the utility function for Grid CE service. The experiments performed by simulation

show that the utility function provides support for the optimization of the computing cluster capacity as it takes into account the costs associated with resources, the estimated revenue from executing the jobs as specified in SLA, and the penalties for breaching the SLA.

We propose to continue the research to implement and test the framework for an adaptive Grid CE which is set up as an autonomic element that can self-manage itself and can be high-level managed by quality requirements expressed in a SLA.

REFERENCES

[1] *A. Stanciu* and *G. Neagu*, "Help desk structure for the support service of a Virtual Organization supported by multiple Grid infrastructures," in 17th Int. Conference on Control Systems and Computer Science (CSCS17), 2009, **vol. 1,** pp. 429–432.

[2] *S. Traugott* and *L. Brown*, "Why Order Matters: Turing Equivalence in Automated Systems Administration," in Pp. 99-120 of the Proceedings of LISA '02: Sixteenth Systems Administration Conference, (Berkeley, CA: USENIX Association, 2002)., 2002.

[3] *S. Childs, B. Coghlan, D. O'Callaghan, G. Quigley,* and *J. Walsh*, "The Second-Generation Grid-Ireland Deployment Architecture with Quattor Centralised Fabric Management," in In Proc. Cracow Grid Workshop, 2005.

[4] *P. Anderson*, "The Complete Guide to LCFG," 2003.

[5] *R. García Leiva, M. Barroso López, G. Cancio Meliá, B. Chardi Marco, L. Cons, P. Poznański, A. Washbrook, E. Ferro*, and *A. Holt*, "Quattor: Tools and Techniques for the Configuration, Installation and Management of Large-Scale Grid Computing Fabrics," Journal of Grid Computing, **vol. 2**, no. 4, pp. 313–322, Apr. 2005.

[6] *M. Jouvin*, "Quattor: managing (complex) grid sites," Journal of Physics: Conference Series, **vol. 119**, no. 5, p. 052021, Jul. 2008.

[7] *P. Marshall, K. Keahey,* and *T. Freeman*, "Elastic Site: Using Clouds to Elastically Extend Site Resources," in Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing, Washington, DC, USA, 2010, pp. 43–52.

[8] *D. A. Menasce* and *M. N. Bennani*, "Autonomic Virtualized Environments," pp. 28–28.

[9] *J. O. Kephart* and *D. M. Chess*, "The vision of autonomic computing," Computer, **vol. 36**, no. 1, pp. 41–50, Jan. 2003.

[10] *M. Burgess*, "Computer Immunology," in Proceedings of the 12th USENIX conference on System administration, Berkeley, CA, USA, 1998, pp. 283–298.

[11] *W. E. Walsh, G. Tesauro, J. O. Kephart*, and *R. Das*, "Utility functions in autonomic systems," pp. 70–77.

[12] *H. N. Van, F. D. Tran*, and *J.-M. Menaud*, "SLA-Aware Virtual Resource Management for Cloud Infrastructures," 2009, pp. 357–362.

[13] *A. Stanciu, B. Enciu*, and *G. Neagu*, "Auto-administration of gLite-based grid sites," presented at the 2nd Workshops on Software Services, Timi□oara, 2011.

[14] *A. Stanciu, B. Enciu*, and *G. Neagu*, "Auto-administrating and self-repairing systems by enforcing a configuration policy," presented at the 18th International Conference on Control Systems and Computer Science, Bucure□ti, 2011.

[15] *N. Nethercote, P. J. Stuckey, R. Becket, S. Brand, G. J. Duck*, and *G. Tack*, "MiniZinc: Towards a Standard CP Modelling Language," in Principles and Practice of Constraint Programming – CP 2007, **vol. 4741**, C. Bessière, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 529–543.