Cloud Service Benchmarking

AWS DynamoDB vs. Google Spanner

Denis Rangelov and Martin Dichev
Group H

I. INTRODUCTION

The purpose of this report is to summarize and evaluate the measurements collected during the performance evaluation process of two Cloud-based services - DynamoDB [4] and Google Spanner [5]. It reveals the benchmarking setup used throughout the project as well as the reasoning behind the selected benchmarking approach. Apart from the current passage, the report comprises of 5 more sections. The first one is *"Systems Under Test"* and illustrates the features that AWS DynamoDB and Google Spanner offer. In the *"Benchmark Design"* section, the main focus is placed on the design objectives, the selected metrics, the workload and the benchmarking setup. The sections that follow present the results collected when measuring the performance of the benchamarking targets (i.e. AWS DynamoDB and Google Spanner) as well as our interpretation of these results. The report concludes with a brief overview of the encountered challenges during the project and also includes an assessment of the degree to which we managed to accomplish the envisioned project goals.

II. SYSTEMS UNDER TEST

"If you know the enemy and know yourself, you need not fear the result of a hundred battles"
- Sun Tzu, The Art of War

Although the quote above is primarily intended as a playful and humorous way to open up the current section, there is a deeper meaning behind it. For the purposes of this writing, the idea of knowing your enemy can be translated as knowing well the systems one is trying to test. More specifically, when benchmarking a service is the main goal of a project, knowing the systems as close as possible is of crucial importance. Therefore, the current section is dedicated to summarizing some of the most well-known properties of DynamoDB and Google Spanner. Important to note here is that many of the properties of cloud services such as DynamoDB or Google Spanner remain "secret" for the general public. Therefore, despite our best efforts, it is possible that we might not be able to cover in much detail and with a complete accuracy how these systems operate behind the scenes. The information presented below was gathered by examining the reports from companies that actively use the services in question and also by reviewing the conference talks and papers the two service providers have published online.

III. BENCHMARK GOALS AND DESIGN

The main objective of this section is to briefly describe how we address the general benchmark design objectives, what performance metrics we decided to focus on and also what type of workloads we choose to utilize. That being said, the first important discussion to kickoff the current section is the benchmarking goal that we worked on throughout the project.

A. Benchmarking Goal

In terms of the benchmarking goal, we decided to aim our attention at the following research question(s):

- What is AWS DynamoDB and Google Spanner's write performance in terms of latency when both stores are introduced with update heavy workloads?
- How does the database record size affect the write latency of the two stores?

The decision to examine these benchmarking scenarios is based on the tendency that many of the modern web applications are handling a huge amount of OLTP workloads (i.e. insert/update/delete requests). Therefore, we decided to apply update heavy stress on the two benchmarking targets which - at least in our experience - is a workload that occurs often times in practice and in real-life use cases (e.g. updating the items in your shopping cart, changing the status in your social media profile, etc.). To be completely honest, the workloads used throughout the project are synthetic (more on that in later sections) which means that it is really impractical to associate the applied workloads with a specific application scenario. Rather, we will use the update heavy workload to test *"small isolated features"* [3] such as the write performance (especially for different record sizes) of the two data stores.

B. Benchmarking Tool

To help us answer the research questions mentioned above, we choose a benchmarking tool called *Yahoo! Cloud Serving Benchmark (YCSB)* [2]. Among the reasons that we decided to work with YCSB are:

- It has out-of-the box drivers for both DynamoDB and Google Spanner
- It offers a large number of predefined workloads to choose from
- It has a multitude of "switches and knobs" that we can turn on and off based on the needs of the project

- It allows us to scale the workloads effortlessly
- It has an awesome documentation

With this in mind, throughout the project we did some additional testing with workloads that are indirectly related to the research goals presented in the beginning of the section. The collected measurements allowed us to acquire a broader perspective of what are the strengths and weaknesses of the systems under test and for this reason we will briefly discuss them in the later sections of this report.

C. Benchmarking Metrics

As mentioned in the "Benchmarking Goals and Design" section, the goal of the experiments conducted in the scope of this project is to collect measurements of the write performance of DynamoDB [4] and Google Spanner [5]. To achieve that goal, as a performance metric we examined the write/update *latency* measured in milliseconds. It is important to mention here that we assumed a client's perspective throughout the benchmarking sessions. The reasoning behind this decision is not only based on the task description (it was suggested to do so in the assignment sheet) but also on the fact that this perspective includes the roundtrip time to the collected measurements. As mentioned in the "Cloud Service Benchmarking" book by Prof. Tai [3], the round trip time has an important impact on the overall latency (network delays are not to be underestimated) and therefore it was a logical choice to adopt the client's perspective in our benchmarking setup.

