Aspect-oriented management of service requests for assurance of high performance and dependability

Paweł Lubomski[1], Paweł Pszczoliński[2] and Henryk Krawczyk[3]

[1] IT Services Centre, Gdańsk University of Technology, Gdańsk, Poland
lubomski@pg.gda.pl
[2] IT Services Centre, Gdańsk University of Technology, Gdańsk, Poland
pszczola@pg.gda.pl
[3] Faculty of Electronics, Telecommunications and Informatics,
Gdańsk University of Technology, Gdańsk, Poland
hkrawk@eti.pg.gda.pl

Abstract. A new approach to service requests management in case of insufficient hardware resources is proposed. It is based on wide aspects of requests analysis and it assures reliable and fast access to priority services. Requests are analyzed for, among others, time of occurrence, category of user who made the request, type of service, current system load and hardware utilization. Deterministic but dynamic rules help to manage system load very effectively, especially in terms of dependability and reliability. The proposed solution was tested on Gdańsk University of Technology central system, followed by the discussion of the results.

Keywords: service requests management, aspect, dependability, reliability, load balancing.

1 Introduction

Ensuring efficient and reliable access to IT services is a serious challenge. It becomes even more difficult when requests for different services fluctuate over the time. That is the reason why cloud computing is used so commonly. It allows to scale hardware resources in a very quick, relatively cheap, and efficient way. But what happens, when hardware resources are limited? Depending on organization where the system is used, load can vary in a daily, weekly, monthly or even yearly cycles. A university is a very good example at this point. Administrative staff and university lecturers use their university system during working hours. Students, on the other hand, prefer evening sessions. In addition, depending on the moment of academic year, also the need for various IT services is different. For example, reading lecture plans is heavily used at the beginning of the semester, but after a few days the number of requests for this service drops drastically. Moreover, it is possible to determine the type of each service – whether reading or writing the data dominates. An example of a university shows that the allocation of resources for specific IT services should not be constant over time but dynamically adjusted. The problem to solve is to develop such

a method of service requests management which in case of insufficient hardware resources will provide reliable and fast access to priority services. Of course, you can try to resolve this problem using different types of load balancing but it does not give you the possibility to prioritize requests, especially when hardware resources are limited. As a result of a simple load balancing, we can achieve the consumption of all available resources and, as a result, the denial of service very quickly. The aim of an effective solution is to use multiple nodes and to handle service requests dynamically depending on the aspect in which they are made. In this approach the critical services will be available regardless of the load at all times.

2 Motivation and related work

The problem of proper and effective load balancing has been under intensive research for years. Load balancing algorithms can be divided into two categories. There are static algorithms commonly used such as Round Robin and Weighted Round Robin. On the other hand, last connection as well as weighted last connection are dynamic algorithms commonly used. There are also works on other issues of cloud or distributed computing, e.g. cost-optimal scheduling on clouds [1], load balancing for distributed multi-agent computing [2], agent-based load balancing in cloud [3], communication-aware load balancing for parallel applications on clusters [4]. Also, there are approaches that apply genetic algorithms to dynamic load balancing [5]. Latest approaches focus on dynamic type-related task grouping on the same nodes [6] or weighted last connection algorithms with forecasting mechanisms [7]. There is also some research related to performance overhead while using virtualization in cloud [8]. Some other research was done focusing on type of communication used in resource allocation inside cluster and on load balancing – blocking or non-blocking connection, especially in the message-oriented model [9]. It is very important when clusters and data are located in different data centres spread all over the world. That impacts on availability very much.

Our aim is to focus not only on availability but also on dependability and reliability of services, where proper prioritization based on wide aspect of requests analysis takes place. It is very important when there are not enough hardware resources to process all requests at a time. This work is the continuation of our research on context analysis for better security and dependability of distributed internet systems [10].

