Scientific Programming 20 (2012) 69–80
DOI 10.3233/SPR-2012-0339
IOS Press

Storage QoS provisioning for execution programming of data-intensive applications

Renata Słota
Department of Computer Science, Faculty of Electrical Engineering, Automatics, Computer Science and Electronics, AGH University of Science and Technology, A. Mickiewicza 30, 30-059 Krakow, Poland
E-mail: rena@agh.edu.pl

Abstract. In this paper a method for execution programming of data-intensive applications is presented. The method is based on storage Quality of Service (SQoS) provisioning. SQoS provisioning uses the semantic based storage monitoring based on a storage resources model and a storage performance management. Test results show the gain for the execution time when using the QStorMan toolkit which implements the presented method. Taking into account the SQoS provisioning opportunity on the one hand, and the increasingly growing user demands on the other hand, we believe that the execution programming of data-intensive applications can bring a new quality into the application execution.

Keywords: Storage resources, Quality of Service, data-intensive application, distributed environment, storage management, grid

1. Introduction

The e-Science paradigm allows for doing experiments at a much higher level of cooperation, scale and cost saving. An e-Science experiment is carried out in a virtual laboratory constructed by applications running in distributed computing environments like Grids or Clouds and shared scientific equipment. It is a common case that such applications are data-intensive and need large storage resources. Since more than one third $(1/3)$ on average of the application execution time is devoted to I/O related operations there is a strong demand for efficient data access methods with predictable performance. In addition there is an increasing gap between computation and I/O performance, which in turn constantly builds up the role of storage in the high performance computing area.

The Quality of Service (QoS) of storage is an essential issue for time constrained applications, e.g., real time visualization, recording/filtering/processing of unique irreproducible data sequences, e.g., from radio telescope. Assuming the possibility of storage QoS (SQoS) provisioning of the given computing environment it could be possible to guarantee or eventually decrease application execution time taking into account the costs associated with running the given application. The SQoS requirements can be defined in the form of Service Level Agreement (SLA).

Storage QoS provisioning in distributed environments is a challenging task according to the heterogeneity of storage resources, the dynamicity of storage nodes load distribution, the uncertainty of user behavior, the resource sharing in virtualized environments introduced for better resource utilization. The introduction of a common model for storage resources, which unifies the storage resources monitoring (and especially the performance monitoring), allows for more efficient management of storage resources for the purposes of SQoS provisioning. The use of such a model not only simplifies the monitoring itself but also makes it possible to introduce a common layer for access to the monitoring parameters including QoS metrics. It is important that the construction of appropriate effective and flexible systems for storage resource management is easier with the use of semantic technologies, also for modeling storage resources.

Taking into account SQoS provisioning on the one hand, and the increasingly growing users demands on the other hand, we believe that the execution programming of data-intensive applications can bring a new quality into the application execution. The term "execution programming" means the ability of formulating the conditions of applications' execution (e.g., SQoS metrics ranges), which need to be met at runtime. If the application does not need to change its requirements at runtime then the conditions can be statically defined before the execution. Otherwise a special application

programming interface for specifying and modifying the conditions at runtime is needed.

In this paper a method for execution programming of data-intensive applications is shown. The method is based on SQoS provisioning. An essential part of the article shows a way for achieving SQoS through the use of the mentioned common model, performance monitoring, and management strategies for storage resources. For that reason the paper has the following structure: after presenting the state of the art in the second section, the C2SM model, and a storage resource monitoring based on it, are described in Section 3. The next section presents how the mentioned monitoring can be used for the QoS-aware data management. The QStorMan toolkit, which implements the idea of using the semantic-based monitoring for management of the data used by data-intensive applications, is described in Section 5. In this section the architecture of the toolkit as well as its interfaces enabling the execution programming of applications are comprised. The next section directly concerns the execution programming, where two basic use cases of QStorMan are shown. Section 7 contains test results, which show a gain for the execution time of data-intensive application.

2. Related work

This paper focuses on the use of the possibility to specify storage QoS requirements for influencing the storage systems' behavior on behalf of data-intensive applications during their execution. Therefore, the selected examples of research in the field of SQoS provisioning, sharing and management of storage resources, as well as I/O operation characteristics are presented below.

2.1. QoS for storage

Many of the studies in the field of SQoS provisioning are dedicated to delivering SQoS for the file systems.

In [2] the authors present a QoS based parallel file transfer to cater customers' requirement. The proposed algorithms find out the best resources, which should be used to transfer the requested data, decide the size of part of each replica which should be transferred, and estimate time and cost to accommodate the requirement. The proposed algorithms enable the economical use of distribute grid resources. In [22] the Bourbon framework is presented. It uses the Ceph peta-scale object-based storage system. Bourbon enables Ceph to become QoS-aware by providing the capability to isolate performance between different classes of workloads. The QoS mechanism in this case accommodates the stripping and randomized distribution of data used in Ceph. The global-level of QoS relies upon the low-level QoS provided by the individual object-based storage devices being part of the system. In [5] a QoS-aware file system called Apollon, which can efficiently handle mixed workload is presented. The authors propose a practical I/O scheduling mechanism to prioritize the incoming disk I/O requests. This scheduling is implemented in a deadline-driven I/O scheduler and admission control module. The Apollon system is reported to have superior performance in guaranteeing the QoS requirement of real-time requests.

In the presented paper we obtain SQoS by a suitable model of storage resources, monitoring storage load and cost estimation of data access, as well as through the use of semantic technologies which help in the storage and data management and the SQoS monitoring.

2.2. Provisioning, sharing and management of storage resources

Given that the research concerning the efficient storage provisioning address many layers in the I/O path it is also important to study the techniques of management and the organization of access to storage resources.

