Computer Science • Vol. 11 • 2010

Dariusz Król[*,**], Włodzimierz Funika[*,**], Renata Słota[*,**], Jacek Kitowski[*,**]

SLA-ORIENTED SEMI-AUTOMATIC MANAGEMENT OF DATA STORAGE AND APPLICATIONS IN DISTRIBUTED ENVIRONMENTS

In this paper we describe a semi-automatic programming framework for supporting users with managing the deployment of distributed applications along with storing large amounts of data in order to maintain Quality of Service in highly dynamic and distributed environments, e.g., Grid. The Polish national PL-GRID project aims to provide Polish science with both hardware and software infrastructures which will allow scientists to perform complex simulations and in-silico *experiments on a scale greater than ever before. We highlight the issues and challenges related to data storage strategies that arise at the analysis stage of user requirements coming from different areas of science. Next we present a solution to the discussed issues along with a description of sample usage scenarios. At the end we provide remarks on the current status of the implementation work and some results from the tests performed.*

Keywords: *data storage, application management, distributed environment*

SEMIAUTOMATYCZNE ZARZĄDZANIE APLIKACJAMI ORAZ SKŁADOWANIEM DANYCH W ŚRODOWISKACH ROZPROSZONYCH Z UWZGLĘDNIENIEM PARAMETRÓW SLA

Artykuł opisuje semiautomatyczny szkielet aplikacyjny służący do wsparcia procesu wdrażania aplikacji oraz składowania dużych ilości danych w środowiskach rozproszonych z uwzględnieniem parametrów jakościowych. Projekt PL-Grid ma na celu wsparcie polskiej nauki w celu umożliwienia naukowcom przeprowadzania złożonych eksperymentów typu in-silico *na skalę wiekszą niż dotychczas. W artykule zostały opisane wyzwania związane ze strategiami zarządzania wielkimi ilościami danych, zdefiniowane w fazie analizowania wymagań użytkowników projektu PL-Grid. Zostały również opisane proponowane rozwiązania omawianych problemów, opis przykładowych scenariuszy użycia oraz aktualny stan prac implementacyjnych i rezultaty przeprowadzonych testów.*

Słowa kluczowe: *składowanie danych, zarządzanie danymi, środowiska rozproszone*

[*] Department of Computer Science, AGH University of Science and Technology, al. Mickiewicza 30, 30-059, Kraków, Poland, {dkrol,funika,rena,kito}@agh.edu.pl
[**] ACC CYFRONET AGH, ul. Nawojki 11, 30-950 Kraków, Poland, {dkrol,funika,rena,kito}@agh.edu.pl

1. Introduction

The development of different areas of science such as chemistry, biology or physics entails an increased complexity of problems which are being solved which is even greater than the observed technological progress. Thus there is a necessity to explore different possibilities to decrease the complexity. One of the possible ways to do this is to apply a more sophisticated algorithm of a smaller computational complexity than the previous one. Unfortunately, some problems (e.g. the travelling salesman problem [1]) do not have fast algorithms which may solve them over a decent period of time, e.g. a few days. In such situations we can try to distribute computation to many nodes (e.g. hundreds) to speed it up. By doing so, however, we may end up with an application of uncontrollable complexity. Another important aspect of maintaining an application in a production environment is the Quality of Service (QoS) which defines thresholds for the application parameters which are related to the non-funtional requirements, e.g. availability or throughput. In most cases the QoS of an application is defined by a contract between an application provider and an application user which is often referred to as Service Level Agreement (SLA). The more complicated the application or the more QoS elements are required, the greater administrative effort has to be put to run the application. In many cases the effort may be too large comparing to the achievable profit which puts into question the profitability of the application. Both aspects, the issue of management and enforcing SLA (and many other issues not mentioned here) are addressed by the Polish national project PL-Grid [2]. It focuses on providing Polish scientists with an infrastructure in terms of hardware and software, which will meet the requirements of modern in-silico experiments. Along with appropriate hardware, dedicated software systems which will fulfil user requirements are necessary. Hundreds of geographically distributed nodes connected together in the PL-Grid production environment is nearly as perfect as possible for the testing of application management algorithms. In the management area, this paper presents a software framework for supporting enforcing SLAs by adjusting the deployment of an application in a distributed environment.