3 Proposed solution

In contrast to a simple load balancing, management of requests based on an aspect of their calls does not allocate resources evenly. Aspect-oriented management of requests means that during the request realization there is not only services invocation but also an additional functionality. This additional functionality includes reading the request attributes and then, on the basis of them as well as the configured rules, deciding which service node should be involved. This additional action is entirely separated

from the main request realization. All requests should be analyzed this way. This is a perfect example of aspect-oriented programming. By using information about who, from where, and when invoke request to the service appears we are able to allocate resources (often very limited) to meet changing demands and at the same time provide the resources needed for critical services dependability. Figure 1. illustrates schematically the definition of the problem and its solution.

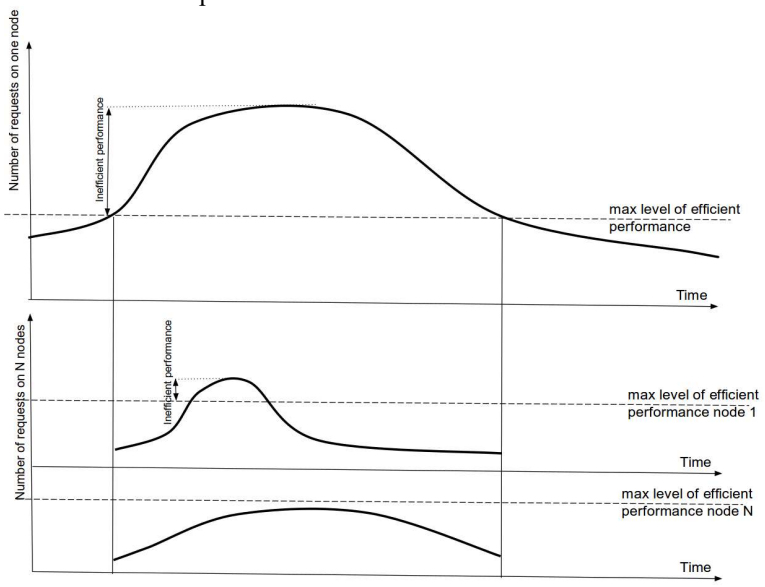

Fig. 1. Management of service requests in terms of performance improvement with limited hardware resources.

System work on a single node during the biggest load is inefficient very often and many requests are serviced in sub-optimal time. Detailed statistics from our case are presented in section 5. Adding more nodes with appropriate management of requests, although they did not allow all services to maintain optimal execution time (node 1), allows the selected priority services to work optimally and without any delay (node N) during the increased load time period. If you have resources to ensure optimal performance of all services during the biggest load, a simple load balancing is good enough to ensure efficient performance. Dynamic management of requests on the basis of modifiable configuration is a necessary solution when it is not possible to ensure sufficient resources for all services and while priority services must be provided with high performance and reliability during the whole period of increased load. In single node configuration during the peak load snow ball effect appears, which means that more and more services response in elongate time because they are waiting for access to resources. In such situation whole system work with poor performance. When we can divide load between many nodes and decide that on some of them only priority services will work those services will not wait for resources. Even in peak load snowball effect will not appear on nodes where demand for resources is less then available.

Of course, the proper system architecture is important to support aspect-oriented management of requests. A perfect solution is to introduce an explicit separation between the user interface layer and the services layer. It is assumed that the user interface layer (web browser with running applications on the client side and the network connection between the browser and the server) does not bring noticeable performance drop during the request handling. Comparing with the time of the request handling in the services layer, the time of transmission and request handling in UI layer can be omitted. The discussed model of system architecture is presented at Figure 2. This architecture is used in the increasingly popular microservices [11].

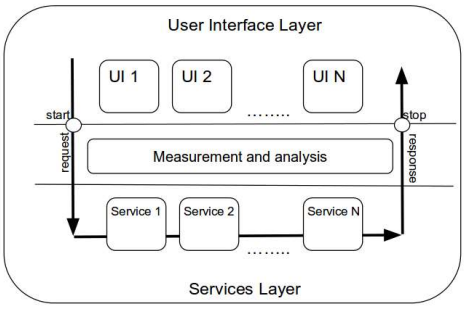

Fig. 2. Model of system architecture with request analysis between UI layer and services layer.