Overview of the problems associated with distributed data sharing, management, and processing in data grids can be found in [20]. The authors provide comprehensive taxonomies that cover various aspects of architecture, data transportation, data replication and resource allocation, and scheduling. The key concepts of data grids are compared with other data sharing and distribution paradigms like: content delivery and peer-to-peer networks and distributed databases. Based on this taxonomy they identify areas for future exploration. In [21] an intelligent storage system is presented. The system focuses on I/O performance optimization for a soft real time application. It uses the knowledge obtained by the intelligence module which determines the class of incoming workload to improve the performance using pro-active storage optimization. Machine-learning based techniques were used to train the intelligence module. The system uses workload monitor and analyzer, and system optimizer. The system is an autonomic system, which does not require any changes to existing interfaces. In [23] the con-

cept of reference storage system (RSS) is proposed. The authors state that the latency and bandwidth are not good requirements for SLA since the users hardly know what values make sense and this is why the RSS interface is proposed. RSS is a storage system chosen by users for which the performance is measured and then used as performance interface between application and the mimicked storage servers. The performance is managed by migrating virtual storage devices. A machine learning model is used to implement the interface.

The above presented examples of storage management methods are very sophisticated. In this paper only simple one based on heuristics is presented, but this method is sufficient to obtain good results.

2.3. I/O characterization

Because of a wide variety of data-intensive applications existence this is a wide variety of workloads generated by these applications. A number of studies assumes that each of the workloads has its own characteristics for which specific actions, e.g., management, tuning, configuring can be performed to achieve optimal throughput.

A methodology for evaluating of I/O performance on computer clusters under different I/O configurations is proposed in [8]. Three levels of the I/O path are considered: application level, I/O system level and I/O devices. The method is based on selecting different system configuration and I/O parameters and evaluating the performance for the given configuration. This information is then used to select the most appropriate configuration for the given application. Another methodology for system wide I/O characterization is proposed in [1]. The authors present a mechanism for capturing detailed application-level behavior, which allows to identify the system-wide trends and the application-specific I/O strategies. The method uses storage device instrumentation, static file system analysis and monitoring. Such studies are interesting and can be used to support the automatic detection of the applications' characteristics, which in turn can help to find what are the exact application requirements. Such aspects are covered in [6], where the behavior-inspired data management is proposed.

Our assumption is that the application behavior is changing during its run-time and that the knowledge about the changes is placed in the application. Therefore, an application interface for SQoS requirements is needed. This interface also allows users to choose a less expensive method of the application execution.

3. Model C2SM and storage resources monitoring

The monitoring of storage resources is essential for the systems where QoS and SLA are involved. The monitoring provides input data to other modules which use these data to make performance influence decisions aimed at meeting the SLA/QoS requirements.

A storage resource can be a complete storage system like Disk Array (DA) or Hierarchical Storage Management (HSM) system, as well a simpler one like HDD (hard disk drive). In order to simplify the monitoring of different kinds of storage resources a common model for representing the storage resource state from a performance point of view is needed. Such a model, the C2SM model, has been developed as part of the OntoStor project [17]. The model consists of a set of performance related parameters and state transition algorithms allowing for future performance evaluation. A simplified version of the C2SM model is shown in Fig. 1. The model has been developed as an extension to CIM (Common Information Model) [3], which is used to model IT components with the goal of information management.

The model defines a generic storage system (AGH_StorageSystem) and two more specific types of storage systems: HSM systems and disk-based systems. The disk-based systems are represented by disk arrays, cluster filesystems (e.g., Lustre) and local disks. It is essential in the model that it defines common performance attributes for various storage systems, which allows for unifying the view of a storage system from the performance related aspects.

The monitoring system consists of sensors and a monitoring service (see Fig. 2). The sensors are installed on the storage access nodes which are attached to the storage resources. They are specialized for the given type of storage resource and send monitoring data to the monitoring service. The interface for accessing monitoring data is common and specified by the C2SM model. This greatly simplifies extending the monitoring systems since only the storage system dependent sensors, which are rather simple ones, need to be added.

The storage systems can differ in terms of the methods of retrieving of performance related data. That is why the common model and the appropriate interface based on it for representing storage performance related parameters are important. Based on the model and the current state of a given storage system represented by values of the parameters defined in the model it is possible to estimate the data access time. The

