Computing and Informatics, Vol. 37, 2018, 1–22, doi: 10.4149/cai_2018_1_1

TWO-LAYER LOAD BALANCING FOR ONEDATA SYSTEM

Łukasz Opioła, Łukasz Dutka, Michał Wrzeszcz

AGH University of Science and Technology
ACC Cyfronet AGH
ul. Nawojki 11
30-950 Kraków, Poland
e-mail: {lopiola, dutka, wrzeszcz}@agh.edu.pl

Renata Słota, Jacek Kitowski

AGH University of Science and Technology
Faculty of Computer Science, Electronics and Telecommunications
Department of Computer Science
al. A. Mickiewicza 30
30-059 Kraków, Poland
&
ACC Cyfronet AGH
ul. Nawojki 11
30-950 Kraków, Poland
e-mail: {rena, kito}@agh.edu.pl

Abstract. The recent years have significantly changed the perception of web services and data storages, as clouds became a big part of IT market. New challenges appear in the field of scalable web systems, which become bigger and more complex. One of them is designing load balancing algorithms that could allow for optimal utilization of servers' resources in large, distributed systems. This paper presents an algorithm called Two-Level Load Balancing, which has been implemented and evaluated in Onedata – a global data access system. A study of Onedata architecture, request types and use cases has been performed to determine the requirements of load balancing set by similar, highly scalable distributed systems. The algorithm was designed to match these requirements, and it was achieved by using a synergy of

DNS and internal dispatcher load balancing. Test results show that the algorithm does not introduce considerable overheads and maintains the performance of the system on high level, even in cases when its servers are not equally loaded.

Keywords: Load balancing, geographically distributed systems, DNS, web clusters

Mathematics Subject Classification 2010: 68M11, 68M14, 68W99

1 INTRODUCTION

The number of Internet users grows every day, and the software and hardware vendors have to face demands for fast, convenient and massively scalable services. At the same time as web services are thriving, data centres around the globe are racing each other to create more and more powerful supercomputers. Many institutions form Grids and Clouds to provide platforms of immense computing power and storage space. Thanks to this, scientists of various disciplines can conduct research and cooperate to solve problems which have been beyond their reach for years. Such approach is called e-Science [9] and has created new trends in distributed computing. Again, there is a demand for scalable web services that will allow to harness the potential of distributed environments and ensure the convenience of using resources offered by data centres.

The answer to current challenges in the sector of web services are web clusters and distributed web systems, composed of tens or hundreds of servers. Recently, more and more systems try to use the advantages of highly distributed architectures. Popular web portals have to constantly extend their pools of servers to handle the growing amount of clients and data produced by them. Among them, there are cloud storage services such as Dropbox, Google Drive or OneDrive [1], which focus on any time and any place access to user data. Cloud solutions are also built for High Performance Computing (HPC) purposes, e.g. Amazon Web Services [2] and IBM HPC Cloud [3], which allow performing data-intensive computations on distributed architectures. Other massively scalable systems that handle millions of requests every minute and store enormous amounts of data in databases, include Facebook [4], Twitter [5] or Gmail [6]. As the complexity and distribution of those systems grow, it is harder to manage their numerous servers. Relevantly, the full, collective potential of multi-node architectures can be unlocked only by use of dedicated solutions that allow for cooperation of the servers. Among them, load balancing is arguably one of the most important. It determines the way that incoming requests are distributed to servers and strives to efficiently use their cumulative resources. The approaches used in modern systems range from simple static to complicated dynamic algorithms supported by server monitoring, dedicated hardware, advanced mathematical models and others. The choice of load balancing algorithm is dependent on many factors

including the system size, its purpose, type of clients and anticipated throughput of requests.

This paper describes an originally created load balancing algorithm called Two-Level Load Balancing. It is dedicated for highly scalable, distributed web systems that are meant to handle numerous requests that vary in size and processing time. It has been implemented and evaluated in `Onedata` [7, 8] to prove its viability in such systems.

`Onedata`, a global data access system, was created as an answer to the requirements of modern science. It virtualizes the storages of globally distributed storage providers and unifies them into one data space. From the user's point of view, it hides the growing complexity of storage systems. Moreover, it facilitates administration with advanced monitoring tools and automated data management. Beside being a new, powerful tool for research teams, it is also a promising choice for everyday users.

The rest of the paper is organized as follows. The second section includes gathered knowledge about existing load balancing strategies. In the third section, the `Onedata` system is described in detail to underline the requirements of load balancing algorithms that can be used in similar systems. In the next sections, the proposed solution is presented and evaluated.

2 STATE OF THE ART

For better understanding and to avoid ambiguities, several terms should be introduced that are used throughout this paper. Starting with architectural terms, a node is a single machine (computer). Nodes can be organized in groups and interconnected to form a cluster of nodes. Finally, a cluster of nodes can communicate with other clusters to form a distributed network of clusters. Another frequently used phrase is web system, which is a collection of hardware and software that constitutes a portal available on the Internet. A web system can be deployed on one of mentioned architectures, by installing server software on the nodes. A node that runs a server application is called a server (or a web server). An application that connects to a web system and sends requests is called a client, and it is most often a web browser. In cases where a web system runs on more than one node, a load balancing algorithm is required. It describes the decision process used to distribute incoming client requests among servers of a web system.

Load balancing is naturally an important aspect of every web system that can be deployed on a cluster of nodes. Hence, in recent years, numerous load balancing algorithms have been invented, tested and documented. The approaches differ depending on many factors, such as the distribution of the system, used hardware, the homogeneity and size of the cluster and more. The main focus of this section is to gather and systematize the knowledge about load balancing, in order to provide a clear background for issues addressed in this article.

2.1 Classifications of Load Balancing Algorithms

Load balancing algorithms can be categorized [10] based on four important features:

- physical location of cluster nodes,
- visibility in IP network,
- OSI model layer on which they operate,
- static or dynamic character.

Physical location of cluster nodes can be classified as local scale-out and global scale-out. Local scale-out is an installation where all the nodes reside in a separate area, close to each other and interconnected. If a web system is composed of geographically distributed servers, it is called a global scale-out setup. The distribution of servers is a key factor when designing a load balancing algorithm.

