CHOOSING THE RIGHT CLOUD ARCHITECTURE
A Cost Perspective

Uwe Hohenstein, Reto Krummenacher, Ludwig Mittermeier and Sebastian Dippl
Siemens AG, Corporate Technology, Otto-Hahn-Ring 6, D-81730 Muenchen, Germany

Keywords: Cloud Computing, Cost-driven Architecture, Cloud Application Design, Windows Azure Platform.

Abstract: Cloud computing offers IT resources and services as a utility, and enables a much quicker move to market at much lower cost, arguably. The initial expenses for effort and hardware are indeed lower, and potential growth is much easier handled due to the inherited elasticity. However, applications in the cloud can cause significant operational costs - different from on-premises operational costs - and hence unpleasant surprises if not architected right. Cost factors should thus become much more of a core consideration when architecting for the cloud. Different scenarios that are discussed in this paper will show how different architectural decisions result in significantly different operational costs.

1 INTRODUCTION

Cloud computing has emerged to be the current highlight in terms of IT as a service. A smart idea is in principle enough to start a new business (Armbrust, 2010): no more need for large cost expenditure, no need for over-provisioning and wasting expensive resources, for not missing potential new clients. The main benefits of cloud computing, without going into technical details yet, are the elasticity and high availability of (at least theoretically infinite) hardware and software resources, the pay-as-you-go pricing model, and the self-service administration of the resources. In more economical terms, cloud computing has a very attractive benefit of turning CAPEX (capital expenses) into OPEX (operational expenses).

Still, none of these features, functional or non-functional, comes for free. A scalable architecture is essential for leveraging scalable cloud infrastructures (Hamdaqa, 2011), or in other words, simply deploying existing enterprise software into the cloud does not make the software any more scalable or cloud-enabled. Cloud architecture best practices are offered by most cloud utility providers (e.g., Amazon AWS (Varia, 2010)) or Microsoft Azure (Pace, 2010) with illustrations of how to design for failure, how to leverage elasticity, how to decouple components and parallelize etc. These important guidelines of how to bring existing and new applications to the cloud are common and valid for all cloud infrastructure offerings, although optimal software engineering decisions might certainly depend on the particular cloud utility for which one implements the cloud-enabled application.

There is, however, one important aspect, as we will argue throughout this paper, which is (too often) forgotten, when specifying solution architectures for the cloud: the operational costs of running an application in the cloud. In particular from an enterprise perspective, the maintenance and operations costs are highly relevant, and they should thus have a significant impact on design decisions, as we exemplify and discuss in this paper. The total costs of running an application are comprised of various individual sources such as the charges for compute instances, storage, bandwidth or different additional services. Depending on the cost model, one or the other individual cost source will dominate the overall bill, and reducing the total cost can only be done when minimizing the use of these dominating resources, already when defining the architecture. Consequently, when architecting for the cloud, cost factors need to be taken into account, and one might consider extending the "4+1 Architectural View Model" by (Kruchten, 1995) with an *operational cost view*. While a modular design helps to reduce maintenance costs and easy evolution, the operational cost view would enable an architect to illustrate the impact of the architectural decision on the overall expenses (Käfer, 2010a).

Hohenstein U., Krummenacher R., Mittermeier L. and Dippl S..
CHOOSING THE RIGHT CLOUD ARCHITECTURE - A Cost Perspective.
DOI: 10.5220/0003918803340344
In *Proceedings of the 2nd International Conference on Cloud Computing and Services Science* (CLOSER-2012), pages 334-344
ISBN: 978-989-8565-05-1

The similar line of arguments was expressed by Todd Hoff on HighScalability.com: “Instead of asking for the Big O complexity of an algorithm we'll also have to ask for the Big $ (or Big Euro) notation so we can judge an algorithm by its cost against a particular cloud profile.” (Hoff, 2009). It shows that while turning to cloud deployments cost-centric architectures becomes even more important, as the costs are more obviously accountable. Although, we are certainly by no means arguing that architectures should be determined by cost measures, we emphasize that discussions about architectural alternatives, about pros and cons with regard to costs have to be taken into account much more prominently when choosing the appropriate architecture for the cloud.

Unfortunately, existing work on cost-centric architectures is very few (cf. Section 5), and most publications and white papers rather relate to a Total Cost of Ownership (TCO) comparison between on premise and cloud deployments, not taking into account the actual architecture; at business-cloud.de, the Experton Group has published a TCO Calculator that helps in assessing the cost advantages of deploying in the cloud.

With this paper we want to counterwork this trend, and showcase with concrete examples how architectures impact the operational costs, once the decision to work in the cloud has been taken. As a technical basis for our work we are using the Windows Azure platform and the corresponding pricing models. The main reason for working with Windows Azure in the context of this paper is the comprehensive PaaS offering that ships with a complete development and deployment environment and various relevant by-products such as persistent storage, access control, or distributed cache. This has also the advantage that there are no problems with the licensing of such products, as these are part of the platform and the cost model. The latter, moreover, makes the calculation of the architecture-dependent costs much easier.

In order to clarify the baseline, we continue the paper with a short introduction to the core concepts of Windows Azure and its pricing model in Section 2. Then, we present two scenarios that are derived from real-world business cases, and based on that we will discuss and analyze the different architectural alternatives in Sections 3 and 4, respectively. In Section 5, we outline some related work with cost-centric aspects. During the practical part of our investigation, we detected some recommendations that are worth being reported on in Section 6, before the paper is concluded with Section 7.

