Uniform and Efficient Access to Data in Organizationally Distributed Environments

Łukasz Dutka[1], Renata Słota[2], Michał Wrzeszcz[1], Dariusz Król[1,2], and Jacek Kitowski[1,2]

[1] AGH University of Science and Technology, ACC Cyfronet AGH, ul. Nawojki 11, 30-950 Kraków, Poland
[2] AGH University of Science and Technology, Faculty of Computer Science, Electronics and Telecommunications, Department of Computer Science, al. Mickiewicza 30, 30-059 Kraków, Poland
{dutka,rena,wrzeszcz,dkrol,kito}@agh.edu.pl

Abstract. In this article, the authors present a solution to the problem of accessing data in organizationally distributed environments, such as Grids and Clouds, in a uniform and efficient manner. An overview of existing storage solutions is described, in particular high-performance filesystems and data management systems, with regard to the provided functionality, scalability and configuration elasticity. Next, a novel solution, called VeilFS, is described in terms of objectives to attain, its architecture and current implementation status. In particular, the mechanisms used for achieving a desired level of performance and fault-tolerance are discussed and preliminary overhead tests are presented.

Keywords: storage system, data management system, virtual file system, distributed environment, Grid.

1 Introduction

We are observing a fast growth of the digital universe [7]. However, the increasing number of powerful computing environments is more challenging than the increase in the overall data volume. The data not only has to be stored, but it also has to be processed. This problem is often called the Big Data revolution [11] and addresses such issues as the variety of data and the processing speed required to unlock the potential of access to such an amount of information. The processing of large volumes of diverse data with a satisfactory performance requires use of appropriate storage systems. Moreover, different requirements of user groups make it necessary to install heterogeneous storage systems managed by advanced data management systems for efficient storage resources usage and provisioning.

We have investigated data access requirements of PL-Grid Infrastructure users [9]. Thirteen groups of users representing essential science disciplines are supported by the PLGrid Plus project, namely: AstroGrid-PL, HEPGrid, Nanotechnologies, Acoustics, Life Science, Quantum Chemistry and Molecular Physics,

M. Bubak et al. (Eds.): PLGrid Plus, LNCS 8500, pp. 178–194, 2014.

Ecology, SynchroGrid, Energy Sector, Bioinformatics, Health Sciences, Materials, and Metallurgy. As the PL-Grid Infrastructure spans across five biggest academic computer centers in Poland, most of these groups need to access their data located in different sites and storage systems. A common use case involves users who schedule computations to all available sites to generate data, e.g. as part of a data farming experiment [10], and then collecting the data in a single site to extract meaningful information.

The PL-Grid Infrastructure provides various storage systems fulfilling different requirements: "scratch" for intermediate job results, "storage" for final job results, and LFC/DPM for long-term data storage. "Scratch" and "storage" systems are file systems used locally within each site, while LFC/DPM is a global solution, accessible through a dedicated API, appropriate for data sharing between sites. Even within a single PL-Grid site, efficient data management is a challenging task, especially when involving the data access quality requirements, due to dealing with workload balancing between multiple storage resources abstracted by a single file system [14,15]. The users' quality requirements, defined using SLA, are an important aspect of resource management, as well as an important aspect of users' accounting [16].

The analysis of the PL-Grid Infrastructure users' requirements has shown that access to data is often too complicated [13]. The variety of possible storage solutions confuses users. Users expect that data access will be simple using one tool. The distribution of large scale computational environments should not create new barriers, but should provide new opportunities such as intra-community data sharing and collaboration.

The data access may be simplified by administrators of computing centers who can create special storage spaces for particular groups of users. However, it complicates the work of administrators and causes problems in the infrastructure maintenance when a lot of storage spaces is created and supported. Tools that will help the administrators with such management of various storage systems will be useful for them.

In this article, the authors introduce a system operating in the user space (i.e., FUSE), called VeilFS, which virtualizes organizationally distributed, heterogeneous storage systems to obtain uniform and efficient access to data. A single VeilFS instance works at a data center site level, but can also cooperate with instances from other sites to support geographically distributed organizations or separated organizations, which share resources on a federation level.

The rest of the article is organized as follows. Section 2 presents other possible solutions to the problem of accessing data in organizationally distributed environments and shows their shortcomings. The proposed VeilFS system architecture is widely described in Section 3. Section 4 shows the current state of VeilFS's implementation and preliminary tests results. Finally, Section 5 concludes the article.

2 State of the Art

In distributed environments, storage resources are provided to users through one of two types of systems: (1) high-performance file systems and (2) data management systems. Solutions of the former type intend to provide access to storage resources in an optimized for performance manner. They are built on top of dedicated storage resources, e.g. RAIDs, and they expose a POSIX-compliant interface. They can be used during computation to store results, which exceed the capacity of a single hard drive. Examples of such solutions include GlusterFS [3], CephFS [1], and Scality RING [5]. Despite providing similar functionality, these systems differ in implementation, which influences their non-functional features. For instance, to resolve the actual location of data, GlusterFS uses an elastic hashing algorithm (each node is able to find the location of the data algorithmically, without use of a metadata server), CephFS uses a metadata server, while Scality RING utilizes a routing-based algorithm within a P2P network. Most of them are scalable, while strictly avoiding architectural bottlenecks. The choice which one to use should depend on the actual requirements, e.g. low latency or support for dynamic reconfiguration. However, they are not suitable for Grids due to limited support for federalization, which is essential in organizationally distributed environments such as Grids.