Building a system that uses architecture of independent services and applications assures flexibility. The problem of providing communication between services and UI application requires an additional usage of REST and JSON protocol at a small effort. The explicit separation of the UI layer from the business services layer provides a number of advantages:

- less impact of errors for overall work of the system - by separating the requests, a part of the system can be in error state while the remaining part is able to work properly,
- scalability – you can easily deploy services that require more resources on machines with greater computing power,
- the ability to customize system architecture to the realities of the organization - this is important when the organization is large and has a complicated structure. It also allows you to deploy selected applications and services for specific groups of users to ensure the improvement of access control,
- the possibility of a partial modernization of the system – this is especially important for large systems where frequent exchange of technology is impossible. With independent services it is easier to upgrade some parts of the system. It is necessary to ensure compatibility at the API level only.

As business services have only programming API and they do not have a user interface, developers are forced to write unit tests because otherwise they are unable to test what they have produced. The key to the implementation of high quality software is

writing automated tests. Unfortunately, when quick results are expected, this activity is often omitted in the first stage of the system production due to the additional time effort. However, when the work methodology forces developers to create tests, then they will guarantee high quality solution at the maintenance stage. That and all previous advantages make the architecture of independent services very functional and easily applicable, especially in systems with microservices architectures [12].

As for an aspect-oriented management of request deployment, the first thing to do is determining the system's characteristics. It means indicating time periods when the system is the most intensively used, when delays in access to services happen which result from inadequate hardware resources for the system's load. The next question is in which services the delay occurs and if it is caused by the massive use of services by users or the services' insufficiency. Another matter is if any services should be distinguished in terms of importance and priority. To answer all these questions you need to perform some suitable measurements. The best way to deal with that matter seems to be gathering traces of performed operations and their execution times on the service layer. Additionally, it is worth saving the aspect of request characterizing the selected call: which user invoked the request (what users' group he belongs to), which application the request was invoked from, when it was invoked and which service it was sent to. After collecting these data and analyzing them, we can determine which services consume the most resources, when, by whom and from what applications they are used. You need to compare the above information with the business environment of the system: who (which group of users) has to be provided with priority access, to what services and in which period of time this access must be provided. In practice, the most often it turns out that various services are not used extensively simultaneously and that miscellaneous groups of users need priority at different time. Such dynamics of the system usage enforces a similar dynamic in allocating resources.
The allocation of limited resources to services must not be constant but variable with the ability to adapt easily, depending on the aspect of the request. For this purpose, the requests management component must be introduced between the UI and the services. That component should be able to analyze the mentioned above aspects of a request. Then, depending on predefined rules, the request should be redirected to one of the N nodes. In this way, you can specify the nodes that will serve a selected group of users and requests to selected services. This allows you to prioritize periodically the selected requests by providing reliable access to critical services and functions of the system.

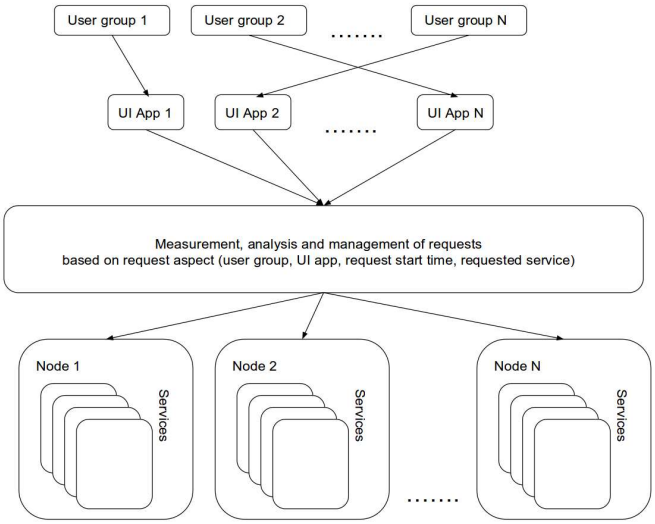

Fig. 3. Diagram of requests redirection to the nodes on the basis of the aspect of the request

