Computer Science • 15 (1) 2014 http://dx.doi.org/10.7494/csci.2014.15.1.89

Michał Soboń
Piotr Nawrocki

PUBLIC CLOUD COMPUTING FOR SOFTWARE AS A SERVICE PLATFORMS

Abstract *As cloud computing becomes increasingly popular among enterprises, organizations, and developers, it is time to consider how Software as a Service solutions can be built rapidly and cost-effectively. This paper presents the possibility of using a service-based architecture operating within the framework of the public cloud computing model and implementing Software as a Service. It also describes the architecture, implementation, and testing of sample applications. The conclusions drawn, with respect to related work and the results obtained, shed more light on the application of public cloud systems for Software as a Service purposes.*

Keywords cloud, service, service-oriented architecture, Service Level Agreement, SaaS

1. Introduction

A cloud is a computer system that provides computing, data storage, and infrastructure as a service. Clouds have been evolving for decades and originated from mainframes, which were big, inflexible systems used by large institutions for computing purposes. However, mainframes quickly became insufficient in the face of increasingly-complex scientific computations. The next stage in cloud evolution was the use of grids – heterogeneous systems created for computing purposes which were based on the temporarily unutilized processing power of mainframes. Grids provided computing resources, but the amount of resources available depended on the current mainframe usage. Thus, grids were perfect for scientific purposes but unsuitable for business ones due to the unreliability of their computing power. Clouds were built based on the experience gathered from grid systems – they provide optimum usage of physical resources combined with a guaranteed quality of service for end users. Moreover, they are based on Internet technologies (which facilitates access) and automatically-managed resources (which improves provisioning and scalability) [19, 16, 6].

Clouds rapidly emerged as a system infrastructure because they provided quickly-scalable resources. Public clouds such as Amazon EC2, Microsoft Azure, and Google App Engine provide reasonably-priced resources on demand. Public clouds have multiple applications: scientific computing, back end for web applications, and providing infrastructure for high-level services like Software as a Service [26]. This paper attempts to evaluate how public clouds can be used to build high-level services such as Software as a Service.

This paper is organized as follows: Section 2 presents related work, Section 3 describes a Software as a Service system based on the Heroku cloud, Section 4 presents the tests performed, and Section 5 presents the conclusion.

2. Related work

The idea for cloud systems comes from grids. There are many similarities between grids and clouds. First of all, both computing models are multi-task and multi-tenant ones. Cloud and grid computing provide service-level agreements (SLAs) for guaranteed uptime availability exceeding 99 percent. Scalability of applications is also accomplished in the same way – through load balancing of separately-running instances [15]. The author points out one main difference – grids may slow down, or computations may even be interrupted, as a result of available resource shortages (which is a serious issue hindering grid usage for business purposes). On the other hand, clouds are capable of providing stable, reserved resources for customers.

In [3], the author presents the evolution of IT services. The starting point for consideration is the positive role of outsourcing. It results in cost reductions, higher quality, a fast development cycle, and performance assurance. Benefits from outsourcing are compared to greater advantages of cloud computing, like better cost optimization and another speed-up in development. As opposed to outsourcing (which

requires significant expenditure), cloud computing does not generate up-front costs. Other advantages of cloud systems include a shorter contract time that allows for more flexibility.

Service-oriented architecture[1] (SOA), combined with cloud computing, can provide multiple benefits. In another article [24], the author presents the challenges and opportunities of such a combination. It is promising due to the improving quality of clouds, and benefits resulting from their interoperability, scalability, and easy service reuse owing to their availability over the Internet [21]. Of course, this solution does introduce some new challenges [8], like service availability that is limited to cloud availability as well as new issues of security, but the authors of the paper believe that the opportunities are greater than the challenges.

Clouds combine four areas of computer science: distributed systems, Internet technologies, resource virtualization, and automated resource management. This idea makes it possible to take advantage of synergies between all these domains. The concept of cloud systems [8] is presented in Figure 1.

The following are major characteristics of clouds [2]:

- access over network;
- access on demand;
- flexibility;
- adaptation to customer needs.