The second type of solutions for exposing the storage resources are data management systems. Such systems are oriented towards high-level data management, e.g. to provide data accessibility between sites, rather than high-perfor-mance data access. They aim at providing an abstraction layer on top of storage resources across multiple organizations, which exposes a single namespace for data storage. In addition, many of such systems facilitate data management by enabling administrators to define data management rules, e.g. w.r.t. archiving the unused data. Examples of such systems include Parrot [17], iRODS system [8], [12], and DropBox [2]. The main goal of these systems is to enable data access from anywhere in a uniform way, e.g. Parrot utilizes the *ptrace* debugging interface to trap the system calls of a program and replace them with remote I/O operations. As a result, remote data can be accessed in the same way as local files. Data integration in the iRODS system is based on a metadata catalog – iCAT – which is involved in the processing of the majority of data access requests. The metadata catalog is implemented as a relational database, hence it can be considered as a bottleneck of the whole system.

The analyzed systems were compared (see Table 1) according to the following features:

- high-performance data access,
- data location transparency,
- support for data management rules,
- POSIX-compliant interface,
- decentralized management.

The first thing to notice is that there is no system, which would provide all the mentioned features. High-performance file systems provide efficient data

Table 1. Comparison of storage exposing solutions

System	High-performance	Location transparency	Management rules	POSIX-compliancy	Decentralized management
GlusterFS	Yes	Yes	No	Yes	No
CephFS	Yes	Yes	No	Yes	No
Scality RING	Yes	Yes	Yes	Yes	No
Parrot	No	Yes	No	Yes	No
iRODS	No	No	Yes	Yes	Yes
DropBox	No	Yes	N/A	No	No

access, but they lack data management capabilities, while data management systems provide management-related capabilities, but they do not ensure high-performance. As a result, the users have to utilize high-performance systems during computations in one site, but a data management system has to be used during access to the generated data from other sites. Moreover, administrators have multiple storage solutions to manage, which can be error-prone and inefficient.

3 A Unified Data Access Solution for Organizationally Distributed Environments – VeilFS

Below we describe our solution to the problem of accessing data stored in organizationally distributed environments in an efficient and uniform way, named VeilFS, implemented as a virtual file system. We start with a summary of the functionality provided by our system. Next, we discuss its architecture and its internal components.

3.1 Functionality Overview

The VeilFS system provides a unified and efficient access to data stored in organizationally distributed environments, e.g. Grids and Clouds. From the user point of view, VeilFS:

- **Provides a Uniform and Coherent View on All Data Stored on the Storage Systems Distributed Across the Infrastructure.** The user of geographically distributed organization can perform many tasks simultaneously. The tasks may be executed in one or many locations. Using VeilFS, all user's processes see all data stored in all sites. If the needed data is not stored locally, it is migrated by the system. The user can also mount VeilFS on her/his PC and access the data as if it was stored on a local hard drive.
- **Supports Working in Groups by Creation of an Easy-to-Use Shared Workspace for Each Group.** Users are able to share data inside a group by simply moving the data to an appropriate directory.

- **Supports Data Sharing.** Users are able to publish a file obtaining a unique URL. Until canceled by the owner, anyone is able to download the file using the obtained URL. However, only the owner is able to modify the file.
- **Serves Data Efficiently.** The system is designed to minimize overheads and provide data from remote storages as fast as possible.

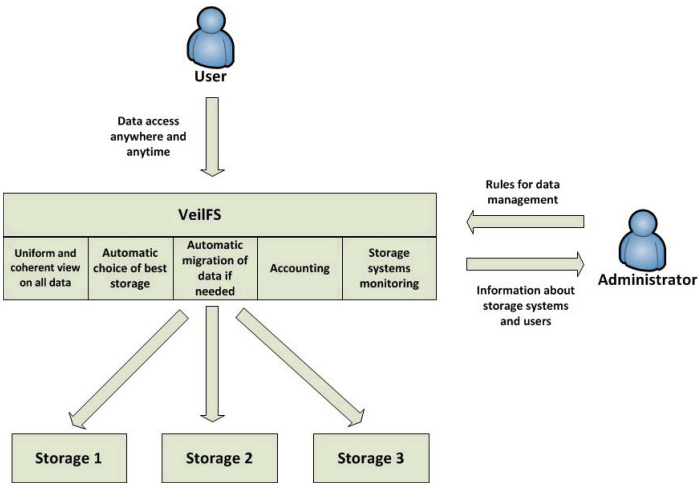

Fig. 1. Functionalities provided by VeilFS

However, simplification of the system for the users results in an increase in the number and difficulty of management tasks that have to be done by administrators or automaton. Hence, VeilFS provides functionalities that also facilitate administrators' work:

- Gathering and visualizing of the infrastructure state monitoring data – each storage system supervised by VeilFS is monitored to provide the administrators with an insight into storage utilization.
- Rule-based data management – automatic data management can be performed on the basis of rules specified by administrators, e.g. the rarely used data can be automatically migrated from fast to cost-effective storage. The optimization of use of storage resources is transparent to the users.
- Data protection from unauthorized access – the system is integrated with grid and Linux security systems to protect the data.

Both points of view are summarized in Fig. 1.

