Behaviour-inspired Data Management in the Cloud

Dariusz Król, Renata Słota, Włodzimierz Funika

AGH University of Science and Technology, Faculty of Electrical Engineering,
Automatics, Computer Science and Electronics, Department of Computer Science,
al. Mickiewicza 30, 30-059 Krakow, Poland
{dkrol, rena, funika}@agh.edu.pl

Abstract — **Open source cloud computing solutions are still not mature enough to handle data-intensive applications, e.g., scientific simulations. Thus, it is crucial to propose appropriate algorithms and procedures to the data management problem in order to adjust Cloud-based infrastructures to scientific community requirements. This paper presents an approach inspired by the observation of the Cloud user behaviour: the intensity of data access operations, their nature, etc. We also describe how the proposed approach influences the architecture of a typical Cloud solution and how it can be implemented based on the Eucalyptus system, which is a successful open source Cloud solution.**

Keywords-cloud computing; data management; monitoring; behaviour patterns.

I. INTRODUCTION

Cloud computing is arguably the most popular buzzword in the tech world today. It promises to reduce the total cost of maintenance of an IT infrastructure with providing better scalability and reliability at the same time. Apart from "unlimited" computational power, the Cloud provides also "unlimited" storage capacity which can be accessed from every device which is connected to the Internet. Therefore, this is not a surprise that many commercial companies along with academic facilities are very interested in this paradigm. As with other paradigms, various research centers test it in a variety of ways in order to unveil its strengths and weaknesses. What makes the Cloud computing special in comparison to other, more academic-related approaches to distributed computing, e.g., Grid computing [2], is the support and investment made by the largest IT companies [3], e.g., Google, IBM, Microsoft and many others. These million dollars investments can be treated as a good omen that Cloud computing can be widely adopted and will not disappear after few years.

From a few years now, Clouds are evolving into a number of different forms but with a common goal, i.e., providing computational power and storage capacity on premise. Existing Clouds can be classified in a few ways each of which relates to one feature from the following list:

- accessibility of the Cloud for users,
- abstraction level on which Cloud users operate,
- resources provided by the Cloud,
- openness of the source code of the Cloud.

The first taxonomy includes public, private and hybrid Clouds. Today, the most popular are public clouds which can be used by anyone. Potential customer needs only to obtain an account on the providers site. This category includes Amazon Elastic Compute Cloud (Amazon EC2) [5], Microsoft Azure [6], Google AppEngine [7] and many others. On the other hand, there are private clouds. In most cases, they are limited to resources and members of a single organization. Also, their accessibility is limited to the organization's intranet. The third group, called hybrid Clouds, concerns a special type of private Clouds whose computation power and storage capacity can be extended by resources of public Clouds. Hybrid Clouds exploit the scalability feature of the Cloud to provide required resources by utilizing publicly accessible Clouds when the in-house infrastructure is not enough.

The second taxonomy concerns the style in which the customers use Clouds. This taxonomy includes:

- Infrastructure as a Service (IaaS) Clouds which provide access to a virtualized pool of resources using which customers assemble Virtual Machines (VMs) on which customers install any technology stack and any applications they need. Probably the most well known example of this group is Amazon EC2,
- Platform as a Service (PaaS) Clouds expose a well defined runtime environment, e.g., Java Virtual Machine or Microsoft .NET and programming services which are used to develop applications without troubling with virtual machines, e.g., queuing systems, (non-)relational databases and more. This group includes Google AppEngine among many other,
- Software as a Service (SaaS) Clouds are about delivering applications which are deployed at the providers infrastructure, e.g., Google Apps [8]. Applications which are exposed with SaaS model are accessible mostly often via a web browser and are provided in the pay-per-use model. This group focuses on customers rather than developers.

The third taxonomy of Clouds focuses on the type of resources provided. Today, this taxonomy includes two elements: compute Clouds and storage Clouds. The first group comprises Clouds which provide access to computational power by running VMs or applications on a

specified virtualized hardware, e.g., in Amazon EC2 the user can choose the size of a virtual machine to use in terms of the available number of virtual CPUs and RAM memory [9]. To mention just a few examples, the customer can choose a small instance of a virtual machine which can be defined as a single, normalized, virtual CPU, 512 MB of RAM and 10 GB of hard drive capacity while a big instance can consist of 8 virtual CPU, 4 GB of RAM and 50 GB of hard drive capacity. On the other hand, the storage Clouds enable users to store data sets in a number of ways, i.e., in files, (non-)relational databases or block devices. In theory, the storage clouds can provide an infinity storage capacity on demand.

The last taxonomy divides Clouds into two groups: open-source Clouds and proprietary Clouds. This classification is important especially from a developer point of view. Open-source Clouds can be modified and analyzed by anyone while proprietary Clouds are commercial solutions mostly often with a closed source code. Hence, their internals are hidden from customers or developers from outside the Cloud provider's company. Thus, in many situations, it is difficult or even impossible to compare these two types of Clouds.

Todays Cloud computing solutions, especially the open source ones, are not mature enough in terms of storage capabilities to handle data-intensive applications which need to store results in Cloud-based storage. One of the issues is the lack of adaptability of data management strategy, e.g., to dynamically changing user requirements or location from where the user access the Cloud storage. Each of these aspects influences the access time to data especially when considering geographically distributed resources which constitute a single Cloud installation, e.g., a user who lived in Europe can download his data from a data center in US instead of a closer data center located in Europe. Therefore, we propose a novel approach, based on autonomic systems (similar to situation-aware systems - [4]) and behaviour observation whose main goal is to adapt data location to the user needs which will result in decreasing data access time and higher utilization factor of resources. We introduce a "Usage profile" concept which describes a piece of data stored in the Cloud storage. The usage profile contains information how the described data is used by Cloud clients. To create such a profile, storage-related operations performed by Cloud users are monitored and analyzed. The approach is designed to be an additional element of the Cloud installation rather than being mandatory. Thus, it can be treated as a plugin for a Cloud solution. Moreover, it is transparent from the Cloud user point of view because it operates on the Cloud provider's side where various management actions can improve the storage performance.

The commercial clouds, e.g., Amazon EC2 which is an IaaS solution, i.e., it allows to manage a computational environment consisting of virtual machines, cannot be easily studied due to proprietary source code, thus in this paper they will not be taken into consideration.

This article is an extended version of the work presented in [1]. The rest of the paper is organized as follows.In Section II, a number of the existing Cloud solutions and Cloud-based storage services are presented. In Section III, we describe a data management algorithm which is based on behaviour analysis. Parameters of the usage profile along with behaviour which is analyzed to create usage profiles are described in Section IV. A prototype implementation of the algorithm is presented in Section V. Directions for future work are discussed in Section VII. The paper is concluded in Section VIII.