A very challenging subject related to the application management is the storage of large amounts of data in a heterogeneous, distributed environment according to the provided non-functional requirements. The data is either generated by applications or comes from some external sources, e.g. from a network of sensors which measure some indicators of an experiment. Today middleware which run on top of heterogeneous environments exploit mechanisms such as virtualization and abstraction to provide users with a coherent interface to computational and storage elements. Unfortunately, this way, details about specific features of each element are lost, e.g. replication support. Whereas this is not an issue in situations where basic operations on storage are only required, data-intensive applications often need more sophisticated features from storage devices, e.g. high availability which can only be achieved by exploiting a replication mechanism in form of e.g. disk arrays. In such cases, the user has to access this kind of storage by theirselves. This can be too difficult for users who are not experts in data storage. The PL-GRID project aims to solve this issue by allowing the user to specify non-functional requirements related to the storage in an easy way, without special expertise. This paper includes a description of a framework which implements this functionality.

The rest of the paper is organized as follows: in the section that follows, essential reasons for the presented research are discussed. Section 3 briefly overviews related work. In Section 4, sample usage scenarios which exploit the approach under discussion are described. Next, we provide remarks about the realized implementation and data on the tests performed based on the defined usage scenarios. Finally, we conclude the paper and highlight directions for future work.

2. Towards autonomic-oriented management

As mentioned above, the traditional methods of application management are not sufficient when considering highly distributed and dynamic environments such as the Grid. The main reasons for this are: a high cost of human resources (mainly in form of administrators) and detailed knowledge which is required to handle a complex application properly, e.g. to recognize symptoms of an incoming disaster. Additional problems arise when our application has to provide a concrete level of QoS, e.g. due to an SLA contract agreed with users. In these cases the administrator needs to find out what is wrong and perform necessary actions to regain a desired state of the application QoS which may take hours or days. Therefore, more and more datacenters aim at the automation of management procedures, e.g. migrating an application between servers or configuring a data storage strategy.

The automation should decrease, on the one hand, the total time of the SLA restoring procedure, and a needed amount of human interaction, on the other hand. However, this automation has to be more *intelligent* to be able to take over administrator's actions, even partially. By speaking *intelligent* we mean possessing additional knowledge about the runtime environment, e.g. information on a current workload on each node, which can be exploited to manage an application more precisely. One of the current trends in building more intelligent and self-management applications is the Autonomic computing initiative [3]. In brief, an autonomic application can be considered as a set of features, including the environment awareness of its state and self-management, which an application has to exploit in a runtime environment. This basic feature fits into our problem quite well. Since distributed applications are in our area of interest, we would like the *autonomic manager* system to run different parts of the application on different nodes according to the current environment state and required QoS. Whenever one of the exploited nodes gets too much workload and this leads to an SLA violation, the manager must migrate this particular part of the application to another node which is not so heavy loaded. The primary objective of the whole procedure is maintaining the application's QoS on a level defined by an SLA.

From the point of view of data management, so sophisticated functionality is not crucial. Due to the differences between the approaches and goals of application and data management, each scenario should exploit separated systems. While in the case of application management the main roles are played by SLA/QoS, data management deals with allowing the users to pose special storage requirements, to define them in a declarative manner and thus to exploit specific features of different storage devices. However, some of the above features, e.g. knowledge about the runtime environment, are still highly desirable. An important feature which was not mentioned above is the *easiness of use*. As the user may not have knowledge about different types of storage devices, defining non-functional requirements should be as easy as possible. These requirements, e.g. a high availability or minimum write/read speed, are used as filters on the list of the currently available storage devices, which is retrieved from a knowledge base of a Virtual Organisation the user belong to. Moreover, a monitoring system is used to retrieve information about the current values of the required features. As a result, the users' data are stored on a device which is the most suitable one from the point of view of the requirements imposed.

3. Related work

In this section we present the ongoing work related to the issues addressed by the paper. First we concentrate on autonomic-oriented frameworks: ProActive Parallel Suite and a SLA-management oriented, mobile-agent based system. Then, we overview the tools which focus on data management, namely dCache and the XtreemFS. We focus on the features which are related to adapting data storage strategies to user-defined requirements.