Such a wide set of functionalities implies that VeilFS can be perceived as a file system (it provides access to files), but on the other hand it can be treated as a data management system, because it manages the data on storage systems beneath it.

3.2 High-Level Architecture of VeilFS

An exemplary environment where VeilFS works is depicted in Fig. 2. This environment contains two sites. In each site an instance of VeilFS is installed. The most important element is Data Management Component, which coordinates the system's operation in the site and processes client requests. It consists of a set of cooperating modules, which may be deployed to a cluster to increase the throughput of the component. It is connected to a database (DB), which stores information about metadata, e.g. the mappings of logical filenames to the actual data location on a storage. Each site has its own Data Management Component with DB.

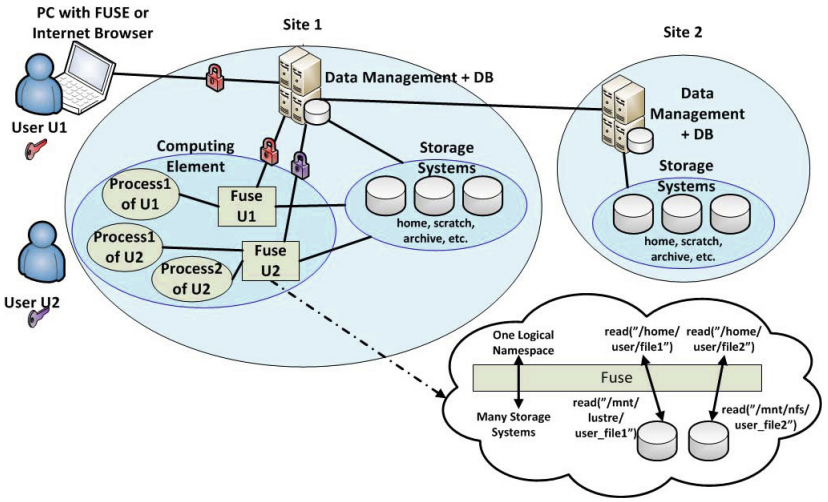

Fig. 2. Exemplary environment with VeilFS

The site usually contains many Computing Elements where users' processes are executed. Computing Elements are connected to Storage Systems where users' data is stored. Storage Systems in sites are usually organized as high performance systems with high capacity (e.g. Lustre) rather than simple hard disks. Not all Computing Elements must be connected to all Storage Systems, in contrary to the machines where Data Management Component is deployed.

There can be three levels of utilization of VeilFS:

- site level, i.e., a single data center being a part of a single organization,
- organization level (also geographically distributed organization), i.e., a single organization, which spans across one or more sites,
- federation level, i.e., separated organizations cooperating together to achieve a common goal, yet maintaining their autonomy.

There is no difference between the deployment of components when instances of VeilFS are cooperating within an organization or a federation. The difference is in the administration – VeilFS instances that belong to the same organization can cooperate more closely.

3.3 System Clients and Protocols

End users access the data stored within VeilFS through one of the provided user interfaces:

- FUSE client, which implements a file system in user space to cover the data location and exposes a standard POSIX file system interface,
- Web-based GUI, which allows data management via any Internet browser,
- REST API.

The simplest way of usage is a Web based GUI. The Internet browser installed on a user's PC connects to Data Management Component using safe protocols. After authentication, the user is able to perform several operations on files, e.g. download, upload or publish.

The FUSE client operates on files as if they were stored locally. It may be installed on Computing Elements within a site, or on a user PC. The client provides a file system in user space, which translates a logical file name to the actual data location on a storage system. To do the translation, the file system communicates with Data Management Component. The system intends to provide efficiency sufficient for high-performance applications, therefore it operates on the data locally whenever it is possible (in many cases Computing Elements are connected using a shared storage within a site). If the data is not reachable locally, the system provides the data from a remote storage system as efficiently as possible. Data Management Component has access to all storages in a site so it may copy the data to the storage, to which the client has direct access, or stream data to the client. VeilFS covers the management of temporary copies according to the rules specified previously by administrators, e.g. only the required blocks of a file can be copied in case of large datasets. If the client is located in a different site than the data, the instances of Data Management Component of both sites cooperate to increase the data transfer speed – the data is copied using an own protocol that allows for utilization of many hosts and many channels at the same time. The client from outside connects to the Data Management Component instance using DNS, where Data Management Component instances register information about users. When the data is not reachable locally, advanced buffering and prefetching algorithms are used.

VeilFS provides a REST programming interface to enable integration with other applications, which handle user data, e.g. domain-specific web portals. Another example is grid scheduling systems, which can decide to execute a grid job in a particular site, based on information about the required data location obtained through the REST API. Moreover, the scheduling system may request a transfer of the remaining data to the site where the job will run while this job is queued.

3.4 Data Management Component Architecture

Inside the Data Management Component we can distinguish between a few logical elements that perform specific tasks. We call them *modules*. The modules are listed below and shown in Fig. 3:

- *fslogic* – it is responsible for mapping the logical filenames to the real locations of data. It handles requests from FUSE clients.
- *dao* – this module performs operations in database.
- *rtransfer* and *gateway* – they are responsible for transfer of data between sites. One *rtransfer* can cooperate with many *gateways*. The *gateway* transfers parts of the data while the *rtransfer* coordinates requests (there may be many requests for transfer of the same data) and splits data between gateways to increase the transfer.
- *control_panel* – this module handles requests from web-based GUI.
- *remote_files_manager* – this module mediates between the FUSE client and storage during I/O operations when the FUSE client is not directly connected to the storage where data is located (e.g. it is installed at user's PC).
- *dns* – it provides answers to DNS queries. It is part of the load balancing system described later in this article.
- *cluster_rengine* – it handles the events used for monitoring and executes rules triggered by these events.
- *rule_manager* – this module allows for definition of management rules by administrators.
- *central_logger* – it gathers logs from all the nodes of the system in one place.