II. RELATED WORK

Cloud computing has been already widely adopted by various commercial companies and academic facilities. While many commercial companies develop their own solutions, e.g., Amazon EC2, Microsoft Azure or Google AppEngine, others use and invest in open source solutions which are especially well suited for situations where the environment has to be adapted to some specific requirements. This feature is very important for scientific community which would like to implement new concepts and approaches, e.g., to optimize data access time or other parameters. In this section, we focus on three well known IaaS environments: Eucalyptus, Nimbus, OpenNebula and OpenStack. We also consider a few commercial products for data management. In addition, two data management systems which are based on the Grid computing paradigm are presented. Hence, we will be able to compare Cloud-based solutions with Grid-based solutions which is valuable for readers with a Grid computing background but who are not familiar with Clouds yet.

A. Eucalyptus

Eucalyptus system [10] is an example of an open source project which became very popular outside the scientific community and is exploited by many commercial companies to create their own private clouds. It was started as a research project in the Computer Science Department at the University of California, Santa Barbara in 2007 and today it is often treated as a model solution of an IaaS Cloud. Eucalyptus aims at providing an open source counterpart of the Amazon EC2 Cloud in terms of interface and available functionality. There are two versions of the Eucalyptus Cloud: Community and Enterprise.

Each Eucalyptus installation consists of a few loosely coupled components each of which can run on a separate physical machine to increase scalability. The frontend of such a Cloud is "Cloud controller" element which is an access point to the virtual machines related features. While "Cloud controller" is responsible for computation, the "Walrus" component is responsible for data storage. It allows to store virtual machine images along with any other files which are organized into a hierarchy of *buckets* and can be treated as a counterpart of Amazon Simple Storage Service (S3) [11] in the Eucalyptus system. Amazon S3 is a Cloud storage service which allows to store any type of data in form of files in a number of buckets (each with a unique name within a bucket) using a simple Application Programming Interface (API), i.e., *put, get, list, del* operations are supported. Each virtual machine is run on a physical host which is controlled by the "Node controller" element. A group of nodes can be gathered into a cluster which exposes a single access point, namely "Cluster

controller" from the virtual machine management side and "Storage controller" from the virtual machine images repository side.

Eucalyptus is based on the Java technology stack and its source code is freely accessible and can be modified as necessary. To mention a few, the current implementation uses web services (Apache Axis [12]) to expose the provided functionality to the external clients, and exposes a web-based user interface developed with Google Web Toolkit (GWT) [13]. It also supports the Xen [14] and Kernel-based Virtual Machine (KVM) [15] hypervisors to run virtual machines on the supervised resources.

The open version of the Eucalyptus system stores data into a single directory on the host on which the Walrus component is installed. Therefore, the only way to distribute the data is to exploit a distributed file system, e.g., Lustre [16] or Oracle Cluster File System 2 [17], which will be mounted to the directory used by the Eucalyptus installation. However, this file system is orthogonal to the Cloud solution, i.e., it does not have access to any information about the Cloud. Thus, it can manage data based on some basic information only, e.g., size of stored files or capacity of available storage resources. Such strategies are very limited and are not customizable for the Cloud computing paradigm.

On the other hand, we have the Enterprise version of Eucalyptus. Among other features, the Enterprise version of Eucalyptus provides an adapter for direct integration with Storage Area Networks (SANs) [18], e.g., Dell Equallogic or NetApp. With this adapter, you can easily configure Eucalyptus to exploit SAN [19] directly as a data storage back end. However, to our best knowledge, this integration does not allow to combine different types of storage systems within a single Cloud installation. Also, a Cloud administrator can`t provide policy for data distribution among available storage resources. The data management is left entirely to the SAN solution which knows nothing about the Cloud, its users or the type of data stored in the Cloud. Though, SANs are enterprise-class solutions for data storage, they do not provide any Cloud-specific storage strategies which would regard, e.g., information about Cloud customers.

B. Nimbus

Nimbus [20] is a toolkit for turning a cluster into an IaaS Cloud computing solution. It is developed by the Globus Alliance [21]. A Nimbus client can lease remote resources by deploying virtual machines on these resources and configure them to fulfil the user requirements. What makes it attractive is support for a communication interface known from the Grid computing, namely Web Services Resource Framework (WSRF) [22]. As in other popular solutions, Nimbus provides an Amazon EC2 compatible interface for Cloud clients, which is de facto a standard of IaaS environment due to its wide adoption in a number of solutions.

A Nimbus installation consists of a number of loosely coupled elements. The center point of the Nimbus architecture is the "Workspace service" component which is a coordinator of the whole installation. It is invoked through different remote protocol frontends, e.g., WSRF or EC2 – compatible services. Another important component is "Workspace resource manager" which runs on each host within the Cloud and is responsible for controlling a hypervisor on the host machine. The current version fully supports the Xen hypervisor and most of the operations on the KVM hypervisor. It is also worth of mentioning that Nimbus installation can be easily connected to a public commercial Cloud, e.g., Amazon EC2 in order to achieve even greater computer power when the in-house infrastructure is not enough.

In terms of data management, the Nimbus project is limited to the virtual machine image repository. There is no component which would provide a functionality similar to that of the Amazon S3. The user can only upload virtual machine images to the Nimbus cloud and store the data stemming from computation on storage devices connected directly to a virtual machine.

The Nimbus project is based on open source tools and frameworks, e.g., Apache Axis, the Spring framework [28] or JavaDB [29]. Therefore, everyone can download its sources from a public repository and modify its functionality as desired.

C. OpenNebula

OpenNebula [30] is a Virtual Infrastructure Manager for building cloud infrastructures based on Xen, KVM and VMWare virtualization platforms [31]. It was designed and developed as part of the EU project RESERVOIR [32], whose main goal is to provide open source technologies to enable deployment and management of complete IT services across different administrative domains. OpenNebula aims to overcome the shortcomings of existing virtual infrastructure solutions, e.g., inability to scale to external clouds, a limited choice of interfaces with the existing storage and network management solutions, few preconfigured placement policies or lack of support for scheduling, deploying and configuring groups of virtual machines (apart from the VMWare vApp solution [34]). Like other of the presented solutions, OpenNebula is fully open source and its source code can freely be downloaded from a public repository.

OpenNebula architecture was designed with modularity in mind. Therefore, it can be extended to seamlessly support a new virtualization platform e.g., in terms of virtual image or service managers. For instance, a procedure of setting up a VM disk image consists of well-defined hooks whose implementation can be easily replaced to interface with the third-party software. To manage an OpenNebula installation, the user can use a simple, dedicated command line interface or Amazon EC2 query interface. Therefore, it can be accessed with the tools originally developed to work with the Amazon EC2 cloud.

In terms of storage mechanisms, it is limited to repository of VM images only. The repository can be shared between available nodes with the Network File System (NFS) [35]. It is also possible to take advantage of block devices, e.g.,

Logical Volume Management 2 (LVM2) [36] to create snapshots of images in order to decrease time needed to run a new image instance.