As far as visibility in IP network is concerned, two classes can be determined – web clusters and distributed web systems. If the whole system is visible under one virtual IP address, it is often called a web cluster or a cluster-based web system. This setup can be accomplished in several ways, usually by directing the incoming traffic to nodes via a front-end network device (switch or router), which often acts as a load balancer. In contrast, distributed web systems are structures where multiple nodes are visible to the clients under their distinct IP addresses. In these cases, mostly DNS servers with dedicated algorithms are employed to provide load balancing [15]. Intuitively, most web clusters are local scale-out setups, and many distributed web systems are built on global scale-out architectures, but this is not always the case.

The next categorization is based on the OSI model layer, on which the load balancing is executed. In case of web clusters, the network device which distributes load among cluster nodes commonly operates on layer 2, 3 or 4 [10]. It is not a rule, though, some algorithms include web switches that are aware of the application content (OSI layer 7) of incoming requests and consider it in decision process [11]. Layer 7 policies can likewise be based on a software dispatcher incorporated in a web server, which reroutes requests internally using the knowledge of request content and application logic.

Finally, load balancing algorithms can be categorized as static and dynamic. Static algorithms base on initial configuration that does not change in time, for example number of servers or their computing power. Round-Robin (RR) and Weighted Round-Robin (WRR) are good examples – they are widely used in web systems with satisfying results, thanks to being uncomplicated and not introducing considerable overheads. For these reasons they are also a good benchmark while testing more complex algorithms. Dynamic load balancing actively uses feedback from servers, for example current load or number of queued requests, to make request distribution decisions. Popular algorithms include shortest queue, least-connection and load-based approaches [11]. Although static algorithms are often sufficient for smaller, less complicated systems, they do not provide elasticity and are poorly fitted for systems with dynamic and database-driven workloads. It has been shown

in many publications that a well-designed dynamic algorithm can manifest better performance than RR or WRR in more complex systems. This is why dynamic algorithms are subject to research and there have been many attempts to create new, effective solutions.

2.2 Related Works

This section describes several examples of dynamic load balancing algorithms which can be found in literature. The first one is a load balancing algorithm presented in [12], intended for web clusters. It is based on an approximation model which helps estimate the utilization and capacity of web servers. In addition, feedback from servers is used to correct the estimation errors of the model. The knowledge gathered in this way is used to make load balancing decisions and the results are very appealing. However, the whole logic is enveloped in a complicated web switch and application servers have to be compatible with it.

Another algorithm for web clusters was presented in [13] and has also been proven to give better results than static algorithms. It utilizes a theoretical model based on Markov chain and a probability generating function. The analysis covers queue lengths and mean cyclic time for queries, and the authors show that the proposed model is realistic and applicable in load balancing. Nonetheless, it is not universal, as it assumes the environment to be a web cluster with a special hierarchical architecture.

Remarkable algorithms have also been developed in the field of distributed web systems. In such systems, load balancing is executed by DNS servers. Basic approach uses Round-Robin policy to distribute load among multiple servers, but often other algorithms are employed with better results than simple DNS RR. The authors of [14] proposed a solution where the DNS server (working in RR mode) does not require modifications, but it is dynamically updated with the list of available web servers. Based on the knowledge of current load of servers and a configurable threshold, the algorithm decides which servers are overloaded and should be temporarily removed from DNS server's list. Its advantage is compatibility with third party DNS server software, as the updates are part of DNS specification (see RFC 2136 [16]). However, the mechanism requires a load balancing module that gathers monitoring data from the servers and performs the updates. It might introduce substantial overheads and delays in reacting to current load fluctuations. What is more, this approach might be insufficient in systems with high incoming network traffic, because of DNS response caching in intermediate DNS servers and in client machines. Overloaded nodes would still receive requests until the expiry of a DNS reply.

Other algorithms include customizations of the DNS servers, for example in [15], where they are modified to direct clients to geographically closest web servers. The purpose is to reduce network impact on communication and it is achieved using a proximity algorithm. Moreover, if a web server becomes overloaded, it delegates some requests with a simple HTTP redirect. This should enhance the Quality of Service of the whole system, but it is not trivial to decide when and where such

redirects should be performed. If this issue is handled wrongly, the response times of the web system might increase. Moreover, if redirected requests reach a server that is overloaded too, it might cause another redirection, which leads to undesirable instability in periods of high network traffic.

To summarize, there are numerous load balancing algorithms documented, some are dedicated for certain systems and not each of them can be adapted to be used elsewhere. To the best of our knowledge, none of discussed policies could be employed with satisfying results in highly scalable, distributed systems with wide request type spectrum, such as `Onedata`. The reasons are usually connected with architectural assumptions of these solutions or the lack of features that are crucial in such systems.

3 LOAD BALANCING IN ONEDATA – REQUIREMENTS ANALYSIS

This section is intended to provide overview on the `Onedata` system, its use cases and main features. It outlines the challenges connected with designing an effective load balancing algorithm that would be compatible with its architecture and have desired qualities.

3.1 `Onedata` Overview

The main goal of `Onedata` is to provide unified and efficient access to data stored in globally distributed environments [8]. The need for such system was initially observed among users of big computing infrastructures [19], such as PL-Grid (The Polish Grid Infrastructure) [17] or EGI (The European Grid Infrastructure) [18]. Many scientists of various disciplines conduct data-intensive research, and sometimes they would like to simultaneously use infrastructures located in different data centres, or cooperate with research associates in other institutions. Currently, it is inconvenient as the users often have to manually manage and migrate their data between different data centres. What is more, various storage systems that they use have different interfaces and often require technical knowledge to use. `Onedata` removes these barriers by virtualizing globally distributed storage systems of different providers. In other words, it introduces a virtual file system that uses the collective resources of all storage systems to which a user has access and gives the user a unified view on his data. With data management and migration handled transparently to the users, it is an excellent tool for any time, any place data access. What distinguishes `Onedata` from other similar tools is its support for High Performance Computing (HPC), which is crucial from the perspective of computing infrastructures users. While `Onedata` was originally dedicated for big data centres and scientists, it might have potential in the commercial market too. It is possible because the today's everyday users are encouraged to store their data in clouds and would like to have a coherent and uniform view on their data.