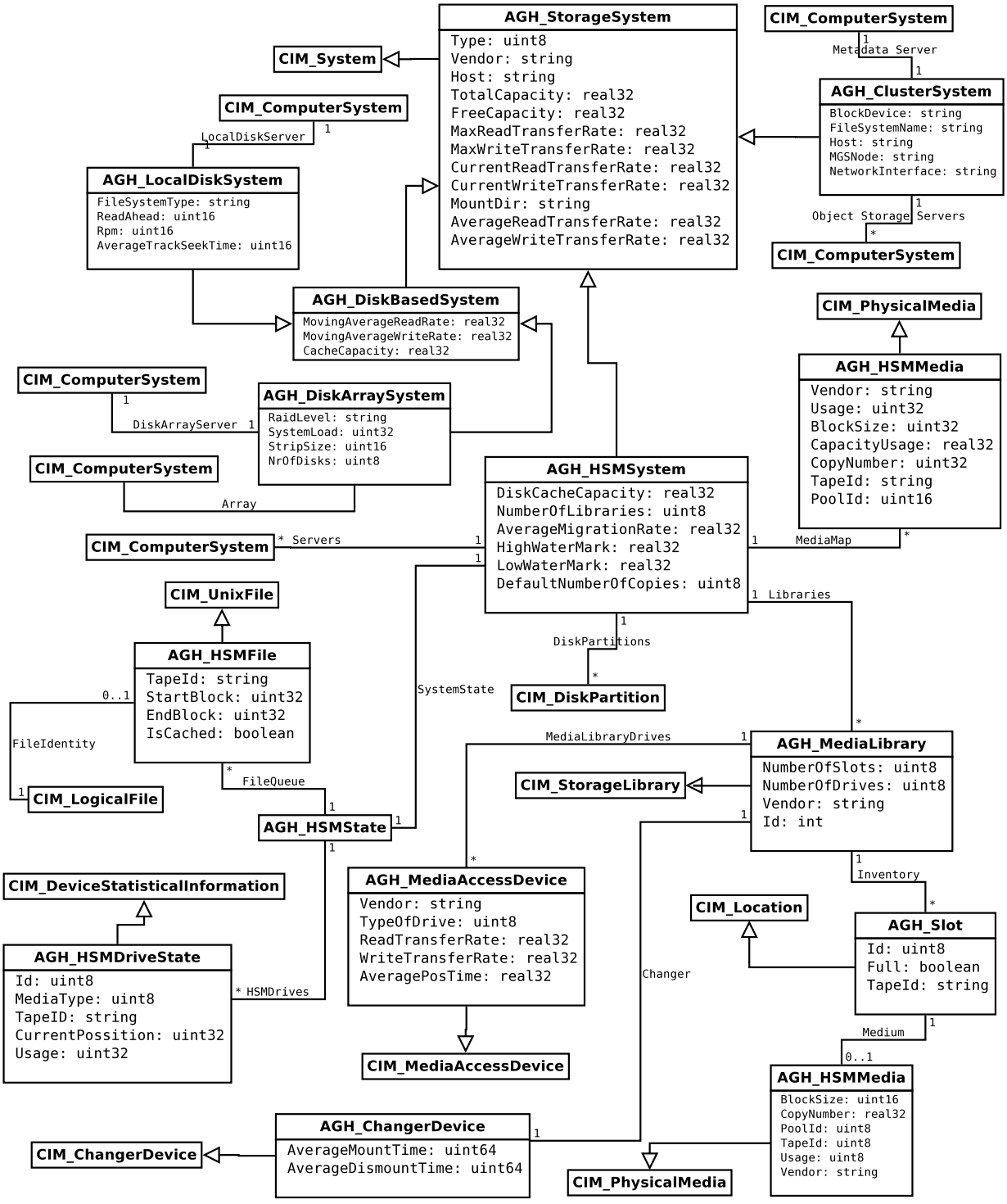

Fig. 1. C2SM model.

C2SM model allows to create accurate estimators, even for complex storage systems such as HSM [7]. The values of the essential performance parameters coming from the monitoring as well as the ones produced by

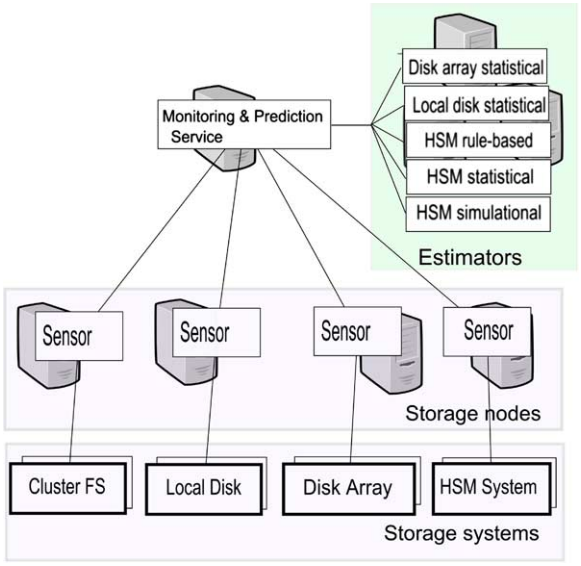

Fig. 2. Monitoring system with estimation capabilities. (Colors are visible in the online version of the article; http://dx.doi.org/10.3233/SPR-2012-0339.)

the estimators (Monitoring and Prediction System in Fig. 2) are used for efficient storage resource management (see for example [14]).

Based on C2SM a relevant semantic model and two ontologies allowing for semantic description of storage resources [10], storage QoS metrics and SQoS monitoring systems [13] were built.

4. Storage management with QoS

The SQoS provisioning can be realized by an appropriate storage resource management, which takes into account the past, current and predicted performance of a storage system.

The idea of storage management with QoS using a storage resources monitoring and prediction system is presented in Fig. 3, in which a part of the distributed storage system is shown. The proposed architecture consists of:

- *monitoring and prediction system*, mentioned in the previous section,
- *storage performance manager*, which analyzes the performance and prediction data and taking into account the given SQoS requirements provides the resource broker with hints about the effective use of managed storage resources. The hints also depend on the current management policy. There can be defined multiple policies emphasizing certain performance attributes.

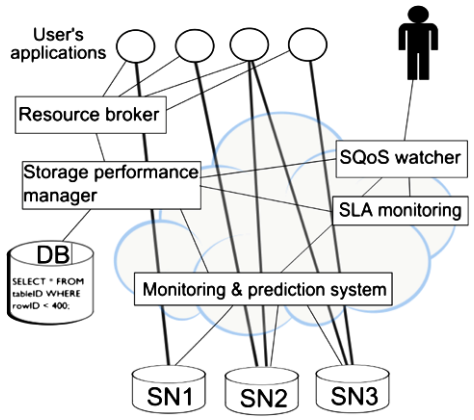

Fig. 3. Idea of data management with storage resources monitoring for SQoS provisioning. (Colors are visible in the online version of the article; http://dx.doi.org/10.3233/SPR-2012-0339.)

- *resource broker*, which schedules client requests assigning storage resources. The broker takes into account the hints received from the performance manager.
- *SQoS watcher*, which monitors the specified SQoS metrics. The SQoS metrics are calculated based on the values of the low level parameters provided by the monitoring system.
- *SLA monitoring*, which periodically verifies for SLA violations and sends alerts to the storage performance manager. This allows for automated performance management for the purpose of keeping the SLA fulfilled. The SLA monitoring keeps a history of SQoS metrics, which allows to calculate the percentage of the requests with unsatisfied SQoS. In this way the SLA monitoring can check if the given SLO (Service Level Objective) is fulfilled over a given time period.