D. OpenStack

OpenStack is a joint effort of NASA and RackSpace. NASA contributed to the project by releasing its middleware, called Nebula [23], for managing virtual machines at physical infrastructure. RackSpace contributed with its storage solution known as Cloud Files [24]. OpenStack [25] is a collection of tools for managing data centers resources to build a virtual infrastructure. In terms of computations, OpenStack provides OpenStack Compute (Nova) solution which is responsible for managing instances of virtual machines. In terms of storage, OpenStack provides OpenStack Object Storage (Swift) which is an object storage solution with built-in redundancy and failover mechanisms. There is also a separate subsystem, called OpenStack Imaging Service, which can be used to lookup and retrieving virtual machine images. Since the first release of OpenStack was in October 2010, there is no evidence about production deployments of the toolkit in either industry or scientific area yet. Thus, there is no information about the performance and stability of OpenStack. Also, OpenStack lacks of an interface that would be compatible with the Amazon clouds which is a de facto standard in the Cloud ecosystem.

E. Flash Cloud

"Flash Cloud" [27] is a commercial service for storing data using the Cloud paradigm. After creating an account, the user gets access to a certain storage capacity which can be scaled up depending on your requirements. Your data can be managed using the following methods:

- Web 2.0 interface,
- native desktop software for PC or Mac and mobile devices including IPhone and BlackBerry,
- Web Distributed Authoring and Versioning (WebDAV)-based [38] API.

The data are stored using industry leading solutions such as Internet Small Computer Interface (iSCSI) [37], File Transfer Protocol (FTP)/Network Attached Storage (NAS) and EVault [39]. However, "Flash Cloud" does not provide any mechanism for integrating with popular computing Clouds, e.g. Eucalyptus, Nimbus, OpenStack. Moreover, the storage capacity limits (250GB for a normal customer and 2000 GB for an enterprise customer) are rather low comparing to the requirements for a Cloud which can be measured in TeraBytes or even PetaBytes. The last drawback is the programming interface which is proprietary and does not follow any popular solutions, though it is based on open WebDAV protocol.

F. EMC Atmos

Another commercial product is EMC2 Atmos which is a complete Cloud Storage-as-a-Service solution [17]. It provides massive scalability by allowing to manage and attach new storage resources from a single control center. Atmos features policy-based information management which allows to define bussiness level policies how the stored information should be distributed among available resources. It also reduces effort required for administration by implementing auto-configuring, auto-managing and auto-healing capabilities. In the newest version 2.0, it also provides a Representational State Transfer (REST)-based API which is compatible with Amazon S3. Although, Atmos provides many interesting features and capabilities, it does not provide integration with existing Clouds, to our best knowledge. It is rather a separate solution oriented to the storage only, i.e., it does not support any functionality related to running virtual machines. Thus, to provide your users with a fully functional Cloud you will need to use a computing Cloud solution besides EMC Atmos. However, the data management within EMC Atmos does not take into account specific information about the computing Cloud part and its users, e.g., access frequency to users data.

G. XtreemOS

XtreemFS [33] is a cluster-oriented, distributed file system developed within the XtreemOS European project. The main goal of the project is to develop an easy to use and 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 such features as: scalability, parallel Input/Output (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 time necessary to write a single file in many locations. Also, there is no Web Service interface available to access the XtreemFS remotely.

H. dCache

DCache [26] is a data management system which implements all the requirements for a Storage Element in the Grid. It was developed at CERN to fulfil the requirements of the Large Hadron Collider 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 the 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 can take as an input 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. DCache is a Grid-oriented tool by design, thus it is not compatible with existing Cloud solutions. DCache provides a programming interface similar to a filesystem interface which is at a lower level of abstraction comparing to the storage cloud interface. However, dCache could be treated as a storage system which is used by a storage cloud rather than being a complete storage cloud solution.

III. BEHAVIOUR-INSPIRED APPROACH TO DATA MANAGEMENT

The most important aspect of the presented approach is its orientation towards the requirements of each user rather than some global optimization such as equal data distribution among available resources. Our approach treats each user individually by monitoring his/her behaviour related to data storage. The monitoring is needed to discover the nature of data automatically, e.g., whether it is read-only or often modified data. With this knowledge, the data can be managed appropriately, i.e., with requirements such as high availability taken into account. Another important feature of the approach which can be deduced from the previous one is its transparency from the user point of view. Thus, it can be applied to any existing solution without any modification required to the user-side code.

The structure of the described algorithm is depicted in Figure 1. There are 3 phases included:

- *The observation phase* where the information about the user behaviours are aggregated. It is a start point of the management iteration. Each operation related to the storage, e.g., uploading or downloading files is recorded along with information about the user who performed the operation and a time-stamp.
- *The profile construction phase* is the one where the gathered behaviour-related data about is analyzed. For each user, a profile which describes how the user accesses each piece of data is created, thus the profile also contains information how each piece of data should be treated.
- *The data management phase* is responsible for modifying the data storage, i.e., applying a dedicated strategy which corresponds to a user profile. Such a strategy can e.g., create many replicas of a piece of data which is read by many users but hardly anyone modifies it or it can move the data closer to the user to decrease its access time.

An important feature of the algorithm is the fact that it never ends. There is no stop condition because such a management process may last as long as the Cloud is running. Each iteration of the loop results in tuning the storage strategy to the observed user behaviour. However, the historical data is taken into account as well and can influence the storage strategy rather than just be omitted. In fact, its importance to the new strategy is one of the parameters of the algorithm.

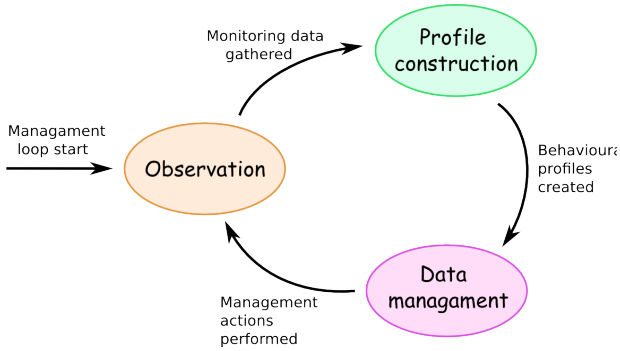

Figure 1. Profile-based data management loop.

Another important aspect of the approach is its influence on the architecture of a cloud solution. The overview of such an architecture is schematically depicted in Figure 2. To underline its most important components, some simplifications were introduced, e.g., the Cloud solution is represented only by "Cloud manager" which is an access point to the cloud infrastructure. "Storage elements" represent physical resources where the data is actually stored. The new components are as follows:

- *Monitoring system* is responsible for gathering information about user actions. The most important operations are those related to data storage, e.g., uploading a file or accessing a file by the user. Information about these actions have to be remotely accessible by an external Cloud client in a programming language independent way.
- *Behaviour data manager* is the main element of this new approach. It performs the analysis of the user behaviours and creates their profiles. Then, it performs all the necessary actions to adjust the storage strategy to the actual profile. In most cases, these actions will be related either to moving data between storage elements with different physical parameters or to managing data replication, e.g., creating new replicas. By combining these two types of operations, we can improve the Quality of Service (QoS) of the cloud storage, e.g., decrease the data access time. It is also possible to apply more sophisticated algorithms for data management as the ones described in [41] and [42]. The communication between “Behaviour data manager” and “Storage elements” is optional. If “Cloud manager” exposes an interface to manage the actual data location, there is no need in “Behaviour data manager” to interact with “Storage elements” directly.