3.1.1 System Architecture

From the global point of view, `Onedata` is composed of many cooperating deployments, one per every provider that decides to enter the `Onedata` system with its storage resources (see Figure 1). Such deployment is called `Oneprovider cluster` and is essentially a web system hosted on a cluster of nodes. The instances of `Oneprovider clusters` cooperate globally to unify user's view on his data. The significant feature of this cooperation is that the providers can retain their autonomy and do not have to trust each other. This is achieved with a mediator called *GlobalRegistry* and other innovative architectural solutions [8]. This paper describes a load balancing algorithm that is limited to the context of a single `Oneprovider cluster` instance.

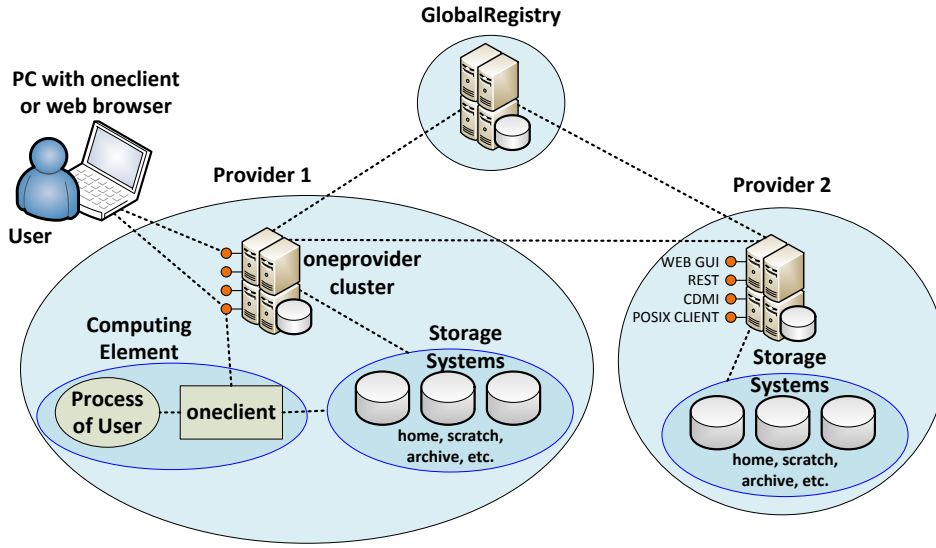


Figure 1. Exemplary environment with `Onedata`

`Oneprovider cluster` is a distributed application deployed on multiple nodes, each having its own, external IP address and each connected to underlying storage systems. The main responsibilities of `Oneprovider cluster` include file metadata management, data access coordination and, most importantly in the context of load balancing, processing of user requests. The application is written in Erlang, which is renowned for its powerful distribution mechanisms, as well as parallel processing capabilities. This choice of technology has proven advantageous in the light of scalability and high availability requirements of `Onedata`.

In reference to classifications presented before, the architecture of `Oneprovider cluster` falls in the category of distributed web systems, and is based on a local scale-out infrastructure. This is an unusual setup, and in connection with mech-

anisms implicitly offered by Erlang, it constitutes a unique case for load balancing.

3.1.2 Client and Request Types

There are several interfaces through which the `Onedata` system can be accessed. Firstly, a client application called `Oneclient`, which exposes a standard POSIX file system interface. Secondly, an intuitive web GUI that provides easy access from any place connected to the Internet. Finally, REST and CDMI (Cloud Data Management Interface) endpoints that are suited for third party service developers.

The `Oneclient` application is based on FUSE (Filesystem in Userspace). It allows for mounting a virtual file system in UNIX-based systems, so that a `Onedata` user can access his data exactly like on a local storage. `Oneclient` communicates with a `Oneprovider cluster` and handles low level file management transparently. This is a perfect solution for a typical user, but can also greatly facilitate computations and experiments performed by scientists in data centres. `Onedata` puts a great importance on support for HPC and the client application is fitted to handle data-intensive operations. If the host of `Oneclient` has direct access to storage systems used by `Oneprovider cluster`, their communication is limited to metadata only and `Oneclient` reads and writes the data directly from the storage systems. In other cases, e.g. when it is installed on a personal notebook, it works in a slower mode, as the `Oneprovider cluster` has to mediate in all file operations. Hence, the client applications can generate various types of requests. In both cases (direct and remote file access) it performs numerous, lightweight metadata requests, such as retrieving physical location of a file. However, a remote `Oneclient` (e.g. installed on a PC) has to send and receive a lot of data through the network connection during I/O operations, which consist of multiple read and write requests. It is also possible for a directly connected `Oneclient` to require data transfers on the wire, for instance when some files are located on the storage of another provider.

The web GUI was designed to provide access to data without installation of any software and ensure a user-friendly experience. It features an intuitive and responsive file manager with data sharing and publishing capabilities, among others. The communication with `Oneprovider cluster` is based on secure HTTPS and websocket protocols, thus some connections can be kept alive for a long time. Noticeably, all the operations must be performed via `Oneprovider cluster`, including those that cause high network interface usage by performing file uploads and downloads. On the other hand, some requests might be lightweight (e.g. renaming a file), or use a lot of computing resources (e.g. removing large directories).

Third party service developers can use the REST and CDMI APIs to integrate with `Onedata`. They both rely on stateless, secure HTTPS connections. The REST endpoints offer most file operations in a robust, concise and well documented API based on interoperable JSON. CDMI complaint interface, that allows both low and high level operations on user data, constitutes a more complicated yet usable and powerful alternative. The great advantage of CDMI is introduction of file opera-

tions that are not available in `Oneclient`, for example issuing copying of large files and directories, often used in pre-staging of content before data-intensive computations. As far as REST and CDMI are concerned, the processing time and volume of requests may vary dramatically depending on the type of operations performed. In addition, some requests do not carry much information on the wire, but might consume considerable amounts of server's processing power or memory (e.g. copying large files). Those interfaces might contribute considerably to `Oneprovider cluster` load, especially when it is deployed on a Grid infrastructure and integrated with middleware or job schedulers.

Noticeably, some of `Oneprovider cluster` load results from communication with *GlobalRegistry* and other `Oneprovider clusters`. It is based on REST interfaces and essentially the performed requests have the same characteristics as described above. Hence, this communication is not perceived as a separate class of requests.