The ProActive Parallel Suite [4] is one of the prominent frameworks for developing distributed, multi-core oriented applications with the Java programming language. Its feature set contains: *active object* support which allows to build standard Java objects which are executed in a distributed environment (e.g. Grid) in a transparent way from the user point of view, exploiting *lazy evaluation* which means that a result from operation is evaluated when required and explicit support for *High-level API* which allows to focus on a business logic of an application rather than implementing well known patterns, e.g. Master-Worker, Calcium or Object-Oriented SPMD. When considering the active object concept in ProActive, it is worth of mentioning that each object can be migrated between available nodes simply by calling a single method (called `migrateTo()`) from the ProActive API. Based on this mechanism, ProActive provides support for load balancing algorithms in form of a class hierarchy which can be extended by the user and which can be integrated in an application. One of the exploited paradigms is Peer-to-Peer, it allows to search nodes which can take over active objects from another node. However, the ProActive approach is oriented to different types of algorithms rather than maintaining SLA contracts which specify the QoS level of the application. It does not provide any support for a third-party monitoring systems which would be a precious source of a monitoring data.

An interesting approach to enforce an SLA in a distributed environment is presented in [5]. This approach is based on a mobile-agent concept that defines autonomic components which can be migrated between available resources in a transparent manner. To determine whether an SLA is fulfilled, each element of a workflow which is realized by a distributed application has to provide information about different quality indicators. By combining these information, the fulfilment of an SLA can be determined. In the presented approach, the global SLA is divided into different levels of abstraction to map business level objectives into more technical ones which can be easily observed. The authors propose a *business application* level for describing indicators which concern such fields as: availability, response time performance, security or help desk quality, *business services* level which refers to the indicators related to services, e.g. an audio service or a database access service, and a *network service* level which involves such indicators as throughput, transmission error rate, or jitter. Whenever an SLA is not fulfilled, appropriate actions have to be taken. Due to the presented separation of a global SLA into different levels each part can be handled separately. When one part of an application is overloaded then an agent who is responsible for it can send messages to a nearby agent that could take over some work to balance the overall workload in the application. Unfortunately, the approach tries to optimize the global QoS by optimizing different parts of the application which can not suffice in general. Although, it is referred to as an autonomic system, it lacks some important features of such systems, e.g. shared knowledge sources or behaviour configuration with policies. Its current version is based on distributed flows of requests/data with QoS in mind rather than allowing the user to provide his/her own set of actions which should be evaluated when an SLA is not fulfilled.

XtreemFS is a grid-oriented, distributed file system developed within the XtreemOS european project [6]. The main goal of the project is to develop an easy to use and to administrate, grid operating system which provides an abstraction layer on top of available resources, both computational and storage ones. From the data management point of view, the project provides a modern file system which is optimized to run in a Grid environment. It focuses on features such as: scalability, parallel IO, replication and extendibility. Like many other distributed file systems, XtreemFS separates metadata information from the actual data in order to provide a coherent logical namespace on the one hand and to distribute actual data among available resources on the other hand. The replication mechanism is introduced to provide high availability of the stored data. While this behaviour is appropriate for crucial data which may not be lost in any case, in other cases the replication mechanism generates only overhead in terms of longer time which is necessary to write a single file in many locations. Moreover, the XtreemFS does not provide any means for declaring other non-functional requirements which could be more viable from the user point of view.

DCache [7] is a data management system which implements all the requirements for a Storage Element in a Grid. It was developed at CERN to fulfil the requirements of the Large Hadron Collider as for data storage. One of its main features is the separation of the logical namespace of its data repository from the actual physical location of the data. DCache exposes a coherent namespace built from files stored on different physical devices. Moreover, dCache autonomously distributes data among available devices according to the currently available space on devices, workload and *Least Recently Used* algorithms to free space for the incoming data. Although dCache distributes data in an autonomic way, there are settings which can be configured to tune the dCache installation to specific requirements of a concrete user. This parameter set contains rules which as an input can take a directory location within the dCache file system and storage information of the connected Storage Systems as well as the IP address of the client and as an output such a rule returns a destination where the data should be sent. However, there is no possibility to provide non-functional requirements, e.g. an availability level of the stored date or a throughput of the storage element device.

4. Usage scenarios and architecture overview

In order to conclude Section 3 we can claim that the existing solutions are lacking tools which would manage data and application with fulfilling the previously formulated requirements related to the Grid environment. Although there are some common requirements in both cases, e.g. knowledge-based support and automation of procedures, each management process should be handled by a different tool.