- *Profile knowledge base* is a repository where the historical profiles for each piece of data are stored along with a record of each performed action. Thus, it can be used by the "Behaviour data manager" to take into account not only the most recent information but also the previous actions.

As we can see, the approach can be easily integrated with any Cloud solution which can be monitored, i.e., each performed operation related to the storage is registered, and the stored data can be moved between available physical resources, either indirectly with an exposed programming interface or directly with accessing storage elements and moving raw data. These requirements are rather easy to meet and in the next section we are presenting an example implementation based on a popular open source Cloud solution. It is also worth mentioning that in most cases the original source code of such a Cloud solution may stay untouched.

IV. USAGE PROFILE

The presented approach highly exploits usage profiles which are based on the observation of the actions performs by Cloud users. In our approach, a usage profile describes how a concrete piece of data in the Cloud is used by customers within a specified period of time in a quantitative way. The quantitative nature of usage profiles is a must in order to enable comparison of usage profiles. We would like to represent each usage profile as an element of a N-dimensional space where N denotes the number of usage profile parameters. By doing so, we can determine the relationship between each two profiles, e.g., whether they are similar, i.e., close to each other or not.

Such a profile aims at describing a behaviour pattern for a piece of data to which the profile is assigned. Thus, it can be treated as a kind of metadata for the actual data stored in a Cloud storage. The necessary data for creating usage profiles is obtained with a dedicated monitoring system. On the other hand, we have a set of behaviour classes which define typical usage patterns in the domain of data storage, e.g., read-only data. In our representation in an N-dimensional space, each behaviour class will be represented by distinct subspaces of the entire space. Thus, we will be able to easily determine to which subspace each usage profile belongs.

The behaviour classes should be defined either by experts of the data storage domain or experienced administrators of storage solutions. To each behaviour class, a set of proper actions is assigned concerning the usage pattern which is described by the class. After creating a usage profile for a given data object, the nearest behaviour class is chosen. Then, the assigned set of data management actions is performed in order to increase desired quality parameters of the Cloud, e.g., throughput, data availability or data access time. Which quality parameters will be increased depends on the behaviour classes and data management actions assigned to the behaviour classes. This algorithm is performed periodically in a loop with a configurable interval between iterations.

For a system prototype, we defined the following parameters which will be included in usage profiles:

- frequency of access to an object, e.g., per hour or per iteration of the algorithm loop,
- number of read/write operations,
- number of different users accessing to the object (separately for read/write operations),
- number of different places (e.g., IP addresses) from which the object was accessed (separately for read/write operations).

Based on these parameters a set of predefined behaviour

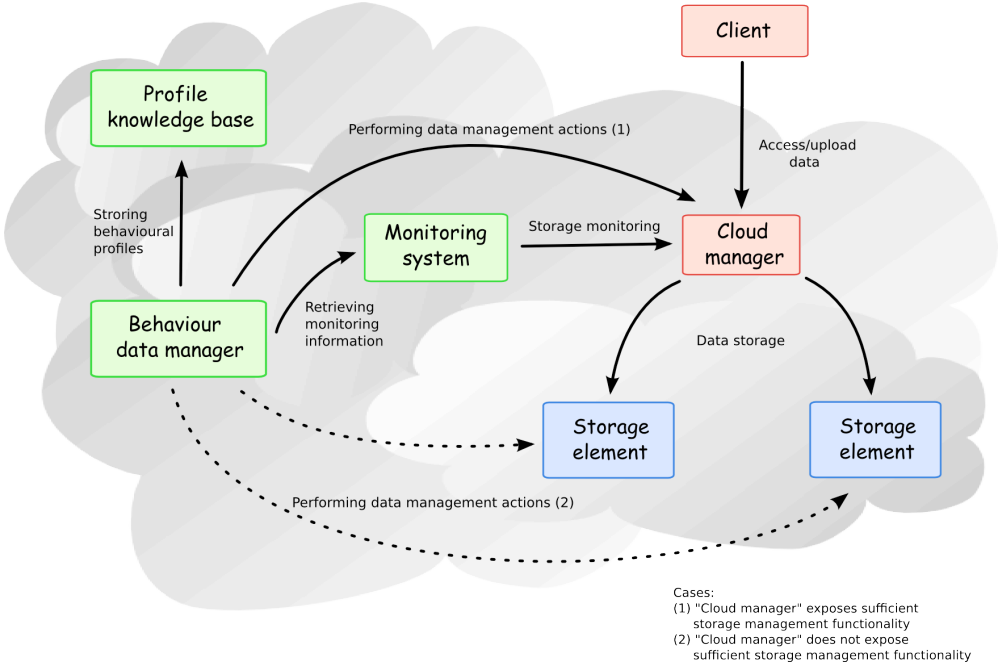

Figure 2. Architecture overview of a cloud solution with “Behaviour data manager” involved.

classes can be created. They should correspond to the well known storage management patterns, e.g., when an object is always read, it should be replicated to multiple physical locations to decrease the access time.

Such a profile is created for each piece of data in the cloud storage (e.g., file) after the first iteration of the algorithm and updated on each subsequent iteration. In each "Data management" phase, for each profile a similarity function to each defined profile type is calculated. Then, the actions related to the most similar profile type are performed.

Therefore, the approach can be easily extended in terms of recognized behaviour, simply by defining new behaviour classes along with related actions.

V. IMPLEMENTATION

To present a sample implementation of the approach, we chose the Eucalyptus system as a basis. This choice was motivated by the large popularity of Eucalyptus and its functionality in terms of storage which is very similar to the Amazon S3 offer, a de facto standard in the Cloud industry.

The Eucalyptus architecture contains a component called "Walrus" which is responsible for the storage-related functionality. "Walrus" exposes an API which comprises methods for creating, updating, and removing objects and buckets from the Cloud storage.

The open version of Eucalyptus implements cloud storage as a designated directory on the local file system. It is rather a minimalistic solution of the Cloud storage, due to very limited ways of data distribution. A feasible way is to mount a distributed file system at the designated directory which will transparently distribute data among a number of storage elements. Unfortunately such a solution does not allow to control data manipulation which cannot be accepted in our situation. Therefore, we extended the storage system in Eucalyptus by an ability to store data in several directories instead of one only, each of which can point to a different physical location, e.g., via NFS. With this extension, the location of the data can be easily controlled, simply by moving files between directories.