2 WINDOWS AZURE AND ITS PRICING MODEL

In this section, we give a short introduction and overview of the core concepts of Windows Azure including the pay-per-use model. Pricing details reflect the status quo when writing this paper and will certainly change again in the future. However, regardless of the pricing details of a specific cloud computing platform, the baseline argumentation of this paper remains the same.

2.1 Core Concepts

Windows Azure provides virtual machines, so-called compute instances that run Windows Server 2008 and are available in two forms: a *Web Role* hosts an IIS (Internet Information Server) and is foreseen to provide the front-ends for web applications such as ASP.NET. In contrast, a *Worker Role* does not possess an IIS and serves mainly as a host for backend processes. The Web Roles offer different thread modes that can be configured, e.g., to have a thread pool with delegating each request to the next thread. In contrast to AWS, the compute instances are redundant with a built-in failover mechanism.

Both types of compute instances can initiate Internet connections, however, instances of Web and Worker Roles are not directly accessible via the Internet. All network traffic coming from outside to Web and Worker Role instances goes through a load balancer; each role can specify an endpoint configuration by which protocol (e.g., HTTP(S)) and by which port it should be accessible. Incoming traffic is routed to role instances in a round robin fashion. As a consequence, if there is more than one instance of a Web Role, subsequent requests will be routed by the load balancer to different instances. Therefore it is not an option to use the local file system of a Web Role for storing HTTP session data. Rather, the Azure storage mechanisms, which are table storage, queue storage and blob storage, need to be used for such kind of data that needs to be processed in subsequent requests. Similarly, SQL Azure, a managed SQL service in the cloud, can be used.

Azure table storage allows for storing data in a manner that is similar to tables, however, it does not enforce a fixed scheme; a row consists of a couple of properties and values, which are stored, without any predefined structure. Azure queue storage allows for FIFO-style message passing between role instances. Each message can be up to 64 KB in size. Finally, Azure blob storage allows for storing binary data such as images or videos, which can be annotated

with metadata. All the Azure storage services can be accessed via a RESTful interface; i.e., an HTTP protocol-based web API. This way, all programming languages with support for HTTP can use of the Azure storage capabilities, from inside the cloud or outside. Apart from that, the Windows Azure storage client library provides a more comfortable way for accessing the Azure storages.

An application built for Windows Azure runs in the context of a so-called hosted service, which defines for instance a public URL prefix as well as the geographical region. Windows Azure applications are uploaded (deployed) to the public cloud environment via the Azure web-based self-management portal to a specific hosted service, either to a production deployment or a staging deployment. The production deployment is accessible via the public URL of the hosted service whereas a deployment that is uploaded to the staging area is for testing purposes and thus only accessible via a URL generated by Azure. Staging and production deployments can be swapped without service downtime.

2.2 Standard Rates

The standard rates for Windows Azure can be found in *http://www.microsoft.com/windowsazure/offers/MS-AZR-0003P* as of January 2012.

Compute instances, i.e., Web and Worker Roles, are charged for the number of hours they are deployed. Even if a compute instance is used for 5 seconds, a full hour has to be paid. There are several instance categories, *small* (S), *medium* (M) etc. As Table 1 shows, the instance categories scale in a linear manner with regard to equipments and prices. That is, a medium instance has double of CPU, disk etc. than a small instance resulting in a double price. The exception is an XS instance category.

Table 1: Prices for compute instances.

	CPU	RAM	HDD (GB)	MBps	$ / h	I/O performance
XS	Shared	768MB	20	5	0.04	Low
S	1,6GHz	1,7 GB	225	100	0.12	Moderate
M	2 x	3,5 GB	490	200	0.24	High
L	4 x	7 GB	1000	400	0.48	High
XL	8 x	14 GB	2040	800	0.96	High

For Azure table, blob and queue storages, the costs depend on bandwidth, transaction, and storage consumption. Storage is billed based upon the average usage during a billing period of blob, table, and queue storage. For example, if 10 GB of storage are utilized for the first half of the month and none for the second half of the month, 5 GB of storage are billed for average usage. Each GB of storage is charged with $0.14 per GB. Storage consumption is measured at least once a day by Azure. Please note that the storage consumption takes into account the physical storage, which consists not only of raw data, but also the length of the property names, the data types, and the size of the actual data.

Moreover, any access to storage, i.e., any transaction, has to be paid: 10000 storage transactions cost $0.01. Bulk operations, e.g., bundling several inserts in one operation, count as one transaction.

All inbound data transfers to the Azure cloud are at no charge since Spring 2011. The outbound transfer to the North America and Europe regions is charged with $0.12 per outgoing GB, the Asia Pacific Region is more expensive. It is important to note that the transferred data has some typical XML overhead according to the protocol.

Data transfer is for free within the same affinity group, i.e., for compute instances that run in the same data center. The affinity group can be specified in the Azure self-service portal.

The costs for SQL Azure are also based upon monthly consumption. The Web Edition costs $9.99 per month for up to 1GB, and $49.95 for up to 5GB. The Business Edition allows for larger databases with similar prices: $99.99 per month for each 10GB (up to 50 GB).

Finally, we want to mention the Azure Access Control Service for authentication, which is charged with $1.99 per 100,000 transactions.

There are also some flat rates where a fixed number of compute instances is paid. For instance, a 6-month commitment (*http://www.Microsoft.com/windowsazure/offers*) mostly offers a 20% off rate for resources. In case the given quotas (e.g., 750 free compute hours) are exceeded, standards rates apply for overages. Furthermore, special offers exist for MSDN subscribers, BizSpark, or MPN members. Those specific rates are out of scope for this paper.