D. Benchmarking Setup

Our experience while working on this project, led us to the conclusion that when it comes to benchmarking cloud services, choosing the correct setup is one of the most challenging aspects of the task at hand. To be precise, there is a large variety of parameters that were relevant for the benchmarking experiments and could not be neglected. Among the most prominent ones (especially for DynamoDB and Spanner) are the geolocation of both - the benchmarking client and the SUTs, the provisioned resources and the data distribution. We will break down the each of these individually below:

Geolocation of the Benchmarking Client:

In the **ideal case**, the client should be geo-distributed so that the benchmark setup can look as close as possible to a real application scenario (e.g. multiple users in different parts of the world trying to access a web service). Unfortunately, the benchmarking client used throughout the project does not support geo-distribution, but instead it was deployed only in one region. The main reason for the usage of a non-distributed client setup is that we don't have the necessary coordination mechanisms among the YCSB clients, that can ensure that the distributed load will "hit" the SUT's in the intended way. One example that simultaneously supports and illustrates this claim is presented in the Cloud Service Benchmarking book [3], where the authors point out that for loads that try to target a specific range of keys (e.g. triggering a hot-key problem), using YCSB in a distributed manner might not actually produce the desired stress on the system because of the probability distributions YCSB uses when generating the workloads.

That being said, even with a non-optimal single client deployment, there is still one important point to be considered - which region to deploy the benchmarking client to? The short answer is that we deployed the benchmarking client on one of our local machines here in Berlin. The long answer, on the other hand, is that when we deployed the benchmarking client on a EC2 instance in another region, we observed a very strange behavior. In particular, the EC2 was giving an unfair advantage to DynamoDB (higher throughput) compared to Spanner. It is really difficult to make a conclusive statement, if this behavior is the norm, but it is very likely that services in the AWS ecosystem communicate with each other in a more efficient manner because of either software or hardware optimizations. For instance, one possible (although speculative) explanation for these results could be that AWS uses high speed interconnects between their data centers.

With this context in mind, we decided to use a more neutral environment for the benchmarking client and that is why we ran the client from our local machine.

Geolocation of the Benchmarking Targets:

Similar to the geolocation of the benchmarking client, in the **ideal case**, both SUTs should be deployed in a distributed fashion so that the benchmarking use case looks very close to a real life scenario. However, to deploy a fully geo-distributed Spanner instance (i.e. with replicas in Europe, Asia and USA), the cost is 9$/hour. Even with the free credits, since we ran the experiments for a relatively long time to ensure reproducibility, it was impossible for us to keep up with the enormous costs (we even used a full-fledged account with 300$ free credits but it was still too expensive to keep the Spanner instance running).

On the DynamoDB side, things did not go so well either. According to our research, the geo-distributed deployment of DynamoDB is called "Amazon DynamoDB global table". After trying to initialize one such instance, we got an Email by Amazon saying that there is *"insufficient replicated write capacity units"* for the free-tier and the student accounts.

Finally, since the geo-distributed deployment did not work as expected, we deployed both data stores in only one - the eu-west-2 (London) - region so that each one of them is equally far from the benchrmaking client's location.

Provisioning resources:

In terms of resources, the two stores do not offer that many configuration options. In fact, both DynamoDB and Spanner are mostly black-box systems that leave almost no decisions for the customer to make. Outside of the geo-distribution mentioned above, there is only one more option left to configure for both systems. With Spanner this option is the number of nodes which is not to be confused with the number of replicas. In the Spanner documentation this detail is explained as follows : *"If you need to scale up the*

serving and storage resources in your instance, add more nodes to that instance. Note that adding a node does not increase the number of replicas (which are fixed for a given configuration)" [6]. Instead of speculating what "scale up the serving and storage resources" precisely means, we just left the default settings which spawns only one Spanner node.

With DynamoDB the only option that we could set is the minimum/maximum provisioned read/write capacity units. Since we did not change the default instance settings for Spanner, we decided to do the same with DynamoDB. In the **ideal case** we would have definitely provisioned both stores in the closest possible manner in terms of resources. Unfortunately, by operating as black-box, these systems leave almost no space for customer configuration and that's why we decided to leave the data stores provisioned with the default provider's settings.