The presented approach allows for storage resource performance management for fulfilling the required SQoS for a given client request. The performance management makes use of such techniques like data replication, prefetching, partitioning/stripping, parallel data transfers.

Data replication is a common technique for increasing the data availability and performance. In a distributed environment there can be many locations (storage nodes), where a new replica can be created when needed. Automation of optimal replica placement is a challenging task since it depends on many factors, some of which are hardly predictable, e.g., the users activity in the near future. Another decision to be made concerns data reading when multiple replicas are avail-

able. In this case the most appropriate replicas need to be selected.

Prefetching is a speculative copying of data in advance to a location where the data can be accessed faster. This (the faster location) can be the disk cache of an HSM system, a storage node with better network connection to the client or just the local clients' storage. The problem in this case is to predict which client will request which data and when.

With the data partitioning the data parts are distributed to more nodes for successive parallel data transfers, which allows for higher throughput.

Depending on the purposes of storage resource management, appropriate management strategies are applied. These strategies called policies provide SQoS as the "best efforts" or by the appropriate queuing and resource reservation to guarantee the delivery of SQoS (even in the form of SLA contracts).

The next section gives a description of how the proposed method has been applied in the implementation of a toolkit for storage QoS provisioning called QStorMan.

5. The QStorMan toolkit

The QStorMan (Quality Storage Management) toolkit is an example of a set of tools for providing storage QoS. The main objective of the QStorMan toolkit is to manage the data coming from data-intensive applications in distributed environments. The toolkit allows users to define non-functional requirements for storage resources explicitly. SQoS provisioning of the QStorMan toolkit is possible due to: (1) semantic descriptions of storage resources and user requirements exploitation, (2) information from storage monitoring system and knowledge base usage to find the most suitable storage system compliant with the defined requirements.

In order to show how QStorMan provides storage QoS the QStorMan architecture and its interfaces are presented below.

5.1. QStorMan architecture

The architecture of the QStorMan toolkit is depicted in Fig. 4. There are the SES (Storage Element Selection) service and libraries, the GOM (Grid Organizational Memory) knowledge base, and the SMED (Semantic Monitoring and Estimation System) monitoring system.

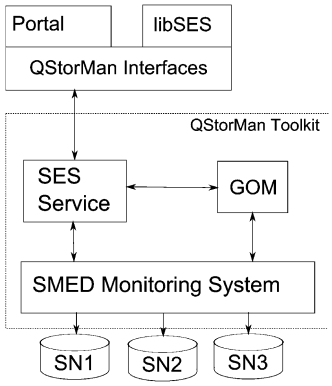

Fig. 4. QStorMan architecture.

The SES service finds the most suitable storage node (SN) according to the user defined requirements and current storage resources workload using policies. The SES service cooperates with the SES libraries. The libraries consist of two separate elements. The first is a programming library which is explicitly used by application developers by calling an exposed API in an application code. The second is a system library which resides on a server where an application is running. This library is responsible for intercepting open file system calls made by the application and for redirecting these calls to the most suitable storage system.

The GOM knowledge base stores semantic descriptions of the configuration of the storage resources along with defined non-functional requirements from the users. It cooperates with the SES service delivering relevant knowledge.

The SMED monitoring system monitors storage resources and provides information about the current or average values of various SQoS parameters through the defined relevant SQoS metrics.

The basic use case for the QStorMan toolkit is as follows (see Fig. 5). The SES library communicates with the SES service sending a query for selecting the storage node, which should be used for I/O operation by the clients' application. The SES service queries the GOM about the available storage resources and next queries the SMED monitoring system about the values of the relevant SQoS metrics. Based on the application's SQoS requirements and the feedback from the SMED system, the SES service chooses the appropriate SN.

If an application is executed in a grid environment, there are two use cases of its execution using the QStorMan toolkit: (1) global selection of computing center, (2) local acceleration of the application execution. The QStorMan interfaces allow the user to select the mode of QStorMan support.

