Application Splitting in the Cloud: A Performance Study

Franz Faul[1], Rafael Arizcorreta[1], Florian Dudouet[2] and Thomas Michael Bohnert[2]
[1]*Swiss Re, Zurich, Switzerland*
[2]*Zurich University of Applied Sciences, Winterthur, Switzerland*

Keywords: Cloud Computing, Performance, Database, Hybrid Cloud.

Abstract: Cloud-based deployments have become more and more mainstream in recent years, with many companies evaluating moving their infrastructure to the cloud, whether a public cloud, a private cloud, or a mix of the two through the hybrid cloud concept. One service offered by many clouds providers is Database-as-a-Service, where a user is offered a direct endpoint and access credentials to a chosen type of database. In this paper, we evaluate the performance impact of application splitting in a Hybrid Cloud environment. In this context, the database may be located in a cloud setting and the application servers on another cloud or on-premises, or the other way around. We found that for applications with low database latency and throughput requirements, moving to a public cloud environment can be a cost saving solution. None of the cloud providers evaluated were able to provide comparable performance for database-heavy database applications when compared to an optimized enterprise environment. Evaluating application splitting, we conclude that bursting to the cloud is a viable option in most cases, provided that the data is moved to the cloud before performing the requests.

1 INTRODUCTION

With the introduction of cloud technology, many new paradigms appeared in the the industry to exploit the concept of on demand, pay-as-you-go resources. Services do not need to remain in the on premise datacenter, but can be spread to an internal private cloud or externally among different public cloud providers. This new concept challenges the traditional IT environment and provides an unprecedented flexibility that pushes traditional data and compute services to their limit. In this paper the focus is on the impact of the Cloud paradigm in regards to the usage of Databases in a large enterprise context.

To measure and compare performance we investigate three different scenarios involving the usage of resources located in a cloud, either private or public. These scenarios essentially differ with the location of the database and compute power (i.e. internal computing resources and external database, as well as the counterpart with an internal database and external cloud computing resources). To achieve more meaningful results, these scenarios were run multiple times using resources from different cloud providers, either on premise (private cloud) or on their hosting locations (public cloud). The concept of Hybrid Cloud is further used to evaluate the feasibility of extending on-premises resources with remote ones from a public cloud provider, which is a technique known as Cloud Bursting. The main topic is not only the latency and bandwidth results but additinally the impact of differently sized cloud resources.

This study is done in the context of Swiss Re (Swiss Re, 2014), one of the largest reinsurance companys worldwide. The core asset of reinsurance is the know-how of past loss events in order to compute the price for new contracts and offerings. These events are extremely diverse and are all taken into account in the pricing algorithms, which explains why large amounts of data need to be fetched from databases and computed in order to achieve cost-optimized results. Exploiting the Cloud means that the data the computing power - or both - are moved to the cloud.

2 OBJECTIVES AND EVALUATION METHODOLOGY

The evaluation documented here has specific goals within Swiss Re but was aimed to make it generic and usable in other environments, which is why the essential focus was on public cloud providers using commonly available products. Additionally, private cloud evaluation is made using an OpenStack (The

OpenStack Foundation, 2014) based cloud, which is a readily available Open-Source cloud management software allowing for benchmark reproducibility in other environments. In all cases we use Commodity Hardware.

The tests were run from the following environments: Amazon Web Services, Google Cloud, the ICC Cloud Lab and the Swiss Re environment. Further the pricing models of two large cloud providers used for the tests are briefly described.

2.1 Benchmarking Methodology

Benchmarks need to be repeatable, observable, portable, realistic and runnable. What exactly is being tested as well as the possible limitations of the benchmarks have to be clear. In our context, the focus is on standardized and industry accepted test suites. As the benchmarking suite needs to run in the cloud provided application server, the scope was set to Java based applications. We identified two software suites which are both meet the requirements:

TPC-C: The most industry-wide accepted benchmarking method for TPC-Benchmark (TPC, 2014a) to evaluate Online Transaction Processing (OLTP) applications. TPC-C generates an easily comparable metric through the simulation of queries sent to a sample OLTP web application. TPC provides Java bindings to run TPC-C through a Java application, which makes it easy to integrate into our testing framework. There are other variants of TPC tests, such as TPC-E. We decided to choose TPC-C since the TPC-C tests are still more common as well as more write intensive (1.9 reads per 1 write) compared to the TPC-E ones (9.7 reads per 1 write). (Chen et al., 2011)

JMeter: Is a commonly used load-testing tool developed by the Apache group. It can be customized visually or through configuration files. JMeter is open-source and completely written in Java. As a very customizable tool, it is used to create specific test-runs adapted to Swiss Re in size and complexity.