Cloud systems can be classified by level of abstraction [2]:

- Infrastructure as a Service (IaaS) – this is a platform that provides virtualized resources (computations, storage, communication) on demand. IaaS can be used as the basis for higher-level cloud systems, such as Platform as a Service or Software as a Service. Amazon EC2 and GoGrid are examples of this type of cloud.
- Platform as a Service (PaaS) – this is a platform that provides virtualized resources enhanced with programming platform elements. These additional features can be used in applications via the provided API; e.g., PaaS can provide authentication or a persistence layer. Platform as a Service very often offers automatic scalability. Google App Engine and Heroku are examples of this type of cloud. Table 1 shows the comparison of most popular PaaS systems.
- Software as a Service (SaaS) – this is the most advanced category of cloud systems. SaaS provides on-demand access to applications located in the computer network. The user does not need to worry about the infrastructure, software, or application maintenance – these are the responsibilities of the provider. Salesforce.com is an example of SaaS [25].

IaaS and PaaS clouds are quite similar in that they provide an environment for applications. PaaS offers solutions for common cases, like authentication, persistence,

[1]Service-oriented architecture is a software-development paradigm in which software delivers desired functionalities through the use of orchestrated services [7].

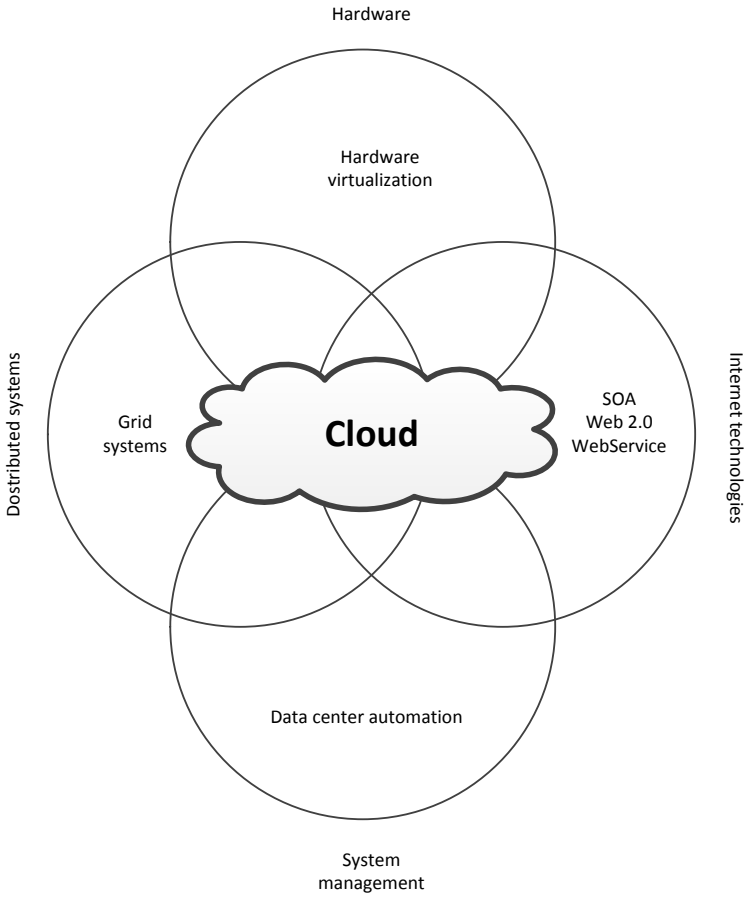

Figure 1. Concept of a cloud.

or big data support. This makes it possible to save time in development, but flexibility is reduced compared to IaaS [9]. PaaS saves time by handling all operational work, such as configuration, optimization, continuous environment updates, and finally, the deployment of applications and their detailed monitoring. This, however, also proves to be a disadvantage of PaaS, due to its issues with complex, highly-customized solutions. IaaS, on the other hand, provides resources (computations, storage, communication) that need to be configured, managed, and monitored by the user. These time-consuming actions allow users to create highly-customized solutions, including PaaS.

Cloud systems are becoming increasingly popular thanks to their development efficiency and cost effectiveness. Easy access over a network makes this technology globally available. In [22], the author divides cloud users into four groups: cloud in-

Table 1
Platform as a Service comparison.

Name	Vendor	Supported programming languages	Features
App Engine	Google	Java, Python, Go	Easy development of scalable applications using BigTable database, multiple features available via API; e.g., authentication [23]
Heroku	Salesforce.com	Ruby, Java, Scala, Python, Node.js, Clojure, PHP	Many programming languages supported, easy management, deployment using Git
Cloud Foundry	VMWare	Java, JavaScript, Ruby, Scala	Support for Eclipse IDE, add-on for Spring Roo framework
Windows Azure	Microsoft	.NET, JavaScript, PHP, Python,	Well integrated with Microsoft services available via API

frastructure developers, service authors, integration and provisioning staff, and end users. Cloud systems are a common tool for these people, as they care about quality. Issues with clouds may be solved on multiple levels. The author calculates the ratio between the number of support and development requests – one request to a cloud infrastructure developer corresponds to 1,000 requests to service authors and 100,000 requests to service integrators. These numbers reflect the flexibility and easy customization of cloud infrastructure.

