Migration of an on-premise application to the Cloud: Experience report

Pavel Rabetski and Gerardo Schneider

Department of Computer Science and Engineering
Chalmers University of Technology, and the University of Gothenburg
Gothenburg, Sweden
rabeckijps@gmail.com, gerardo@cse.gu.se

Abstract. As of today it is still not clear how and when cloud computing should be used. Developers very often write applications in a way that does not really fit a cloud environment, and in some cases without taking into account how quality attributes (like performance, security or portability) are affected. In this paper we share our experience and observations from adopting cloud computing for an on-premise enterprise application in a context of a small software company. We present experimental results concerning a comparative evaluation (w.r.t. performance and cost) of the behavior of the original system both on-premise and on the Cloud, considering different scenarios in the Cloud.

1 Introduction

Cloud computing refers to a utility-based provisioning of virtualized computational resources over the Internet. Even though *computing as a utility* is not a new concept [15], it has only recently become commercially available due to new technological shifts in virtualization, distributed computing and communication technologies. From a long-held dream cloud computing has turned into a new promising trend of the IT industry that is about to change the way computational resources and software are designed and purchased. Bottery et al [3] believe that the emergence of cloud computing will fundamentally transform the economics of the multi-billion dollar software industry. Strategy consulting firm AMI-Partners predicts that small business spending on cloud computing will hit $100 billion by 2014 [7]. Despite such promising predictions, there is a big confusion among potential adopters as cloud computing is not mature enough. Indeed, it is not clear what cloud computing is and when it is convenient to use it [1]. According to the Gartner report [6], cloud computing will become the preferred option for application development only around 2015, despite initial growth. Moreover, the lack of standards and keen competition on the new market has led to a variety of idiosyncratic cloud platforms. Cloud giants like Amazon, Google, Microsoft, and SalesForce are trying to establish their rules and promote their franchise. Choosing a proper cloud provider additionally complicates the migration planning, especially for smaller companies that do not have resources for extensive research on cloud computing.

The main objective of this work is to analyze what it means to migrate an on-premise application to the Cloud and what are the consequences of the migration. We perform our study on a specific industrial case study described in detail later in the paper. Our main contributions are: 1. The migration of an industrial enterprise web application to the Cloud; 2. Experiments concerning performance and costs of the migration. Based on our experimental results we draw conclusions on the consequences of the migration and provide suggestions on how to extrapolate our experience to other software systems.

The rest of the paper is organized as follows: Section 2 gives necessary background information. Sections 3 and 4 describe the migration of an industrial enterprise system to the Cloud, and our experimental results. Section 5 presents related work, and the last section summarizes the results.

2 Cloud computing

In this section we give a definition of cloud computing along with its key characteristics, and we describe two existing cloud classifications.

Cloud computing usually refers to a utility-based provisioning of computational resources over the Internet. Widely used analogies to explain cloud computing are electricity and water supply systems. Like the Cloud, they provide centralized resources that are accessible for everyone. Also, in the Cloud you only pay for what you have used. And finally, resources are usually consumed by those who have difficulties to produce necessary amounts by themselves or just do not want to do that. Despite the description by analogy, it is difficult to give a unique and precise definition. The definitions proposed are often focused on different perspectives and do not have common baselines. Vaquero et al [20] gives a definition highlighting three features that most closely describe cloud computing: *scalability*, *pay-as-you-go utility model* and *virtualization* [20].

There are two widely used cloud computing classifications. The first one describes four cloud types depending on the *deployment location*: public, private, community, and hybrid clouds [13]. The second classification is a widely used cloud ontology describing three cloud models depending on the *provided capabilities* [22]: i) Infrastructure as a Service (IaaS), ii) Platform as a Service (PaaS), and iii) Software as a Service (SaaS). It is also called a *cloud stack* because they are somehow typically built on top of each other. They can exist independently or may co-exist. Due to the lack of standardization it is not very clear where the exact boundaries lie between the components of the cloud stack.

- *IaaS.* The infrastructure layer represents fundamental resources that are the basis for the upper layers. It is very similar to a regular virtual server hosting. IaaS is built directly on the hardware, providing virtualized resources (e.g. storing and processing capacities) as a service. Examples of public IaaS providers are Amazon Web Services and GoGrid.
- *PaaS.* This layer is usually built on top of IaaS. The platform layer provides a higher level software platform with extended services where other systems can run. It delivers a programming-language-level environment with a set of

Advantages:	no upfront investments; on-demand capacity; focus on core applications; potential for more sales; easier customer maintenance; platform-provided features
Challenges:	security and privacy; availability; performance; compliance requirements; vendor lock-in; environment limitations (e.g. sandbox); multi-tenancy and licensing

Table 1. Summary of advantages and challenges of cloud computing

language-integrated APIs for implementing and deploying SaaS applications. Microsoft Azure and Google App Engine are examples of PaaS.
- *SaaS.* The services exposed in this layer represent alternatives to locally running end-user applications. They are usually interesting for a wide market, compared to IaaS or PaaS. They can also be composed from other services available in the Cloud. Normally, SaaS applications are accessed through web-portals for some fee. Microsoft Office365 or Gmail are examples of SaaS.