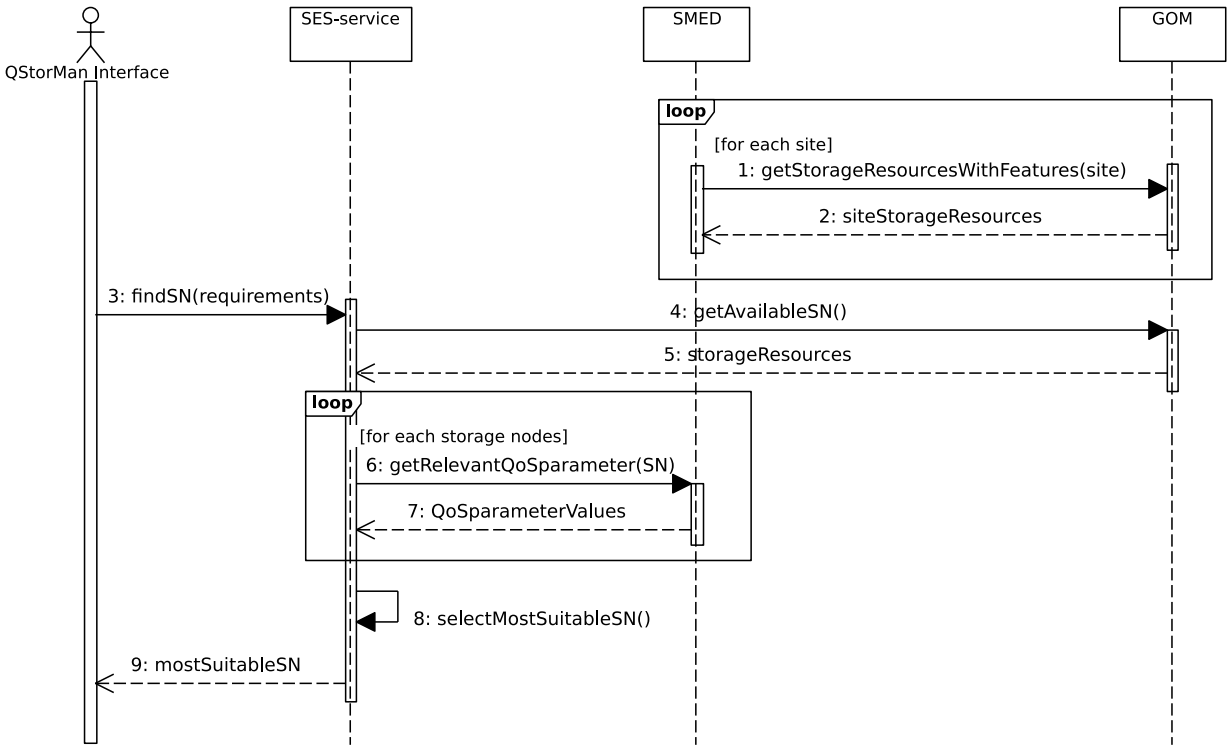

Fig. 5. Cooperation between the main modules of QStorMan for choosing the most suitable storage node.

5.2. QStorMan interfaces

The QStorMan toolkit provides interfaces for defining non-functional requirements. Using these interfaces the end user can influence the distribution of data produced by his/her application by specifying only SQoS high-level requirements.

There are two QStorMan interfaces for defining non-functional requirements: a web-interface and an Application Programming Interface (API).

The QStorMan web-interface is used to define non-functional requirements, which are stored to the GOM knowledge base. These requirements are used to accelerate application execution on a selected CN (by the user or by QStorMan). In this case the requirements are defined once per application run time.

The QStorMan API is used to define requirements from an application code. Using the API it is possible to change the requirements at application run time.

The QStorMan API allows for creating, opening and closing files and changing a storage policy (predefined strategy for assigning SN for I/O operations). The C++ API contains the following functions:

- _createFile(fileName:char*, policy:StoragePolicy*) : int
- _openFile(fileName:char*) : int
- _closeFile(fileName:char*) : void
- _changeStoragePolicy(fileName:char*, policy:StoragePolicy*) : void

The policies define methods of data application management. According to the user SQoS requirements and the chosen policy, an appropriate SN is assigned for the incoming I/O operations. The SMED monitoring system provides the current values of SQoS parameters based on the defined SQoS metrics. The proposed polices are:

- Best Storage Node (BSN): the current SQoS metric value (CV) for SN has to accomplish the following requirement: $CV > \alpha \times RV$, where RV is a requested metric value and α is an experimentally chosen coefficient ($\alpha > 1$). SN with the greatest value of the difference $CV - RV$ is chosen.
- Round Robin with Monitoring (RRM): those SNs which have the current SQoS metric value greater than the required one are taken into account, and they are assigned in round robin fashion.
- Strict QoS (SQ): the SQ policy is implemented by scheduling the applications' data transfers in such a way that no more than Nmax concurrent transfers per SN are performed.

- QoS with Monitoring (QM): the QM policy is implemented by scheduling only those transfers for which there is an available bandwidth to meet the SQoS requirements. The available bandwidth is obtained from the monitoring system just before the scheduling. The following heuristic is used:

$$\frac{sn_{\text{load}}}{np+1} > rb \times k,$$

where sn_{load} is the current storage node load, np is the number of scheduled active data transfers, rb is the requested bandwidth and k is the overhead coefficient.
- Round Robin (RR): SNs are assigned in round robin fashion, no information from the SMED monitoring system is taken into account; RR policy is defined for testing purposes.

In the next section usage of the above described QStorMan toolkit for the programming of data-intensive applications is shown.

6. Programming of execution of data-intensive application

The execution time of data-intensive application depends among others on the number of I/O operations. The overall execution time for such applications can be lowered via reducing the time consumed by I/O, e.g., with selecting the most appropriate storage system which can meet the given requirements.

The ability of specifying the storage QoS requirements as well as the ability of meeting them are the necessary conditions for reducing the execution time. In the previous sections we showed a method for SQoS provisioning and its implementation in form of the QStorMan toolkit. In this section we show examples of execution programming of data-intensive application conditioned by the mentioned SQoS provisioning.

We distinguish two types of applications. The first type represents the applications for which the SQoS requirements are defined once, before the application execution, and they do not change during run time. The second type relates to more demanding applications which change their SQoS requirements during execution. These changes are needed, e.g., due to changes to the amount of data being read or written. Such requirements may also take into consideration the cost of data access.

The SES libraries of the QStorMan toolkit allow the execution programming of the above mentioned two types of data-intensive applications. Each of the SES libraries is used in different use cases:

- The system library supports the applications which are not aware of the library existence. These include legacy code applications which cannot by modified anymore or applications whose SQoS requirements are not complex and can be defined once for all I/O operations before the application is executed.
- The programming library supports development of new applications which require special treatment of the processed data.

Two use cases of the SES libraries for programming of execution data-intensive application in distributed environments are described below. The first one describes the usage of the system library for computational node selection and I/O operation acceleration. The second one, which uses the QStorMan API, relates to the possibility of making dynamic changes to SQoS requirements during the execution of the application.

6.1. Computational node selection and I/O acceleration

An example of execution programming of data-intensive application in Grid environments is the QosCosGrid (QCG) [19] execution environment. This environment is intended for executing large jobs, which need thousands of cores geographically distributed across some number of computer centers. The QosCosGrid environment allows for reservation of such an amount of resources. There is a possibility for speeding-up of data-intensive applications by selecting the appropriate storage system at the computing site level. The acceleration is provided by the QStorMan toolkit, which is integrated with QCG. The current version of the QCG software allows for specifying that the application is data-intensive. In this case the QStorMan system libraries are started (corresponding system variable is set) and storage resources are assigned according to some pre-defined policy, for example assigning the currently least loaded resources.