In many papers [22, 17, 3], the authors note multiple issues connected with cloud security. The most common are related to the security of multi-tenancy, data privacy, and limited SLA. Security in cloud-based systems is frequently mismanaged – in order to maximize effectiveness while also minimizing costs as well as the risk of security breaches, security and privacy must be considered from the initial planning stage at the beginning of the system development life-cycle. Addressing security issues after implementation and deployment is not only much more difficult and expensive, but also considerably riskier [10].

Despite the complex issues related to security, SaaS becomes an increasingly-popular solution for cloud computing. One of the proposals to increase the security level is a DIFC (Decentralized Information Flow Control) model for SaaS application security (referred to as SAS-DIFC) [20]. This controls the data flow among SaaS applications as well as the external untrusted environment, and also protects the privacy of user data.

Important issues in the context of the cloud are performance and scalability. In [5], the authors present performance evaluation and scalability measurement issues for

Software as a Service (SaaS) in the Amazon EC2 cloud environment. In this article, the authors propose formal and graphic models with metrics, which allow for efficient measurements of SaaS performance and scalability in a cloud.

In addition to the many benefits [11], the use of cloud computing in the SaaS model brings some disadvantages as well. In [4], the author lists the main disadvantages of SaaS, such as low confidence in data security in situations where critical data (human resources, billing, etc.) is located on servers outside the company, and integration with the rest of the system applications. The problem of integration is a result of simultaneous use of data located in the cloud and on the local system. Another issue regarding SaaS is the existence of a single point of failure for all applications. This makes development and maintenance more complicated due to the need to provide high availability.

An important issue is the problem of integrating existing software into the SaaS environment. In [14], the authors use a master table and master code to effectively transfer existing software and data from the ASP (Application Service Provider) to the SaaS environment.

On the basis of conclusions from the papers discussed above, we have decided to evaluate public cloud systems as an infrastructure background for a cloud operating at a high level of abstraction; i.e., Software as a Service.

3. Implementation of Software as a Service

Building Software as a Service on the basis of physical, non-virtualized resources is complicated and expensive. We need to ensure maintenance staff and provide additional resources for future growth [12, 18]. A solution to these issues is to build an SaaS system based on public cloud technology, which provides reasonably-priced, scalable, and easily-obtainable resources. We have built a sample SaaS system based on the Heroku cloud. Heroku itself is a Platform as a Service system built over the Amazon IaaS.

The selected Heroku platform has many advantages, including automated failover, disaster recovery, and bit rot prevention. The platform supports agile development methodologies and allows for seamless software development. Heroku allows for the use of the Postgres database or other external databases, and provides a "worker role" which makes it possible to run long, asynchronous tasks [13, 1].

The main part of the sample SaaS system is the Portal. It is a gateway for users which allows access to applications that are provided as services. The Portal is also responsible for user management, single sign-on, and application-access management. It may contain community elements such as newsletters, forums, or notifications. The next part of the system is the application group; this provide business functionalities for customers (software services). Access for users is available on demand, and customers may be charged depending on software usage.

Figure 2 presents the system architecture.

It contains five layers:

- the external resource layer containing identity providers federated with the Portal using the OpenId/OAuth protocol stack and other resources managed by other organizations;
- the system layer containing the Portal is responsible for authentication, authorization, and user account management;
- the communication layer is the Internet-based part of the system responsible for gluing all layers together;
- the application layer contains multiple applications that implement business features provided as a service by the Portal;
- the user layer is based on end-user computers with web browsers that allow access to the system.