	Oneclient direct i/o	Oneclient remote	web GUI	REST & CDMI
light	++	+	+	+
small transfer	+	++	+	+
large transfer	–	–	++	++
computationally requiring	–	–	+	++

Table 1. Request types in `Onedata`

Considering the impact of requests on resource usage, we classified them into four categories: *light*, *small transfer*, *large transfer* and *computationally requiring*. *Light* requests have high priority and should be processed with no delays, i.e. metadata requests. They do not carry much information on the wire nor use much computational resources. *Small transfer* requests carry minor amounts of data back and/or forth, for instance read/write operations generated by `Oneclient` working in remote mode. Requests classified as *large transfer* carry big data payloads that can cause high usage of network interfaces. Finally, *computationally requiring* requests consume significant amounts of computing power to be processed. Table 1 contains a summary of request categories in `Onedata`. The number of '+' signs indicates how often such requests appear for different interfaces.

The variety of requests is significant in the context of load balancing, as different requests have different impact on load of cluster nodes, both network and computational (CPU, memory). It is also important that in the `Oneclient` application we have full control over server preference when connecting to `Oneprovider cluster`. However, in case of other interfaces (all HTTP based), the only way to control the choice of servers by the clients is to use a DNS server. Even then, we still cannot influence the way the DNS resolvers and web browsers work. For those reasons, `Onedata` is a great example of a distributed web system that cannot fully use all of its design advantages without a well suited load balancing algorithm.

3.1.3 `Onedata` Use Cases

To better explain the influence of different users and requests on a `Oneprovider cluster` in the context of load balancing, some concrete use cases of different interfaces are presented below.

As far as web GUI is concerned, one of the key functionalities is upload and download of files. This relates both to private content and shared files that can be downloaded by anyone possessing a valid URL. Such operations cause a substantial load on network interfaces and generate I/O operations on the server.

Considering `Oneclient`, when a data center user is performing data-intensive computations on a directly connected storage, hundreds of light metadata requests are generated and they should be handled quickly so as not to limit the `Oneclient` performance.

Further on, a researcher might want to use a copy of a huge file with gigabytes of data as an input to his simulation. He would then use the CDMI interface, most probably indirectly by middleware that can pre-stage his data. Such request requires lots of CPU time, RAM and I/O operations to be processed.

3.2 Load Balancing Scenarios

Bearing in mind the above-mentioned characteristics of `Onedata`, we identified three representative scenarios that depict requirements of load balancing for `Oneprovider cluster`:

- sharp load fluctuations,
- unbalanced use of resources,
- node failure.

The first scenario includes a set of clients (of any type) connected to a node of `Oneprovider cluster` (*node A*). Assume that all other nodes maintain similar number of connections at the time, and the cluster load is balanced. However, it is possible that suddenly some of the clients of *node A* greatly increase the intensity of their requests, as, for instance, a large grid job has started. *Node A* becomes overloaded and there is no possibility of redirecting the clients to other nodes without breaking the connections. A mechanism is required that would allow for internal rerouting of requests to nodes which have more free resources.

Another example shows why the payloads and processing time of requests should be taken into account. Consider two nodes (*A* and *B*), which hold a comparable number of connections. However, *node A* has received a lot of file upload/download requests from web clients and its interfaces are practically exhausted. Nevertheless, it still has reserves of computing power. On the other hand, *node B* is busy with processing multiple, computationally requiring CDMI requests which causes full CPU utilization, while its network interfaces stand practically unused. The cluster should direct new connections to *node B*, but at the same time internally delegate

some requests from *node B* to *node A* to achieve optimal utilization of resources on both nodes.

The next case regards a cluster of nodes and a situation where one of the nodes (*node A*) becomes temporarily nonoperational. The reason might be either a network malfunction, software failure or overload. While it is not possible to amend existing connections of *node A*, the system should be able to stop new clients from connecting to it and direct them to healthy nodes.

All the aforementioned scenarios are possible, and while such cases might not occur often, the system must be able to react rationally in any circumstances. Beside the specific scenarios where a load balancing algorithm is indispensable, there are also general requirements that it must fulfil, as described below.

3.3 Load Balancing Challenges

Firstly, the main reason of introducing any load balancing algorithm is achieving scalability. Basically, it means ability to increase system's performance by extending the cluster with new nodes. Naturally, the bigger the cluster, the less improvement will be introduced with new extensions, as the overheads of managing such an infrastructure and maintaining communication between the nodes become too substantial. The desired load balancing algorithm should allow for building clusters big enough to handle expected number of clients.

Other crucial features for distributed systems are High Availability (HA) and Quality of Service (QoS). The former means that a service should remain fully operational despite failures of some nodes or components. The latter, from the user's point of view, is how efficient, responsive and failure-free the service is. Load balancing algorithm can have a great influence on both of these aspects if it can instantly react to undesired situations and minimize traffic on overloaded nodes.

An optimal load balancing solution should incorporate mechanisms that would allow for maintaining comparable resource utilization on all nodes of a cluster, which includes computational resources as well as network interfaces. It is especially important that none of the nodes becomes significantly more loaded than the others. Another feature that is somehow connected is avoiding bottlenecks. When this is neglected, it might ruin the system's scalability and the potential of parallel processing.

Manageability is also a relevant aspect. The system administrators should be able to easily modify the size of a cluster or migrate some applications between servers, while the load balancing algorithm adapts dynamically to the new circumstances.

Last, but not least, the distributed architecture of target systems must be considered when designing a load balancing algorithm. It must be well integrated, but should also exploit the benefits of distribution.

4 TWO-LEVEL LOAD BALANCING

Considering all the aforementioned requirements, an original load balancing algorithm, called Two-Level Load Balancing (TLLB), has been designed and evaluated. The two levels refer to DNS servers and internal, application layer (OSI 7) dispatchers. A sequence diagram for reference is presented in Figure 2. It presents the complete flow of requests, starting from domain resolving and ending with a HTTPS reply from a server, with symbols indicating where each level of load balancing is applied. The whole algorithm is discussed in detail later on.