3 Case study: Migrating DC system to the Cloud

We describe here the migration of an enterprise system to the Cloud. We follow the migration process suggested in [19]. First, we describe the current system implementation. Then, we describe the new cloud architecture for the migrated application along with identified compatibility issues. We also suggest several system improvements to further leverage the cloud environment.

3.1 Preliminary analysis

Before doing the migration we have done a careful analysis of advantages and disadvantages of cloud computing. A summary of the results is presented in Table 1. In addition to that, we have studied existing public cloud platforms, namely, Amazon Web Services, Google App Engine, and Microsoft Azure in order to find the most suitable one. See [16] for a more detailed description.

3.2 Current DC implementation

InformaIT company *InformaIT* is a small independent software vendor (ISV) that focuses on document management systems. Most of the systems are based on Microsoft products and technologies. Being an innovative company, InformaIT is very interested in modern IT trends. The *Document Comparison system* (DC) was selected as a candidate for experimenting on the migration of applications to the Cloud. DC is a small web-based enterprise solution that enhances document management processes. The main purpose is to provide a fast and easy way to compare textual and graphical content of different digital documents.

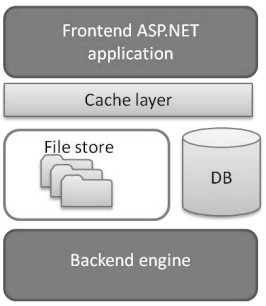

Fig. 1. DC components

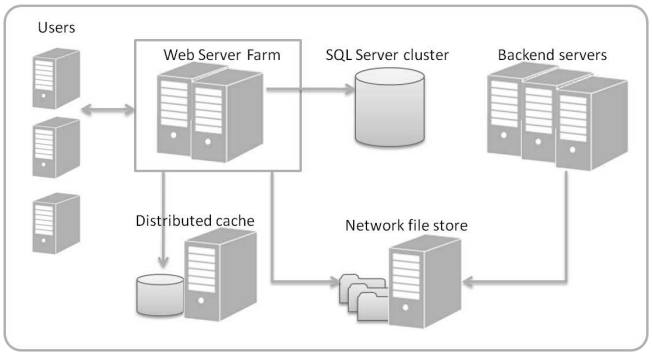

Fig. 2. On-premise distributed deployment model of DC

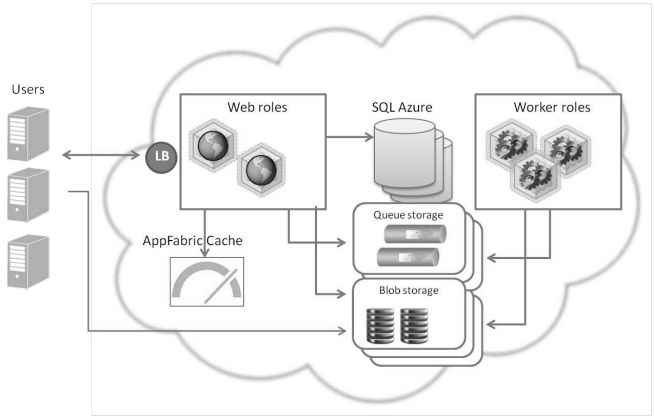

Fig. 3. Cloud deployment model of DC

DC architecture The system is implemented using Microsoft .NET 2.0 framework and various programming languages including server-side C# and C++, and client-side JavaScript. DC contains five main components: i) frontend web application, ii) backend engine, iii) distributed cache, iv) database, and v) shared file store (see Fig. 1).

The frontend is a simple ASP.NET web application running under IIS on Windows OS. It provides web interfaces for end-users, so they can upload digital documents, change configuration settings, analyze the result, and generate reports. The frontend extensively uses ASP.NET session state to track processed information and a current user status.

The backend is implemented as a Windows service (.NET based). It performs long running computational tasks e.g. the rasterization of digital documents. A special commercial library is used to fasten this process which accesses the files via regular file system API. It also requires the registration of a COM component.