As mentioned above, there are a few components to add to the Eucalyptus architecture in order to implement the approach under discussion. Such an extended architecture is depicted in Figure 3. There is the "Walrus" component which exposes an API to external users for storing data in the Cloud. Apart from storing the custom data, e.g., results coming from a running simulation, "Walrus" stores two other types of objects. The first one is a VM image which is uploaded by the user and then is run on the Eucalyptus

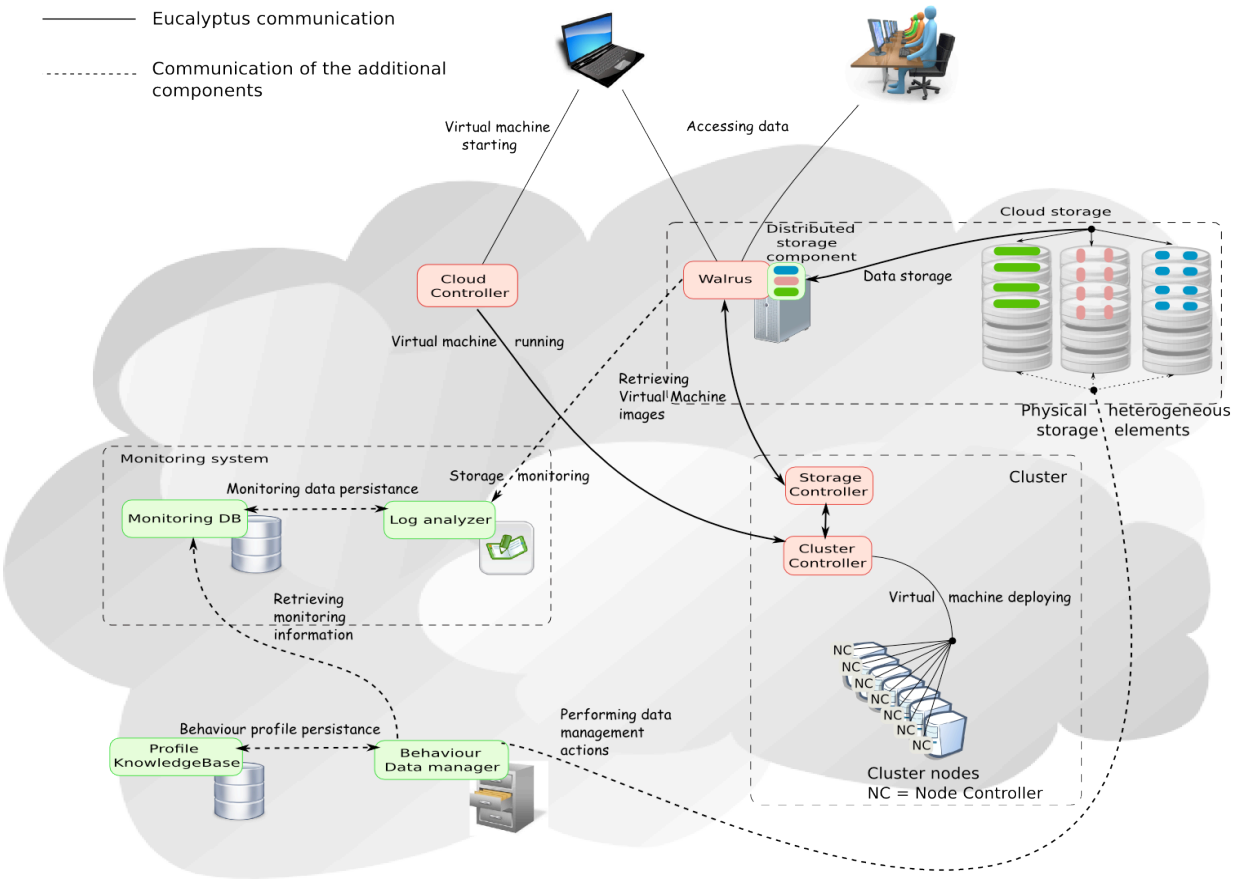

Figure 3. Eucalyptus system architecture extended by "Behaviour Data Manager", "Profile KnowledgeBase", "Monitoring system", and "Distributed storage component".

infrastructure. Although, the user communicates with another component, called "Cloud controller", the images are actually stored with "Walrus". The second type of objects is "Block storage". It is used as a mountable partition to store data during a VM run, similarly to a local file system. Moreover, a "Block storage" object can store data between two subsequent runs of a VM and as opposed to a VM virtual disk, the data is not erased after a VM shutdown. Such a partition is stored within the Cloud storage with "Walrus". All these three types of objects are stored with "Walrus" on the same rules and thus can be uniformly managed with our system.

In the following subsections, we describe the implementation of the previously mentioned components, i.e. a monitoring system, the behaviour data manager and the profile knowledge base. However, in order to implement these components we had to extend the Eucalyptus implementation to support the distributed storage and to provide some additional information about storage-related operations when they occur.

When designing the extension to Eucalyptus, we focused on making it as non-intrusive as possible. Thus, we decided to replace an existing implementation of a Java class responsible for storing the data to the storage resources. By doing so, we can activate this functionality with few modifications to the Eucalyptus source code.

A. *Storage-related operation monitoring*

The first of the additional elements added to the architecture is a dedicated “Monitoring system”. It consists of “Log analyzer” which periodically reads the Walrus log where each storage-related operation is recorded and a relational database where information who and which operation performed are stored. Thus all the necessary data to create usage profiles is prepared in a technology neutral form.

The Walrus component logs an occurrence of each storage-related operation to a common log file, i.e., create and delete buckets, put, get and delete objects from buckets. A log entry which corresponds to a storage-related operation contains information about the type of operation, the user who performs the operation and information about the subject of the operation. A sample entry describing a *putObject* operation is depicted below:

```
07:53:32  INFO 342  WalrusRESTBinding      | <?xml
version="1.0" encoding="UTF-8"?>
| <euca:PutObjectType
xmlns:euca="http://msgs.eucalyptus.com">
|   <euca:WalrusDataRequestType>
|     <euca:WalrusRequestType>
|       <euca:EucalyptusMessage>
|         <euca:correlationId>58b85aeb-29f4-4125-
8f60-da95baa4422e</euca:correlationId>
|         <euca:_return>true</euca:_return>
|       </euca:EucalyptusMessage>
|       <euca:accessKeyID>WKy3rMzOWPouVOxK1p3Ar1C2
uRBwa2FBXnCw</euca:accessKeyID>
|<euca:timeStamp>2011-06-
11T05:53:32.006Z</euca:timeStamp>
|      <euca:bucket>testing_bucket_1</euca:bucket>
|       <euca:key>file_1024_5</euca:key>
|     </euca:WalrusRequestType>
|     <euca:randomKey>testing_bucket_1.file_1024_5
.Dpq-OXrOgNtjnQ..</euca:randomKey>
|   </euca:WalrusDataRequestType>
|   <euca:contentLength>1024000000</euca:contentLe
ngth>
|   <euca:metaData/>
|   <euca:accessControlList>
|     <euca:grants/>
|   </euca:accessControlList>
|   <euca:contentType>binary/octet-
stream</euca:contentType>
| </euca:PutObjectType>
```

Every entry has a structure similar to an XML document where data is put within tags which describe the semantic of the data, e.g., `<euca:timeStamp>2011-06-11T05:53:32.006Z</euca:timeStamp>`.