Analysis of exemplary use cases can help understand the functionality of each module. When a user writes some data using FUSE client on his/her PC, the client sends a DNS request to identify, to which machine it should connect to perform further operations. The answer is formulated by the *dns* module. Next, the client sends a request to *fslogic* to resolve the location of data. *fslogic* obtains this information from the database through *dao*. When the data should be written to a remote storage, the client writes this data sending it to *remote_files_manager*.

Afterwards, when the user wants to read other file, the client again uses *fslogic* to get location of the desired data. If the data is located in other site, it returns the information where the data will be copied and sends a request to *rtransfer* to download the needed data. *rtransfer* chooses instances of the *gateway* module that should perform the transfer and sends requests to them. The data may be split across many *gateways* when a large amount of data is requested (it is possible due to the use of the prefetching algorithms).

Administrators can define some rules, e.g. quota check every 10 000 writes performed by the user. It is done by defining a rule via the web-based GUI. GUI requests are handled by the *control_panel* module, which, in this case, sends a request of rule definition to *rule_manager*. *rule_manager* sends the definition of the rule to *cluster_rengine* and requests all connected clients to produce write events. The newly connected clients (e.g., recently installed on users' PCs) will download a list of events to be produced during the initialization. Then, when the

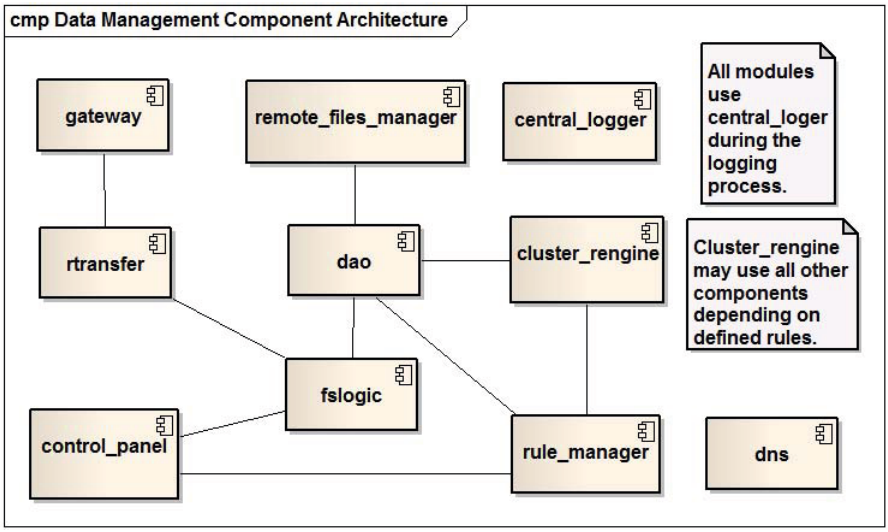

Fig. 3. Data Management Component modules

user performs a write operation, the write event is produced. If the administrator wishes to, the events are initially aggregated at the client-side, to reduce the load of network interfaces, and then sent to *cluster_rengine*, which counts events and performs the rule when at least 10 000 events have appeared.

3.5 Security

Data Management Component should be deployed on machines that are connected by internal network protected by a firewall. Only two ports should be opened: 53 for DNS and 443 for communication with clients, which is possible only using encrypted transmission (SSL).

The VeilFS authentication mechanisms are integrated with mechanisms already existing in PL-Grid so the user may use the PL-Grid certificate to mount the FUSE client or log-in to the web-based GUI. Additionally, proxy certificates generated with the PL-Grid certificates are also supported to enable the use of the FUSE client inside grid jobs. The web-based GUI cooperates with PL-Grid OpenID to automatically download additional information about users so the user is also able to log-in to GUI using the PL-Grid username and password.

The authorization mechanism is based on the fact that the sender of each request can be identified using data from his/her certificate or OpenID. Thus, the users' spaces in the database, that store information about data location, can be separated – the space connected with the request is automatically chosen. It is impossible to modify the data from other users' spaces.

The PL-Grid user accounts are automatically mapped to operating system accounts on all Computing Elements inside the sites. The data on the storage

is created with adequate permissions so it is not possible to read the data that belongs to other user directly from the storage system.

FUSE clients installed outside all sites (on users' PCs) perform operations on the storage through Data Management Component that works with root permissions. Data Management Component verifies if the sender of a request has appropriate permissions to operate on a chosen data, because one might try to operate on other user's data by sending specially prepared requests of storage operations.