2.3 Special Quotas and Limits

There are some quotas active that define upper thresholds. For instance, every account may run 20 concurrent small compute instances (which is equal to 10 medium instances or 5 large instances) and possess 5 concurrent storage accounts, each having its own credentials for access. Higher numbers can be ordered, however, require negotiation with the Azure customer service. Besides this, there are a couple of technical restrictions such as the payload limit of 64 K for queues.

3 SCENARIO 1: MASS DATA STORE

This paper relies on two typical scenarios that occur quite often in reality. However, the scenarios were simplified in order to ease the discussion and to obfuscate the business details.

The first scenario is concerned with mass data storage. Several data providers (DPs) of given organizations provide data for a cloud-based mass data storage. The data in the cloud storage is used and processed by applications for analysis or other purposes; e.g., business intelligence or production process optimization. A more concrete example is a fleet management system that manages cars; each car sends data about its current state or position to a central cloud service. In this case the organizations are car fleets, the data providers are individual vehicles or fleet owners, and the collected data is processed further on to optimize fleet usage or the traffic management.

The following discussion is based upon several assumptions; most of them will be relevant for the presented cost calculations:

- There are 5 organizations with 20 DPs each for a total of 100 DPs. Each DP sends 10 data items à 1 KB per second to storage. Both the frequency of 10 items per seconds and the payload are assumed to be constant (varying loads are discussed later in Subsection 3.5). In summary, 1000 items are thus arriving per second (100 DPs * 10 items/s) for a total payload of 1000 KB/s.
- No data will be removed; there is an increasing amount of data maintained in storage.
- There is a transport latency from outside the cloud to the inside and vice versa, which might of course vary depending on the overall network congestion. However, the impact on the architecture is neglectably small since we assume an asynchronous HTTP communication link between data providers and the cloud storage. Hence, the DPs are just firing without waiting for a confirmation.

Please note the main purpose of this paper is neither to present a particular application and its costs, nor to define the cheapest architecture for such. The given numbers are (realistic) assumptions taken to calculate and *compare* occurring costs in the cloud. Moreover, it is not the intention to assert a certain type of architecture, rather we want to show how architectural choices can affect costs more or less dramatically. We also recognize that changing the assumed numbers and SLAs could lead to different costs and ranking of architectures. There are also further possible architectures that are not discussed.

3.1 The Web Role Approach

A couple of Web Roles (with threads running in it) receive data from all data providers, no matter of what organization. Threads of an appropriate number of Web Roles store data into organization-specific storages. Thus, every organization has some cloud storage of its own to maintain its data – not at last due to security considerations: an organization's data must not be accessible by other organizations. According to the incoming load, more or less Web Roles can be started, having the IIS load balancer in front of them.

For the purpose of this paper, we fix some further system parameters:

- Small compute instances are taken for the Web Roles.
- We assume that each Web Role can run 10 threads without system overload. This is a reasonable number that corresponds to Microsoft's recommendations (Best Practices, 2011). Our tests have shown that small Azure compute instances are already quite busy having 10 threads running.
- Referring to the benchmarks in (MS Extreme Computing Group, 2011), we assume that storing data from a Web Role into cloud storage is typically done in 30ms.
- For client authentication, authorization, and data pre-processing, some additional 40ms are assumed at the Worker Role, including a database access for getting the credentials.

Summing up, this means that the processing of each incoming storage request in a Web Role has some 70ms compute latency including all storage accesses. As a consequence, one Web Role thread is able to handle about 14 requests in a second. Handling the 1000 incoming items/s (10 data items per second from 100 data providers) thus requires minimally 70 threads. According to the assumption that each Web Role can run 10 threads, 7 Web Roles with 10 threads each are needed to handle the requested throughput; otherwise the IIS queue of the Web Roles will fill up, letting data providers experience more and more latency. With a constant load, the IIS queues will never be able to shrink, which moreover increases the risk of losing data.

According to (Microsoft Extreme Computing Group, 2011), any Azure storage solution should be able to handle a write throughput of 1000 items/s performed by 7 Web Roles with 10 threads.

As stated previously, essential for this paper are the operational costs of architectures. The monthly costs for this first solution are as follows:

- The complete inbound traffic to the Web Role is free of charge; since July 2011.

- Seven small Web Roles à 12ct per hour for 30 days cost $604.80.
- Table storage (no removal assumed) with a daily increase of 82.4 GB (1 GB/month à 14ct) results in further $178.81 if we consider the worst case that Azure monitors storage consumption at the end of a day: 82.4 GB for the 1st day, a total of 2*82.4 GB for the 2nd day etc. sum up to 38316 GB in a month.
- 1000 storage transactions per second lead to 2,592,000,000 per month à 1ct per 10000: $2592.

The total costs are $3375.61. A quick conclusion shows that transactions produce the main costs; Web Roles and the storage also affect the costs.

An aspect not yet discussed is access control and security. Authentication becomes necessary when working with Web Roles, as those are able to access all storage components directly. In this architecture, authorization/authentication can be performed by the Web Role, which is both an advantage and a disadvantage: On the one hand, this provides better flexibility. But on the other hand, an additional authorization/authentication component is required that incurs further costs, either for using Azure Access Control ($1.99 for 100,000 transactions) or implementing one's own component. Anyway, the Web Roles have access to all cloud storages since they serve all organizations.

Of course, the Web Roles can also perform some pre-processing, for example, extracting data from XML input, transforming data, or condensing data.

3.2 Queues at the Front-end

In an alternative architecture, each organization obtains one dedicated cloud queue at the frontend. Data providers of each organization then put data items directly into their respective queue using the provided REST interface for the queue storage.