The file store serves as a shared storage for system components. It keeps persistent data and organizes asynchronous communication between the frontend and the backend.

The cache layer keeps frequently used data, which increases the performance of the system. For example, the frontend stores the latest document comparison result there.

Unlike many document management systems, DC is not database-centric. The amount of data in the database is quite small and is used infrequently.

Deployment model DC is deployed on servers located in the data centers of customer organizations. This means customers have to take care of the infrastructure, and have technical personnel to maintain it. The amount of required hardware depends on the amount and the complexity of processing data. A single server is usually enough for small companies, while big organizations need several machines to run the system. DC also requires preinstalled Windows Server 2003/2008 with Microsoft SQL Server.

An on-premise distributed deployment model of DC is shown in Fig. 2. ASP.NET applications are composed into a Web Server Farm. They store frequently used data in a distributed cache that is usually located on a separate server. Backend engines are deployed separately as well. They require more powerful servers for heavy computations. A customer can choose the number of frontend and backend servers to achieve the required performance. A shared network folder plays the role of persistent file storage. Microsoft SQL Server is used as a database. End-users are usually located in the same environment where the system is running

This on-premise deployment model gives several advantages. First, it keeps data and code physically close. It results in very low latencies and no bandwidth limitations. Second, sensitive data never goes outside the organization, which provides a high level of security. In some cases when users need to access the system outside the organization, a VPN connection is established to keep the transferred data protected.

Organizations are charged per installation depending on the number of users. There are different types of licenses available, including a personal license and a concurrent license.

Motivation for the migration There are several disadvantages of running DC on-premise in a customer environment. We briefly discuss here some of them as well as the benefits of migrating the application to the Cloud.

The biggest opportunity is the potential for more sales. DC is currently oriented to big and medium organizations that have enough resources, own infrastructure, and technical personnel to install and run the system. Furthermore, the license cost is quite high. Potential customers such as small companies cannot afford DC, facing too big financial commitments. Some of them would like to use the system inconstantly and pay only for the amount of compared data. SaaS version of DC can bring the product to such customers.

A cloud computing advantage would be easer installation and upgrade procedures. The system is currently distributed across many customers. InformaIT

has to convince each customer to replace an on-premises package and then assist during the actual upgrading. Some customers still run older versions of the system, which brings an additional support overhead. The simple maintenance model of cloud computing will help to distribute resources more efficiently, leading to cost savings and business agility.

3.3 Suggested cloud DC architecture

Developers usually face a range of alternatives when implementing cloud-based systems. In this section we describe the chosen approach for our case and discuss different alternatives that can affect cost, architectural quality, and the amount of required changes.

Choosing a cloud provider The first step when moving an on-premise application to the Cloud is to choose a proper cloud provider. We examined three major cloud providers (Amazon Web Services, Google AppEngine, and Microsoft Azure). Based on our finding we conclude that Google AppEngine is the worst candidate for DC because it does not support .NET applications, while Amazon AWS and Microsoft Azure both fit for the migration quite well. After further analysis we prefer Windows Azure to Amazon AWS for several reasons: i) it requires less configuration effort, ii) it has a faster deployment model, and iii) it allows consistent development experience for applications that are well-versed in Microsoft technologies.

Cloud DC architecture Once we have chosen a public cloud provider, we need to show how existing architectural components are mapped to abstractions provided by the platform. In our case this platform is Microsoft Azure.

The frontend. Azure Web Role is an obvious choice for our ASP.NET frontend. Web Role has a preconfigured IIS and a built-in load balancer for web applications. Still, there are some limitations to keep in mind. For example, the Azure load balancer is not sticky, meaning that two requests from the same user can be processed by different Web Role instances. Also, Web Role supports only IIS 7.0 and requires .NET 3.5/4.0.

The backend engine. The backend maps to a Worker Role, since it suits perfectly for long running background tasks. It is worth noting that roles do not have administrative privileges in the environment. It restricts the execution of tasks that change OS configuration e.g. registration of a COM component or changing OS registry.

The distributed cache. Microsoft Azure has only one service for distributed cache so far, AppFabric Cache. Alternatively, cached data can be stored in either SQL Azure or regular Azure Storage. Though AppFabric Cache is considered to have better performance compared to the alternatives [16], it is quite expensive and limited in size (4GB maximum). We choose AppFabric Cache under the assumption that the size of data stored in cache will be significantly reduced. Otherwise we suggest using Table Storage.