Figure 3 shows a formal diagram of request redirection to the nodes on the basis of the aspect of the request and the defined configuration. The ideal solution is defining the rules in the way which will ensure equal responsive access to all services for all users. However, with a large number of users and services and with limited resources it may be impossible. Then you have to choose which critical services should be available in which periods of time.

The rules should be under modification as long as we reach the satisfactory level of responsiveness of critical services. At the time of insufficient resources, low priority services may not be available but those with a high priority will be able to work properly. Figure 4 shows a life cycle of configuration. It will be usually a daily cycle, but it can also be adapted to another time quantum, depending on the needs. Continuous monitoring of requests and the collected data analysis allows you to customize the configuration of dynamic requests to the nodes allocation. This way you can achieve the highest possible efficiency as well as ensure the dependability of key services.

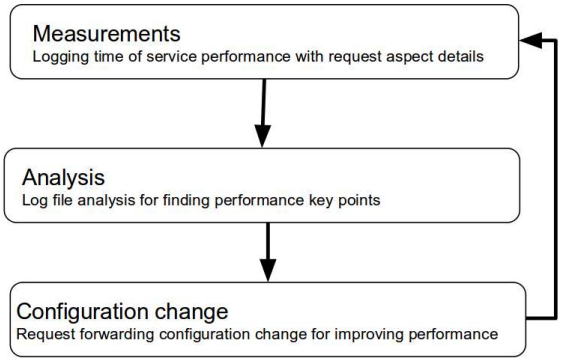

Fig. 4. Life cycle of requests redirection configuration

4 Verification method – case study

The proposed solution was implemented in the system "Moja PG" at Gdańsk University of Technology. A component of request management was introduced between the portal (which is a collection of independent user applications) and services layer (in which business components are embedded). Another node was added on which all services are available in the same way as on the original node. The requests manager distributes requests from user applications into these two independent nodes.

Using exactly the same UNIX operating system and the same version of the JVM on both nodes provides the same threading, concurrency and parallelism mechanisms. Exactly the same system software was used before and after adding the request manager. Thus it is possible to compare the results of those two measurements without the need to consider the impact of the operating system or JVM.

Dynamic requests management solution was introduced in the production environment at the beginning of the academic year 2016/2017. It was a specific period of time because you had to ensure reliable access to services for 30,000 users within a few days. Students wanted to see their timetables, sign up for elective subjects or extend the validity of their electronic ID cards. At the same time employees had to issue the necessary certificates, give students their final grades and supervise the teaching process. Student actions during the working day got lower priority in the access to resources on the basis of the aspect of the request. However, full access to resources was granted in the evening when university employees did not use the system.

Introducing such a separation, the average execution time was reduced. There was no noticeable drop in performance during the first days of the academic year, which occurred in the previous years. Services were available, there was no moment in which the system was overloaded. Thanks to the dynamic request management and despite a

sudden increase in the number of system users, there was no snow ball effect. This effect means lengthening the service response time as a result of taking over more and more resources by other services. Reliable access to critical services was guaranteed all the time because they were invoked on a separate node. The change from a traditional balanced load distribution to dynamic request management helped to maintain full reliability of critical system functions.

The process of determining the rules for the separation of requests between nodes was discrete and was repeated once a day. Analysis was performed automatically on the basis of log files in which service requests were saved along with their execution times. As a result, we were able to calculate the following statistics:

- average time of service execution divided into groups of users,
- the total time of the service execution during the day divided into groups of users,
- the number of requests to the service divided into groups of users.