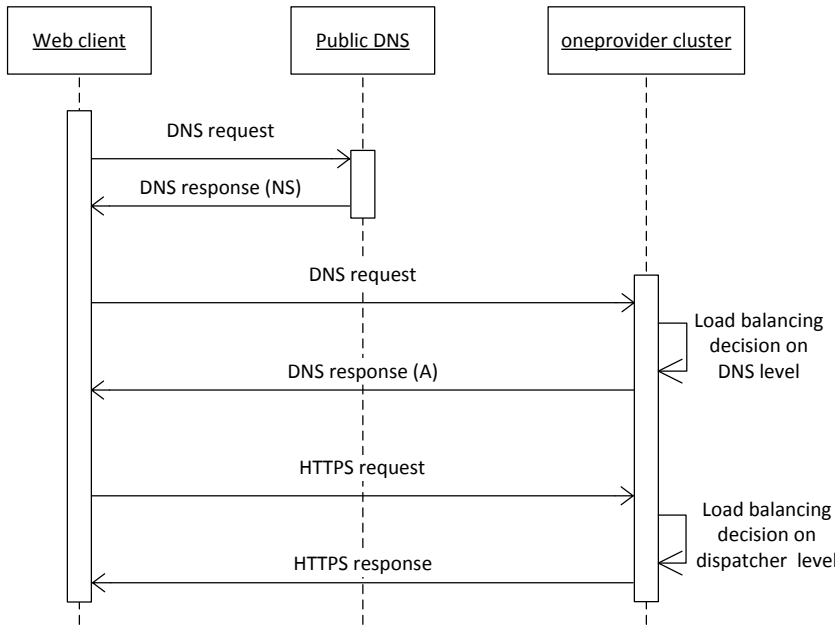


Figure 2. Two-level load balancing – sequence diagram

A significant aspect of the presented approach is that both levels are handled by software that is fully integrated in the cluster. Every node has its own external IP address and contains a DNS server and a dispatcher module. Both modules operate as part of the `Oneprovider cluster` application. It means that they can communicate with other components of the system, one of which is Central Cluster Manager (CCM). One of its responsibilities is collecting monitoring data from `Oneprovider cluster` nodes, which contains current load (CPU, memory) and network interfaces usage. This data can be used for load balancing decisions on both levels. Obviously, gathering monitoring data and processing it introduce overheads, and this fact is usually considered an important factor when comparing a dynamic algorithm to static algorithms like RR. It often happens that complicated supervision methods

and decision algorithms slow down the whole system so much that there is no point in using them. However, the monitoring in `Oneprovider cluster` was designed for advanced diagnostic and administrative tools. It is essential anyway, thus it can be used in load balancing with practically no additional cost. The algorithms are uncomplicated, and the data is processed by the CCM and served to DNS servers and dispatchers in a form of ready-to-use load balancing instructions. They are updated periodically in short intervals so that the cluster can quickly react to load fluctuations.

4.1 First Level – DNS Server

By default, all nodes IP addresses appear in the DNS response. High Availability (HA) is achieved by including multiple addresses in responses and the ability to temporarily exclude nodes that are nonoperational or unreachable. The nodes with lower load have proportionally greater probability to be placed at the top of the list (see Figure 3). While this cannot ensure that they will be preferred, most web browsers and operating system level resolvers will be more eager to choose addresses that come first. Hence, this algorithm produces the desired effects. Currently, there is no method to impose the record choice priorities on clients of `Onedata`, which are all based on HTTP protocol. The SRV DNS records could be an answer to this, but the HTTP protocol does not assume its use and popular web browsers do not support it. The load of a node, L, is calculated using a weighted average, as in Equation (1).

$$L = \frac{\alpha * net\_load + \beta * cpu\_load + \gamma * mem\_load}{\alpha + \beta + \gamma} \tag{1}$$

where *net_load* is network interfaces load, *cpu_load* is the CPU usage and *mem_load* is the memory usage. They are expressed in per cents and so is the resulting load. The network usage ratio depends on maximum interface throughput. The values of α, β and γ are determined experimentally. If α coefficient is dominating, the DNS server will be eager to lighten the network traffic to nodes with heavily utilized interfaces, which is a desired feature. Nonetheless, computational load must be also taken into consideration so that more clients can be directed to nodes with free resources. In Figure 3, an example DNS instructions has been shown, where one of the nodes has been temporarily removed because of a failure or connection problem. The sequence in exemplary response has been randomized as described before.

4.2 Second Level – Dispatcher

The dispatcher module has been introduced in order to increase control, refine the load balancing algorithm and handle edge cases. The instructions for dispatchers are created based on computational load of a node, according to Equation (2).

$$L = \frac{\beta * cpu\_load + \gamma * mem\_load}{\beta + \gamma} \tag{2}$$

where *cpu_load* is the CPU usage and *mem_load* is the memory usage. The β and γ coefficient values are selected experimentally. The network traffic is omitted, as the dispatcher does not influence the external interfaces usage, regardless of its decisions. In natural circumstances, when the nodes of `Oneprovider cluster` are similarly loaded, the dispatcher has a very low impact on request processing time as it simply follows the incoming requests to handler modules residing on the same node. However, it is crucial in cases when the load is fluctuating sharply. If the node which received the request is significantly more loaded than the others, the dispatcher will delegate such request to another one. This is determined using a threshold coefficient (Equation (3)).

$$\frac{L}{L_{min}} > \rho => overloaded. \tag{3}$$

The ρ coefficient value is selected experimentally. The load of each node – L – is compared to the lowest load in the cluster – L_{min}, and if the ratio exceeds the ρ threshold, the node is considered *overloaded* and the generated instructions for dispatchers will strive to correct that situation. They include information how often and to which nodes should requests be delegated (see Figure 3). The target node for delegation is chosen in weighted random manner, where the least loaded nodes are most probable to be chosen. The randomization approach introduces very low overheads while giving satisfying results, and allows for parallelization of request redirecting as the load balancing instructions are processed in read-only mode. Naturally, the request rerouting itself increases its processing time. However, given that the `Oneprovider cluster` nodes are interconnected with high performance interfaces and communication mechanisms in Erlang are greatly optimized, it is still beneficial to delegate the request as it will be processed faster than on a heavily loaded node. This is especially true for requests that consume a lot of resources to be processed.

4.3 Request Flow in TLLB

Figure 4 presents a `Oneprovider cluster` composed of two nodes, outlines all modules and entities that take part in load balancing process and shows the requests flow during web GUI usage.