The database. On-premise DC version uses a Microsoft SQL Server database. We find SQL Azure to be a perfect cloud alternative. In most cases switching

Compatibility issue	Required modification
Current solution uses .NET 2.0 and VS2005 that are not supported by Microsoft Azure. The platform uses the latest product versions.	The system should be migrated to .NET 3.5/4.0 and VS2010. This modification is quite simple due to full backwards compatibility of .NET 4.0 and 2.0.
The system cannot register COM components directly from code due to environment limitations.	There are some workarounds that allow using COM components for Azure applications: Registration-Free COM [18] and role startup scripts. We suggest using startup scripts because it is the easiest solution.
A local folder cannot be shared across Azure roles. Furthermore, suggested Blob Storage and Queue Storage have APIs that are not compatible with regular file APIs currently used by DC.	Change the code for accessing data in the file storage to use Blob Storage and Queue Storage APIs. Azure Drive is an alternative solution that eliminates these changes.
Standard ASP.NET session state modes do not suit Azure environment. In-Process mode is not an option because of a non-sticky load balancer.	The system needs distributed session storage in order to scale. We suggest using AppFabric Cache (or optionally Table Storage). Microsoft Azure offers an easy way of using these storages.

Table 2. Identified compatibility issues

to SQL Azure is as simple as changing the connection string. In [8] it is argued that SQL Azure can become a bottleneck for systems that concurrently operate large amounts of data. However, it is not the case for DC.

The file store. We have found out that the local file storage is not persistent and cannot be shared with other roles. All data stored locally gets lost if the role dies. The only persistent option for Azure applications is Azure Storage. We suggest using Queue Storage for messaging and Blob Storage for the files shared among roles. This approach leverages the cloud platform as much as possible. First, all data are automatically replicated and scaled. Second, Azure Storage can be accessed directly via REST calls, reducing the load on the frontend. Last but not least, Queue Storage provides a built-in reliable communication mechanism.

Fig. 3 presents a proposed deployment model of the system in the Cloud.

Identified compatibility issues Even though Microsoft Azure fits well for the migration of DC, we have identified some compatibility issues that require changes in the current implementation. These issues are described in Table 2. In what follows, we recommend some design modifications in order to tune system performance, increase portability, and make the migration as smooth as possible.

Separate data layer from business logic layer. InformaIT wants the system to be easily portable across both environments. However, this is not easy to achieve because of the need to switch from regular file system API to Azure Storage API. We suggest separating a data access layer from a business logic layer in order to increase portability. In other words, instead

of using APIs directly, a business logic layer calls a data access layer interface. This loose coupling allows using regular file system or Azure Storage depending on the deployment environment.

Become as stateless as possible. Large amount of cached data will not only degrade the performance but also increase the cost. An additional 1GB of AppFabric cache costs around 100$, which is 1000 times more than Azure Storage cost. The bigger the session size, the more time required to serialize/deserialize it. DC currently stores megabytes of data per a session, which is a big overhead. We suggest reducing the amount of cached data, making DC as stateless as possible. This suggestion can be applied for any web application that extensively uses session data.

Extensively use logging. Logging is very important for cloud applications, since debugging is impossible in the cloud environment. Logging helps developers to trace the behavior of the system and determine the reason of system failures. Furthermore it might be useful for identifying the level of resource utilization or just collecting statistical information.

4 Experiments

In this section we present experiments concerning cost and performance of running the DC application on the Cloud (under different conditions), and we compare those results with the on-premise implementation of DC. We are not concerned here with other issues as security and privacy.

In what follows we describe the environment these experiments are performed in. Experiments and measurements are done for North Europe deployment location of Microsoft Azure. This is the geographically closest location to the client testing environment located in Sweden (Gothenburg). All Azure compute instances have a small size, which provides 1.75 GB memory, 225 GB local disk space, moderate I/O performance, and CPU performance equivalent to one 1.6GHz core. We use small instances as a part of Azure free trial subscription, which gives necessary resources to perform our experiments for no fees. Testing on the client side is executed in a non-virtualized environment, external to the Cloud, with a direct connection to the Internet via a high-speed wired Ethernet. However, the cloud deployment location and the client environment are changed for some experiments. All experiments are performed at least 100 times to confirm that the results are stable.

4.1 Performance

As we identified earlier, a cloud environment entails increased latencies and unknown hardware underneath. Therefore, DC can have the following performance bottlenecks in the Cloud: the execution of heavy computational tasks (like digital document rendering) that require efficient hardware; and session handling that is latency sensitive. These operations represent the highest risk when moving DC to the cloud environment, because they might lead to significant system performance degradation.