On the basis of those statistics it is easy to determine which services are the most often invoked, by whom, and which occupy resources for the longest time. Of course, changing the configuration directly affects the performance of the execution of services, so you need to monitor regularly the statistics to ensure that the rules are properly applied. Besides, changes in the way of using the system make a continuous control essential, too.

5 Measurements results

Measurements of system "Moja PG" in the production environment were made on log files generated by the system. A sample log file part is shown below:

```
2016-09-26 00:00:05,170 [StudentManagerBean t-146] START: getObjectById
2016-09-26 00:00:05,205 [StudentManagerBean t-146] STOP getObjectById: 34ms
2016-09-26 00:00:05,222 [StudentManagerBean t-18] START: getCardBySubject
2016-09-26 00:00:05,662 [InventionsManagerBean -83] START: searchInventions
2016-09-26 00:00:05,794 [InventionsManagerBean t-83] STOP searchInventions: 132ms
2016-09-26 00:00:05,809 [InventionsManagerBean t-157] START: getInventorsByInventionId
2016-09-26 00:00:05,822 [InventionsManagerBean t-157] STOP getInventorsByInventionId:12ms
2016-09-26 00:00:05,835 [InventionsManagerBean t-146] START: getAdministrativeUnitsByInvenId
2016-09-26 00:00:05,838 [InventionsManagerBean t-146] STOP getAdministrativeUnitsByInvenId: 2ms
2016-09-26 00:00:05,844 [StudentManagerBean t-18] STOP getCardBySubject: 621ms
```

Some part of the following lines content logged by the system had to be hidden for security reasons. Pairing log lines talking about the beginning of the service request processing (START) and its completion (STOP) was performed on the basis of the task ID (e.g. "t-146"). Each STOP line contains also execution time in milliseconds. Services are executed asynchronously, so log lines appear in a file in a random order, too. In each line there is also included the timestamp, the name of the package of

services, e.g. StudentManagerBean or InventionsManagerBean, and the name of the proper service.

The first log analysis checked only correctly completed invocation of services and summed up their execution times. Then, on the basis of the total time in one-day periods, a ranking list was created showing which services performed the longest. The ranking presents services that are invoked very frequently or which are invoked rarely but with a long execution time. Therefore, it is also worth analyzing the average time of service execution. First, the measurements were performed on a loaded production system without the additional node. Then, the services which occupied the first node resources for the longest time were redirected to the second node. After 24 hours log file analysis was repeated, this time on both nodes. The collected results are shown in Tables 1, 2 and 3 and in Figures 5 and 6.

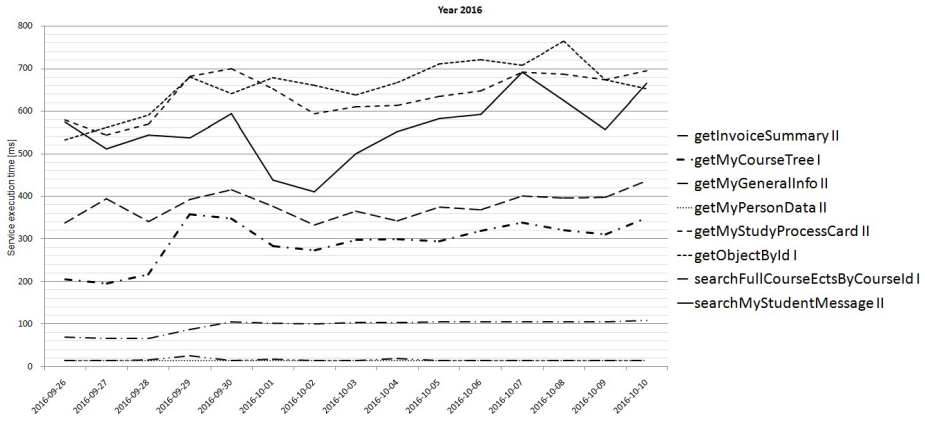

Fig. 5. Chart of the average service execution time on selected days of the year 2016.