Since there is no longer a front-end Web Role, authorization and authentication becomes an issue: it must be ensured that a data provider is only allowed to store in the queue of its organization. In Azure, the credentials are bound to a storage account, i.e., all queue or table storages belonging to the same account share the same credentials. This implies that each organization would require an account of its own as otherwise every DP would inherently get access to all queues. The quota of five storage accounts that are granted per Azure account are just sufficient for our example; otherwise additional storage accounts would have to be explicitly requested, however, without any further expenses.

Threads in a Worker Role pick up data items from the queues and transfer them to cloud storage. The number of requested Worker Roles (threads) depends on the time for emptying queues and on the required timeliness of data in cloud storage. In fact, the queue length must be close to empty, otherwise the queue will permanently increase since the assumed load is constant. If data must be up-to-date in cloud storage within fractions of a second, more Worker Roles (threads) are required to perform the transfer. However, data provider throughput is not throttled by a too low number of Worker Roles. There is no risk of data losses since queues are persistent, but an overflow might become critical.

We assume a queue read latency of 30ms and a typical storage write latency of 30ms according to (Microsoft Extreme Computing Group, 2011) for a total of 60ms. Then, one Worker Role thread is able to transfer an average of 16.67 items per second; 60 Worker Role threads distributed over 6 Worker Roles (because of the 1/10 Worker Role/thread ratio) are required to keep pace with each of the 5 queues being filled up with 200 items/s. It does not matter whether Worker Roles are assigned to specific queues or serve all queues.

Scalability with regard to incoming data is limited only by the queue throughput. The requested 200 items/s are easily achievable by Azure cloud queues according to (Microsoft Extreme Computing Group, 2011). If necessary, more queues could be set up, e.g., one for each data provider; however, to note again, the quota for storage accounts is five. The number of queues and accounts does not affect the total operational costs as only the queued data and the transactions are charged but not the number.

The monthly costs for such a queue-based architecture are computed as follows:

- Incoming requests to the front-end queue are again for free.
- The background storage costs remain at $191.58.
- The storage transactions for background storage are also the same $2592 as before.
- There are five newly introduced front-end queues with each queue getting in average 200 messages per second. As already mentioned, the Worker Roles will empty the queues in order to keep pace with the input stream. But even if there are 10 messages in the queue at any point in time, requiring 50KB storage (5 queues * 10 KB) over 30 days, results in the micro-costs of 0.0007 ct.
- There are three kinds of inbound and outbound transactions for the queues, one to read a message, another to store, and a third one to delete the message; Azure does not offer a mean to read and delete with one operation. This means enormous costs of $7776 = 3 * $2592.
- Six Worker Roles are used each for a price of 12ct

per hour for 30 days: $518.40.

Concluding the calculation, we quadruple the transaction costs from $2592 (Subsection 3.1) to now $10268 with the benefit of reducing 7 Worker Roles to 6 Web Roles and saving 86.40$ in a month for computation ($518.40 instead of $604.80). In addition, there are smallest amounts of costs for queue storage (0.0007ct). Hence, this architecture produces costs of $11077.98 per month and is thus about $7700 more expensive than the previous one.

Technically speaking, this architecture has some advantages. First, queues allow for more flexible reactions to load changes. Queues can fill up (without causing dominating costs) to be emptied at later points in time, during low load times, if no time critical data is involved. Consequently, an interesting alternative to the proposed setting could have the queues fill up due to fewer Worker Roles, and use – if cheaper – operating hours at night to transfer the data items to the backend storage. Having storage queues filled up does not call upon the same risks as IIS queues, as storage queues are persistent. Second, the architecture can rely on Azure queue authentication as a queue belongs to only one organization, and the data providers of one organization can only fill their organization's queue. However, authentication becomes less flexible.

As another disadvantage, additional implementation effort is required to set up the Worker Roles in a multithreaded manner. In contrast, multithreading is for free in Web Roles because of configurable instantiation models. Moreover, the transfer has to be fault-tolerant due to the lack of storage-spanning transactions, deleting data in the queue and inserting it into the backend storage. And the implementation must be able to determine what data from the queue can be skipped if a crash occurs after transferring to the backend but before deleting in the queue.

A further disadvantage of this architecture is the fact that the payload of queue messages cannot exceed a 64 KB threshold in Azure. Hence, if the payload is unknown or might increase, a complete redesign is required: one possibility is to use blobs for storing data, and to put a reference (URI) to the blob into the queue. This causes additional storage and transaction costs for blobs and an additional delay for data providers due to blob handling.

3.3 Bulk Operations

This architecture is based upon the previous one, however, attempts to reduce the number of expensive transactions by means of bulk operations. Azure provides to this end a mean to build bulks of operations of the same kind.

At the front-end, there is no opportunity unless the data providers collect data in bulks and submit bulks to the queues. However, bulk operations can be used during the internal processing: a Worker Role is able to fetch bulks of items from the queue, to remove them in bulks, and to submit bulks to the backend storage. This will in fact cause some delay in processing and lacks a little of less timeliness. Moreover, some implementation overhead occurs since it is necessary to wait for complete bulks. Some fault-tolerance is again required: a Worker Role might crash while just having cached a bulk. Data is not lost in that case since queues are persistent and still contain the data.

Even if the bulk size for queues is limited to 32 at maximum, it is possible to divide the transaction costs drastically by 32. However, the queue API offers only the possibility to get data in bulks, but not to delete bulks. Consequently, the cost for getting data from the queue can be reduced from $2592 to $81, but the other transactions stay at $2592. Moreover, bulk operations are possible for the backend storage. The table storage offers writing bulks operations of at most 100 entities and 4MB of size. This also reduces transaction costs from $2592 to $25.92 for retrieval. Compared to Subsection 3.2 the total transaction costs of $5290.92 (2*$2592 + $81 + $25.92) remain high.