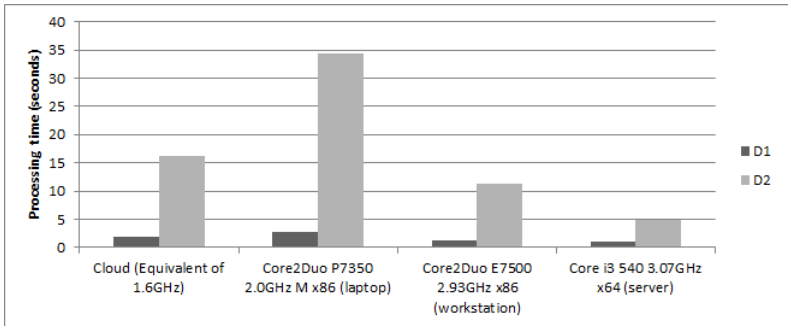

Fig. 4. Execution time comparison for cloud and on-premise environments

Execution time We have analyzed a production set of documents in order to suggest testing data for this experiment. We have classified two dominant types and picked up one document of each type (we reference to them as D1 and D2 accordingly). We then execute CPU heavy code for both documents and compare run time for cloud and on-premise DC versions. For a cloud version we use a small Azure compute instance (that has CPU performance equivalent to 1.6GHz), while on-premise installations have Core2Duo P7350 2.0GHz M x86 (laptop), Core2Duo E7500 2.93GHz x86 (workstation), Core i3 540 3.07GHz x64 (dedicated local server). Fig. 4 illustrates the results of our experiments. We have observed notably worse performance of one DC instance in the Cloud rather than on the dedicated server with powerful Intel Core i3 CPU (16.1 sec compared to 4.9 sec for D2). This means the system needs about three times more instances of the backend engine in the Cloud to achieve the same throughput.

Note that in contrast to the on-premise version where all files can be stored locally, cloud application needs to download and upload files to Blob Storage in order to process documents. However, it turned out that download and upload time together never exceeds 13% of total run time. Thus, our conclusion about computing capacity in the Cloud is still relevant.

Session storing/retrieving time In this section we compare on-premise and cloud DC session handling performance and also test two alternatives in Azure platform. For on-premise installation we evaluate standard ASP.NET in-process and state server modes. In-process mode stores session state data in memory, while state server mode uses a special process (separate from the ASP.NET worker process) for it. For cloud installation we evaluate AppFabric Cache, and a custom session handler that uses Azure Table. Session handling is very important for the frontend ASP.NET application, because it retrieves and stores session data on every page load as a part of the ASP.NET application lifecycle.

After putting an object into session, we measure the time it takes to load and save the session when handling an http request. We perform this experiment against different storages and object sizes: 1Kb, 1Mb, and 10Mb (assuming that session should not exceed 10Mb). Every object contains randomly generated

	On-premise DC installation		Cloud DC installation	
Session size	In-process	State server	AppFabric Cache	Table Storage
1 Kb	0.0/0.0	0.0/0.0	0.004/0.008	0.094/0.113
1 Mb	0.0/0.0	0.008/0.009	0.098/0.143	0.292/0.548
10 Mb	0.0/0.0	0.161/0.173	0.435/0.583	1.167/1.861

Table 3. Storing/retrieving time in seconds for session data

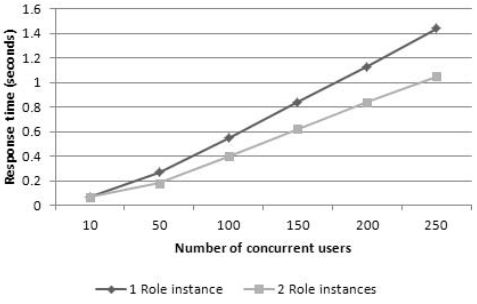

Fig. 5. Cloud DC page response time

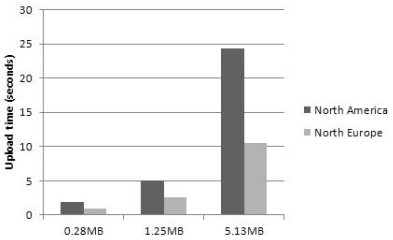

Fig. 6. Response time file uploading

binary data. It is worth noting that serialization time depends on the number of objects stored in the session. In our case there is only one object. We also use the local Web server for state server mode, while a remote Web server would considerably increase session handling time.