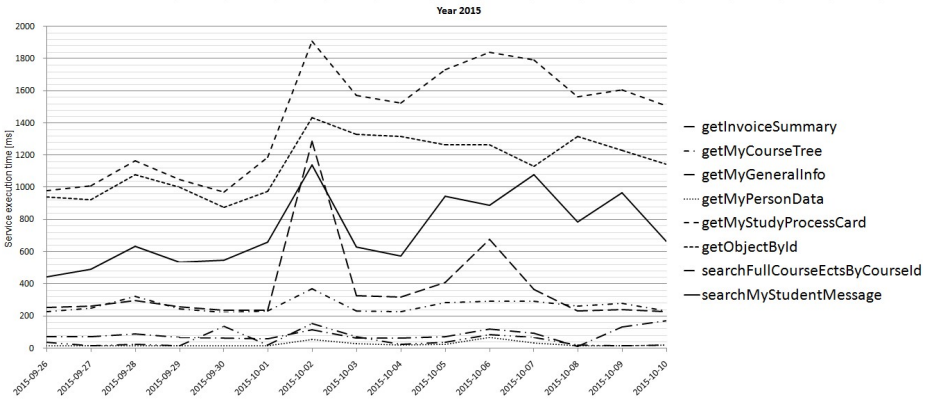

Fig. 6. Chart of the average service execution time on selected days of the year 2015.

The charts on Figures 5 and 6 show the average execution times of selected services at the beginning of the academic year in 2015 and 2016. In 2015 “Moja PG” system worked only on a single node without aspect-oriented management of requests. In 2016 an additional node was used and service requests were distributed between nodes on the basis of their invocation aspects. The noticeable fact is the reduced service execution time (the longest service executions in 2016 lasted 765 milliseconds and in 2015 – 1907 milliseconds). In addition, on the graph of 2015 you can notice a significant increase in services execution time on 2015-10-02 – all students started to use the system intensively at the beginning of the academic year. Then a snowball effect occurred – an extension of services execution time because the user load caused that each invoked service was executing even longer. In the chart of the year 2016 there was an extension of services execution time on 2016-09-30, when the students started to use the system intensively. However, thanks to using two nodes, there were enough resources to prevent any snowball effect and the duration of execution time of services did not exceed 800ms.

Measurements from 2016 presented in Table 1 clearly show that the priority services running on the second node had a single-digit increase in execution time under load (from 1.33% to 8.03%), while services operating in the more loaded first node had an increase in execution time under load in the range from 10.58% to 26.58%. On the basis of those results it can be stated that the critical services running on a node II worked stable and reliably, despite the system maximum load of 30 thousand users. If the load had been distributed equally between the two nodes, some of the resources would have been used by the services with a lower priority, and consequently the priority services execution times would have been extended, affecting negatively on the dependability of key system functionality. Table 3 shows compiled results of the comparison of services execution time between selected time periods in 2015 and 2016.

Table 1. The results of service performance measurement in 2016

Year 2016				
Service name	Max execution time [ms]	Avg execution time without load [ms]	Avg execution time with load [ms]	The increase in execution time between the loaded and no-loaded system [%]
getInvoiceSummary II	108.00	90.50	96.13	6.22
getMyCourseTree I	317.00	200.50	253.80	26.58
getMyGeneralInfo II	437.00	348.50	368.07	5.61
getMyPersonData II	15.00	14.03	14.33	2.16
getMyStudyProcessCard II	700.00	581.50	628.20	8.03
getObjectById I	765.00	547.50	638.53	16.63
searchFullCourseEctsByCourseId II	20.00	15.00	15.20	1.33
searchMyStudentMessage I	692.00	523.00	578.33	10.58