Each of these tools has to provide important architectural features to be compliant to the existing Grid environment. The application management tool (called *Autonomic Manager*) has to retrieve data about the current state of the runtime environment from a monitoring system. As there are many solutions available, it is more beneficial to adapt an existing system rather than to create a new one from scratch. Another important, external element is a shared knowledge base which can be used for storing information by other actors of the system, e.g. an SLA contract uploaded by a user, or information for the Autonomic Manager itself, e.g. decisions, which can be used in the future. The knowledge base can be used to store a configuration of the behaviour of the Autonomic Manager in form of *policy objects*. Each policy object contains information about the needed QoS level of an application capability in form of a metric threshold which can be monitored with a monitoring system. When the QoS level exceeds the threshold, the Autonomic Manager performs indispensable operations to restore the required level. In the research under discussion, the only considered action type is a *migration* of parts of the application between the available resources. This type of intervention is the least invasive one because it does not affect the application business logic, instead it optimizes the application QoS level by changing the deployment layout of the application which can be transparent to the application code. Also, there is a high probability that an increase in workload on the resources is the main reason for QoS dropping, thus making them inadequate for maintaining the SLA agreed.

On the other hand, a data management tool is responsible for retrieving non-functional requirements for the storage from the user, e.g. a type of device or required values for different characteristics. We need to provide access to both a monitoring system, which can measure required indicator values, and a knowledge base, which stores information about the available resources. The last but not least element of the tool design is integration with concrete types of storage elements where the user's data will be actually stored. Implementing this integration heavily depends on the technological aspects of the architecture. Whereas many different storage devices may be supported, this implies the necessity provide a coherent interface to all these devices. Therefore, we assume that the level of a file system will be sufficient for our needs. It allows the user to create files where data will be stored without a need to know its physical location. Also, there is an already defined standard interface for accessing different file systems so there is no need to develop a new one. The designed architecture is depicted in Figure 1.

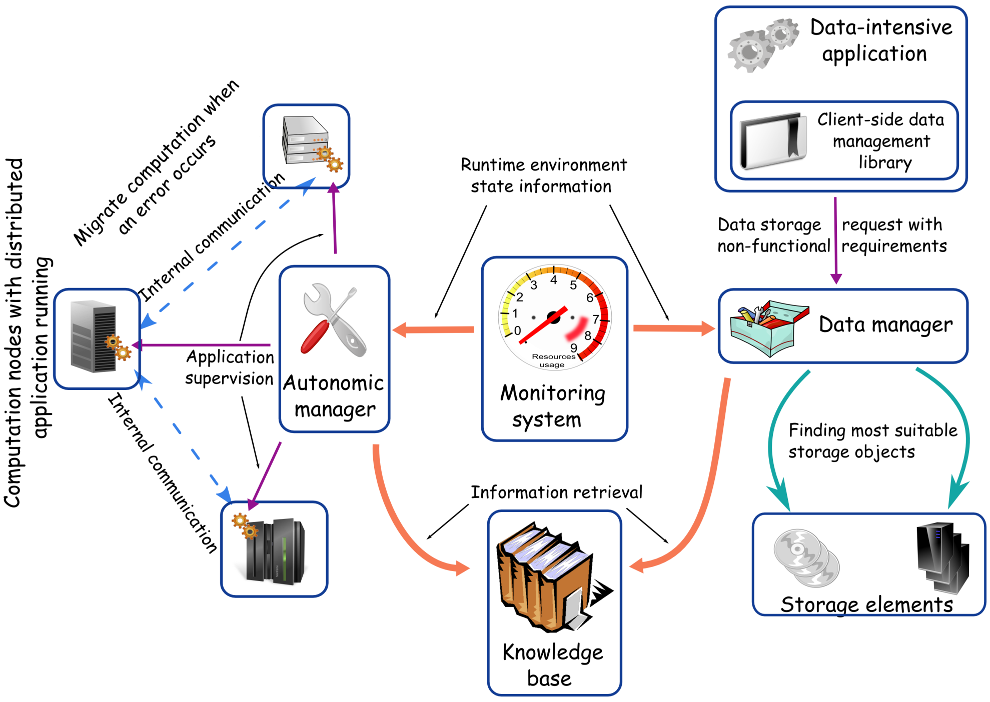
Data-intensive application
Client-side data management library
Migrate computation when an error occurs
Runtime environment state information
Data storage non-functional
request with requirements
Computation nodes with distributed application running
Internal communication
Application supervision
Autonomic manager
Monitoring system
Resources usage
Data manager
Finding most suitable storage objects
Internal communication
Information retrieval
Storage elements
Knowledge base