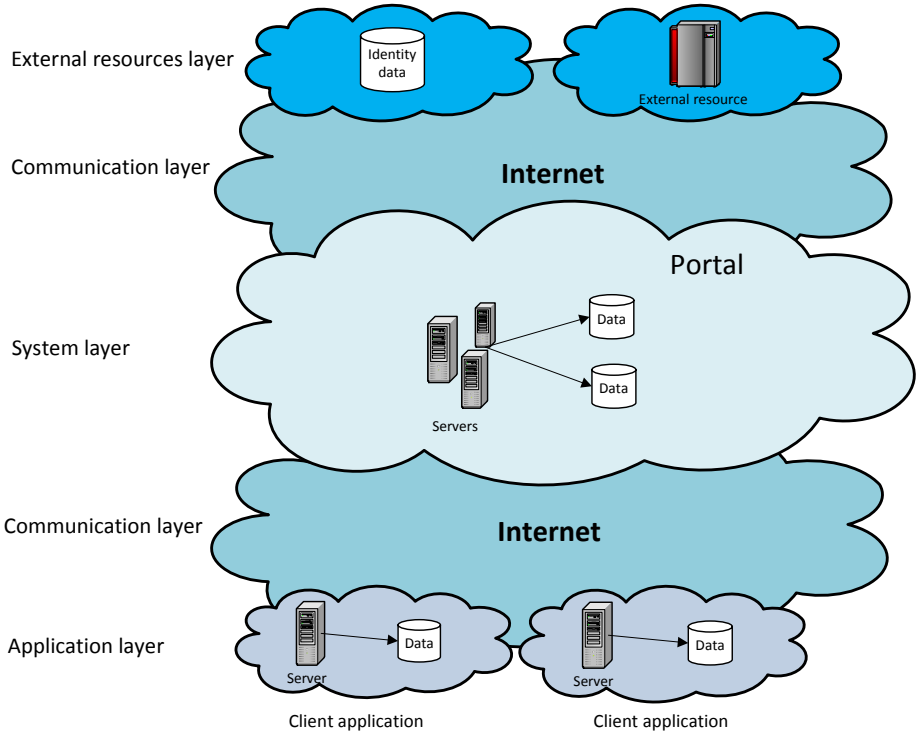

Figure 2. Portal architecture.

The Portal is responsible for providing single sign-on[2] for applications. Figure 3 shows the sequence associated with user login to the system. The user displays the

[2]Single Sign-on (SSO) – a system feature enabling common authentication for multiple independent applications (Service Providers – SP) using credentials from the Identity Provider (IdP). If the user is authenticated with the IP, there is no need to log in again when changing applications. This

application page; if the request does not contain a valid SSO token, the user is then redirected to the Portal login page. After successful authentication, an SSO token is returned. This token is attached to the next request, and the application validates it with the Portal. After successful verification, a user session is created and the session ID is returned to the user.

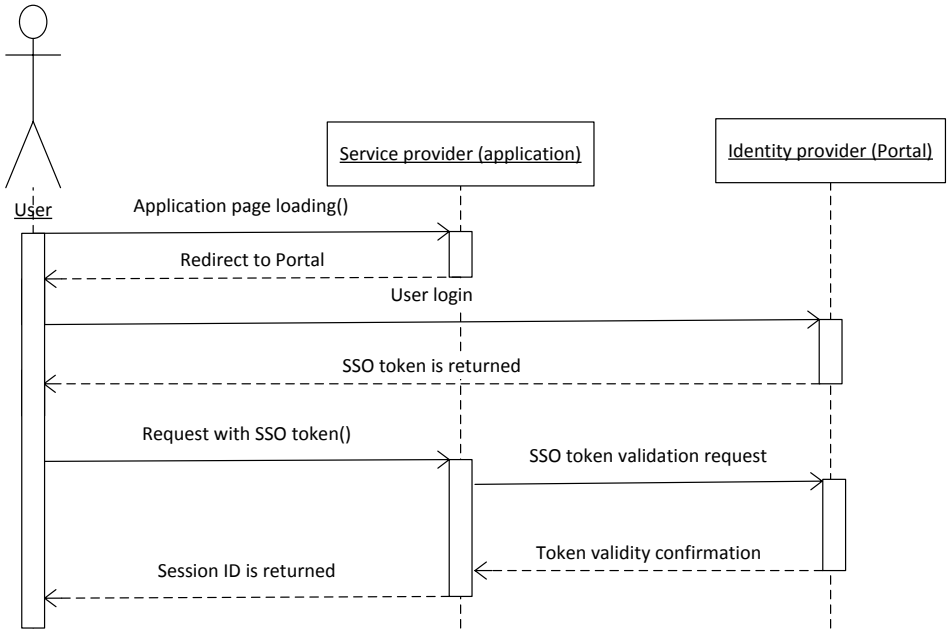

Figure 3. Single sign-on login sequence using a provider on the basis of the Portal's embedded database.