While TPC-C benchmarks are made to simulate traditional applications, typically reproducing a warehouse storing problem, it was necessary to benchmark something closer to the real-life workloads in the Swiss Re environment to validate the generic results. In this regard, real life tests based on an application productively used in Swiss Re were built on top of JMeter. The application chosen provides a much more complex database structure. It is known what queries are the most frequent ones and which queries are more work intensive. These were chosen to benchmark the performance of the underlying infrastructure. The data used was randomly generated but followed the application guide lines.

The following section describes these two test suites and how they are used in the framework of our evaluation in detail.

2.2 Benchmark Details

2.2.1 TPC-C

To define industry-wide accepted transaction processing and database benchmarks, the Transaction Processing Performance Council (TPC) was founded as a non-profit organization in 1988. TPC-C was introduced by the TPC in August 1992 after a development process of more than two years (Levine, 2014a) (Raab, 2014).

TPC-C is a benchmark for OLTP workloads. It was developed based on the old TPC-A test, which did not work with multiple transaction types and had a less complex database and execution structure (TPC, 2014b). TPC-C was designed to accelerate the process of benchmarking by defining an industry-wide accepted standard that can be easily interpreted and compared. This was realized by working together with 8 vendors during the design process (Levine, 2014b).

The TPC-C Benchmark consists of read- and update-intensive transactions that simulate activities of a complex OLTP application (TPC, 2014). It measures the number of business transactions that create orders and uses it as comparable value when tested against different systems. The metrics for these tests are represented as successful business transactions per minute (tpmC).

The specifications for the TPC-C benchmark can be found on the official TPC website (TPC, 2014). The current version is 5.1.1 and was defined on February 2010. There are already several implementations available to use. OLTP Benchmark provides an out of the box implementation that allows the fine-grained setting of TPC-C configurations. This includes the distribution between the different queries used in TPC-C. In this setup, the default settings were used.

During the test, TPC-C simulates the OLTP workload of an artificial wholesale supplier company. The Database Setup consists of 9 separate tables as explained in figure 1. No views are used in the setup.

The test consists of five types of transactions: new order, payment, order-status, delivery and stock-level transaction. The most important type is the new or-

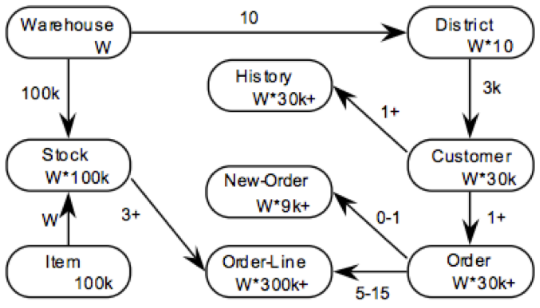

Figure 1: TPC-C Table Structure.

der, which enters a complete order in a single database transaction. It was designed to simulate the variable workload found in productive OLTP environments.

The payment transaction lowers the customer's balance and adds it to the warehouse sales statistics. Compared to the new order query it is lightweight. The order status transaction gets the status of the customer's last order. It is a read-only transaction that has a low execution frequency. The delivery transaction processes a batch of 10 new, not yet processed orders. It is not executed very frequently, and can take a little longer than the others, as each order requires a read and write operation. The stock-level transaction is used to determine the number of recently sold items that are now below a certain threshold. It is a heavy read-only transaction.

The fictional company has a configurable number of warehouses that serve a district. Each warehouse serves 10 districts. Each district again has 30000 customers. The warehouses keep stock for the 100000 different items the company sells. Each warehouse has a defined number of terminals. This is a representation of workers in the warehouses who process the orders made by the customers. (TPC, 2014)

2.2.2 JMeter as a Test Suite

Apache JMeter is an open source software, a 100% Java application designed to load test functional behaviours and measure performance. It was originally designed for testing web applications but has since expanded to other test functions (Apache, 2014).