Fig. 1. Architecture overview of the application and data management system

As the figure suggests, the Autonomic Manager component resides between the monitoring system and the resources on which a distributed application is running. Thus, it creates a classical control loop known from the autonomic computing concept. On each of the supervised nodes, an *agent-like* component has to be installed to accept and perform migration requests from the Autonomic Manager. Once an application on these nodes is started, each part has to be registered with the Autonomic Manager along with a description of the QoS level which has to be kept, e.g. by storing these information in the knowledge base. Then, the Autonomic Manager starts its own algorithms whose main goal is to provide a load balancing procedure with the migration mechanism, which ensures that the SLA is maintained.

On the side of data management, the architecture is even simpler. As with the application management tool, there are necessary connections with the monitoring system and the knowledge base. However, at this aspect similarities end. When considering application management, the Autonomic Manager operates on an application, in case of data management it is the application that is responsible for connecting to a component called *Data Manager*. The direction of a request flow is reversed. The Data Manager can be also regarded as a proxy between a data-intensive application and a set of storage services. There is no need for the application to choose a concrete storage device explicitly, thus it only defines requirements and relies on the Data Manager which compares and selects the most suitable storage device. This way the application can focus on its primary goal rather than on taking care of data storage details.

To present how these two tools work in practice, we will consider two scenarios. The first one is oriented to the application management issue. Let's suppose, a few companies want to cooperate and share information and resources in order to achieve a common goal. They decide to create a Virtual Organisation with a concrete SLA which defines what resources on what terms will be exploited. One of the companies decides to provide an application in the manner of Software as a Service (SaaS) which allows to perform a distributed simulation of some physical problem. Based on the company experience, an SLA contains an entry about the maximum time of a single application run, e.g. 7 days. The company calculated this threshold with a few assumptions:

- the exposed application is divided into three distinct parts which are distributed onto three different machines;
- the first part is computation-intensive and thus requires 90% of CPU time on a machine, let's assume that all machines have an identical CPU;
- the second part is memory-intensive and thus requires 3 GB of RAM memory;
- the third part is so called *master node* and it is responsible for distributing work and collecting results.

As the company wants to guarantee that the SLA will be maintained at any moment, a few more machines are delegated to be ready in case any of the primary machines is overloaded but they can also be used by another application at this time. In a common situation, the company should hire (or delegate at least) an administrator to check whether the SLA is not violated from time to time. However, with the Autonomic Manager system there is no need for doing so. The only additional action which has to be performed after deploying the application is to register the parts of the application, each having the corresponding SLA to the Autonomic Manager along with a configuration description of the available machines. The details of these actions will be presented in the section that follows.

The second scenario presents a use case of the Data Manager tool. Let's suppose, a scientist, e.g. a physicist who is participating in a huge High Energy Physics oriented experiment, has to develop an application which will be responsible for storing data generated by the experiment which generates different types of data: one which is very important and has to be stored at a high availability level, and a second type which is not so crucial but there is a intensive stream of this data which has to be temporarily stored to be next filtered and further processed. The scientist has access to a set of storage devices of different kinds, characteristics and geographical localizations. Unfortunately, an average scientist does not possess knowledge what storage devices should be used to store particular types of data. In a normal situation, he/she would have to spend a lot of time learning about the available devices and its configuration instead of developing the application business logic. Fortunately, the Data Manager can do this work for the scientist. What it needs is a list of available storage devices with a list of characteristics required. Then, the scientist can explicitly pass the requirements for storing different data on different types of storage devices. We will come back to this scenario in Section 5 for more details.

5. Implementation and tests

In this section we study the implementation of both the previously presented tools in more details. Then, we provide results from the tests aimed at simulating the usage scenarios presented in the above section.