Acceleration of the application execution at runtime directly with the SES system library is also feasible. In this case the user should:

- declare the non-functional requirements in the GOM knowledge base (it can be done with the QStorMan portlet),
- export LD_PRELOAD=<path_to_libses_wrapper_library>.

The execution programming of data-intensive application can also be realized via the QStorMan web interface. The web interface is integrated with a portal which is used for placing the users' requirements into the GOM. Through the portal the QStorMan toolkit proposes a computing site which can meet the applications' SQoS requirements. Using the proposed site ensures more efficient application execution (see the test results in Section 7). When using the portal for such a use case the user should do the following three steps:

- declare the non-functional requirements in the QStorMan portlet,
- copy and paste the returned text from the portlet to the JDL file,
- send the job with the application to the grid environment.

It is possible to use the both use cases at the same time: the selection of computing nodes and the local acceleration by the SES system library.

6.2. *Use of QStorMan API for dynamic changes in QoS requirements*

In this case the execution programming of data-intensive applications takes place through changing the SQoS applications' requirements from the application code.

The QStorMan API allows for specifying the SQoS requirements in the application source code. So it is possible to change the SQoS requirements during runtime according to the applications' control flow. The API is similar to the standard file-related API known from the standard C++ library. The most important deference is an additional parameter, called StoragePolicy. The parameter holds non-functional SQoS requirements for a new file. The storage policy is configured by setting various attributes of the policy. An example of use of the C++ programming library is shown in Fig. 6.

The following attributes of the standard policy are set up: file size and average write transfer rate requirements, both being dependent on the file size predicted under the next computational step of simulation.

7. Test examples

For verification of the proposed execution programming method the following tests of the QStorMan toolkit were carried out, using the real test environment provided by the PL-Grid [18].

(1) Test set I (TS1) – computing nodes selection tests in a distributed Grid environment,
(2) Test set II (TS2) – data-intensive application acceleration tests in a given computing site,
(3) Test set III (TS3) – QStorMan policy tests for SLA/QoS requirement fulfillment.

The purpose of the TS1 was to verify the selection of a computing site for executing the given data-intensive application in compliance with SQoS requirements. The selected site should be able to meet the SQoS applications' requirements. After the selection, the application is sent to the job queue of the proposed site. On the contrary, the same application, but without use of the QStorMan functionality, is sent to the next site in round robin fashion for each iteration of the test. As we can see in Fig. 7 the time decrease may be significant.

The tests from TS2 are focused on the execution time of the application, which is run in one site. The site, in which the tests were carried out, uses the Lustre [16] filesystem for storing the application data. The Lustre filesystem provides disk pools allowing for better control/allocating of the Lustre disk resources. The pool to be used for I/O by the application is recommended by the QStorMan toolkit (using a policy which balances the load for the pools – the BSN policy in Section 5.2). These tests simulate the storage activity for a certain number of users running the same simulating data-intensive applications on the Grid. Only a part of the users makes use of the QStorMan toolkit. The algorithm of the simulating application is shown below.

```
1   sleep(t_start);
2   for (int i = 0; i < N; i++) {
3      writeData(amount);
4      sleep(t_cont);
5    }
```

The sample test results for 10 simultaneous users of the mentioned application, with $N = 10$ and amount $=$ 50 GB, are depicted in Fig. 8. In general, the average execution time decrease, in the local site, gains about 20–50% during the tests.

The tests from TS3 concern the ability of the QStorMan policies to meet SQoS requirements. The only storage QoS requirements for these tests is that the average I/O bandwidth measured for each application (data transfer) should be greater than 10 MB/s. The results for selected policies (SQ and QM, see Section 5.2) are shown in Figs 9 and 10. For these tests additional background storage load is applied. We can

```
1    #include <LustreManager.h>
2    #include <StoragePolicyFactory.h>
3    using namespace lustre_api_library;
4    LustreManager manager;
5    StoragePolicy policy;
6    int file_size, descriptor;
7    void *buf;
8
9    for  (int i=0; i<MAX_STEP; i++) {
10     file_size = computational_step ();
11   // setting the file size requirement, depending on the computation results
12     policy.setCapacity(file_size);
13   //setting the average write transfer rate, depending on the file size
14     if (file_size < 4000)
15       policy.setAverageWriteTransferRate(50);
16     else  policy.setAverageWriteTransferRate(100);
17
18     descriptor = manager.createFile(file_name, &policy);
19     for (int j=0; j< MAX_DATA; j++) {
20       buf = data_block(j);
21       write(descriptor, buf, BLOCK_SIZE);
22     }
23     manager.closeFile (file_name);
24   }
```

Fig. 6. Usage of QStorMan API – example.

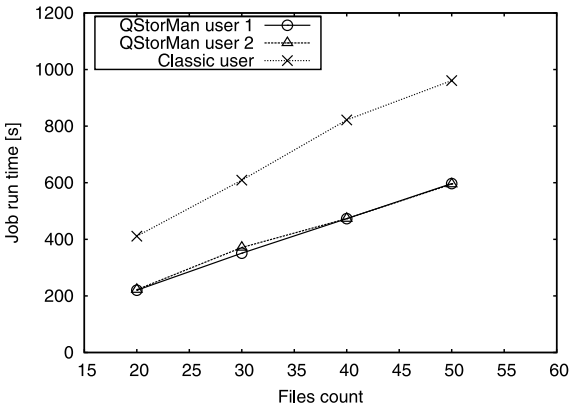

Fig. 7. TS1 – time decrease of data-intensive application due to choosing the appropriate computing node.