As mentioned before, the CCM module prepares instructions for DNS and dispatcher modules based on monitoring data and propagates them periodically (step (0) in Figure 4). When a user wants to use the web GUI, firstly his browser has to resolve the `Oneprovider cluster` domain into an IP address. The domain must be public and recognizable by global DNS servers. The client performs a DNS request to a public DNS server, and receives a response indicating where it should

a) DNS instructions

	IP	probability
Node A	200.0.0.1	0.26
Node B	200.0.0.2	0.31
Node C	200.0.0.3	0.43
Node D	---	---

Exemplary DNS response:

```
200.0.0.3
200.0.0.1
200.0.0.2
```

b) dispatcher instructions

Node OK

	probability
Node A	1.0

All requests processed locally

Node overloaded

	probability
Node A	0.20
Node B	0.32
Node C	0.48

80 % of requests delegated to other nodes

Figure 3. a) DNS and b) dispatcher instructions

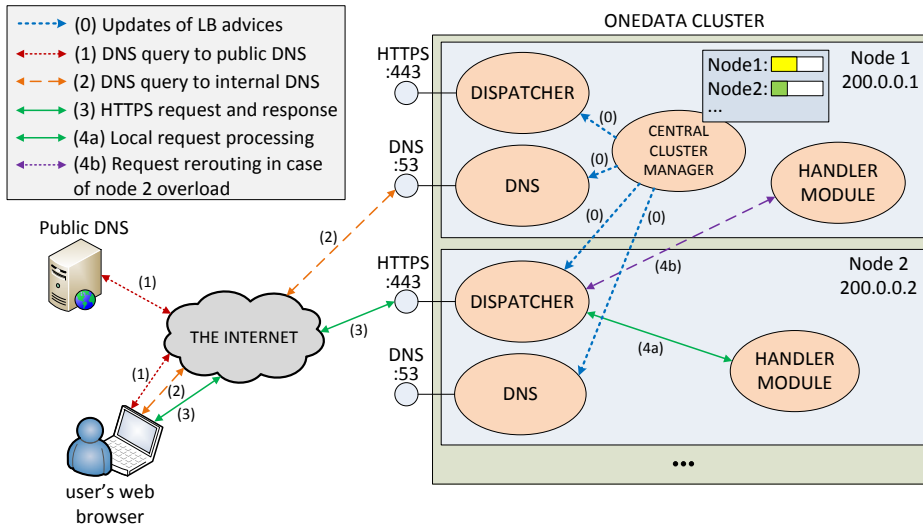


Figure 4. Request flow in Two-Level Load Balancing

ask again (step (1)). The response list contains hostnames of all the `Oneprovider cluster` nodes. The client continues to resolve the domain by asking one of the internal DNS servers (step (2)). The queried server returns a list of IP addresses of cluster nodes (sorted in a manner mentioned before), and one of the addresses is finally chosen. The client performs a HTTPS request, which reaches the dispatcher (step (3)). If the target node is not highly loaded, the request is processed locally (step (4a)) and the response is returned to the client. However, in case of an overload, delegation is performed and another, less loaded node evaluates the request (step (4b)). Eventually, the client receives the response, which has been processed

unnoticeably longer. It is important that all nodes in the cluster are able to handle any request, so the delegation algorithm is uncomplicated.

4.4 Synergy of the Two Levels

Ultimately, it should be justified why both load balancing levels shall be used instead of just one. To start with, DNS servers are indispensable in distributed web systems, i.e., when the system is reachable under multiple IP addresses. This issue could be settled by using a static, third party DNS server that uses Round-Robin shuffling. However, the proposed solution gives great elasticity and ability to quickly react to changes in the cluster structure and possible node failures. Moreover, by having access to the system status, the load balancing can be finer and better distribute incoming connections. In addition, the integration of DNS servers increases maintainability of the whole system. Secondly, the presence of dispatchers is necessary, as DNS load balancing has too high inertia. It means that it cannot responsively control the flow of requests, and it is mostly because of its responses Time To Live (TTL). For `Oneprovider cluster` it is set to a very low value of 60 seconds, but during this time the cluster status might change dramatically while the clients will still be using cached DNS responses. Hence, dispatchers are indispensable for finer and more responsive load balancing. Arguably, the two levels of load balancing create a synergy, which utilizes their best features.

5 TEST RESULTS

To evaluate the TLLB algorithm, a test environment has been set up. It was composed of multiple, homogeneous virtual nodes with enabled network emulation. `Oneprovider cluster` instances of different sizes were deployed on some nodes, while other served as clients that generated requests. The conducted tests included scalability tests based on throughput measurements. In addition, a test scenario which emphasises the two-level synergy has been evaluated. The obtained results have been normalized for more convenient analysis.

The clients' behaviour was simulated, using requests of various types and sizes – and the configurations were constant in the scope of each test. Each test was repeated multiple times and the outcomes were averaged. The repeatability of test results was high.

The aim of these tests was to assess the behaviour of the TLLB algorithm in a virtual environment, identical with target physical environment that the system could be deployed on. The tests were designed to ensure assumed features of the TLLB algorithm, such as the ability to maintain efficient system scaling or delegate requests on dispatcher level. This way, the algorithm can be safely introduced in production environment, where further testing and tuning will be performed.

The purpose of the first test was to examine the scalability of `Oneprovider cluster` depending on the use of different load balancing levels. This way, four combinations were obtained:

1. none,
2. dns,
3. disp,
4. dns_disp (see Figure 5).

Combination 1. included a DNS server working in RR mode and disabled dispatcher, i.e. all requests were processed on the target node. It served as a reference measurement as the simplest, static solution with none of proposed load balancing algorithms enabled. The next combinations were: 2. enabled DNS level and disabled dispatcher, 3. disabled DNS (RR mode) and enabled dispatcher and 4. Two-Level Load Balancing. The results are presented in Figure 5.

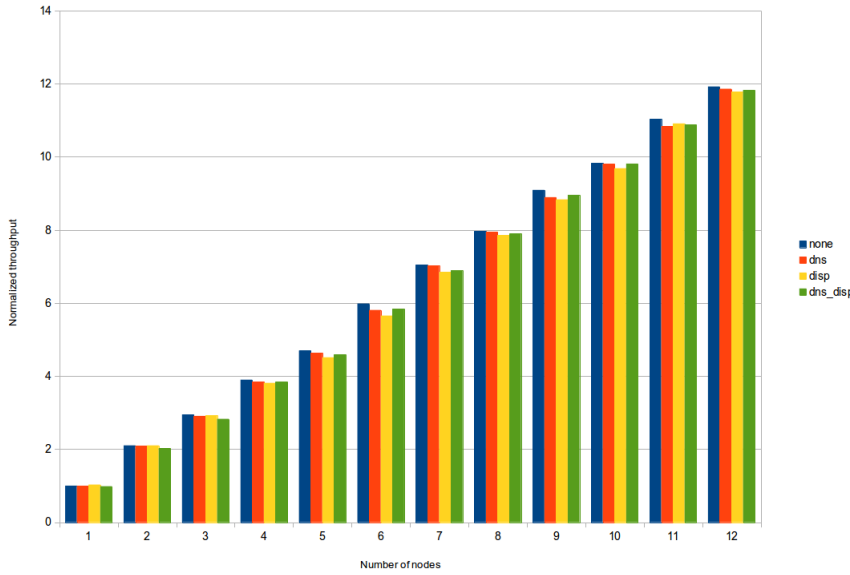


Figure 5. Scalability – throughput (normalized)