The monitoring system is implemented with the Python programming language. It uses mainly the standard library of Python to parse and retrieve relevant information from Cloud log files. The monitoring system analyzes the size of a log file in a loop to find out whether or not the file contains new information. If the size of the file grows between two iterations, the monitoring system analyzes only this additional data. Using a regular expression, the storage-related operations are extracted.

Whenever the monitoring system parses a log entry which describes a storage-related operation, it inserts information about this fact to a shared relational database which acts as the Profile Knowledge base. To connect with the database, the MySQLdb Python connector is used. The schema of the knowledge base is depicted in Figure 4. There are several tables which are filled with monitoring information. Most of them are self-explanatory. The “Users”, “Buckets” and “Objects” tables contain information on the elements, i.e., buckets and objects, stored in Eucalyptus Cloud and on the users of the cloud. The information about the performed storage-related operations are stored in the “BucketOperationHistory” and “ObjectOperationHistory” tables. The “Operations” table contains information about possible types of operations.

B. *Behaviour-inspired Data Manager*

While the monitoring system gathers information about the state of a Cloud, another component called “Behaviour-inspired Data Manager” analyzes this information and manages the data stored in the Cloud.

The Data Manager performs the following actions periodically on the information gathered by the monitoring system:

- create usage profiles for each stored object,
- classify the usage profiles to one of the defined behaviour classes,
- manage stored objects based on actions which are assigned to behaviour classes.

The procedure of creating usage profiles is in counting different types of operations performed on each stored object by different users. Currently, a usage profile consists of information about the performed *puts* and *gets* operations and the number of different users who have accessed the object over a period of time. Each created usage profile is stored in the "UsageProfile" table in Profile Knowledge Base with information about the time period to which the profile refers.

The second step of the management process is the classification. Each usage profile created in the previous step is assigned to one of the defined behaviour class. In the presented prototype, behaviour classes are defined manually by a Cloud administrator. Each behaviour class refers to a management pattern which will be exploited in the next phase. For testing purposes, we defined three sample classes for describing three main behaviour patterns:

- "Read-only" class which describes objects which are mostly read,
- "Write-only" class which describes objects which are often changed,
- "Nothing-do" class which describes objects which are hardly used.

Each class is defined in the Profile Knowledge base as a tuple *<typical_number_of_gets, typical_number_of_puts, typical_number_of_users>*. Based on the created usage profiles and behaviour classs, the Data Manager calculates Euclidian distances between profiles and classes and assigns the nearest class to each stored object.

The last phase is the phase where the actual management actions take place. After the classification phase, each object stored in the Cloud has a behaviour class assigned. Each behaviour class refers to a management pattern which is a set of actions which should be performed in a situation described by the behaviour class. To present the idea, we provided sample actions for each of the previously defined behaviour classes:

- Create a new replica of a "read-only" object,
- Remove one of the existing replicas of a "write-only" object. Also if the user number is equal to one then move the "write-only" object closer to the client.
- Do nothing for a "nothing-do" object.

The replication is a common mechanism for increasing

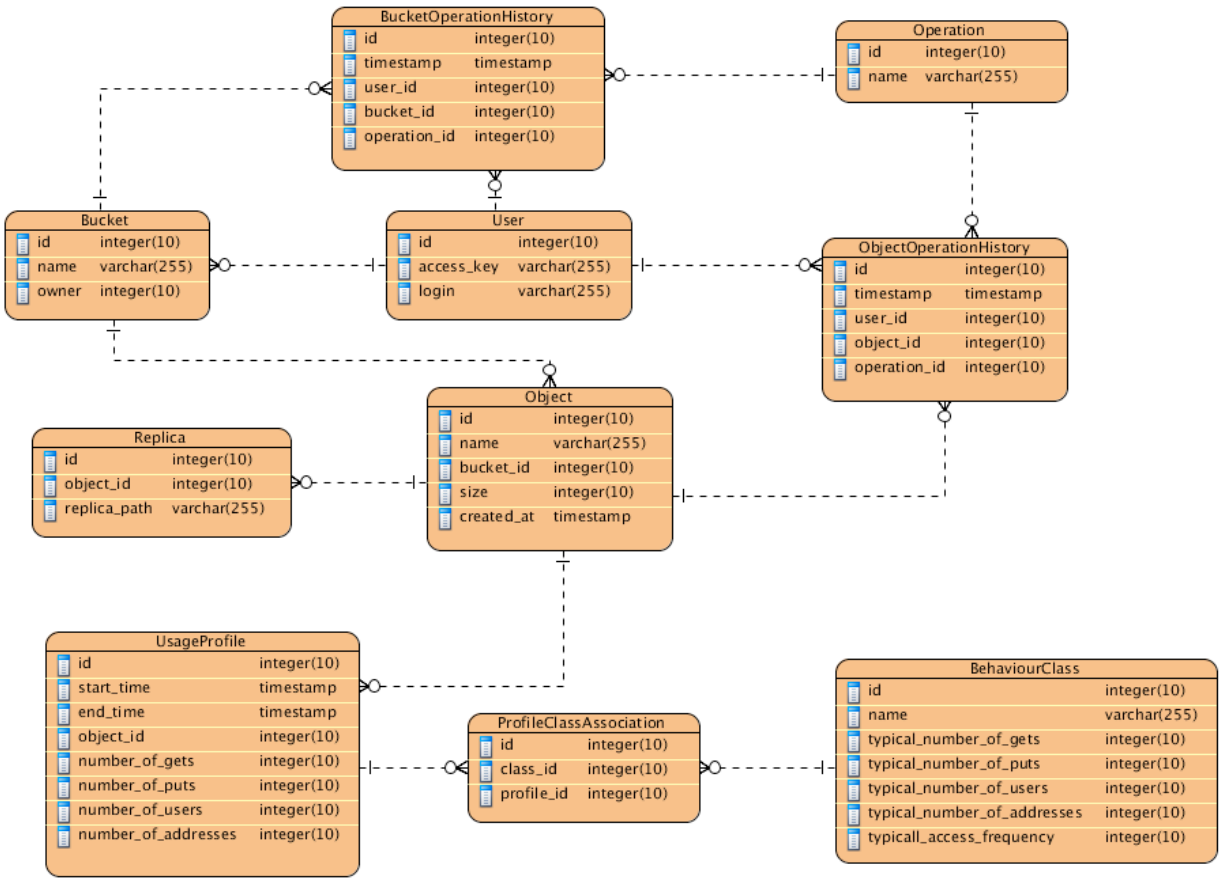

Figure 4. Data model schema of a relational database which constitutes the Profile Knowledge base.

read performance for files which are often read. By changing the replication level of a stored object dynamically, we can respond to changes in the object usage.