The wrapper supports the remote start of different predefined tests that run in the JMeter application. The idea behind JMeter is that the user creates test cases (i.e. specific queries) which can then be executed on different machines. The results of these test cases are the metrics defined.

The tests were split up in single steps with different specifications.

All the tests are set up to have 10 Threads which run in parallel. The amount of query executions in a single thread varies per tests.

Table 1: Description of the different JMeter tests.

Test name	Description
Small	Simple query with a return of 1 row and a size of 773 Bytes
CCAS	Multiple queries on 5 different tables and return 30 rows of a total size of 1667 Bytes
RuleSet	Multiple queries on 3 different tables and return 864 rows of a total size of 46440 Bytes
StressTest	Simple query with a return of 3250 rows and a size of a total size of 17579 Bytes
Account	Simple query on one table with 8000 rows returned in a total size of 422936 Bytes

- 5,000 Small tests
- 10,000 CCAS tests
- 10,000 RuleSet tests
- 20,000 StressTest tests
- 5,000 Account tests

For the JMeter tests, a part of a database structure which is currently in use in an application at Swiss Re has been chosen. The content of the tables in the database structure is randomized. The database consists of approximately 60 tables and has a hierarchical structure. The structure can be split into two sections, the rule section and the reference data section. They are connected via mapping entities. On this overview all the tables which are used in the test queries and their relations are displayed. The JMeter test cases for all tests are performed with queries.

2.2.3 Application

The cloud deployable web application is written in Java 6 and optimized for Tomcat. All the interactions are done using three different servlets. One for triggering the TPC-C tests and two for the SQL and HTTP JMeter tests. The application is self-contained and can be easily deployed to the various cloud providers using provided APIs. The result of the tests can be downloaded directly through the application. Tomcat was chosen because it is the most prominent application server (Salnikov-Tarnovski, 2014). It is lightweight, available on all the selected cloud providers and is easy to install. The configuration is the same on all the environments (default configuration as of version 7).

MySQL is available on all the evaluated public cloud providers as a service (SaaS model) with the

Figure 2: High level table structure overview.

possibility of choosing the size of the VM backing the database. We chose to evaluate this service model as it is what a typical customer would do if they simply wanted a database running on the Cloud. On the other hand, the MySQL service was manually installed on the ICC Lab, as there is no MySQL-as-a-Service component on this purely IaaS Cloud. The default configuration as of version 5.6 was selected.

2.3 Expected Issues

It is expected that when connecting to the database through the internet, the throughput will be much less than in a LAN. This is not only due to the lower bandwidth, but also due to the higher latency between the application and the database. The latency has a high impact when, like a heavy database reliant application, many calls to the database are done. If packages could be grouped, the impact of the latency could be lowered and the performance increased. An example impact of latency can be found in table 2 showing expected impact.

Unpredictability of results is also one of the expected issues, which is why each test will be repeated several times over multiple days to evaluate the variance, if any, of our results.

These issues are expected, and it will come as no surprise that an application connecting to a distant database will be slower. Despite this, it is still important to measure how much these applications are slowed down and in which cases they remain usable and economically viable.

Table 2: Example calculation of latency impact.

# of SQL Calls	Location	Lat [ms]	Hours
1’000’000	Local	1	3
1’000’000	CH to CH	10	28
1’000’000	CH to NY	100	278
1’000’000	CH to SY	300	833

2.4 Tested Cloud Providers

The list of public clouds which will be evaluated is shown in table 3. For all compute nodes, meaning the nodes where the application server is running, we selected the standard size available. For the databases we chose different sizes to compare the throughput. In the AWS cloud, 3 different database tiers have been tested. For the Google cloud two have been chosen.

3 EXPERIMENTS

For the evaluation of the performances between the different cloud set ups, we evaluated three different scenarios, with a generic summary shown in Figure 3.

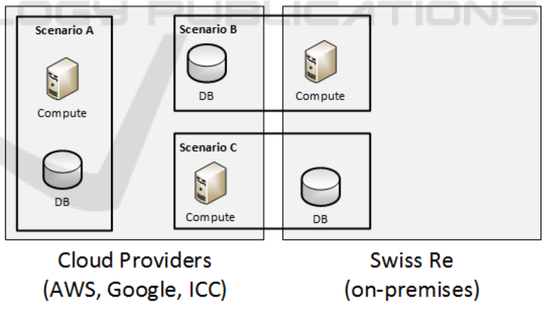

Figure 3: Test scenarios overview.

