NoSQL SaaS Comparison*

Extended Abstract†

J. McFadden‡
Institute of Technology
Univ. of Washington: Tacoma
Tacoma, WA 98402
mcfaddja@uw.edu

Y. Tamta§
Institute of Technology
Univ. of Washington: Tacoma
Tacoma, WA 98402
yashaswitamta@gmail.com

J. N. Gandhi¶
Institute of Technology
Univ. of Washington: Tacoma
Tacoma, WA 98402
jugalg.uw.edu

ABSTRACT

Two different No-SQL database cloud services, from two different providers are compared. These comparisons will be based, broadly, on the performance, features, usability, and cost of each service. The performance metric includes measurements of both data throughput and query latency under a variety of load conditions; while the usability metric will focus on the number of available means for interacting with the database, as well as the ease of utilizing those means. Along with comparing standard NoSQL database features, our features metric will also consider tools for interacting with the each database which available from the respective provider.

CCS CONCEPTS

•**Computer systems organization** → **Cloud based systems;** *Software as a Service;* •**Database systems** → NoSQL; •**Cloud Systems** → Perfomance; Cost;

KEYWORDS

ACM proceedings, LaTeX, text tagging

ACM Reference format:
J. McFadden, Y. Tamta, and J. N. Gandhi. 1997. NoSQL SaaS Comparison. In *Proceedings of TCSS 562: Cloud Computing Term Project, Tacoma, WA USA, April 2017 (TCSS 562: Spring 2017),* 4 pages.
DOI: 10.475/123_4

1 INTRODUCTION

Initially, we were going to do a software/systems comparison, with the software/systems chosen for comparison in this project being two different NoSQL database system. These systems would have been deployed/run/operated in several different ways, including *SaaS*[1] implementations, *containerized* implementations, and *native installations*. The goal of the project would have been to understand the performance characteristics of each deployment method ***and*** to quantify the costs of each deployment method, with the costs being calculated based on the hourly cost to operate, the initial time & costs required for setup, and the maintenance requirement of a deployment. Additionally, performance of the systems and deployments would have been measured using the time required to carry out various database operations, under a set of several different conditions, as well as the CPU, memory, and network loads imposed by the various deployments under the same set of conditions. As we proceeded with the project, however, we discovered that there was no way to properly cover all of these topics within a single project

1.1 Changed Questions

Ultimately, we decided that our initial approach was too complex and beyond the intended scope of the project; therefore, were limited ourselves to comparing DynamoDB from AWS and BigTable from Google's Cloud Services. We also changed our comparison of the services to be based on metrics measuring their performance, features, usability, and cost; while also setting a specific and narrow scope for each metric. The performance metric is limited to separately measuring both data throughput and query latency under a variety of load a replication conditions; while the features metric will concentrate on features available from each provider, such as provisioning and tools for interacting with their respective service. Our usability metric will focus on both the ease of implementing software to use each service, as well as the availability in terms of the number of supported languages, the number of available APIs & SDKs, and the availability of third-party software for interacting with each service. Finally, our cost metric will focus on the costs associated with the setup and operation of each service, however, it will also attempt to cover costs which might be encountered when implementing software making use of each service.

1.2 Service Change

Unfortunately, BigTable is difficult to deploy for even a simple test case, as it requires the configuration of multiple cloud services on Google's cloud and the installation of multiple Linux-only software packages, namely rBase & Hadoop. Thus, we were again forced to alter our plan, this time by swapping BigTable from Google for CosmosDB from Azure. This swap also lead to an additional feature to consider and condition to test under, the replication two or more of each cloud service's availability zones.

*Produces the permission block, and copyright information
†The full version of the author's guide is available as `acmart.pdf` document
‡
§
¶
[1]**SaaS :** Software as a service.

TCSS 562: Spring 2017, Tacoma, WA USA

DOI: 10.475/123_4

2 RESEARCH QUESTIONS

We decided on our metrics of performance, features, usability, and cost based on five research questions. Each question was designed based on one or two metrics choices from our first three metrics (performance, features, and usability), with cost being considered separately over all relevant questions. The research questions we developed are

1): How do the costs and provisioning implementations compare?

2): Are we getting what we pay for (*provision*) W.R.T.[2] capacity and space?

3): How efficient is use of the service W.R.T. latency and throughput?

4): How easy is it to implement connections in software or applications? How easy is it to push/pull data? What kind of tools are available from each provider for interacting with the database?

5): What kinds of replication are available and how are they accomplished?

2.1 Provisioning Costs and Implementations

When answering, "***How do the costs and provisioning implementations compare?***" we are interested in exploring two things. The first thing in which we are interested is each service's implementation of database provisioning, with specific attention being paid to how provisioning is characterized by each service. Specifically, we what to know how each service parameterizes the provisioning of databases in their NoSQL services. To cover this part of our first research question, we have the following "sub-questions" :

1 - A): Are reads and writes provisioned separately or together?

1 - B): What counts as a database "*item*"?

1 - C): Is there an assumption with regard to the size of database "*items*" or is that an additional provisioning parameter?

1 - D): Is the disk space required to store the database used as a parameter?

The second and last thing we wish to explore is the cost model used by each provider, especially how it relates to database use and data size. To cover the second part of this research question, we will add the following "sub-questions" :

1 - E): Are reads and writes charged at the same or different rates?

1 - F): Does the provisioned cost of database depend on its level of use?

1 - G): Does the size of database "items" affect the cost of provisioning?

1 - H): Is the disk space required to store the database included in the previous provisioning costs or does it constitute another, additional cost dimension?