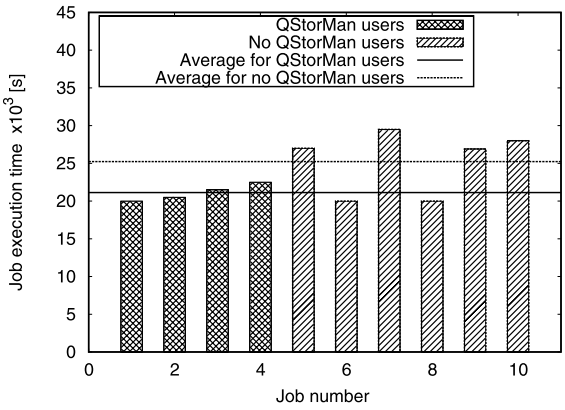

Fig. 8. TS2 – time decrease of data-intensive application due to the local storage load balancing.

see that the policy, which uses monitoring – QM has better SQoS fulfillment. Even though for some of the transfers the SQoS requirements are not met, they are still very close to target rate.

It should be noted that the additional load of the QStorMan toolkit posed to the Grid environment was thoroughly studied. The elements, which influenced the storage performance most, were the monitoring sensors doing active storage performance measurements of the SNs. Due to proper configuring of the sensors the introduced monitoring overhead to the Lustre filesystem was less than 5%.

8. Conclusion

Most of the applications solving the present scientific challenges under the e-Science paradigm need to access large amount of data. That is why the research

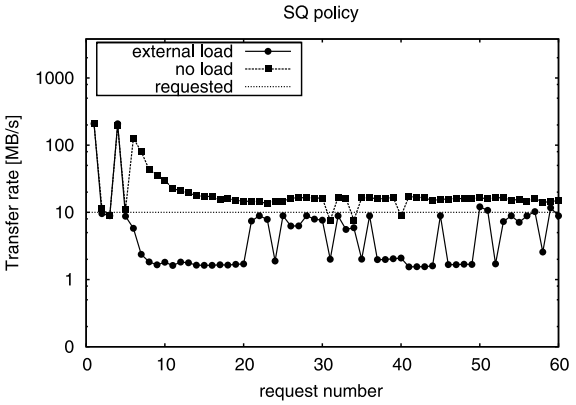

Fig. 9. TS3 – SLA/QoS fulfillment: test results for the SQ policy.

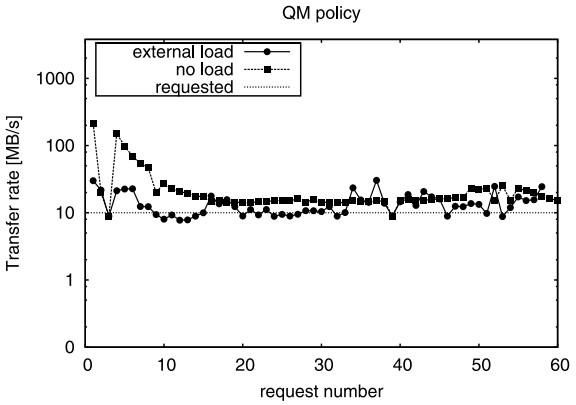

Fig. 10. TS3 – SLA/QoS fulfillment: test results for the QM policy.

in the field of execution programming of data-intensive application is important. For this purpose the research in the field of storage including the quality of storage services is essential.

The ability to specify storage QoS requirements for data-intensive applications can influence the way, in which their data are stored and retrieved, in the environments which support SQoS. Applying SQoS using the best effort model allows for storage performance-related load balancing and better chances for meeting QoS demands. Enriching this model with the appropriate policies of storage resource management using scheduling and resource reservation enables the delivery of a high level SQoS and fulfillment of SLA.

The management of storage resources is complicated given the complexity and heterogeneity of storage systems. The storage related management can appear at any level of IT system – the storage hardware, the storage system itself, the operating system, the filesystem, cache management, etc., which additionally raise the level of non-triviality.

A concept of SQoS provisioning and its implementation in form of the QStorMan toolkit has been presented in this paper. We have shown a method of using these tools for the goal of execution programming of data-intensive applications. The presented tests prove the usability/validity of the proposed approach.

The presented examples of using the QStorMan toolkit for execution programming of data intensive applications assume that a certain storage access pattern occurs when the application is doing I/O operations. This pattern is defined by such attributes like frequency of I/O operations, file size, number of files. We believe that the design of the QStorMan toolkit focused on its flexibility concerning the monitoring of heterogeneous storage systems, the creation of new QoS metrics, as well as the creation of new storage resource management methods, gives the possibility to adopt the toolkit for other I/O patterns. Tests for another type of application are still in progress. This application is an out-of-core application requiring access to many small files. The preliminary tests showed that a 15% speedup can be expected when switching to a newly developed policy.

The implementation of such a sophisticated toolkit is not a trivial task. Its main assumption is that performance monitoring of storage resources allows for better management of these resources. The implementation of such monitoring should take into account the heterogeneity of storage resources and should not introduce significant overhead. In the case of QStorMan and the SMED monitoring system the success lies on the using of C2SM model, which allows for unifying the access to the monitored resources and easy distributing of the monitoring system (see [12]). Using of unified SQoS metrics allows for easy construction of the upper system layers (e.g., storage performance manager, resource broker). The role of semantic technologies is very essential here. The ontologies have been used for resources description and monitoring metrics description. The developed ontologies and their usage for resource description, SQoS requirements specification, SLA monitoring and policies realization can be found in [9,11,13,15].

The QStorMan toolkit is an example of implementing SQoS provisioning for the goal of execution programming of data intensive applications, which are supposed to be the dominating type of applications in the era of the 4th paradigm of science [4].