As an alternative, table storage could be used instead of queues. It offers bulk operations even for reads, writes and deletes. The challenge now is to mimic the queue behavior. One possible way is to use table storage for each organization and the data provider's id as a partition key. Hence, it is easily and efficiently possible to fetch the eldest 100 data items for a given data provider (using the timestamp in a query), to store those items in the backend storage, and to remove them. In fact, there is some implementation effort, e.g., to be sure that a bulk of 100 is available in order to avoid polling, and to coordinate the Worker Role threads, i.e., who is accessing which table. The transaction costs for the Worker Role can be divided by 100 from $7776 to $77.76. This makes the solution a little cheaper than 3.1 since $86.40 for compute instances are saved.

3.4 Direct Access to Cloud Storage

Another approach gets rid of compute instances, i.e., Web or Worker Roles, in order to save costs. Data providers can store their data directly into blobs or tables; the post-processing applications then access the data provider's storage directly.

Both blob and table storage are possible in this type of architecture. However, blob storage has an important advantage over table storage: it offers fine-granular security rights. Blobs are stored in containers and the access rights of each container can be controlled individually even if the containers belong to the same account. In contrast, table stores of the same account share the same credentials. Hence, blobs are used in the following, each organization obtaining a container of its own. Note that the number of containers does not affect the operational costs.

The throughput depends on the access capabilities of blobs; the requested throughput of 200 items/s for each organization/container should be possible. Otherwise, additional accounts or containers have to be ordered.

The costs in the first month are here as follows:

- The data storage costs remain the same, and sum up to $191.58.
- The costs for storage transaction are still at $2592.

The conclusion is quickly made. With less than $2784 operational costs in the first month, this is the by far cheapest architecture – if applicable. The major benefit of this architecture is in fact the reduction of compute instances.

While financially the clear winner so far, technically this approach brings along several disadvantages. First, the backend storage is not shielded from data providers and the system fully relies on the authentication of the storage only. Furthermore, the same storage technology must be appropriate for both data providers and processing applications at the backend, but both might have different demands with regard to throughput or query functionality. If blob storage (or table storage alternatively) does not offer the requested functionality for backend applications, data will have to be transferred into an alternative cloud storage, which again requires additional Worker Role(s) and lets become the architecture similar to Subsections 3.2 or 3.3.

3.5 Load Variations

So far, we have discussed some constant load. We modify this assumption by assuming the same overall load per day, however varying over the day. For example, the load in a typical fleet management might be higher at 8-9 am and 5-6 pm.

Referring to Subsection 3.1, the IIS queues for Web Roles fill up during heavy load. The requested throughput must be handled by setting up additional Web Roles; the costs should be similar to a constant load if the data amount and transactions are the same over the whole day, i.e., there are less Web Roles at non-peak times. However, we pay a Web Role for one hour least. Such an hourly rate could produce higher costs! Moreover, we have to bear in mind the time for provisioning compute instances.

In Subsection 3.2, the front-end queues fill up, but no more Worker Roles are required since the queues are persistent. If there are timeliness constraints, i.e., if data must be mostly accurate in the back-end store, additional Worker Roles can reduce queues. One important question is whether the throughput of the front-end storage is enough. Well, there is still the opportunity to react on too high load with setting up more queues, which requires much effort if to be performed online.

The same holds for the architecture in 3.3: if the throughput of the front-end storage is not sufficient, higher load could be handled with more accounts.

Handling load changes by the number of Web/ Worker Roles, an hourly high load is more positive than arbitrary load changes since charging is done for full hours. In this respect, Worker Roles are more advantageous, because there is a chance of having less Worker Roles: input throughput can be handled over a long period of time without corrupting the required throughput. If the payment model offers a reduced overnight rate, there will also be a chance of using Worker Roles over night at less cost.

4 SCENARIO 2: DATA DELIVERY

The second scenario has been originally presented in (Käfer, 2010b). We suppose a large scale data delivery service being managed in the cloud: data is pushed into the system and is maintained in some central cloud storage. At the front end, customers expect to obtain their specific data from a cloud-based delivery service. An example could be found in logistics where post orders to a wholesale chain need to be collected, centrally managed and forwarded to individual suppliers and freight carrier services. In order to better model this scenario, we assume the backend storage to be filled once in the morning by some data provider for the purpose of a higher throughput. We again postulate some basic assumptions:

- There are 16000 clients receiving items: 0 items for 8000 clients (50%), 1 item for 3200 (20%) and 2 items for 3200 (20%), and 5 items for 1600 (10%). This sums up to 17600 items per day ((3200*1 item + 3200*2 items + 1600*5 items).
- Since each item has a payload of 50 KB, a total daily payload of 880000 KB is produced.

- Searching one item in the storage takes 300 ms even if none is found.
- 4000 clients all want to fetch their items at 8 am, 12 am, and at 5 pm; 4000 clients are equally distributed over the remaining times.
- As an SLA, clients should not wait longer than 1.5 seconds for being served.

4.1 The Web Role Approach

In the first architecture, several Web Role threads serve the clients: clients queue up in the load balancer in order to ask a Web Role for data: a Web Role thread accesses the storage to determine the data for that client and deliver the data while the client is waiting. The appropriate number of Web Roles depends on the number of clients and the given SLAs to clients.