3.6 Deployment

Data Management Component has to manage thousands of clients working simultaneously. Furthermore, each client can generate several requests per second. Thus, one of the main non-functional requirements for Data Management Component is providing high-performance in terms of processed requests per second. For this reason, Data Management Component was written in Erlang, which provides very lightweight processes in comparison with standard operating system processes. To provide the implementation of basic language elements such as lightweight processes, Erlang has a dedicated Execution Environment – Erlang Virtual Machine. Erlang supports two types of applications – Erlang Application and Erlang Distributed Application. Erlang Application is executed inside a single Erlang Virtual Machine while Erlang Distributed Application links several Erlang Virtual Machines. At the start, Erlang Distributed Application is initialized on many Erlang Virtual Machines. However, all but one instances are paused immediately after the initialization. If the working instance fails, one of the paused instances is automatically resumed.

Data Management Component is deployed on a dedicated cluster of nodes using two types of applications that cooperate to provide the needed functionality:

- Erlang Worker Application (EWA), which hosts the modules (see subsection 3.4). It is a standard Erlang Application.
- Erlang Management Distributed Application (EMDA), which manages the cluster. It is an Erlang Distributed Application.

On each node, where Data Management Component is deployed, EWA is started. EMDA is started on chosen nodes. EWA and EMDA do not share Erlang Virtual Machine – if the node hosts them both, two instances of Erlang Virtual Machine are launched.

The nodes are physical or virtual machines. It is recommended to use physical machines to increase hardware fault tolerance (if a deployment is based on virtual machines, a hardware problem may cause a crash of many Virtual Machines – in the worst case it can be a crash of Virtual Machines that host all instances of EMDA). A deployment on a 5-node cluster is depicted in Fig. 4. Summarizing, the cluster nodes can be split into three groups:

- Management Master Node that hosts EWA and EMDA. On this node EMDA is working.

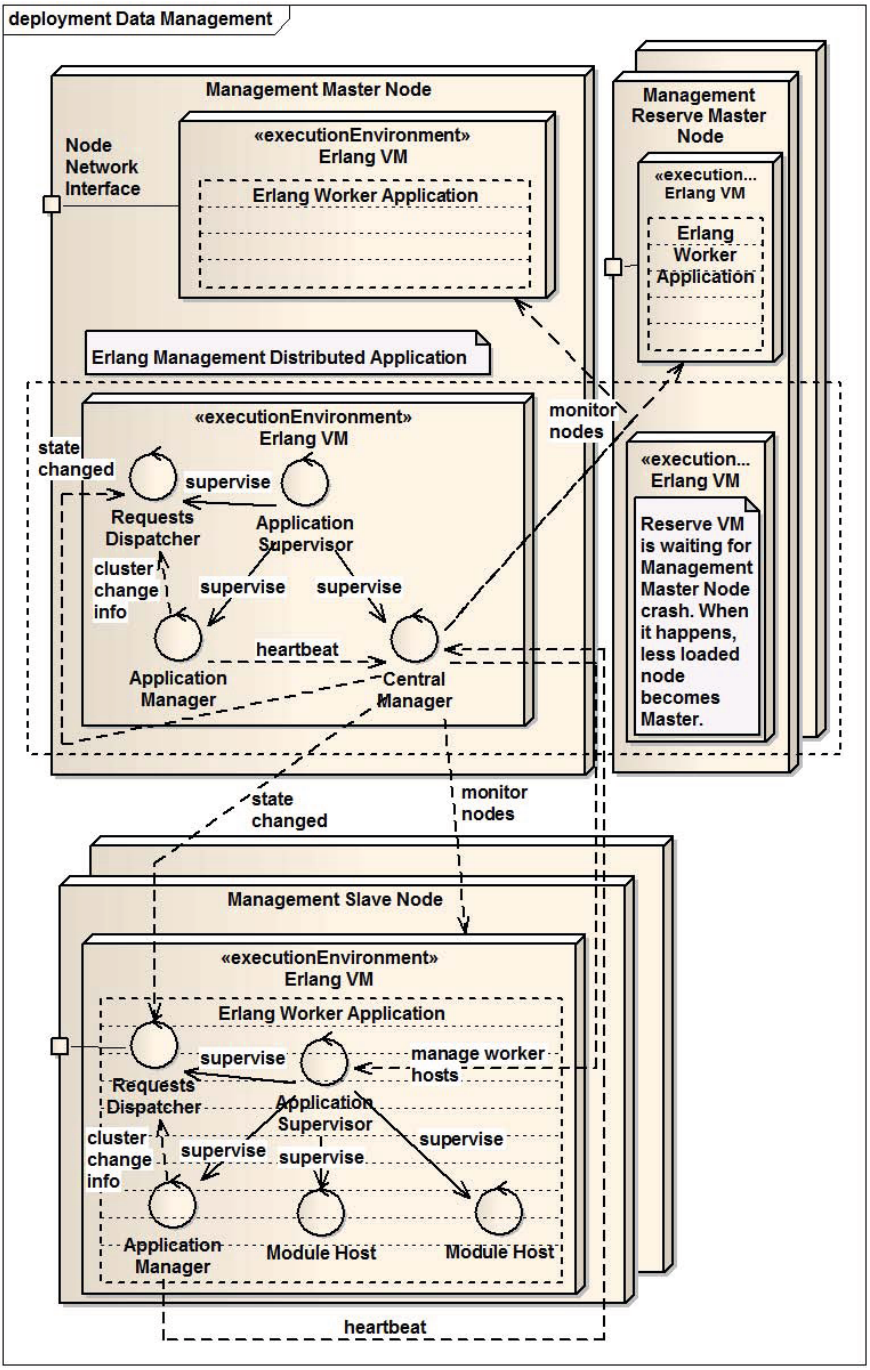

Fig. 4. Data Management Component exemplary deployment

- Management Reserve Master Nodes that host EWA and EMDA. Instances of EMDA are paused on these nodes.
- Management Slave Nodes that host only EWA.