The Autonomic Manager tool is based on Java language-related libraries and frameworks. Its core is based on the ProActive parallel suite, due to the provided support for the migration mechanism. The Autonomic Manager assumes that the supervised application is developed with the ProActive suite, thus different parts of the application can be migrated between the available resources. Each of these parts (called *active object* or component in the ProActive terms) has to be registered with the Autonomic Manager like a listener subscribes for an event. In the presented prototype, the user has to use a prepared library which contains Java class definitions which are intended to be used as base classes for user objects. The prepared classes provide a few methods for controlling the deployment of an application. The Java RMI mechanism is exploited to enable remote procedure calls. We addressed the ProActive suite in Section 3. The knowledge base component is implemented as an ontology which is processed by the Autonomic Manager with the Jena Semantic framework. The ontology defines a semantic model of data, which is divided into four main categories:

- *Resources* which describe objects from a supervised domain, e.g. cluster, CPU or an Active Object. Everything that is important from the point of view of Autonomic Manager and has some capabilities which can be monitored can be an element of this set.
- *ResourceCapabilites* contain features which can be directly monitored with a monitoring system, e.g. available memory or service response time. Each of these features is associated with a concrete `Resource` instance.

- *Metrics* describe how a capability is monitored. Each metric can combine different capabilities to provide a meaningful value for the user, e.g. "mean response time of a service over last 5 minutes". The monitoring system can be inquired of the value of a particular metric.
- *Conditions* can be used to construct an SLA which defines a required QoS level, in form of a threshold for a metric which has to be maintained. In addition to the threshold, a relation which associates the threshold with the metric value has to be provided as well. In the current prototype, only basic mathematical relations are supported, e.g. "less than" or "greater than".

An important element of the constructed control loop is a monitoring system. It should be able to process a data model described previously. It would also be desirable that the monitoring system could send notifications when a defined `Condition` is violated. The more sophisticated the monitoring system in terms of the provided features is, the less complicated the Autonomic Manager has to be. If we would like to enable support for any existing monitoring system, we would have to implement a transformation between an ontology based data model to the monitoring system specific model and a notification mechanism based on the values provided by a monitoring system, which would be an unnecessary effort. The SemMon system [8] provides all of the necessary features thus it is well suited for our case. It reads an ontology to get information about the resources which are going to be monitored and thus can exploit other, more appropriate monitoring systems, connected with dedicated adapters, to collect all the necessary data.

On the other hand, we have the Data Manager tool which requires different technologies to be implemented. The implementation is in progress, however we can already provide information about various exploited technologies. The first difference between the Data Manager and the Autonomic Manager is the separation within the former tool of the client side and the server side. The former one is a classic programming library, written with C/C++ programming languages, to be explicitly used by the user applications. Its main goal is to collect information about storage requirements posed by an application, pass them to the server side which finds the most suitable storage device among the available resources, and, finally, returns this information to the application in form of a file descriptor in a distributed file system which is located on the found device. The on-going implementation uses the Lustre file system [9] which is a widely adopted system, developed by the Sun microsystems. It provides a logical coherent namespace with different physical localization of each file with a support of the stripping mechanism. It also provides a *pool mechanism* which allows to group a set of storage devices, e.g. according to its functionality, behind a single name, thus a created file is automatically stripped onto a particular set of devices. The server side is responsible for finding the most suitable storage element that meet the passed requirements. To do this, it combines *static* information, e.g. the availability of a device or its capacity, and *dynamic* information, e.g. current read/write ratio, to find an optimal element. As a knowledge base, the Grid Organizational Memory (GOM) [10] service has been used. It provides methods for accessing an ontology-based model remotely. Once properly configured, it can contain information about available resources, e.g. locations and characteristics of the storage devices.