We first focus on the three peak load times: the architecture has to serve 4000 clients at each peak time with clients of four types A to D:

A. 800 clients accessing 1 item (served within 300 ms)
B. 800 clients accessing 2 items (within 600 ms)
C. 400 clients accessing 5 items (within 1500 ms)
D. 2000 clients accessing 0 items (within 300 ms)

At first, we need to calculate the number of Worker Roles that are required to satisfy the SLA of being served within 1.5 sec. The number of Worker Roles obviously depends on the arrival of client. The lowest number of Worker Roles is required in the following situation:

- 400 times: a client of type C arrives and is served in 1500ms; each C client requires an own thread.
- 400 times: a sequence of client types B,D,D,D (the last client of type D finishes before 1500 ms)
- 400 times a sequence of B,D,D,A
- 80 times a sequence of A,A,A,A,A

This optimal schedule is rather unrealistic because it usually depends on the arrival and the load balancer. Even in this best case 1280 Web Role threads (400+400+400+80) are required all together. This results in 128 Web Roles with 10 threads each.

The costs can be calculated as follows for each peak time a day (there are three peaks a day):

- 128 Web Roles: Although the Web Roles are only required for 1.5 seconds, we have to pay for the full hour à 12ct/h, i.e., $15.36.
- Storage transactions are required for getting and deleting data. The costs are 0.8ct (4000 clients * 2 accesses * 1ct/10000).
- Outbound data transfer: 4400 items have to be delivered at each peak time for 3.15ct (4400 items * 50 KB * 15ct/GB).
- The backend storage is out of scope here.

Hence, we are charged $15.40 for each of the 3 peak times, i.e., $46.20 for all peak times. In addition, 1 further Web Role is needed for the remaining non-peak time of 21 hours:

- The Web Role costs $2.52 (21 hours * 12ct).
- Storage transactions (2 times for get/delete): 0.8ct (4000 clients * 2 accesses * 1ct/10000).
- Outbound data transfer: $2.97 (4400 items * 50 KB * 15ct/GB = 3.3ct).

The total costs are $51.70 ($46.20 + $5.50) per day.

The major disadvantage lies in the fact that every client checks periodically for newly received data even if none has arrived. This produces a lot of load which in turn requires Web Roles.

4.2 Storage-based Architecture

As a storage-based alternative, we introduce a client-specific storage: there is one account for each client in case of a table or queue store; using blob storage and client-specific containers requires only one global account for all clients, but client-specific credentials for containers. Worker Roles fetch data from the global storage and distribute the data to those client-specific storages. Clients remove their data from this storage right after pick up.

The client service time depends on the transport and access latency for storage. If one blob storage account is used for all client blobs, 4400 accesses occur at each peak time. In fact, according to (Microsoft Extreme Computing Group, 2011), the available throughput is enough to fulfill the SLA that the waiting time for being served does not exceed 1 sec.

The number of Worker Roles and their starting time is only important to deliver items before each peak time; obviously, starting Worker Roles earlier reduces the number of required Worker Roles. Furthermore, the delivery of messages into the global storage from outside is important. We here assume that data delivery has finished before any client wants to receive his data.

If one Worker Role performs a “full scan” on all incoming 17600 items once a day and assigns the items to the client storages, then retrieval takes less than 88 min (5280 sec = 17600 * 300 ms). If one Worker Role is started with 10 threads, then the Worker Role must start 9 minutes before the first peak time. Afterwards, all the items are distributed.

The number of Worker Roles and threads is mostly irrelevant, since a Worker Role is paid for each hour in use according to the payment conditions. The more threads (or Worker Roles) are applied, the later processing can start, which is of

few benefit only. However, a strategy to define which thread searches what becomes necessary.

Let us now calculate the overall costs for one day, assuming that client storages are filled in one step once a day, being ready for the first client.

- The number of Worker Roles is mostly irrelevant; it determines just when to start delivery at the latest: 1 Worker Role with 10 threads is enough and costs 12ct a day.
- Additional storage costs arise for the client storages from the beginning to the point a client fetches items. A precise calculation is unnecessary since even the worst case of keeping the client items the whole day is ignorable: storing 880000 KB to be delivered in the client storage for one day costs less than 14ct a month and 0.5ct per day.
- The additional outbound traffic from the backend store to the distributing Worker Role is for free, if the storage and the Worker Role are in the same data center region.
- The daily transactions (17600 gets and deletes à 1ct/10000 for the backend store, 16000 read accesses for storage in client storage) cost 3.36ct.
- The outbound data transfer from the client storage to the client costs 12.58ct (880000 KB * 15ct/GB) a day.

As a conclusion, there is an enormous cost reduction since much less Worker Roles are used than in Subsection 4.1: this type of architecture produces only 17ct per day instead of $51.

A variant of this architecture could transfer data by the Worker Role threads before each peak time: then less storage costs are consumed, but three Worker Roles are required per day. Hence, there is no benefit because of cheaper storage prices and more expensive Worker Roles. This approach might become useful if data will be delivered to the global storage several times a day.

5 RELATED WORK

A number of researchers have investigated the economic issues around cloud computing from a consumer and provider perspective. Indeed, (Armbrust, 2010) identifies short-term billing as one of the novel features of cloud computing. And (Khajeh-Hosseini, 2010) considers costs as one important research challenge for cloud computing. But only little research has been done in this direction.

(Youseff, 2008) discusses three pricing models that are used by cloud service providers: with *tiered pricing*, different tiers each with different specifications (e.g. CPU and RAM) are provided at a different cost per unit time. A large tier machine has better equipment but also has higher costs. *Per-unit pricing* is based upon exact resource usage; for example $0.15 per GB per month. Finally, *subscription-based pricing* is common in SaaS products such as Salesforce's Enterprise Edition CRM that charges each user per month.