When the Management Master Node fails, one of Management Reserve Master Nodes resumes EMDA and becomes the Management Master Node.

To provide the requested functionality and meet non-functional requirements, several application elements have been implemented using Erlang (see Fig. 4):

- Module Host – an element that executes the code of a chosen module (see 3.4).
- Central Manager – an element that coordinates all nodes, which are part of Data Management Component.
- Application Manager – an element that monitors the state of the node where the application is deployed and provides information about the node state to the Central Manager.
- Application Supervisor – an element that monitors the execution of application code and repairs the application after an error (Supervisor is an Erlang element that monitors Erlang processes and restarts them in case of failure).
- Requests Dispatcher – an element that is responsible for forwarding the requests to the appropriate Module Host.

EWA includes instances of Module Host, which provide modules' functionalities. Requests are processed by the modules concurrently – each request has its own Erlang process inside Module Host. The nodes and EWAs are independent. They cooperate due to the use of Central Manager, which coordinates their work. The Central Manager is the most important element of EMDA. It is not a single point of failure owing to the properties of Erlang Distributed Application described above. The set of nodes may change dynamically through the use of Application Managers. Application Manager periodically sends a heartbeat to Central Manager so Central Manager is able to discover new EWAs. Afterwards, Central Manager monitors Erlang Virtual Machine inside, which EWA works to notice when Erlang Virtual Machine is stopped, e.g. due to a failure. Application Manager also checks periodically if Requests Dispatcher (see subsection 3.7) has up-to-date information about the modules and triggers an update if needed.

Application Manager constantly monitors the load of a node. Module Host monitors the load of a module. Central Manager periodically gathers information about the load from all instances of Application Manager and Module Host. On the basis of this information, it dynamically starts/stops instances of Module Host working for chosen modules to provide load balancing and high availability of each module (for more details see subsection 3.7). Central Manager uses Application Supervisor available on each node to start/stop instances of Module Host. Application Supervisor is an element that monitors Erlang processes and restarts them in case of a failure so Central Manager is not involved in repairs.

Such a design of Data Management Component makes it scalable and resistant to failures. Central Manager is able to capture any changes and respond to them. Moreover, all nodes are able to work independently in case of a network failure, because they have own Supervisors. When the network is repaired, they connect once more and Central Manager is again able to reconfigure the node if needed.

3.7 Notifications and Requests

Clients send synchronous requests to Data Management Component to get/update metadata, e.g. get the mappings of logical filenames to the actual data locations on a storage or initiate metadata for the newly created files. Beside the standard requests connected with metadata, VeilFS heavily utilizes asynchronous notifications, which are processed by Data Management Component to exchange monitoring information across the system. Notifications provide useful information about the infrastructure state to administrators. The administrators use this information to create or parameterize data management rules, which are the basis of the effective data management subsystem, e.g. the rules that control data migration or system quotas. When the rules are defined, notifications are used as triggers of the rules (see exemplary use cases described in subsection 3.4). A typical rule concerns data management within a site. Data management that involves different sites is described in subsection 3.8. Notifications also enable detailed accounting. Beside the information about the amount of data stored in the system, other information such as the load of storage systems by read/write operations generated by user's processes can be controlled.

Types of requests/notifications may change with time. To meet this challenge, a scalable and elastic way of request handling has been designed. Central Manager controls DNS to inform clients, at which nodes a particular module works. However, Module instances may be started/stopped dynamically when types of incoming requests/notifications change. Therefore, a Module instances location may change before the mapping of modules to nodes provided by DNS is expired. Moreover, when many clients work simultaneously, a situation when no mapping is valid may never occur. For this reason, an instance of Requests Dispatcher works inside each application. It is used to route requests from a network interface to the nodes where a desired module is working. Having a Dispatcher instance, control over DNS is not required (requests are always redirected to an appropriate node), but is profitable, because an extensive use of DNS decreases the network traffic inside a cluster (requests usually go straight to the node, which hosts the required module). Additionally, Dispatcher provides a load balancing capability. The communication between instances of Module Host and Requests Dispatcher is not visualized in Fig. 4 – it would make the image too complicated. Instead, a request flow is depicted in Fig. 5.

3.8 Data Access in Organizationally Distributed Environments

The deployment of VeilFS is similar in the context of distributed organization and federation. The access to the data located in different sites was described in subsection 3.3. However, some requirements have to be fulfilled to provide a uniform and coherent view on data as well as an efficient access to data in the federation case. VeilFS supports rules, which operate between sites. Administrators of all organizations should agree on the management rules. Typically, the newly created data should be migrated to a site where the user has a granted storage when the data is no longer used by the process that has created it.