3.1 Test Scenarios

Scenario A is a simple comparison of the different public cloud providers. We execute the tests on different servers of each cloud provider with the database provided by the same cloud provider. With this scenario we can identify substantial differences between the cloud provider with the set ups we have chosen. We added a test where the compute and the database layer are on the same local machine.

In scenario B we connect from a WebSphere instance on premise to the different databases provided

Table 3: Considered Cloud Platforms.

	Tier	Zone	DB Version	Price	Network attachment
AWS: Database Small	db.t1.micro	Ireland	MySQL 5.6	$0.035/h	Very Low
AWS: Database Med	db.m1.medium	Ireland	MySQL 5.6	$0.120/h	Moderate
AWS: Database xLarge	db.m1.xlarge	Ireland	MySQL 5.6	$0.495/h	High
Google: D1	Std-D1-512MB	europe-west1-a	MySQL 5.5	$1.46/d	N/A
Google: D8	Std-D8-4GB	europe-west1-a	MySQL 5.5	$11.71/d	N/A

by the different cloud providers. With this scenario we want to see if outsourcing the database layer to a cloud provider is a viable option.

In scenario C a connection from AWS web servers to Swiss Re is made. This is not a direct database connection. The connection is made through a servlet which makes database calls and converts the results to JSON and sends those back as an answer. This scenario aims to evaluate the feasibility of using external compute power for compute-heavy applications without moving the data.

In each case, no other user-specific application is running during experiments.

4 RESULTS

4.1 Scenario A

As one can see from the test results of the cloud providers, ICC in the current setup is the best performing. The cloud the authors could work with in ZHAW is infrequently used yet and can be described as almost dedicated. It actually is a private cloud for researchers and students rather than a public cloud. The whole network in the ICC Lab environment is set up with 10Gbit Ethernet and not very saturated, which allows a very high throughput.

The communication in AWS and Google Cloud internally is not very different. This was an unexpected result as AWS promotes their bigger instances with better network connectivity. We concluded that this result is because the network is a shared resource and limited though Quality of Service (QoS).

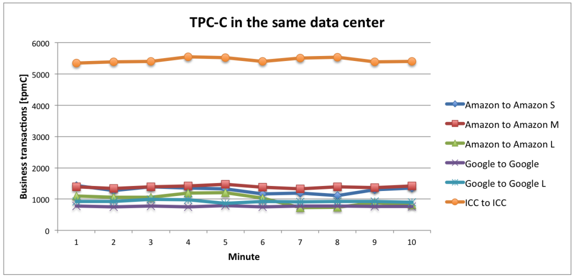

Figure 4: TPCC in datacenter.

4.2 Scenario B

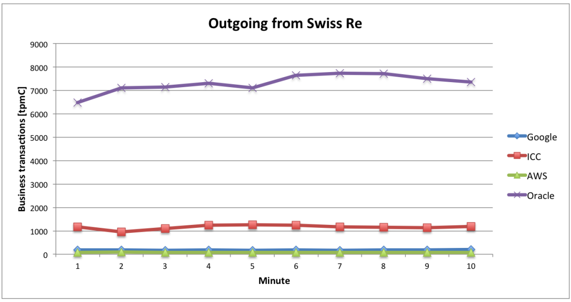

Figure 5: TPCC Swiss Re to different DB.

It can clearly be seen that when working with Swiss Re internal compute nodes and external databases the performance is almost unusable. The access through the Internet slows the performance and in this case it is the bottleneck. The tests were concluded without a Coarse-grained API concept, as an additional transformation from JDBC would have been required, this is not a service provided by cloud providers.

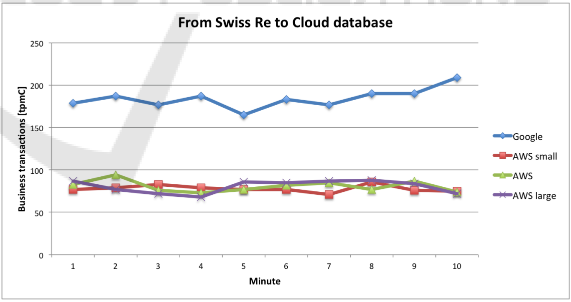

Figure 6: TPCC Swiss Re to Cloud DB.