(Walker, 2009) performs cost comparisons between cloud and on-premises. He states that lease-or-buy decisions have been researched in economics for more than 40 years. Walker compares the costs of a CPU hour when it is purchased as part of a server cluster, with when it is leased. Considering two scenarios – purchasing a 60000 core HPC cluster and purchasing a compute blade rack consisting of 176 cores – the result was that it is cheaper to buy than lease when CPU utilization is very high (over 90%) and electricity is cheap. The other way around, cloud computing becomes reasonable if CPU utilization is low or electricity is expensive. Walker focuses only on the cost of a CPU hour. To widen the space, further costs such as housing the infrastructure, installation and maintenance, staff, storage and networking must be taken into account as well.

(Klems, 2009) also addresses the problem of deciding whether deploying systems in a cloud makes economic sense. He discusses some economic and technical issues that need to be considered when evaluating cloud solutions. Moreover, a framework is provided that could be used to compare the costs of using cloud computing with an in-house IT infrastructure. Unfortunately, two presented case studies are more conceptual than concrete.

(Assuncao, 2009) concentrates on a scenario of using a cloud to extend the capacity of locally maintained computers when their in-house resources are over-utilized. They simulated the costs of using various strategies when borrowing resources from a cloud provider, and evaluated the benefits by using performance metrics such as the Average Weighted Response Time (AWRT) (Grimme, 2008), i.e., the average time that user job-requests take to complete. However, AWRT might not be the best metric to measure performance improvement.

(Kondo, 2009) examines the performance trade-offs and monetary cost benefits of Amazon AWS for volunteered computing applications of different size and storage.

(Palankar, 2008) uses the Amazon data storage service S3 for scientific intensive applications. The conclusion is that monetary costs are high because the service covers scalability, durability, and performance, which are often not required by data-intensive applications. In addition, (Garfinkel, 2007)

conducts a general cost-benefit analysis of clouds, however, without any specific application.

(Deelman, 2008) highlights the potentials of using cloud computing as a cost-effective deployment option for data-intensive scientific applications. They simulate an astronomic application named Montage and run it on Amazon AWS. Their focus was to investigate the performance-cost tradeoffs of different internal execution plans by measuring execution times, amounts of data transferred to and from AWS, and the amount of storage used. Unfortunately, the cost calculation is not precise enough because of the assumption that the cost of running instances on AWS EC2 is calculated on a per-CPU-second basis. However, AWS charge on a per-CPU-hour basis: launching 10 instances for 1 minute would cost 10 CPU hours (not 10 CPU minutes) on AWS. They found the cost of running instances (i.e. CPU time) to be the dominant figure in the total cost of running their application. Another study on Montage (Berriman, 2010) concludes that the high costs of data storage, data transfer and I/O in case of an I/O bound application like Montage makes AWS much less attractive than a local service.

(Kossmann, 2010) presents a web application according to the TPC-W benchmark with a backend database and compares the costs for operating the web application on major cloud providers, using existing relational cloud databases or building a database on top of table or blob storages.

Hence, (Deelman, 2008) and (Kossmann, 2010) are two studies that take roughly our direction.

6 RECOMMENDATIONS

We want to present a couple of recommendations that we derived from our investigation.

While a couple of papers such as (Deelman, 2008) have identified compute instances as the dominating cost factor, we have seen in the first scenario that transactional costs cannot be neglected. Here, bulk transactions could help, if applicable.

Indeed, compute instances are quite expensive, however, if compared to storage. Hence, one should try to minimize the number of compute instances, to allocate only instances when really needed, and to stop compute instances that are no longer needed. Strategies to adopt the number according to the recent load can help to react on varying load. But caution has to be taken since collecting performance and diagnosis data produces additional storage and transaction costs. Note also that stopped compute instances cause the same costs as running instances: an instance should be *deleted* to avoid running costs while still retaining the service URL.

It is also important that the costs for the staging area are the same as for the production environment. Hence, one should not forget to delete staging deployments between or after test phases.

Acquired resources should be used efficiently. For example, it is possible to run several web sites and web applications in one Web Role thanks to full IIS support. The more cores a compute instance has (determined by the instance category), the more parallel work a Web Role can handle. However, in most cases, it is better to use smaller instance categories such as XS or S: smaller instances offer a better scaling granularity while costs scale in a linear manner. In fact, there are also scenarios where a higher equipment such as 8 CPUs (XL), larger main memory, or bandwidth is reasonable, e.g., to allow multi-core programming to a larger extent.

If the load seems to be quite constant, some special offers such as a Subscription Offer (*http://www.Microsoft.com/windowsazure/offers*), enterprise agreements or long-term subscriptions might be a choice to save costs.

7 CONCLUSIONS

Based on Microsoft's Windows Azure platform offering, we have argued in this paper for the importance of taking operational costs into account when designing the architecture for a cloud-based offering. Given examples have shown the impact particular design decisions have on the cost of cloud applications; this is the important message of the paper. However, the paper by no means values one architecture decision over another but emphasizes the importance of considering the use and type of storage, compute instances and communication services already at early stage, in particular with respect to their impact on the operational costs.

The different architectural approaches and the resulting overall costs that at least partly diverge significantly reveal one important problem when it comes to migrating software to the cloud; the many dimensions of design decisions, and certainly the many dimensions of the pricing models. The pricing ladders not only differ between different providers in terms of charging units, special offers and free services, but already within the offers of single companies, cf. davidpallmann.blogspot.com on August 14, 2010: "The #1 hidden cost of cloud computing is simply the number of dimensions there

are to the pricing model. In effect, everything in the cloud is cheap but every kind of service represents an additional level of charge. To make it worse, as new features and services are added to the platform the number of billing considerations continues to increase". In other words, there is much more to cost-effective architecture design than choosing the number and size of compute instances, or deleting stopped staging deployments when not used.