The results show that none of the proposed load balancing levels introduce significant overheads as their performance is comparable to static algorithms. In this case, an improvement over static algorithms was not anticipated because all the nodes were similarly loaded during the tests. Nevertheless, the test results justify the use of dedicated DNS servers in `Oneprovider cluster`. They achieve similar performance as standard RR algorithms, but also ensure elasticity and HA of the system by the ability to temporary exclude nonoperational nodes.

For the next test, a specific scenario was designed to evaluate the dispatcher efficiency. Only half of the nodes were receiving requests to verify if the dispatchers can cope with such situation. For reference, the tests were repeated with dispatcher

load balancing turned off, which is indicated by 'none' data series. The 'dispatcher' data series includes results when dispatchers were enabled. The results are depicted in Figure 6.

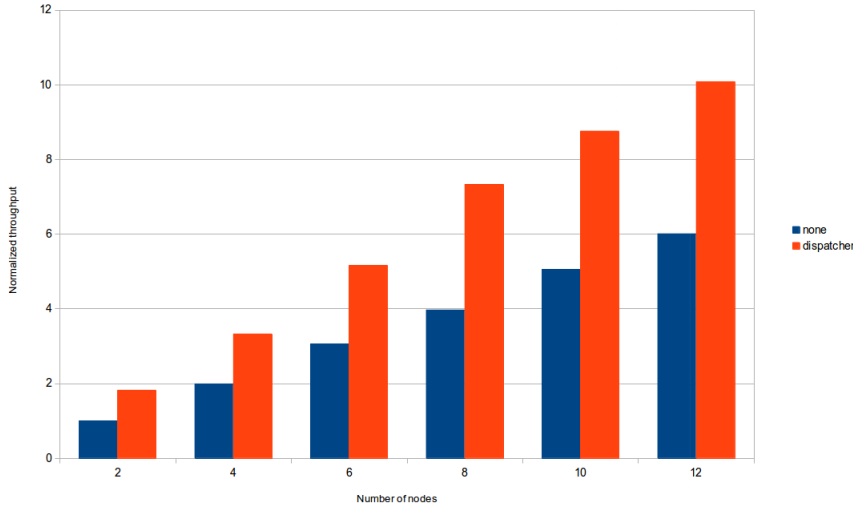


Figure 6. Dispatcher performance with uneven node loads – throughput (normalized)

Obtained results show that dispatchers were able to use free resources on the nodes that did not receive requests. The overall system throughput was nearly two times greater compared to the case when dispatchers were disabled – as requests were rerouted internally to the free nodes, they could be processed more quickly. It was not possible to double the throughput as the requests rerouting introduces overheads on the processing time, but the results prove the value of dispatchers. Their use ensures resistance to load fluctuations and efficient use of collective resources of the whole `Oneprovider cluster`.

6 CONCLUSIONS AND FUTURE WORK

The complexity and size of distributed web systems increases and so does the demand for effective load balancing algorithms. Their requirements have been identified using the case of `Onedata`, considering the various use cases of the system and diversity of requests that it processes. Finally, an innovative solution has been designed, called Two-Level Load Balancing. The first level refers to DNS servers, and the second to dispatchers that operate on application level and are able to reroute requests inside a cluster of nodes. Both levels are encapsulated in the `Oneprovider cluster` application, which allows for low-cost use of monitoring data. The DNS servers distribute load among network interfaces of the cluster in a balanced manner, while dispatchers correct load fluctuations.

The proposed Two-Level Load Balancing has been implemented and evaluated. It has proven to be a suitable choice for the `Onedata` system. The obtained results show that a `Oneprovider cluster` instance can benefit from the use of TLLB in cases of unbalanced or fluctuating loads, while during smoother periods it does not impair the performance of the system. One of the reasons the overheads are negligible is the simplicity of the algorithms used on both levels. While the two level approach is not novel, the proposed algorithm features a highly distributed architecture based on Erlang language, where load balancing modules are fully integrated with the system. This way, it becomes a unique approach. What is more, it has the potential to be used in other scalable web systems, not only in the field of global data access.

The proposed algorithm still has the potential for improvements. It has been tested in environment with simulated clients and yielded satisfying results. However, further testing and tuning should be performed in production environment. What is more, there are techniques that could potentially improve its performance. They include probabilistic load prediction based on history of requests, admission control or content awareness. Importantly, their introduction must be careful and well thought out, as it might cause overheads that could impair the system's performance.

REFERENCES

[1] Martin, A.: OneDrive vs. Google Drive vs. Dropbox: The Best Cloud Storage Service of 2017. Online. Accessed 13.09.2017. `http://www.alphr.com/dropbox/7034/onedrive-vs-google-drive-vs-dropbox-the-best-cloud-storage-service-of-2017`.

[2] Amazon Web Services (AWS) for HPC: Online. Accessed 13.09.2017. `http://aws.amazon.com/hpc/`.

[3] IBM HPC Cloud: Online. Accessed 13.09.2017. `https://ibm.com/systems/spectrum-computing/solutions/hpccloud.html`.

[4] facebook: Online. Accessed 13.09.2017. `https://facebook.com`.

[5] twitter: Online. Accessed 13.09.2017. `https://twitter.com`.

[6] gmail: Online. Accessed 13.09.2017. `https://gmail.com`.

[7] onedata: Online. Accessed 13.09.2017. `https://onedata.org`.

[8] Dutka, Ł.—Wrzeszcz, M.—Lichoń, T.—Słota, R.—Zemek, K.—Trzepla, K.—Opioła, Ł.—Słota, R.—Kitowski, J.: Onedata – A Step Forward Towards Globalization of Data Access for Computing Infrastructures. Procedia Computer Science, Vol. 51, 2015, pp. 2843–2847, doi: 10.1016/j.procs.2015.05.445.