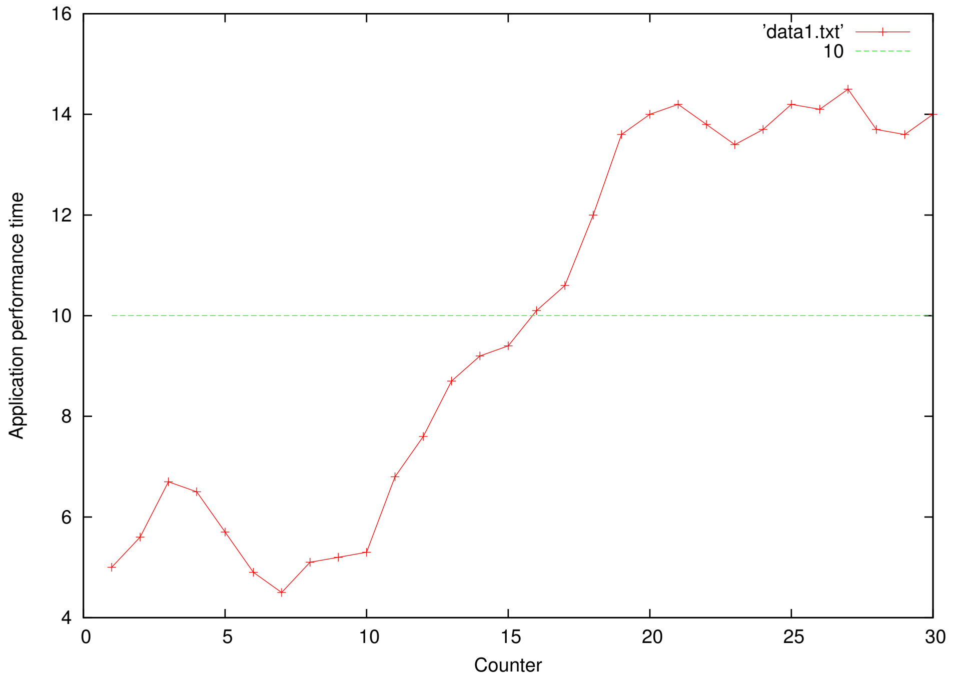

Fig. 2. The `Application performance time` metric plot – 'no Autonomic Manager user' scenario

To conclude this Section, we provide some numbers and plots which depict how the Autonomic Manager can influence a distributed application in a completely automatic way. The tests were based on a usage scenario described in Section 4. However, to speed up the tests, the time scale in the SLA has been changed. Instead of 7 days as a threshold for the application performance time, a 10 minutes threshold has been used. Other units have not been changed. To exclude possible unpredicted Grid phenomena we decided to run tests on several, connected machines. We have run the application 30 times in 2 series. In the first series (Figure 2), we did not used Autonomic Manager to explore how the response time behaves when a workload on a single node is increased suddenly. In the second series (Figure 3), Autonomic Manager has been turned on. As we can see, there is a visible difference between these two series. While, in the former one, when a workload is increased, the SLA is violated until the workload is decreased in a natural way which virtually may not happen. In the second series, the Autonomic Manager collects monitoring data from the SemMon system and can respond to an increasing workload by migrating part of the application from this node to a less loaded node.

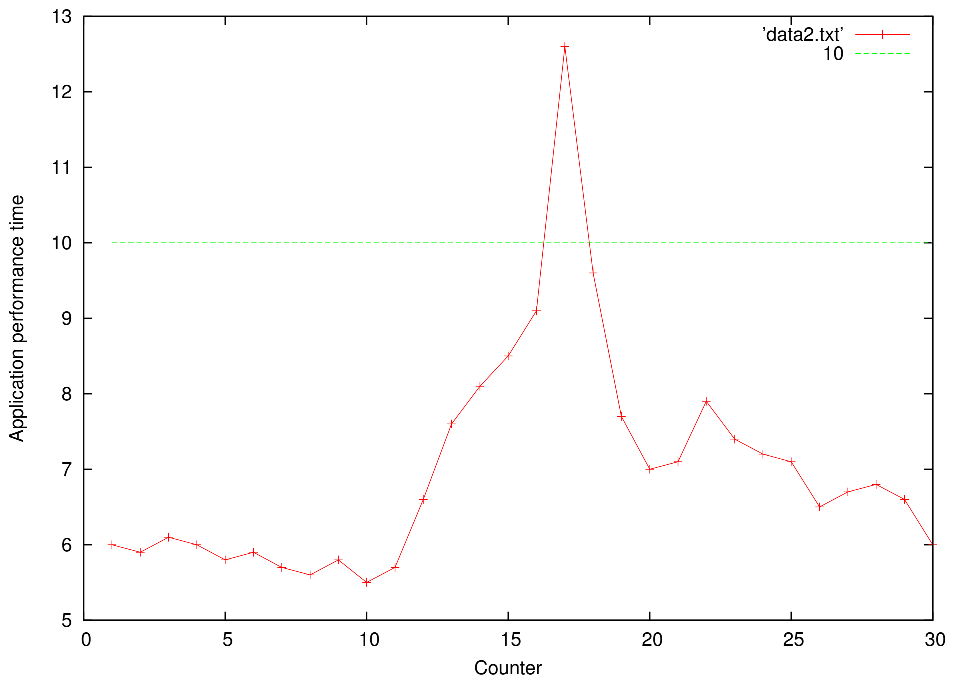

Fig. 3. The `Application performance time` metric plot – ‘Autonomic Manager employed‘ scenario