While we have supported our arguments with tangible examples and experiences we have gained from working with Windows Azure, there is further work required towards concrete guidelines and best practices in cost-effective architecture design for the cloud; also taking into account further features such as Azure AppFabric Cache, special long-term subscriptions, and other cloud offerings such as for example Amazon AWS or Google AppEngine.

REFERENCES

Armbrust, M., Fox, A., Griffith, R., Joseph, A., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I. and Zaharia, M. (2010): *A View of Cloud Computing*. CACM, 53(4), April 2010.

Assuncao, M., Costanzo, A. and Buyya, R. (2009). *Evaluating the cost-benefit of using cloud computing to extend the capacity of clusters*. In HPDC '09: Proc. of 18th ACM int. symposium on High performance distributed computing, Munich, Germany, June 2009.

Calder, B. (2010). *Understanding Windows Azure Storage Billing – Bandwidth, Transactions, and Capacity*. http://blogs.msdn.com/b/windowsazurestorage/archive /2010/07/09/understanding-windows-azure-storage-billing-bandwidth-transactions-and-capacity.aspx.

Berriman, B., Juve, G., Deelman, E., Regelson, M. and Plavchan, P. (2010). *The Application of Cloud Computing to Astronomy: A Study of Cost and Performance*. 6th IEEE Int. Conf. on e-Science.

Deelman, E., Singh, G., Livny, M., Berriman, B. and Good, J. (2008). *The cost of doing science on the cloud: the Montage example*. In SC '08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing, Oregon, USA, November 2008.

Garfinkel, S. (2007). Commodity Grid Computing with Amazon S3 and EC2. In login 2007.

Greenberg, A., Hamilton, J., Maltz, D. and Patel, P. (2009). *The Cost of a Cloud: Research Problems in Data Center Networks*. ACM SIGCOMM Computer Communication Review, 39, 1.

Grimme, C., Lepping, J. and Papaspyrou, A. (2008). *Prospects of Collaboration between Compute Providers by means of Job Interchange*. In Proceedings of the 13th Job Scheduling Strategies for Parallel Processing, April 2008, Lecture Notes in Computer Science (LNCS), 4942.

Hamdaqa, M., Liviogiannis, L. and Tavildari, L. (2011): *A Reference Model for Devloping Cloud Applications*. Int. Conf. on Cloud Computing and Service Science (CLOSER) 2011.

Hoff, T. (2009). *Cloud Programming Directly Feeds Cost Allocation Back into Software Design*. Blog on HighScalability.com, March 6, 2009.

Käfer, G. (2010a): *Cloud Computing Architecture.* SEI Architecture Technology User Network Conf (SATURN) 2010. http://www.sei.cmu.edu/library/ assets/presentations /Cloud Computing Architecture - Gerald Kaefer.pdf

Käfer, G. (2010b): *Cloud Computing Architecture – How to reconcile business, technical, and legal requirements.* CloudConf 2010. http://cdn1.hlmc.de/ tl_files/cloudconf/Downloads/Downloads 17.11.2010 / Cloud Computing Architektur.pdf

Khajeh-Hosseini, A., Sommerville, I. and Sriram, I. (2011). *Research Challenges for Enterprise Cloud Computing*. 1st ACM Symposium on Cloud Computing, SOCC 2010, Indianapolis.

Klems, M., Nimis, J. and Tai, S. (2009). *Do Clouds Compute? A Framework for Estimating the Value of Cloud Computing*. Designing E-Business Systems. Markets, Services, and Networks, Lecture Notes in Business Information Processing, 22.

Kondo, D., Javadi, B., Malecot, P., Cappello, F. and Anderson, D. P. (2009). *Cost-benefit analysis of Cloud Computing versus desktop grids*. In Proc. of the 2009 IEEE international Symp. on Parallel&Distributed Processing, May 2009.

Kossmann, D., Kraska, T. and Loesing, S. (2010). *An Evaluation of Alternative Architectures for Trans-action Processing in the Cloud*. ACM SIGMOD 2010

Kruchten, P. (1995). *Architectural Blueprints – The "4+1" View Model of Software Architecture*. IEEE Software 12 (6), November 1995.

Microsoft Extreme Computing Group (2011): *All Azure Benchmark Test Cases.* Website: http://azurescope .cloudapp.net/BenchmarkTestCases/.

Microsoft (2011). *Best Practices for Developing on Windows Azure*. http://azurescope.cloudapp.net/ BestPractices.

Pace, E., Betts, D., Densmore, S., Dunn, R., Narumoto, M., and Woloski M. (2010). *Moving Applications to the Cloud on the Microsoft Azure™ Platform*. Microsoft Press, August 2010.

Palankar, M., Iamnitchi, A., Ripeanu, M. and Garfinkel, S. (2008). Amazon S3 for Sciene Grids: A Viable Solution? In: Data-Aware Distributed Computing Workship (DADC), 2008.

Varia, J. (2010). *Architecting for the Cloud: Best Practices*. Amazon Web Services, January 2010-2011.

Walker, E. (2009). *The Real Cost of a CPU Hour*. Computer, 42, 4.

Youseff, L., Butrico, M. and Da Silva, D. (2008). *Toward a Unified Ontology of Cloud Computing*. In Grid Computing Environments Workshop (GCE '08), Austin, Texas, USA, November 2008.