To implement the Behaviour-inspired Data Manager we used the Ruby programming language along with a few common Ruby libraries (called Ruby Gems) such as:

- ActiveRecord for Object-Relation mapping,
- YAML for reading configuration files,
- ftools for manipulating files in a file system.

C. Profile Knowledge Base

The Profile Knowledge Base provides a shared data model for the monitoring system and the data manager components. It is implemented as a relational database whose schema is depicted in Figure 4.

By sharing a common data model, the monitoring system and the data manager can be implemented as loosely coupled components. Also, the shared data model allows to easily attach new components to the system in the future. The presented prototype of the system is based on the MySQL database [40].

VI. EXPERIMENTAL EVALUATION

In order to evaluate the implemented prototype, a proper testing infrastructure has been configured and a number of tests were performed. The evaluation aimed at finding how the proposed system influences Cloud performance.

A. Testing environment

Testing environment is a very important aspect of the experimental evaluation. Thus, we prepared a sample configuration for building a small Cloud installation based on a blade-class cluster nodes and a disk array. As a base server for an extended version of the Eucalyptus cloud we use a worker node with the following parameters:

- 2x Intel Xeon CPU L5420 @ 2.50GHz (4 cores each)
- 16 GB RAM
- 120 GB hard drive (5400 RPM)
- Ubuntu Linux 10.04.1 LTS

Apart from the Cloud front end where the Cloud controller and Walrus components were installed, we also have three similar nodes for running virtual machines connected with the front end by Gigabit Ethernet. However, a more interesting part of the environment is the storage. As the main storage resource for our Cloud installation we used part of a disk array accessible via iSCSI protocol, of 6 TB capacity. Such a disk array, however, with a greater storage capacity available, could be used in a production cloud. As an additional storage, we decided to use hard drives from additional worker nodes which are exposed via the NFS protocol. To summarize, we depicted a map of the testing environment in Figure 5. In our opinion, the presented environment can be effectively used to evaluate different storage strategies because it contains heterogeneous storage resources such as hard drives and disk array distributed among a few machines all being connected with open protocols and a commodity network fabric.

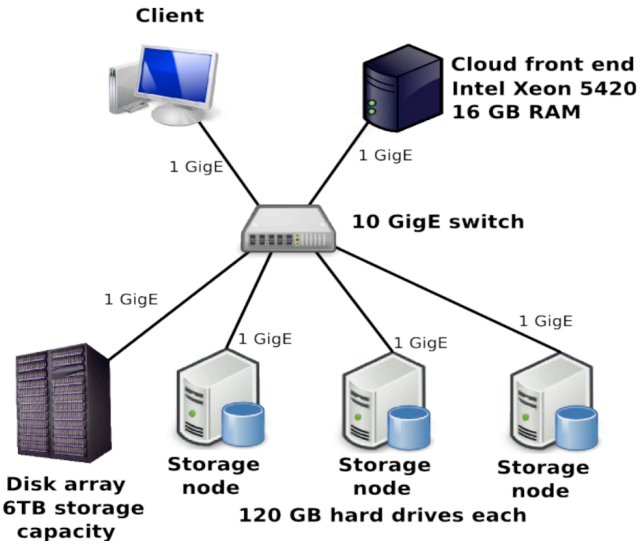

Figure 5. A map of physical resources which constitute a testing environment.

B. Testing scenario

Due to the limited throughput of the network interface from each worker node (1 GbE) which is lower than the overall throughput of the available storage devices, we had to configure the Cloud in a specific way. We decided to use the three storage nodes as a Cloud storage back end and the disk array as a user device, i.e., a device to which users download data. Using this configuration we were able to show the positive influence of the "Behaviour-inspired Data Manager" on the data access time stored in the Cloud despite the limited network throughput. It is worth mentioning that such a configuration was feasible to prepare using our extension to the Eucalyptus Cloud which supports distributed storage.

To compare a standard Eucalyptus installation with an installation supported with the behaviour-inspired Data Manager, we prepared a test case which focused on the read operation to check the dynamically increasing replication feature. In this scenario, we assume there are three files stored in the Cloud of the same size which equals 5 GB. The number and size of the files were selected to obviate the cache mechanism of the storage resources. The files were downloaded several times by a number of users simultaneously. We ran the test case starting with 1 user and ending with 10 users. Also, each test case was performed three times and a mean value was calculated to abate possible noises within the infrastructure during tests.

C. Evaluation results

The results obtained from the above described test case are presented in Figure 6. The chart in this figure presents overall data read time for each number of users. As long as

the number of users is less than 5, the overall data read time is similar for both Cloud installations: with and without support from the Behaviour-inspired Data Manager. This similarity is anticipated since the Data Manager reacts only in the situations described by behaviour classes. In our case, we only defined behaviour classes for read-only and write-only data objects. Since the number of users, less than 5, do not generate enough workload, the Data Manager was virtually idle. . Then, for 5 users and more, the observed usage of each data object is classified as relevant to the read-only object. Thus, the Data Manager replicated data objects among available storage resources. This operation reduces data read time.

The mean gain from using the Behaviour-inspired Data Manager for more than 5 users is about 10% which is pretty high due to the limited throughput of physical network interfaces in the Cloud installation. Moreover, the measurements from different series for the same number of users were very similar and repeatable which implies that both presented solutions: support for distributed storage and the Behaviour-inspired Data Manager are stable.

VII. FUTURE WORK

While the presented prototype is fully functional, there is still some place for enhancements. From the conceptual point of view, the approach lacks of a well-defined set of behaviour classes along with related actions. However, these classes will be crystallized during real life tests when the behaviour of the real users will be observed and analyzed.

Also, the implementation of the described components can be improved. The monitoring system will be extended with analysis of semantic relationship between observed storage-related operations. Such an analysis will lead to detection of different performance issues. The Data Manager will be extended by analysis of trends in storage-related operations. An analysis of such trends will enable the Data Manager to perform more suitable management actions. Additionally, we plan to include some AI-based mechanisms to discover new management actions based on the observed behaviour.

VIII. CONCLUSIONS

Although, there are several open source Cloud solutions available today, none of them provide a storage system which would be able to adapt to the user needs automatically. Instead, only basic functionality, e.g., storing VM images or custom objects is supported. The approach presented in the paper aims at providing a sophisticated data management functionality which would be flexible enough to be applicable to different Cloud solutions and which would manage data according to observed behaviour of Cloud users. We have presented the main assumptions of the approach along with phases of the management process. As an example of its implementation a prototype version of the system based on Eucalyptus is described. Due to the limited functionality of the Eucalyptus system, an extension which provides a real distributed data storage to multiple locations has been implemented.

The performed tests show positive influence of the described components on the performance of the Cloud in common use cases. The tests prove also the stability of the implemented prototype.

Figure 6. Summary read time of 3 files (5 GB each) for each user for different number of users.