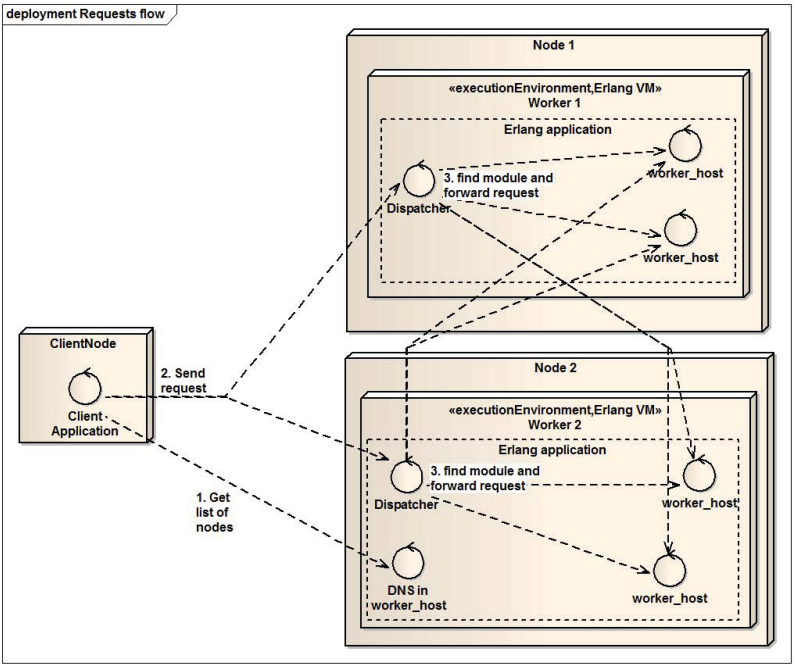

Fig. 5. Client request flow within VeilFS

However, when the user has an access granted to storage resources in multiple sites, different possibilities appear. For example, the data may be migrated to a site where it is used most frequently if all sites have activated rules that permit data send/receive in this case. It decreases the network traffic, because – once migrated – the data will be more frequently read locally than streamed.

4 Implementation Status

A prototype version of the presented system was implemented and evaluated. To provide high performance and scalability, Erlang, C programming languages and a NoSQL database were used. Erlang offers massive parallelism through its lightweight process mechanism, which is very important in the data management part, because Data Management Component cooperates with thousands of Computing Elements simultaneously. On the other hand, the C language enables efficient implementation of low level operations on the physical data. The information about metadata and the system state is stored in a fault-tolerant, high-performance, distributed NoSQL database to avoid performance bottlenecks and guarantee data security.

The implemented prototype version provides unification of namespace, support for group working and results publication. The FUSE client, web-based GUI

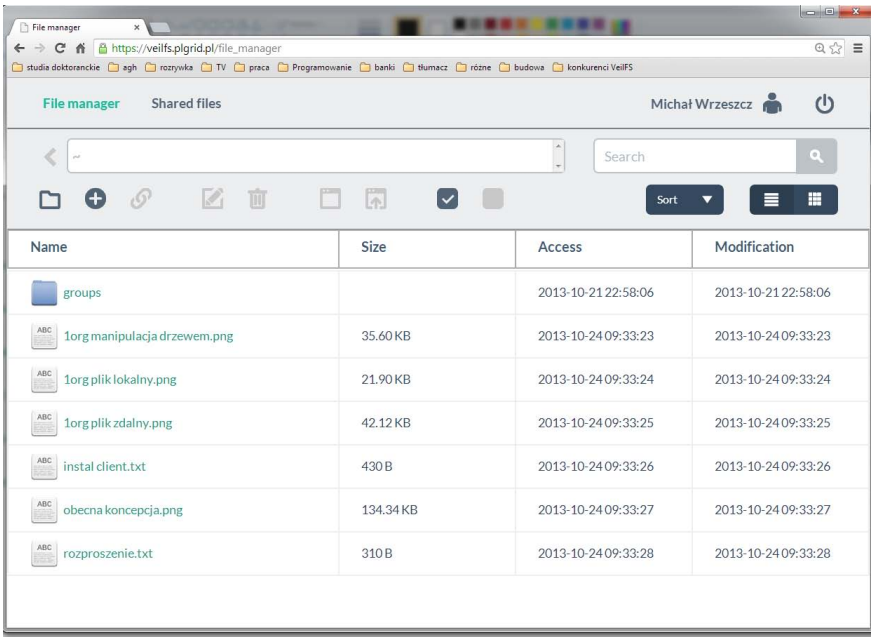

Fig. 6. Web-based GUI for VeilFS

Table 2. Preliminary overhead tests results

Number of threads used	1 thread			16 threads		
Test mode	RW	WR	RD	RW	WR	RD
Without VeilFS [Mb/s]	2,65	14,69	11,27	6,05	13,66	11,25
With VeilFS [Mb/s]	2,76	14,52	11,21	5,63	13,50	11,26
Difference [Mb/s]	0,10	-0,17	-0,06	-0,42	-0,15	0,01
Difference [%]	3,79	-1,12	-0,55	-6,99	-1,13	0,05

(see Fig. 6) and the REST API have been implemented using secure communication methods (SSL and GSI). The Data Management Component has been equipped with mechanisms that provide load balancing and high availability. Simple installators of client, Data Management Component and DB have also been created.

The components implemented using different technologies are able to cooperate. In case of the FUSE client, the component implemented in the C language communicates with modules on the server side implemented in Erlang. The requests are processed concurrently by light Erlang processes to minimize the answer time on multi-core machines. Preliminary tests results have shown that the overhead of VeilFS is low (see Table 2). The transfer rates measured by

SysBench [6] were similar when operations were performed directly using NFS [4] and when they were performed using VeilFS that exploited NFS to store data.