Acknowledgements

This work is partially supported by PL-Grid project POIG.02.03.00-00-007/08-00 and AGH-UST grant 11.11.120.865. Thanks go to D. Nikolow and J. Kitowski for scientific cooperation; to the PL-Grid team, the QStorMan team and especially to D. Król, K. Skałkowski and M. Orzechowski for technical support. This research used resources of the PL-Grid project.

References

[1] P. Carns, K. Harms, W. Allcock, C. Bacon, S. Lang, R. Latham and R. Ross, Understanding and improving computational science storage access through continuous characterization, *Trans. Storage* **7** (2011), 8:1–8:26.

[2] H. Ching-Hsien, C. Tai-Lung and L. Kun-Ho, QoS based parallel file transfer for grid economics, in: *International Conference on Multimedia Information Networking and Security, MINES'09*, Vol. 1, IEEE Computer Society, Los Alamitos, CA, USA, 2009, pp. 653–657.

[3] Common Information Model (CIM) Standards, available at: http://www.dmtf.org/standards/cim/.

[4] T. Hey, S. Tansley and K. Tolle (eds), *The Fourth Paradigm – Data Intensive Scientific Discovery*, Microsoft Research, 2009.

[5] T. Kim, Y. Won, D. Kim, K. Koh and Y. H. Shin, Apollon: file system level support for QoS augmented I/O, in: *Advances in Multimedia Information Processing – PCM 2005*, Lecture Notes in Computer Science, Vol. 3768, Springer, Berlin, 2005, pp. 59–70.

[6] D. Król, R. Słota and W. Funika, Behaviour-inspired data management in the cloud, in: *Proceedings of Cloud Computing 2010 the First International Conference on Cloud Computing, GRIDs, and Virtualization*, Lisbon, Portugal, November 2010, IARIA, pp. 98–103.

[7] M. Kuta, D. Nikolow, R. Słota and J. Kitowski, Data access time estimation for the CASTOR HSM system, in: *Parallel Processing and Applied Mathematics: 6th International Conference, PPAM 2005*, R. Wyrzykowski, J. Dongarra, N. Meyer and J. Waśniewski, eds, Poznan, Poland, September 11–14, 2005, Revised Selected Papers, Lecture Notes in Computer Science, Vol. 3911, Springer, Berlin, 2006, pp. 148–155.

[8] S. Mendez, D. Rexachs and E. Luque, Methodology for performance evaluation of the input/output system on computer clusters, in: *2011 IEEE International Conference on Cluster Computing (CLUSTER)*, September 2011, IEEE Computer Society, Los Alamitos, CA, USA, pp. 474–483.

[9] D. Nikolow, R. Słota and J. Kitowski, Storage QoS aspects in distributed virtualized environment, in: *Proceedings of Cloud Computing 2010 the First International Conference on Cloud Computing, GRIDs, and Virtualization*, Lisbon, Portugal, November 2010, IARIA, 2010, pp. 110–115.

[10] S. Polak, D. Nikolow, R. Słota and J. Kitowski, Modeling storage system performance for data management in cloud environment using ontology, in: *Proceedings of 2011 International Workshop on Semantic Interoperability. In conjunction with ICAART 2011*, S. Pileggi, ed., Rome, Italy, January 2011, SciTePress, 2011, pp. 54–63.

[11] S. Polak and R. Słota, *Organization of Quality-Oriented Data Access in Modern Distributed Environments Based on Semantic Interoperability of Services and Systems*, River, 2012, to appear.

[12] K. Skałkowski, J. Sendor, R. Słota and J. Kitowski, Application of the ESB architecture for distributed monitoring of the SLA requirements, in: *Ninth International Symposium on Parallel and Distributed Computing, ISPDC 2010*, Instanbul, July 2010, IEEE Computer Society, Los Alamitos, CA, USA, pp. 203–210.

[13] R. Słota, D. Nikolow, P. Młocek and J. Kitowski, Semantic-based SLA monitoring of storage resources, in: *Proceedings of Parallel Processing and Applied Mathematics – PPAM 2011, 9th International Conference*, Torun, Poland, September 2011, Lecture Notes in Computer Science, Vol. 7204, Springer, Berlin, 2012.

[14] R. Słota, D. Nikolow, S. Polak, M. Kuta, M. Kapanowski, K. Skałkowski, M. Pogoda and J. Kitowski, Prediction and load balancing system for distributed storage, *Scalable Computing Practice and Experience* **2**(11) (2010), 121–130, Special Issue: Grid and Cloud Computing and their Application.

[15] R. Słota, D. Nikolow, K. Skałkowski and J. Kitowski, Management of data access with quality of service in PL-Grid environment, *Computing and Informatics* **1** (2012).

[16] The Lustre filesystem web site, available at: http://www.lustre.org/.

[17] The OntoStor project web site, available at: http://www.icsr.agh.edu.pl/ontostor/.

[18] The PL-Grid project web site, available at: http://www.plgrid.pl/en.

[19] The QoSCosGrid middleware web site, available at: http://www.qoscosgrid.org/.

[20] S. Venugopal, R. Buyya and K. Ramamohanarao, A taxonomy of Data Grids for distributed data sharing, management, and processing, *ACM Comput. Surv.* **38**(1) (2006).

[21] Y. Won, H. Chang, J. Ryu, Y. Kim and J. Shim, Intelligent storage: Cross-layer optimization for soft real-time workload, *Trans. Storage* **2** (2006), 255–282.

[22] J.C. Wu and S.A. Brandt, Providing quality of service support in object-based file system, in: *Proceedings of the 24th IEEE Conference on Mass Storage Systems and Technologies*, Washington, DC, USA, IEEE Computer Society, Los Alamitos, CA, USA, 2007, pp. 157–170.

[23] X. Zhang, Y. Xu and S. Jiang, YouChoose: Choosing your storage device as a performance interface to consolidated I/O service, *Trans. Storage* **7** (2011), 9:1–9:18.