Experiment observations are presented in Table 3 where we can see that on-premise DC requires significantly less time for session handling compared to the cloud installation. In-process mode is obviously the fastest, since all data is kept in memory all the time. However, when data is stored in another location like AppFabric Cache, it should also be serialized and de-serialized accordingly. We have observed that AppFabric Cache shows considerably better performance than Table Storage, especially for small amounts of data. It is approximately 3 times faster for 1Mb and 10Mb cases, and 17 times faster for 1Kb case (4/8ms compared to 94/113ms). Consequently, DC can have close to on-premise performance in the Cloud when operating smaller data amounts (kilobytes) stored in AppFabric Cache. Table Storage increases response time by 1.167+1.861, that is approximatively 3 sec, when storing 10Mb of data in session. On the other hand, it is much cheaper and has no capacity limits. Table Storage also shows a lower correlation between the time and session size, apparently caused by HTTP latencies to transfer the data.

Response time We have also tested response time of the frontend web application against different deployment locations and different scales. We try to reflect the actual time from the end-user perspective, because perceived response time dictates user-friendliness of the service. Response time can be decomposed into four parts: the latency to send a request from a client; the time to redirect

the request by a load balancer (if there are several role instances); the time to process it by the application; and the latency to get a response back from the server. Even though these factors depend on network locality and traffic congestion, the main purpose is to show the difference in response time depending on different conditions. Variable conditions in our experiments are deployment location, number of role instances (scale), and load. In order to measure response time purely for a Web Role, we use a stateless .aspx page that does not include any external factors like session handling or document page rendering.

The first experiment evaluates page response time for a different number of role instances. The page makes some calculations and then generates dynamic output content. This dynamic data is needed to ensure that the page is not cached by any CDN service or in the client environment. We perform the experiment with a variable number of simulated clients accessing the service concurrently. In order to measure response time, we use Visual Studio 2010 Load Test[1] based on a Web Test that simply requests the page. All testing is done from outside the Cloud. We run the Load Test for a period of five minutes and perform it many times to confirm that the results are stable.

Fig. 5 shows the observed response time for both single instance and dual instance setups with an increasing number of concurrent users. Page time starts at about 80 ms for both cases and then grows linearly with different angular coefficients. Results show that an additional role instance decreases response time, especially for a heavy load. For 250 concurrent users a dual instance setup performs 400ms faster than a single instance setup.

The main goal of the second experiment is to show the difference in response time across different deployment locations. To do so, we execute Visual Studio 2010 Web Test that uploads a document to DC that is running in the cloud environment. This scenario reflects latency and bandwidth in a better way. We have picked up three random files of different sizes from the production document set: 0.28Mb, 1.25Mb, and 5.13Mb. The experiment is executed for two deployment locations: North America and North Europe, with testing performed in Sweden (Gothenburg). We repeat the experiment multiple times to confirm that the results are stable. The observed response time is presented on Fig. 6.

All three cases show approximately twice faster uploading time for North Europe zone compared to North America zone. The biggest file (5.13 MB) is uploaded to the first zone for 10 sec, while the second zone requires almost 24 sec. Consequently, a proper deployment location can significantly improve user experience by reducing interaction latencies.

4.2 Cost

In this section we estimate the cost of DC in the Cloud. For this purpose we model two real life scenarios that describe how cloud DC can be used. The cost for every scenario is estimated based on the Microsoft Azure pricing model.

[1] `http://msdn.microsoft.com/en-us/library/ee923688.aspx`

	Scenario 1		Scenario 2	
Service	**Used capacity**	**Cost ($)**	**Used capacity**	**Cost ($)**
Compute Instance	11 small instances (7920 hs)	950	3-11 small instances (3920 hs)	470
Relational database	1 GB	9.99	1 GB	9.99
Storage	500 GB	75	500 GB	75
Storage transactions	5000k transactions	5	5000k transactions	5
Data transfer	1000 GB	150	1000 GB	150
AppFabric Cache	512 MB	75	0-512 MB	55
Total:		1264.99 (1084.99)		764.99 (664.99)

Table 4. DC estimated cost for Scenario 1 and Scenario 2

	Compute instance	**Storage**	**Data transfer**	**Storage transactions**	**Cache**	**Database**
Scenario 1	75%	6%	12%	0%	6%	1%
Scenario 2	61%	10%	20%	1%	7%	1%

Table 5. Cost distribution for Scenario 1 and Scenario 2

Scenario 1: production installation In Scenario 1 DC is used as a production installation with throughput equivalent to one dedicated server (without elastic scale). For this scenario we require DC to show the same throughout as the on-premise installation that is running on a server with Core i3 540 3.07GHz x64 processor, 500 GB available local storage and 4GB of memory. It uses three out of four cores for the backend and the rest one for the frontend. As we observed earlier, the backend engine shows three times worse performance in the Cloud. That means we need nine small compute instances for the Backend. The frontend application requires two small compute instances, since we do not expect big performance degradation for the ASP.NET application. We also include 512Mb AppFabric Cache. We perform all calculations for a 30 days period which is equivalent to one month. So we totally need 30*24*11 = 7920 compute hours that costs 2160*0.12 = 950 US dollars. Data storage costs 500*0.15=75$; outgoing traffic is 1000*0.15 = 150$; 5 million transactions cost only 5$; and 1 GB SQL Azure is 9.99$. Table 4 presents the total cost for this scenario, and Table 5 illustrates the cost distribution. The total cost in brackets represents an upfront payment case (using a subscription). For more information see the official Microsoft Azure page.