The Portal also provides another way of getting the authentication token. This can be done using an external identity provider over the OpenId[3]/OAuth[4] protocols. The Portal can work with public-identity providers such as Google, Twitter, Facebook, etc. or can be connected to a dedicated provider associated with the customer's domain network. This solution is helpful for both users (who can reuse their existing accounts) and organizations (which gain 'one click' account management of both internal and external resources without sharing their database directly).

mechanism also provides the Single Sign-Off feature, enabling a common logout from all associated applications.

[3]OpenId – a distributed, open authorization standard. It helps with the distribution of identity elements like name, email and address across co-operating applications.

[4]OAuth – an open standard enabling resource sharing without providing username and password and instead using client-agent redirects. The OAuth standard is complementary to OpenId.

4. Case study

For prototyping purposes, we have created the Portal as a core part of the system. It provides user management, application access management, and single sign-on features. The Portal also provides a mechanism enabling usage monitoring of the software provided. We have also developed two sample applications which provide business functionalities: Staff Manager and Warehouse Manager. Staff Manager is responsible for staff management at an enterprise and can be used by the HR department for storing basic information of each employee. The second application is Warehouse Manager, which optimizes the management of goods at a warehouse, thus facilitating the employees' work. The developed applications can form the basis of an application suite for enterprises such as factories, logistics centers, or wholesalers.

Both the Portal and the applications themselves are web-based applications. They have been created using Play Framework – an emerging framework designed for building MVC[5] applications. The back end has been developed in Java, while the front end is based on HTML and CSS. Play provides a flexible template language, facilitating the creation of views. It also contains support for ORM[6], which is provided by EBean ORM. We have taken advantage of one of Play's plugins – Play Authenticate – which makes it possible to easily federate Play applications with OpenId/OAuth identity providers.

4.1. Performance evaluation

The Portal and applications were deployed to the Heroku cloud. Each application ran on a single dyno[7] environment (free plan). In order to evaluate the SaaS system based on the Heroku platform, we performed three tests:

- a test measuring the load time of the Portal's login page depending on client location;
- a test measuring the time cost of authentication depending on authentication provider complexity;
- a test measuring the maximum performance of the application based on one dyno.

4.1.1. Test measuring the load time of the Portal's login page depending on client location

Software as a Service applications are available over Internet, and thus, may be used in multiple locations around the world. Due to client-server distance, the increase in RTT[8] may inconvenience users. We can assume that a fast application needs around

[5]MVC – Model-View-Controller – a design pattern for handling user interfaces.

[6]ORM – Object Relational Mapping; middleware, enabling the use of SQL databases in object-oriented software.

[7]Dyno – an independent process execution environment in Heroku Cloud, which provides 512 MB of RAM and a single CPU.

[8]RTT – Round Trip Time.

250 ms to load a page on the client side[9]. Heroku is hosted on the Amazon EC2 cloud located in Virginia (U.S.) – this is the home of our application. The purpose of the test was to find locations where our system could be considered fast.

For testing purposes, we used the blitz.io tool, a cloud-based system for performing load tests. It provides clients in multiple locations:

- U.S. (Virginia, Oregon, California);
- Brazil;
- Japan;
- Singapore;
- Australia;
- Ireland.

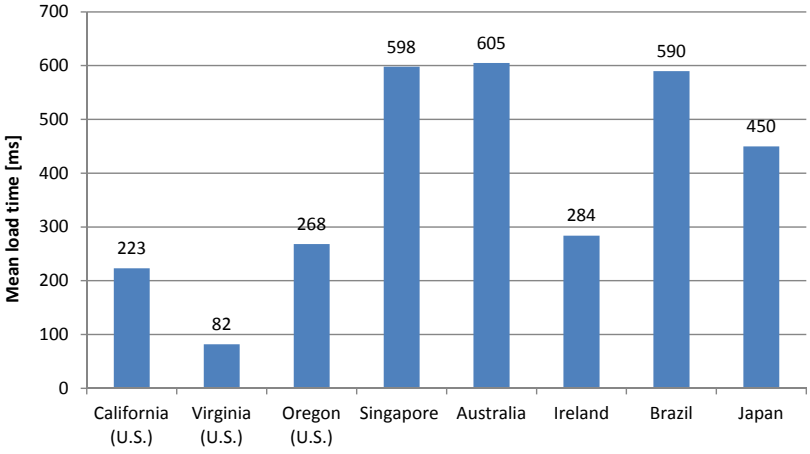

Figure 4. Portal login page mean load time.

During the test, 250 threads (clients) from different locations were continuously loading the login page. Each sequence lasted for 60 seconds to yield results that were more accurate. Figure 4 shows the mean load time of the Portal login page for various client locations. The chart shows that load time for four locations was around or below 250 milliseconds. Thus, it can be stated that our system will be considered fast by clients in the U.S. and Ireland.