Table 2. The results of service performance measurement in 2015

Year 2015				
Service name	Max execution time [ms]	Avg execution time without load [ms]	Avg execution time with load [ms]	The increase in execution time between the loaded and no-loaded system [%]
getInvoiceSummary	172.00	71.17	94.67	33.02
getMyCourseTree	370.00	241.67	324.67	34.34
getMyGeneralInfo	1286.00	258.00	455.00	76.36
getMyPersonData	69.00	17.67	32.44	83.65
getMyStudyProcessCard	1907.00	1060.67	1670.78	57.52
getObjectById	1434.00	966.33	1270.56	31.48
searchFullCourseEctsByCourseId	157.00	43.33	56.11	29.49
searchMyStudentMessage	1078.00	552.67	820.22	48.41

Table 3. The results of comparative measurements of service performance in 2015 and 2016

Service name	Increase in max execution time between 2016 and 2015 [%]	Increase in avg execution time without load between 2016 and 2015 [%]	Increase in avg execution time with load between 2016 and 2015 [%]
getInvoiceSummary	-37.21	-10.54	-33.55
getMyCourseTree	-14.32	-17.03	-21.83
getMyGeneralInfo	-66.02	-4.52	-19.11
getMyPersonData	-78.26	-20.58	-55.82
getMyStudyProcessCard	-63.29	-45.18	-62.40
getObjectById	-46.65	-43.34	-49.74
searchFullCourseEctsByCourseId	-87.26	-65.38	-72.91
searchMyStudentMessage	-35.81	-5.37	-29.49

Without load the percentage differences in execution time ranged from 4.52% to 65.38%, which means that the system running without load on one node selected services (getMyGeneralInfo, searchMyStudentMessage or getInvoiceSummary) handled at the optimal time. This is why differences are so small (from 4.52% to 10.54%), while the execution time of remaining services even without load were reduced from 17.03% to 65.38%. This indicates that one node even without load did not have enough resources for optimal work. The comparative analysis of the maximum service execution times between unloaded system and system under load shows an increase of execution time from 14.32% up to 87.26%. The services of higher priority that were transferred to the second node (II) were being executed shorter from 37.21% to 87.26%, while those of lower priority which were executed on more loaded first node (I) showed a shorter duration of execution time in the range from 14.32% to 46.65%. Summing up the results of the measurements, it can be stated that with the selection of priority services and redirection of their execution on less loaded node a shorter execution time and reliable access, even at maximum load, were provided. Services with less priority which were running on the more loaded node also recorded

a reduction of operating time (by distributing the execution of services on two nodes). However, it was not as significant as in the case of priority services. The measurements' results show that using an aspect-oriented management of service requests can assure dependable access to priority services which can be moved to a separated node. Of course, when there are unlimited resources, a dynamic load-balancing is enough for making all services unfailing. However, when there are insufficient resources, the services should be prioritized and the most important ones should work properly regardless of the load. In the production environment we were able to achieve our main goals by adding another node and using an aspect-oriented management of service requests. In this way:

- the execution time of all services decreased because more resources were available,
- the execution time of priority services decreased significantly because they were invoked on a separated node,
- the most important goal – priority services guaranteed reliable performance regardless of the load.

Priority service are used by finite and well known group of users. We were able to estimate how much of resources were necessary for this group of users to work on priority services. During work day of this group of users required resources were granted to priority services which gave us confidence that they will work with high performance and dependability. When priority services were not used, thanks to dynamic configuration free resources were used by other services which needed them.
Management of service request allows to react to a changing load and assure necessary resources for the chosen services. When high performance and dependability cannot be assured for all services because of the lack of resources, then a configurable management of service requests is a good solution. Thanks to that the priority services can perform efficiently.