As figure 5 does not show the difference between the cloud offerings well, the Oracle instance and the ICC lab were removed in figure 6.

The same pattern as in the test from ICC in regard to the AWS servers can be observed. The database tier did not make a difference when connecting from Swiss Re.

4.3 Scenario C

When very small packages are sent, the latency can vary depending on whether traffic shaping is activated

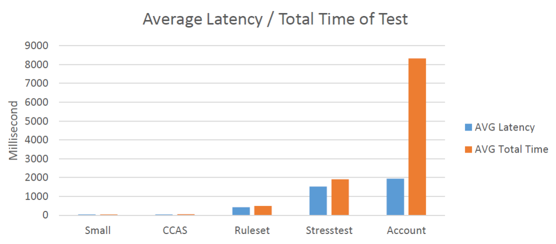

Figure 7: JMeter Proxy Test with AWS.

or not. Traffic Shaping is used to improve latency and optimize performance by delaying some packages and transferring them as a batch. The results of the different test cases from AWS to the Swiss Re database are compared, one can see that the latency and the total time deviate with bigger package size. This is expected, since the more data has to be transferred, the more packages have to be created and sent through the Internet.

4.4 Tests in Non-Swiss Re Environment

The TPCC tests have shown that in-cloud communications are still useable for web applications, but expanding a private cloud with compute power from outside and working with internal data causes a huge performance impact. The latency for packages The JMeter tests support the conclusions drawn with the TPCC tests. It also shows that traffic shaping is a bad idea when big batches of data have to be transferred.

4.5 Tests in Swiss Re Environment

Swiss Re has a stable and fast network, which has grown over the years with huge unused potential. This would be a good baseline for setting up a private cloud

The JMeter tests with the Servlet have shown that small data transfer is feasible with the application in the cloud and the database in Swiss Re. The response time is slower than within Swiss Re but is still short enough for applications with small data traffic. Interactive applications would run better than performance dependent batch calculations.

4.6 Performance

As expected, the performance in the cloud is weaker than the performance in the existing enterprise environment. The split-up of applications is even more critical than moving the combined application and database to the cloud. The overall performance for small applications that are not database intensive is still sufficient (See figure 4). When the application and the database are moved to a cloud as a package, even moderately database-dependent applications can be operated in a public cloud environment.

Applications that rely on the database heavily should be operated within the internal environment. This can be in a traditional IT setup or a private cloud. Current enterprise environments perform a lot faster than any of the tested cloud environments (See figure 4).

Connecting to a database hosted externally from an internal compute resource showed one of the biggest performance differences (See figure 6). The throughput declined drastically. If because of a burst in the on premise private environment, additional compute power from an external cloud provider is required, the transport time for the data has to be considered (See table 2). However, if the application itself relies heavily on the database, such as in model calculations the performance gain will not be drastic unless everything is moved to the cloud (See figure 4 and 6).

When connecting from outside of a cloud, the difference in performance of the various database tiers can almost be ignored (See figure 5). This only has an impact when connecting from the cloud itself, but also marginal (See figure 4). Tests showed that the application does not require all of the available CPU or memory, especially on systems with a slow network connection and it was able to run as many requests as possible.

5 CONCLUSION

5.1 Use Case

One reason to go to a public cloud is to lower operational effort. This can be done by letting the cloud provider operate the infrastructure. The enterprise just uses it as a service and take advantage of the cloud providers economics of scale (Urquhart, 2014). This would be mainly feasible for non-production applications, where performance and data security are not the main focus.

For short tests and new environments made of small applications, the cloud can help to keep Capital Expenditure (CAPEX) low, as no hardware needs to be acquired, and the pas-as-you-go model allows for short time bursting at a controlled price.

When moving to the cloud, it is mandatory to take into account the initial data transfer, especially if the use case involves temporary cloud usage, as the cost of transfer is high. Additionally, a process needs to be put in place to freeze the current database when moving it to the cloud. This is not always feasible for production databases which can not be stopped at all, in this case techniques such as ones used in VM Live Migration (Svärd et al.,) can be exploited to seamlessly transfer hard drive and live memory data.