Scenario 2: production installation with scaling In Scenario 2 DC is used as a production installation with throughput equivalent to one dedicated server (using elastic scale). In this scenario we use the same capacities as in Scenario 2, but leveraging cloud elastic scalability. We assume DC has a typical enterprise system load pattern: high load during working hours (10 hours from 8AM to 6 PM) and almost no load during the rest time. That means we can scale our

system down when the load is very low. We scale it down to three small instances to keep the system available. Also, the cache service is not needed when we have one Web Role. Assuming that there are 22 working days during a month we will need 30*24*3 + 22*10*8 = 3920 hours. The first term means that we need 3 instances all the time, and the second term means that we add 8 more instances during high load periods. The cache will cost 75*(22/30) = 55. However, using elasticity does not affect storage and outgoing traffic. The estimated cost is presented in Table 4. Table 5 shows the cost distribution among different services.

Based on our estimations we can conclude that compute services dominate in all scenarios. It makes up 75%, and 61% of the total cost for Scenario 1, and 2 accordingly. On the other hand, storage transactions have the least cost. SQL Azure also has a small cost share of 1%. However, this is because DC is not database centric. We found that the cost of DC can drop by 40 percent (764.99$ compared to 1264.99$) when leveraging elastic scalability. Even though choosing a proper scaling strategy is pretty straightforward for enterprise applications like ours, it might not be so trivial for other systems.

5 Related work

Some work have been presented on the benefits, challenges, and consequences of adopting the Cloud. Armbrust et al [1] described their vision of cloud computing, emphasizing elasticity as an important economic benefit. Motahari-Nezhad et al [14] added that cloud computing significantly reduced upfront commitments and potentially reduced operational and maintenance costs are also important benefits of cloud computing from business prospective. Chappel [4] elaborated on different opportunities that cloud computing brings to ISV, including the potential for more sales and easier customer upgrades. Kim et al [9] made and extensive research on cloud computing issues, emphasizing security and availability as the most challenging ones. Security and privacy seems to be one of the mostly discussed obstacles for cloud computing adoption [5][21].

Various papers evaluated existing cloud implementations. Rimal et al [17] made a comparative technical study of cloud providers and suggested taxonomy for identifying similarities and differences among them. Later, Louridas [12] discussed the migration of applications to the Cloud, examining key features of cloud offerings based on the taxonomy from [17]. Li et al [10][11] suggested a set of metrics related to application performance and cost in a cloud environment, comparing cloud providers based on these metrics. The authors concluded that none of the cloud providers is clearly superior, even though they observed diverse performance and cost across different platforms.

However, we have not observed many publications on the consequences of the migration that would include for example cost, performance, or security comparison. Tran et al [19] provided a simple cost estimation model for cloud applications, based on the identified influential cost factors. Babar et al [2] shared experiences and observations regarding the migration of an existing system to a cloud environment, which also included some guidelines and suggestions. Still,

none of the papers compared system behavior before and after the migration (or choosing different migration strategies), like we do in this paper.

6 Conclusion

In this paper we have shared our experience of cloud computing adoption based on a real case study from the industry.

We have implemented a cloud version of the on-premise enterprise application for Microsoft Azure platform. High compatibility with Azure and easy deployment were the main reasons for choosing this platform. The application cloud prototype was used to evaluate the performance and the cost of the system in a cloud environment. We have investigated the behavior of the system against different deployment locations, testing materials, scale and load. We could then make some extrapolations and suggest common practices based on our results. Our finding helped InformaIT to make a final decision regarding cloud adoption. Together with partners from InformaIT we have concluded that DC cloud implementation is feasible. We also found the estimated cost reasonable, especially when the system is dynamically scaled based on the load.