ACKNOWLEDGMENTS

This research is supported partly by the European Regional Development Fund program no. POIG.02.03.00-00-007/08-00 as part of the PL-Grid Project. The authors are grateful to prof. Jacek Kitowski for valuable discussions.

REFERENCES

[1] Krol, D., Slota, R., Funika, W., „Behaviour-inspired Data Management in the Cloud“, in: Proc. of CLOUD COMPUTING 2010 The First International Conference on Cloud Computing, GRIDs, and Virtualization November 21-26, 2010 - Lisbon, Portugal, IARIA, 2010, pp. 98-103.

[2] Foster, I., and Kesselman, C. (Eds.), “The Grid: Blueprint for a New Computing Infrastructure”. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1998.

[3] Cloud Computing – Google: The Best Cloud Computing Investment. [on-line: http://www.cloudtweaks.com/2010/02/cloud-computing-google-the-best-cloud-computing-investment/, as of June 13, 2011].

[4] Han, J.H., Lee, D.H., Kim, H., In, H. P., Chae, H.S., and Eom Y.I., “A situation-aware cross-platform architecture for ubiquitous game”, Computing and Informatics, vol. 28(5) (2009) 619-633.

[5] Amazon Elastic Compute Cloud (Amazon EC2). Amazon Inc., 2008, [on-line: http://aws.amazon.com/ec2, as of June 13, 2011].

[6] Microsoft Windows Azure Platform (Windows Azure). Microsoft, 2010 [on-line: http://www.microsoft.com/windowsazure/, as of June 13, 2010].

[7] Google AppEngine. Google Inc., 2008 [on-line: http://code.google.com/intl/pl-PL/appengine/, as of June 13, 2011].

[8] Google Apps website [online: http://www.google.com/apps/intl/en-GB/business/index.html, as of June 13, 2011].

[9] Amazon EC2 instances [on-line: http://aws.amazon.com/ec2/instance-types/, as of June 13, 2011].

[10] Eucalyptus Systems Inc. [on-line: http://www.eucalyptus.com/, as of July 24, 2010]/

[11] Amazon Simple Storage Service (Amazon S3). Amazon Inc. [on-line: http://aws.amazon.com/s3/, as of June 13, 2011].

[12] Apache Axis website [on-line: http://ws.apache.org/axis/, as of June 13, 2011].

[13] Google Web Toolkit website [on-line: http://code.google.com/intl/pl-PL/webtoolkit/, as of June 13, 2011].

[14] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, and A. Warfield, "Xen and the art of virtualization," in SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles. New York, NY, USA: ACM, 2003, pp. 164-177 [on-line: http://dx.doi.org/10.1145/945445.945462, as of June 13, 2011]

[15] Kernel-based Virtual Machine project wiki [on-line: http://www.linux-kvm.org/page/Main_Page, as of June 13, 2011].

[16] Lustre filesystem wiki [on-line: http://wiki.lustre.org/index.php/Main_Page, as of June 13, 2011].

[17] Oracle Cluster File System 2 (OCFS2) project website [on-line: http://oss.oracle.com/projects/ocfs2/, as of June 13, 2011].

[18] Eucalyptus Enterprise version website [on-line: http://www.eucalyptus.com/products/eee, as of June 13, 2011].

[19] Introduction to Storage Area Networks, IBM redbook, [on-line: http://www.redbooks.ibm.com/abstracts/sg245470.html?Open, as of April 16, 2011].

[20] Kielmann, T., “Cloud computing with Nimbus”, March 2009, EGEE User Forum/OGF25 & OGF Europe's 2nd International Event.

[21] Globus Alliance website [on-line: http://www.globus.org/, as of June 13, 2010].

[22] Foster, I., Frey, J., Graham, S., Tuecke, S., Czajkowski, K., and Weerawarana, S., “Modeling Stateful Resources with Web Services”, 2004 [on-line: http://www.ibm.com/developerworks/library/ws-resource/ws-modelingresources.pdf, as of June 13, 2010].

[23] NASA Nebula website [on-line: http://nebula.nasa.gov/, as of April 16, 2011].

[24] RackSpace CloudFiles solution website [on-line: http://www.rackspace.com/cloud/cloud_hosting_products/files/, as of April 16, 2011].

[25] OpenStack project website [on-line: http://www.openstack.org, as of April 16, 2011].

[26] G. Behrmann, P. Fuhrmann, M. Gronager, and J. Kleist, “A distributed storage system with dCache”, in G .Behrmann et al Journal of Physics: Conference Series, 2008.

[27] Flash Cloud web. [on-line: http://www.flashcloudstorage.com/, as of June 13, 2010]

[28] Spring Framework website [on-line: http://www.springsource.org/, as of June 13, 2010].

[29] Java database website [on-line: http://developers.sun.com/javadb/, as of June 13, 2010].

[30] B. Sotomayor, R. S. Montero, I. M. Llorente, and I. Foster, "Capacity Leasing in Cloud Systems using the OpenNebula Engine." Cloud Computing and Applications 2008 (CCA08), 2009.

[31] VMware website [on-line: http://www.vmware.com, as of June 13, 2010].

[32] RESERVOIR project website [on-line: http://62.149.240.97/, as of June 13, 2010].

[33] F. Hupfeld, T. Cortes, B. Kolbeck, E. Focht, M. Hess, J. Malo, J. Marti, J. Stender, and E. Cesario. “XtreemFS - a case for object-based file systems in Grids.”, In: Concurrency and Computation: Practice and Experience. vol. 20(8) (2008).

[34] VMware Virtual Appliances website [on-line: http://www.vmware.com/appliances/getting-started/learn/, as of June 13, 2010].

[35] Network File System version 4 protocol specification [on-line: http://tools.ietf.org/html/rfc3530, as of June 13, 2010].

[36] Logical Volume Management 2 (LVM2) website [on-line: http://sourceware.org/lvm2/, as of June 13, 2011].

[37] Internet Small Computer Interface (iSCSI) RFC document [on-line: http://www.ietf.org/rfc/rfc3720.txt, as of June 13, 2011].

[38] Web Distributed Authoring and Versioning (WebDAV) RFC document [on-line: http://www.ietf.org/rfc/rfc3744.txt, as of June 13, 2011].

[39] EVault Data Backup Software website [on-line: http://www.i365.com/products/data-backup-software/evault-backup-software/, as of June 13, 2011].

[40] The MySQL database website [on-line: http://www.mysql.com/].

[41] Slota, R., Nikolow, D., Kuta, M., Kapanowski, M., Skalkowski, K., and Kitowski, J., “Replica Management for National Data Storage”, Proceedings PPAM09, LNCS6068, Springer, 2010, in print.

[42] Slota, R., Nikolow D., Polak, S., Kuta, M., Kapanowski, M. , and Kitowski, J., "Prediction and Load Balancing System for Distributed Storage", Scalable Computing Practice and Experience, 2010, in print.