[9] Bubak, M.—Kitowski, J.—Wiatr, K. (Eds.): eScience on Distributed Computing Infrastructure. Springer, Lecture Notes in Computer Science, Vol. 8500, 2014. ISBN 978-3-319-10893-3, doi: 10.1007/978-3-319-10894-0.

[10] Gilly, K.—Juiz, C.—Puigjaner, R.: An Up-to-Date Survey in Web Load Balancing. World Wide Web, Vol. 14, 2011, No. 2, pp. 105–131.

[11] Tiwari, A.—Kanungo, P.: Dynamic Load Balancing Algorithm for Scalable Heterogeneous Web Server Cluster with Content Awareness. Trendz in Information Sciences Computing (TISC), 2010, pp. 143–148, doi: 10.1109/TISC.2010.5714626.

[12] Sharifian, S.—Motamedi, S. A.—Akbari, M. K.: An Approximation-Based Load-Balancing Algorithm with Admission Control for Cluster Web Servers with Dynamic Workloads. The Journal of Supercomputing, Vol. 53, 2010, No. 3, pp. 440–463, doi: 10.1007/s11227-009-0303-8.

[13] Bao, L.—Zhao, D.—Zhao, Y.: A Dynamic Dispatcher-Based Scheduling Algorithm on Load Balancing for Web Server Cluster. Web Information Systems and Mining (WISM 2010). Springer, Lecture Notes in Computer Science, Vol. 6318, 2010, pp. 95–102, doi: 10.1007/978-3-642-16515-3_13.

[14] Moon, J.-B.—Kim, M.-H.: Dynamic Load Balancing Method Based on DNS for Distributed Web Systems. E-Commerce and Web Technologies (EC-Web 2005). Springer, Lecture Notes in Computer Science, Vol. 3590, 2005, pp. 238–247, doi: 10.1007/11545163_24.

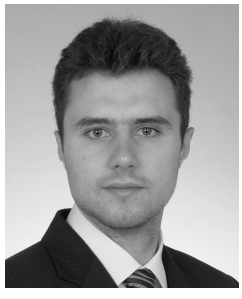
[15] Cardellini, V.—Colajanni, M.—Yu, P. S.: Geographic Load Balancing for Scalable Distributed Web Systems. Proceedings of the 8th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, 2000, pp. 20–27, doi: 10.1109/MASCOT.2000.876425.

[16] RFC 2136: Online. Accessed 13.09.2017. https://ietf.org/rfc/rfc2136.txt.

[17] PL-Grid Infrastructure: Online. Accessed 13.09.2017. http://plgrid.pl/en.

[18] European Grid Infrastructure: Online. Accessed 13.09.2017. https://egi.eu/.

[19] Słota, R.—Dutka, Ł.—Wrzeszcz, M.—Kryza, B.—Nikolow, D.—Król, D.—Kitowski, J.: Storage Management Systems for Organizationally Distributed Environments – PLGrid PLUS Case Study. Parallel Processing and Applied Mathematics (PPAM 2013). Springer, Lecture Notes in Computer Science, Vol. 8384, 2014, pp. 724–733, doi: 10.1007/978-3-642-55224-3_68.



Łukasz Opioła received his Master's degree in computer science from the University of Science and Technology (AGH), Cracow, Poland in 2015. He is currently a Ph.D. student at the Faculty of Computer Science, Electronics and Telecommunication at AGH and an employee of the Academic Computer Centre CYFRONET-AGH. His research areas include data synchronization in distributed systems, as well as authentication and authorization infrastructures.

Łukasz Dutka has significant expertise in cloud systems, large-scale systems, development of application for business purposes, team and project management in commercial projects as well as EU IST projects. He received his Ph.D. in computer science from the AGH University of Science and Technology, Cracow, Poland. He has a longstanding experience with managing large development teams. His scientific interests include large-scale computer system, system architectures, component approaches. He is the author of a modern software development architecture called the Component-Expert Architecture combining expert systems with component architectures, with successful applications in commercial and scientific environments. He has actively participated in a number of EU funded projects including Indigo DataCloud, EGI Engage, Helix Nebula Science Cloud and many others. Currently he is the Technical Director of PL-GRID Plus project and leader of Onedata team.

Michał Wrzeszcz is a Ph.D. student of computer science at the AGH University of Science and Technology in Cracow, Poland and an employee of the Academic Computer Centre CYFRONET-AGH. He is the author or co-author of over 20 scientific papers and conference contributions. His research interests are transparent data access and distributed computing as well as artificial intelligence and social networks.

Renata Slota Ph.D., D.Sc., works at the Department of Computer Science of the AGH University of Science and Technology (AGH) in Cracow, Poland. She is the author or co-author of about 130 scientific papers. Topics of interest include parallel and distributed computing, distributed systems, grid and cloud environments, data management and storage systems, knowledge engineering. She has been involved in many national (recently: PL-Grid Core, PL-Grid NG) and international projects most notably in EU IST, recently: PaaSage, and VirtRoll. Among others, she worked on the development of Onedata and Scalarm systems. Member of the Program Committee of the International Conference on Computational Science (ICCS), and the International Conference on Parallel Processing and Applied Mathematics (PPAM). Reviewer of: Future Generation Computer Systems (FGCS), Computing and Informatics (CAI), and Computer Science (CSCI) journals. Currently, she is the Deputy Dean of the Faculty of Computer Science, Electronics and Telecommunication of AGH.

Jacek Kitowski is Full Professor of computer science and Head of Computer Systems Group at the Department of Computer Science of the AGH University of Science and Technology in Cracow, Poland and Head International Affairs at the Academic Computer Centre CYFRONET-AGH, responsible for international collaboration and for developing high-performance systems and grid/cloud environments. He is author or co-author of about 240 scientific papers. His topics of interest include large-scale computations, Grid services and Cloud computing, distributed storage systems, high availability systems, network computing, knowledge engineering. He is a member of program committees of many conferences, participant of many international and national projects, funded by the European Commission, European Defense Agency, Polish National Centre for Research and Development and Polish National Science Centre. Director of PLGrid Consortium running PLGrid e-infrastructure in Poland for scientific computing. Member of Ministry Expert Body for Scientific Investments.