To our best knowledge there is no a unique metric that defines how well an application fits a cloud environment. The decision should be made separately for every system, based on the tradeoff between advantages and challenges. Existing systems are likely to face more challenges than new applications, due to the technological constraints of cloud platforms. In general, existing systems that are based on service oriented architecture with a focus on statelessness and low coupling fit the Cloud pretty well. Still, applications might require certain changes before being able to fully leverage a cloud environment. These changes are usually caused by environment limitations or the singularity of cloud storages. Based on our observations, the cloud version of a system is likely to show worse performance because of higher latencies and inferior computing hardware underneath. In order to tune system performance, we suggest eliminating unnecessary transfers between different system components, meaning both the amount of data and the number of calls. In particular, web applications should reduce the amount of data stored in session or become completely stateless; data intensive applications should also consider using local cache to store frequently used data. HPC applications will usually require more CPU cores (compute instances) in the Cloud to show the same throughput. Thus, such applications are likely to be costly. Last but not least, we suggest leveraging dynamic scalability in order to reduce the cost of a cloud application. This is especially important for systems with a changeable load. For example, enterprise application should scale up only during working hours; university web sites should scale up during application periods. However, monitoring is necessary when the load does not have a particular pattern. Furthermore, it might be ambiguous what metrics are the most relevant to monitor.

An extended version of the paper may be found online at `www.cse.chalmers.se/~gersch/ESOCC13-extended_version.pdf`.

References

1. M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. A view of cloud computing. *Commun. ACM*, 53:50–58, apr 2010.
2. M. A. Babar and M. A. Chauhan. A tale of migration to cloud computing for sharing experiences and observations. In *SECLOUD'11*, pages 50–56. ACM, 2011.
3. P. Botteri, D. Cowan, B. Deeter, A. Fisher, D. Garg, B. Goodman, J. Levine, G. Messiana, A. Sarin, and S. Tavel. Bessemer's top 10 laws of cloud computing and saas, 2010.
4. D. Chappell. Windows azure and isvs: A guide for decision makers, Jul 2009.
5. R. Chow, P. Golle, M. Jakobsson, E. Shi, J. Staddon, R. Masuoka, and J. Molina. Controlling data in the cloud: outsourcing computation without outsourcing control. In *CCSW'09*, pages 85–90. ACM, 2009.
6. M. Driver. Cloud application infrastructure technologies need seven years to mature. Research report, Gartner Inc., Stamford, USA, 2008.
7. A. R. Hichkey. Smb cloud spending to approach $100 billion by 2014, 2010.
8. Z. Hill, J. Li, M. Mao, A. Ruiz-Alvarez, and M. Humphrey. Early observations on the performance of windows azure. In *HPDC'10*, pages 367–376. ACM, 2010.
9. W. Kim, S. D. Kim, E. Lee, and S. Lee. Adoption issues for cloud computing. In *iiWAS'09*, pages 3–6. ACM, 2009.
10. A. Li, X. Yang, S. Kandula, and M. Zhang. Cloudcmp: comparing public cloud providers. In *IMC'10*, pages 1–14. ACM, 2010.
11. A. Li, X. Yang, S. Kandula, and M. Zhang. Comparing public-cloud providers. *Internet Computing*, 15:50–53, 2011.
12. P. Louridas. Up in the air: Moving your applications to the cloud. *Software, IEEE*, 27:6–10, 2010.
13. P. Mell and T. Grance. The nist definition of cloud computing. Technical report, National Institute of Standards and Technology, 2011.
14. H. M. Nezhad, B. Stephenson, and S. Singhal. Outsourcing business to cloud computing services: Opportunities and challenges. Technical report HPL-2009-23, HP Laboratories, 2009.
15. D. F. Parkhill. *The Challenge of the Computer Utility*. Addison-Wesley, US, 1966.
16. P. Rabetski. Migration of an on-premise application to the cloud. Master's thesis, Software Engineering and Management, Dept. of Computer Science and Engineering, Univ. of Gothenburg, Sweden, 2011.
17. B. Rimal, E. Choi, and I. Lumb. A taxonomy and survey of cloud computing systems. In *5th Int. Joint Conf. on INC, IMS and IDC*, pages 44–51. IEEE, 2009.
18. D. Templin. Simplify app deployment with clickonce and registration-free com, 2005.
19. V. Tran, K. Keung, A. Liu, and A. Fekete. Application migration to cloud: a taxonomy of critical factors. In *SECLOUD'11*, pages 22–28. ACM, 2011.
20. L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner. A break in the clouds: towards a cloud definition. *SIGCOMM Comp. Com. Rev.*, 39:50–55, 2009.
21. M. Vouk. Cloud computing - issues, research and implementations. *CIT*, 16(4):235–246, 2008.
22. L. Youseff, M. Butrico, and D. da Silva. Toward a unified ontology of cloud computing. In *GCE'08*, pages 1–10. IEEE, Nov 2008.