6. Conclusions and future work

In this paper, we focused on the on-going research related to the application and data management within the PL-GRID project. The main goal of the research is to automate as many activities as possible related to administration of a distributed application according to a defined SLA. On the data management side, it allows users to provide non-functional requirements for storage in an declarative way, which are next mapped onto the available storage devices, to find the most suitable one. In order to demonstrate possible usage of the developed approach, a few use cases were prepared based on the real-live examples from the HEP area. By examining these scenarios, it was possible to get a valuable feedback on the approach. Some important changes were introduced based on the gathered feedback. The designed tools can be exploited by the application developers within any distributed environment, e.g. Grid, with hardly any extra effort needed. Results from the presented tests are promising and show that the described approach can be applied to different scenarios and some positive effects can be achieved, i.e. maintaining the QoS level.

However, the approach is still evolving and some new directions and enhancements are considered. The most important further work includes:

- *Dynamic reconfiguration* of the Autonomic Manager by providing new SLA entries or changing the existing ones. It is a highly desirable feature especially when considering long-lasting applications.
- *Support for different types of rescue operation* when an SLA contract is violated. Currently, only the migration mechanism is exploited but there are other worth considering, e.g. process replication to many machines.
- *Graphical User Interface* is an enhancement of the administrative part of the Autonomic Manager. The presented prototype is a pure, console-based tool which can be rather discouraging from a non-export point of view, thus a GUI tool could enlarge the number of users of the Autonomic Manager tool.
- *Support for different storage device types* is a future direction for Data Manager. It is more difficult to achieve the integration of different types of storage, e.g. file systems, disk arrays and hierarchical systems, is a must in a highly distributed and heterogeneous environments.

Acknowledgements

The research presented has been partially supported by the European Union within the European Regional Development Fund program no. POIG.02.02.00-00-007/08-00 as part of the PL-Grid Project.

References

[1] Applegate D. L., Bixby R. E., Chvátal V., Cook W. J.: *The Traveling Salesman Problem: A Computational Study.* Princeton University Press, ISBN 978-0-691-12993-8.

[2] *The PL-GRID project website.* `http://www.plgrid.pl/`.

[3] *An architectural blueprint for autonomic computing.* `http://www-01.ibm.com/software/tivoli/autonomic/pdfs/AC_Blueprint_White_Paper_4th.pdf`.

[4] Caromel D., Delbe Ch., Costanzo di A., Leyton M.: *ProActive: an integrated platform for programming and running applications on Grids and P2P systems.* Computational Methods in Science and Technology, vol. 12, 2006, pp. 69–77.

[5] Schmidt H., Kapitza R., Hauck F. J., Reiser H. P.: *Adaptive Web Service Migration.* Lecture Notes in Computer Science, vol. 5053, Springer Berlin/Heidelberg, 2008, pp. 182–195.

[6] *The XtreemOS project website.* `http://www.xtreemos.eu/`.

[7] Fuhrmann P.: *`dcache`: the commodity cache.* Proc. of 12th NASA Goddard and 21st IEEE Conference on Mass Storage Systems and Technologies, 2004.

[8] Funika W., Godowski P., Pegiel P.: *A Semantic-Oriented Platform for Performance Monitoring of Distributed Java Applications.* Proc. of International Conference on Computational Science, 2008.

[9] *Lustre file system website.* `http://wiki.lustre.org/index.php/Main_Page`.

[10] Kryza B., Pieczykolan J., Kitowski J.: *Grid Organizational Memory: a versatile solution for ontology management in the Grid.* Proc. e-Science 2006 – 2nd IEEE International Conference on e-Science and Grid Computing, Amsterdam, Netherlands, 2006.