6 Summary

The proposed solution is based on the aspect-oriented management of services requests. It was introduced in the system "Moja PG" at Gdańsk University of Technology that supports 30,000 users. After adding an additional node, the services execution time was reduced, thanks to a dynamic management of requests. On the basis of the request aspects and configurable rules, a snowball effect was avoided in the time of the most intensive system use. Execution time of priority services got reduced by an average of more than 60%. In the case of services with lower priority, the execution time was reduced by more than 30%.

Summing up, when computing resources are limited, then appropriate arrangement of rules for requests management provides a stable and fast access to critical services, without the risk that less important services will lead to the seizure of resources. Of course, the rules must be constantly modified, based on the performance results in

order to adapt them to the changing conditions of system usage. As a result of the need for regular performance analysis and customization of the rules used in the configuration, there arises a question if the system should be allowed to configure itself automatically. Some work on the mechanism of dynamic recommendations was started. At this stage we assumed that the system should suggest which rules will improve the efficiency of access to services. However, the administrator will configure these rules in the production environment manually. When the recommendation engine is refined, an ultimate goal is to implement a module which will automatically reconfigure the system due to changing user activities.

References

1. R. Van den Bossche, K. Vanmechelen, and J. Broeckhove, "Cost-Optimal Scheduling in Hybrid IaaS Clouds for Deadline Constrained Workloads," in *2010 IEEE 3rd International Conference on Cloud Computing*, 2010, pp. 228–235.
2. M. A. Metawei, S. A. Ghoneim, S. M. Haggag, and S. M. Nassar, "Load balancing in distributed multi-agent computing systems," *Ain Shams Engineering Journal*, vol. 3, no. 3, pp. 237–249, Sep. 2012.
3. J. O. Gutierrez-Garcia and A. Ramirez-Nafarrate, "Agent-based load balancing in Cloud data centers," *Cluster Computing*, vol. 18, no. 3, 2015.
4. Xiao Qin, Hong Jiang, A. Manzanares, Xiaojun Ruan, and Shu Yin, "Communication-Aware Load Balancing for Parallel Applications on Clusters," *IEEE Transactions on Computers*, vol. 59, no. 1, pp. 42–52, Jan. 2010.
5. A. Y. Zomaya and Yee-Hwei Teh, "Observations on using genetic algorithms for dynamic load-balancing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 12, no. 9, pp. 899–911, 2001.
6. C.-C. Lin, H.-H. Chin, and D.-J. Deng, "Dynamic Multiservice Load Balancing in Cloud-Based Multimedia System," *IEEE Systems Journal*, vol. 8, no. 1, pp. 225–234, Mar. 2014.
7. X. Ren, R. Lin, and H. Zou, "A dynamic load balancing strategy for cloud computing platform based on exponential smoothing forecast," in *2011 IEEE International Conference on Cloud Computing and Intelligence Systems*, 2011, pp. 220–224.
8. P. Lubomski, A. Kalinowski, and H. Krawczyk, "Multi-level Virtualization and Its Impact on System Performance in Cloud Computing," in *Communications in Computer and Information Science*, vol. 608, 2016, pp. 247–259.
9. C. Zenon, M. Venkatesh, and a Shahrzad, "Availability and Load Balancing in Cloud Computing," *International Conference on Computer and Software Modeling IPCSIT vol.14 (2011) IACSIT Press, Singapore*, vol. 14, no. September, pp. 134–140, 2011.
10. P. Lubomski and H. Krawczyk, "Clustering Context Items into User Trust Levels," *Advances in Intelligent Systems and Computing*, vol. 470, pp. 333–342, 2016.
11. J. Thones, "Microservices," *IEEE Software*, vol. 32, no. 1, pp. 116–116, Jan. 2015.
12. A. Balalaie, A. Heydarnoori, and P. Jamshidi, "Microservices Architecture Enables DevOps: Migration to a Cloud-Native Architecture," *IEEE Software*, vol. 33, no. 3, pp. 42–52, May 2016.