4.1.2. Test measuring the time cost of authentication depending on authentication provider complexity

The Portal is a Single Sign-On identity provider. This test compares the time effectiveness of two kinds of authentication – one based on an embedded database and

[9]Blitz.io documentation (`https://www.blitz.io/docs/overview`) suggests that a fast application page should load in 250 ms.

the other using an external, federated authentication provider over OpenId/OAuth protocols. The results show the token revalidation time (including the Portal page reload time) using the Portal's database and external providers (Google, Facebook, Twitter).

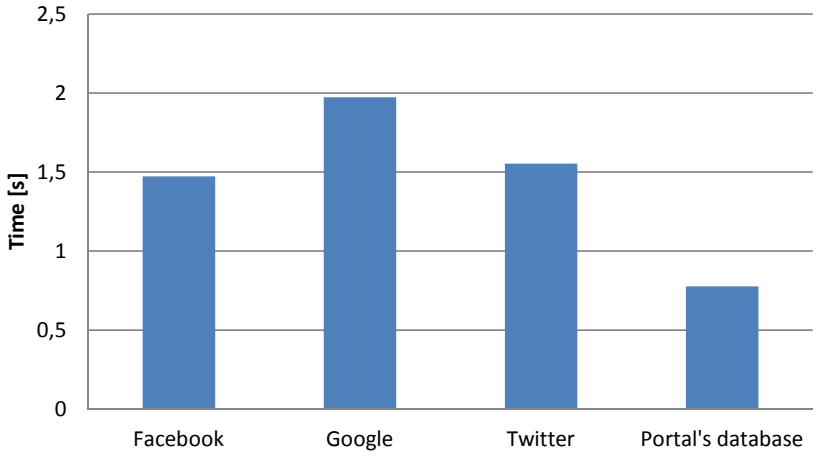

Figure 5. Token revalidation time depending on identity providers.

Figure 5 shows that the usage of OpenId/OAuth protocols is time expensive – it requires about twice as much time as internal database authentication. This is caused by the complexity of protocols and multiple HTTP requests that are sent between the Portal and the external identity provider. The cost is significant, so this should be considered in order to reduce the impact; e.g., by creating a dedicated cache or, where possible, by using less-complicated identity providers.

4.1.3. Test measuring the maximum performance of the application based on one dyno

The main goal of this test was to find the maximum number of requests that can be processed by a system based on a single dyno. During the test, we used the authentication service provided by the Portal, which verifies if the specified single sign-on token is valid.

The free blitz.io plan limits the maximum number of simultaneous clients (threads) to 250 – so, in order to generate additional traffic, we used a PC along with the JMeter tool, which allowed us to create an additional 300 threads.

Figure 6 shows the number of threads as a function of test time. The dotted line shows the maximum number of correctly-completed requests per second. Our system, which was based on one dyno, was able to fulfill about 410 requests per second.

Test results demonstrate that it is possible to build an efficient application based on the Heroku cloud. However, as long as it is the sole infrastructure provider, Heroku

remains the single point of potential failure for the system. Moreover, we should be aware that the user's experience regarding system speed varies depending on his/her location. This issue is caused by RTT. In order to avoid this while building a fast, global system, it is necessary to find a cloud system with multiple locations around the world.

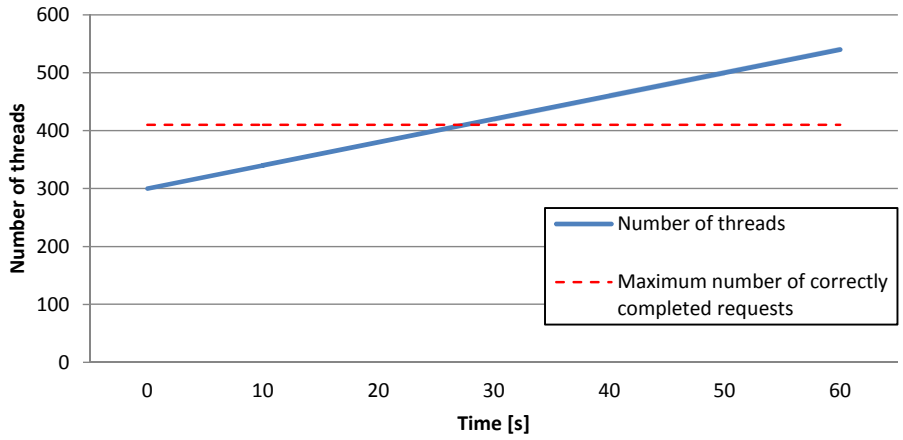

Figure 6. Number of threads as a function of time.