If because of short bursts, additional compute power should be added to the private cloud, to create a hybrid offering, the throughput difference between the internal and the external compute nodes need to be considered (See figures 5 and 6). If the bursts can be foreseen, the data can be replicated to the cloud and the performance loss is not as high as if the compute nodes need to talk to the on premise database through a web service, as most company do not expose their databases to the internet. However, the additionally gained performance for database intensive applications will be drastically lower than the ones of the private cloud. As this would most probably be done only in the production environment, it currently is not recommended.

When the data is required in the cloud and on premise, an asynchronous data replication should be considered; this helps to keep the performance up but could lead to data mismatch until the data is fully transferred to the other location.

5.2 Suggestions

As a remainder, the suggestions below are made according to our findings in our tests and relative to the use case described in the introduction. Conclusions would be different if the application considered can be re-engineered, as we have focused on use cases where no changes can be performed on the database and a JDBC connection is mandatory between application servers and databases.

In essence, public cloud can be a cheap and fast alternative, especially with low data exchange volume. On the other hand, when performance is a strict requirement, we find that a private cloud with enterprise ready databases offers the best match.

When migrating to the cloud for development and testing, an important characteristic to consider is that the different tier types of databases do not have a very big impact. An evaluation has to be conducted internally to verify if the small performance benefit or larger offering is worth the additional price and resources.

The lower performance can be accepted for applications that run small queries, but when having database intensive applications the situation can lead to negative feedback of the users.

In web applications which have low traffic and in which the number of concurrent users is not too high, a replacement of the traditional infrastructure can be considered. For example a seasonal application, with a relatively small number of users, is a good candidate for migration. The benefit is that the application would cost less and with the few users, transaction time with the database of less than 1 second can be guaranteed. This time span is according to Nielsen (Nielsen, 1993), the maximum time allowed for an application with user interaction. This basic advice regarding response times has been around for the last thirty years.

Overall we conclude that development and test systems can be easily moved to the public cloud without a large impact on user satisfaction. Web applications with few database interactions could also be considered to be moved to a public cloud. Systems in a productive Environment which are time-critical should not be moved to a public cloud. Adding temporary computing resources for those is also not recommended.

REFERENCES

Apache (2014). Apache JMeter. Retrieved 2014-06-02, from http://jmeter.apache.org/

Chen, S., Ailamaki, A., Athanassoulis, M., Gibbons, P. B., Johnson, R., Pandis, I., and Stoica, R. (2011). Tpc-e vs. tpc-c: Characterizing the new tpc-e benchmark via an i/o comparison study. *ACM SIGMOD Record*, 39(3):5–10.

Levine, C. (2014a). Standard benchmarks for database systems - tpc-c overview. Retrieved from http://www.tpc.org/information/sessions/sigmod/sld007.htm

Levine, C. (2014b). Standard benchmarks for database systems - what is tpc. Retrieved from http://www.tpc.org/information/sessions/sigmod/sld 006.htm

Nielsen, J. (1993). *Usability Engineering*. US: Morgan Kaufmann.

Raab, F. (2014). TPC-C – The Standard Benchmark for Online transaction Processsing (OLTP). Retrieved from http://research.microsoft.com/en-us/um/people/gray/benchmarkhandbook/ chapter12.pdf

Salnikov-Tarnovski, N. (2014). Most popular application servers. Retrieved 2014-06-02, from http://java.dzone.com/articles/most-popular-application

Svärd, P., Hudzia, B., Walsh, S., Tordsson, J., and Elmroth, E. The noble art of live vm migration-principles and performance of precopy, postcopy and hybrid migration of demanding workloads. Technical report, Technical report, 2014. Tech Report UMINF 14.10. Submitted.

Swiss Re (2014). Swiss re homepage. Retrieved from http://www.swissre.com/

The OpenStack Foundation (2014). Openstack homepage. Retrieved from http://www.openstack.org/

TPC (2014). TPC Benchmark C - Standard Specifications. Retrieved from http://www.tpc.org/tpcc/spec/tpcc current.pdf

TPC (2014a). TPC benchmarks. Retrieved from http://www.tpc.org/information/benchmarks.asp

TPC (2014b). Tpc-c. Retrieved from http://www.tpc.org/tpcc/

Urquhart, J. (2014). Apache JMeter. Retrieved 2014-06-05, from http://www.cnet.com/news/james-hamilton -on-cloud-economies-of-scale/