2.2 Getting what we pay for

Given that using a NoSQL database on a cloud service incurs an hourly cost that can be significant, it is natural to wonder, "***Are we getting what we pay for (provision) W.R.T. capacity and space?***". To answer this question, we will test each service using two different methods of testing. In the first testing method, we will push a large dataset up to each database while measuring the throughput in **items written per second**. Similarly, in the second method, we will pull a large dataset down from each database while again measuring the throughput in **items written per second**.

2.3 Efficiency of throughput and latency

No matter the type of database, the primary metric of any heavily used database is latency of queries, measured in **ms**.[3] This led us to ask, "***How efficient is use of the service W.R.T. latency and throughput?***" as one of our research questions. To answer this question, we will run queries on the database under three different conditions with each condition representing a different level of load on the database service. These load levels are

3 - A): **None :** No load on the database other than the query being measured.

3 - B): **Half-load :** Pushing or pulling data to or from the database at half of the provisioned write or read capacity.

3 - C): **Full-load :** Pushing or pulling data to or from the database at the full provisioned write or read capacity.

2.4 Ease of use

The goal of having any type of database, regardless of database type or location, is to efficiently store data for retrieval by means of some software or application. Furthermore, in most cases, a database must be populated with data before it can be used, therefore there must be a way of initially populating the database after its creation. These considerations led us to ask, "***How easy is it to implement connections in software or applications?***" and, "***How easy is it to push/pull data?***" which we combined into our fourth research question. To answer these questions we considered the general areas of database APIs/Models, software SDKs/APIs, and tools provided by each service.

When looking at the Database APIs/Models area, we wanted to look at the database systems or data storage/retrieval models offered by each service. Specifically, we wanted to know how many

[2] W.R.T. = with respect to

[3] **ms** = milliseconds

different database implementations/languages each service supported or provided. As both services were NoSQL databases, we also wanted to see if either service offered any unique or interesting data storage/retrieval models. In-line with looking at different database languages, we wanted to look at what programing languages or SDKs could be used for writing software to connect to each database service. Importantly, we wanted to know how well developed the libraries and APIs were in each supported language and what, if any, costs would be incurred in order to use a particular supported language. Additionally, since the software or application connected to that database could also be provided by a third-party, we wanted to see what third-party software was available for each service.

Finally, we wanted to see what tools each cloud provider had available for interacting with their NoSQL database services, as well as compare the tool or tools offered by each service. When comparing the tools, we wanted to know if there were any costs associated with using them and if those costs were one-time (*buy the tool*) or if the costs would recur in some way. The second, and last, aspect of exploring the provided tools was a comparison of each tool's ease of use.

2.5 Replication

The final area we were interested in covered database replication, as replication of a database improves its reliability and can improve its performance. This lead us to ask, "***What kinds of replication are available and how are they accomplished?***" as our fifth and final research question. Finally, this question also led us to our first discovery. We found that **AWS's DynamoDB** does not offer replication as a single database between different AWS availability zones, while **Azure's CosmosDB** does offer it between Azure's different availability zones. As we will see when we describe the experiments we ran, this also led to us attempting to run experiments on **Azure's CosmosDB** with and without replication between different Azure availability zones.

3 EXPERIMENTS

The experiments we ran can be classified according to four distinct criteria, all based on one or more of our research questions. These four criterial are as follows :

1): The subject of the experiment, where the subject is which service is being tested and if replication is being used, based on the availability of replication from the service in question.

2): The primary metric of the experiment, with available primary metrics being either total data throughput[4] and query latency.

3): The throughput type being used or measured in the experiment, with the possible types of throughput being pushing data up to the database or pulling data down from the database.

4): The load conditions under which the experiment was conducted, where the load condition is characterized by the level[5] and type[6] of load.

3.1 Subjects

We had three different classifications for the ***subject*** of an experiment with one from AWS and two from Azure. These experimental subject classifications are as follows :

A): AWS's DynamoDB

B): Azure's CosmosDB ***without*** replication across availability zones.

C): Azure's CosmosDB ***with*** replication across availability zones.

3.2 Primary Metrics

There were two primary metrics used in this project, with each experiment only measuring a single primary metric. The primary metrics we used are :

A): Data Throughput, measured in terms of the number of database entries/items being read and/or written per second (**items / sec** or **R/U**).

B): Query Latency, which we measured in terms of the time (**milliseconds** or **ms**) required to return the result of a simple query[7].

3.3 Throughput Types

We considered just two types of Data Throughput and we only considered them separately[8]. These types of Data Throughput are as follows :

A): Pushing, where we pushed data up to the database.

B): Pulling, where were pull data down from the database.

3.4 Load Conditions

When running an experiment, we would use one of five distinct kinds load conditions, where each load condition is characterized in

[4] total data throughput = total number of database entries/items being pushed up to or pulled down from the database

[5] level = measured as a fraction or percentage of the total provisioned read and/or write capacity.

[6] type of load = pushing data up, pulling data down, or some combination.

[7] simple query = no joins, sorts, transforms, etc. just submit an index value and return the corresponding entry.

[8] "*considered them separately* = i.e. one or the other, not both

terms of two different parameters. The parameters used to characterize each load condition were the level of load[9] and the direction of the load. The types of load conditions we use are as follows :

- **A**): **None :** No load on the database other than the query being measured.
- **B**): **Half-load pushing :** Pushing data to the database at half of the provisioned write capacity.
- **C**): **Half-load pulling :** Pulling data to the database at half of the provisioned write capacity.
- **D**): **Full-load pushing :** Pushing data to the database at the full provisioned write capacity.
- **E**): **Full-load pulling :** Pulling data to the database at the full provisioned write capacity.

4 RESULTS

[9]level of load = as a fraction or percentage of the total provisioned I/O capacity