5. Summary

This paper provides a brief survey of cloud systems, mainly focusing on Software as a Service. This kind of cloud can be built on the basis of physical, non-virtualized resources, but this approach entails significant entry and maintenance costs and is not easily scalable (obstacles that are important in business applications). The disadvantages listed above can be overcome by developing Software as a Service based on public clouds that are easily scalable, reasonably priced (based on usage), and come with a maintenance staff. The main goal of our work was to determine if public clouds are suitable for Software as a Service infrastructure.

Cloud systems may cause numerous security issues; one such issue is the fact that incorrect configuration of virtualization may allow unauthorized access to data or loss of data integrity. Some organizations that handle sensitive data cannot afford to transfer their data outside their boundaries, because they would not be able to supervise data processing and access. This is an important constraint for public cloud technology; but in some cases, the solution may be a private cloud system.

One should be aware of the complexity of the Single Sign-On process. It can be quite simple and fast when based on an embedded identity provider using the local database, but there is also the possibility of building complex distributed solutions

using external identity providers. This allows identity sharing across organizations, but also consumes time and introduces an external failure point to the application.

Our test has demonstrated that public clouds are capable of hosting Software as a Service, but we must be aware of certain limits. First of all, it is difficult to develop fast applications for clients in all locations around the world – this is limited by RTT issues. The second limit is more fundamental – the maximum availability of our application is limited by the cloud SLA. In direct use, the cloud is not sufficiently safe to host critical software. There is a need for more-sophisticated research on development of middleware that would increase application availability. This middleware should be based on multiple independent public cloud systems in order to reduce the risk of application outages caused by infrastructure failure.

Acknowledgements

The research presented in this paper was partially supported by the Polish Ministry of Science and Higher Education under AGH University of Science and Technology Grant 11.11.230.015 (statutory project).

References

[1] Aggarwal V., Sengupta S., Sharma V. S., Santharam A.: A Scalable Master-Worker Architecture for PaaS Clouds. In: *Proceedings of the 2012 SC Companion: High Performance Computing, Networking Storage and Analysis*, SCC '12, pp. 1268–1275. IEEE Computer Society, Washington, DC, USA, 2012. ISBN 978-0-7695-4956-9. http://dx.doi.org/10.1109/SC.Companion.2012.153.

[2] Buyya R., Broberg J., Goscinski A. M.: *Cloud Computing Principles and Paradigms*. Wiley Publishing, 2011. ISBN 9780470887998.

[3] Dhar S.: From outsourcing to Cloud computing: Evolution of IT services. In: *IEEE International Technology Management Conference*. 2011. http://dx.doi.org/10.1109/ITMC.2011.5996009.

[4] Ganore P.: *Advantages and disadvantages of SaaS and PaaS*. 2011.

[5] Gao J., Pattabhiraman P., Bai X., Tsai W. T.: SaaS performance and scalability evaluation in clouds. In: *SOSE*, J. Z. Gao, X. Lu, M. Younas, H. Zhu, eds., pp. 61–71. IEEE, 2011. ISBN 978-1-4673-0411-5.

[6] Hamiga M., Jarząb M.: An analysis of methods for sharing an electronic platform of public administration services using cloud computing and service oriented architecture. *Computer Science*, 13(4): 115–132, 2012.

[7] Hewitt E.: *Java SOA Cookbook – SOA implementation recipes, tips, and techniques*. O'Reilly, 2009. ISBN 978-0-596-52072-4.

[8] Hofmann P., Woods D.: Cloud Computing: The Limits of Public Clouds for Business Applications. *IEEE Internet Computing*, 14(6): 90–93, 2010. ISSN 1089-7801. http://dx.doi.org/10.1109/MIC.2010.136.

[9] Inc. A.: *Cloud Platforms vs. Cloud Infrastructure*. Tech. rep., 2009.

[10] Jansen W., Grance T.: *SP 800-144. Guidelines on Security and Privacy in Public Cloud Computing.* Tech. rep., Gaithersburg, MD, United States, 2011.

[11] Jeong H. Y., Hong B. H.: The Identification of Quality Attributes for SaaS in Cloud Computing. *Applied Mechanics and Materials*, vol. 300, pp. 689–692, 2013. http://dx.doi.org/10.4028/www.scientific.net/AMM.300-301.689.