The current version of the system provides a complete set of functionalities for the end user – it provides a uniform and coherent view on all user's data, supports work in groups and data publication. Further work is going on to increase the functionality from the administrator point of view.

5 Conclusions and Future Work

A need for easy data access arises due to the continuously increasing diversity of storage systems. A users' requirements analysis has shown that access to files is often too complicated. Although there are a lot of tools that provide a single user interface for various storage management systems, they are too cumbersome to use in globally distributed environments.

The presented system addresses the issue of easy access to data along with administrators' requirements for effective managing federated, heterogeneous storage resources. The described solutions not only provide users with easy access to data anywhere and anytime, but also give administrators powerful tools for automated infrastructure monitoring and management. Moreover, the proposed architecture is able to process a large number of requests and notifications, which is needed to offer the described functionality. We believe that the system will be useful for all PL-Grid Infrastructure users and more users will be able to use all functionalities currently offered by the PL-Grid Infrastructure through the simplification of data management by VeilFS. We hope that the new functionalities offered by the system, e.g. a simple data sharing and publishing, will be appreciated by its users.

Future work will focus on further development of storage resources management, particularly in the areas of migration, caching and prefetching of data, as well as on creation the tools for administrators. Additionally, the authors are working to add a global level of VeilFS, which allows users data migration between different organizations that are not federated and globally unifies the access to data – the user will see all the data stored in all sites that belong to different federations regardless of the actual access point.

Acknowledgements. Thanks go to the rest of VeilFS team, especially to Rafał Słota, Łukasz Opioła, Darin Nikolow, Paweł Sałata, Beata Skiba and Bartosz Polnik for their support.

References

1. Ceph Filesystem web site, `http://ceph.com/docs/next/cephfs/`
2. Dropbox web site, `https://www.dropbox.com/`
3. GlusterFS community web site, `http://www.gluster.org/about/`
4. Nfs version 3 protocol specification, `http://tools.ietf.org/html/rfc1813`

5. Scality web site, http://www.scality.com/products/what-is-ring/
6. Sysbench: a system performance benchmark, http://sysbench.sourceforge.net/index.html
7. Gantz, J., Reinsel, D.: The Digital Universe in 2020: Big Data, Bigger Digital Shadows, and Biggest Growth in the Far East (2012), http://www.emc.com/leadership/digital-universe/index.htm
8. Hunich, D., Muller-Pfefferkorn, R.: Managing Large Datasets with iRODS: a Performance Analysis. In: Proceedings of the 2010 International Multiconference on Computer Science and Information Technology (IMCSIT), pp. 647–654 (2010)
9. Kitowski, J., Dutka, Ł., Mosurska, Z., Pająk, R., Sterzel, M., Szepieniec, T.: Development of Polish Infrastructure for Advanced Scientific Research – Status and Current Achievements. In: Proc. of IEEE Conf. 12th Inter. Symposium on Parallel and Distributed Computing (ISPDC 2013), Bucharest, Romania, pp. 34–41 (2013)
10. Kryza, B., Król, D., Wrzeszcz, M., Dutka, Ł., Kitowski, J.: Interactive cloud data farming environment for military mission planning support. Computer Science 13(3), 89–100 (2012), https://journals.agh.edu.pl/csci/article/view/19
11. Mills, S., Lucas, S., Irakliotis, L., Rappa, M., Carlson, T., Perlowitz, B.: DEMYSTIFYING BIG DATA: A Practical Guide to Transforming the Business of Government. Technical report (2012), http://www.ibm.com/software/data/demystifying-big-data/
12. Roblitz, T.: Towards Implementing Virtual Data Infrastructures – a Case Study with iRODS. Computer Science 13(4) (2012), http://journals.agh.edu.pl/csci/article/view/43
13. Słota, R., Dutka, Ł., Wrzeszcz, M., Kryza, B., Nikolow, D., Król, D., Kitowski, J.: Storage Systems for Organizationally Distributed Environments – PLGrid Plus Case Study. In: Wyrzykowski, R., Dongarra, J., Karczewski, K., Waśniewski, J. (eds.) PPAM 2013, Part I. LNCS, pp. 724–733. Springer, Heidelberg (2013)
14. Słota, R., Król, D., Skałkowski, K., Kryza, B., Nikołow, D., Orzechowski, M., Kitowski, J.: A Toolkit for Storage QoS Provisioning for Data-Intensive Applications. In: Bubak, M., Szepieniec, T., Wiatr, K. (eds.) PL-Grid 2011. LNCS, vol. 7136, pp. 157–170. Springer, Heidelberg (2012)
15. Słota, R., Nikolow, D., Kitowski, J., Król, D., Kryza, B.: FiVO/QStorMan Semantic Toolkit for Supporting Data-Intensive Applications in Distributed Environments. Computing and Informatics 31(5), 1003–1024 (2012), http://dblp.uni-trier.de/db/journals/cai/cai31.html#SlotaNKOK12
16. Szepieniec, T., Tomanek, M., Radecki, M., Szopa, M., Bubak, M.: Implementation of Service Level Management in PL-Grid Infrastructure. In: Bubak, M., Szepieniec, T., Wiatr, K. (eds.) PL-Grid 2011. LNCS, vol. 7136, pp. 171–181. Springer, Heidelberg (2012)
17. Thain, D., Livny, M.: Parrot: an Application Environment for Data-Intensive Computing. Journal of Parallel and Distributed Computing Practices, 9–18 (2005)