[12] Ju J., Wang Y., Fu J., Wu J., Lin Z.: Research on Key Technology in SaaS. In: *Proceedings of the 2010 International Conference on Intelligent Computing and Cognitive Informatics*, ICICCI '10, pp. 384–387. IEEE Computer Society, Washington, DC, USA, 2010. ISBN 978-0-7695-4014-6. http://dx.doi.org/10.1109/ICICCI.2010.120.

[13] Kemp C., Gyger B.: *Professional Heroku Programming.* Programmer to programmer. Wiley, 2013. ISBN 9781118508992.

[14] Kim W., Lee J.H., Hong C., Han C., Lee H., Jang B.: An innovative method for data and software integration in SaaS. *Comput. Math. Appl.*, 64(5): 1252–1258, 2012. ISSN 0898-1221. http://dx.doi.org/10.1016/j.camwa.2012.03.069.

[15] Myerson J.: *Cloud computing versus grid computing.* 2009.

[16] Plummer D. C., Smith D., Bittman T. J., Cearley D. W., Cappuccio D. J., Scott D., Kumar R., Robertson B.: *Gartner Highlights Five Attributes of Cloud Computing.* (Vol G00167182), pp. 1–5, 2009.

[17] Ren K., Wang C., Wang Q.: *Security Challenges for the Public Cloud.* pp. 69–73. 2012.

[18] Rhoton J.: *Cloud Computing Explained: Implementation Handbook for Enterprises.* Recursive Limited, 2009. ISBN 0956355609.

[19] Sotomayor B., Montero R. S., Llorente I. M., Foster I.: Virtual Infrastructure Management in Private and Hybrid Clouds. *IEEE Internet Computing*, 13(5): 14–22, 2009. ISSN 1089-7801. http://dx.doi.org/10.1109/MIC.2009.119.

[20] Tingting L., Yong Z.: A Decentralized Information Flow Model for SaaS Applications Security. *2013 Third International Conference on Intelligent System Design and Engineering Applications*, vol. 0, pp. 40–43, 2013. http://dx.doi.org/http://doi.ieeecomputersociety.org/10.1109/ISDEA.2012.17.

[21] Tsai W.T., Sun X., Balasooriya J.: Service-Oriented Cloud Computing Architecture. In: *Proceedings of the 2010 Seventh International Conference on Information Technology: New Generations*, ITNG '10, pp. 684–689. IEEE Computer Society, Washington, DC, USA, 2010. ISBN 978-0-7695-3984-3. http://dx.doi.org/10.1109/ITNG.2010.214.

[22] Vouk M. A.: Cloud Computing – Issues, Research and Implementations. *CIT*, 16(4): 235–246, 2008.

[23] Wang R., Chen S., Wang X.: Signing Me onto Your Accounts through Facebook and Google: A Traffic-Guided Security Study of Commercially Deployed Single-Sign-On Web Services. In: *Proceedings of the 2012 IEEE Symposium on Security*

and Privacy, SP '12, pp. 365–379. IEEE Computer Society, Washington, DC, USA, 2012. ISBN 978-0-7695-4681-0. http://dx.doi.org/10.1109/SP.2012.30.

[24] Wei Y., Blake M. B.: Service-Oriented Computing and Cloud Computing: Challenges and Opportunities. *Internet Computing, IEEE*, 14(6): 72–75, 2010. ISSN 1089-7801. http://dx.doi.org/10.1109/mic.2010.147.

[25] Yang G., Zhou F., Zhu Z.: The Application of SaaS-Based Cloud Computing in the University Research and Teaching Platform. In: *Proceedings of the 2011 International Conference on Intelligence Science and Information Engineering*, ISIE '11, pp. 210–213. IEEE Computer Society, Washington, DC, USA, 2011. ISBN 978-0-7695-4480-9. http://dx.doi.org/10.1109/ISIE.2011.19.

[26] Youseff L., Butrico M., Da Silva D.: Toward a Unified Ontology of Cloud Computing. In: *Grid Computing Environments Workshop*, 2008. GCE '08, pp. 1–10. 2008. http://dx.doi.org/10.1109/GCE.2008.4738443.

Affiliations

Michał Soboń
AGH University of Science and Technology, Krakow, Poland, mikesobon@gmail.com

Piotr Nawrocki
AGH University of Science and Technology, Krakow, Poland, piter@agh.edu.pl

Received: 8.02.2013
Revised: 26.09.2013
Accepted: 20.12.2013