Data distribution
Since the benchmarking targes are distributed data stores, they allow the customer/developer to decide how the data will be distributed across the network. For DynamoDB the data partitioning is determined by the partition key chosen by the customer at the table initialization. With Spanner it is determined by the primary key also chosen by the customer when defining the database schema. To make the two deployments as similar as possible, we chose the same partition key for both stores.

E. Assesment of the Benchmark Design Objectives

According to S. Tai, D. Bermach and E. Wittern, there are five general benchmark design objectives - relevance, reproducibility, fairness, portability and understandability [3]. In this subsection, we will briefly assess how our benchmark performs based on these objectives.

Relevance:
The envisioned benchmark targets the write performance of DynamoDB and Google Spanner in terms of latency. Since OLTP (i.e. insert/update/delete) workloads are common in today's web applications [1] (e.g. e-commerce stores) , we can safely say that using a update heavy workload addresses a realistic enough scenario. Additionally, it is reasonable to assume that real-life use cases that include a huge amount of write operations are here to stay in the future. Taking this information into account, we can conclude that the envisioned benchmark is relevant.

Reproducibility:
To ensure reproducibility, we ran the prepared experiments multiple times at different times of the day. Based on our observations, both stores perform very similarly and they always produce approximately the same results. Although in the Cloud there is always the possibility for unstable behaviour, we can safely say that at least at the time of this writing the benchmark satisfied the reproducibility as a design objective.

Repeatability:
Since YCSB as a benchmarking tool operates with synthetic workloads, the benchmark we developed for the project does not fulfill the repeatability design objective. The reason is that with synthetic workloads, there is always a random element attached to each individual benchmark run which hinders us from repeating the *exact* same experiment multiple times.

Fairness:
In general, the benchmark is not completely fair. The reason for that is that most modern NoSQL and NewSQL stores are optimized for write heavy workloads. This means that if the benchmark experiment is executed on a traditional SQL store (e.g. Postgres, MySQL, DB2, etc.), this might put the NoSQL stores in a advantageous position. However, if we take a step back and focus only on DynamoDB and Spanner, the benchmark in its current form does not give any advantage to either of the stores. Therefore, we can conclude that in the more general cases the benchmark is not treating all data stores in a fair manner, but in the scope of the project, the benchmark treats DynamoDB and Google Spanner fairly.

Portability:
With the help of the YCSB tool, it was not difficult to ensure portability for the benchmark. YCSB offers an enormous number of adapters for almost all popular cloud-based (and not only) data stores. This means that the benchmark can be executed against a large number of systems (which is also the definition of "portability" presented in "Cloud Service Benchmarking" [3]) and DynamoDB and Google Spanner are no exception.

F. Workload Types

The type of the workloads used throughout the project can be examined from multiple perspectives. Firstly, the workloads that YCSB offers are **synthetic** by nature - which explains why the workloads we applied are also synthetic. As for the granularity level of the benchmark is concerned, we are using **micro-benchmarks**. The explanation is that as we already mentioned in the "Benchmarking Goals" subsection, we are testing a relatively small, highly specific feature of both data stores. Lastly, in terms of the workload generation model, based on our understanding, YCSB uses the "closed" model, although in practice it is not the most *"realistic"* approach [3].

IV. Benchmark Execution

The purpose of this section is to describe the process of conducting the experiments planed in the scope of the project. Some of the aspects that are relevant for this section, were already discussed in section "Benchmark Goals and Design" (e.g. **provisioning of resources**, **geolocation** of the benchmarking clients and SUT's). Therefore they will not be explicitly repeated unless necessary.

A. Experiment Phases

The experiments we conducted as part of the assignment consist of multiple phases - pre-loading, warmup, actual experiment phase, data collection and cleanup. Each one of these is described below:

Pre-loading Phase:
Since we are benchmarking two data stores, the pre-loading phase is of crucial importance. Therefore, during this phase we loaded the records used during the actual experiments. The duration of the pre-loading phase varied drastically based on the workload settings. For instance, with our initial setup, we tried to insert 10 000 000 records which resulted in an estimated loading time of 3 days for DynamoDB. Therefore we reduced the number of records to reach a more reasonable loading time of around 45 minutes up to 1 hour - i.e. we conducted the experiments with 30 000 records. Also, during this phase we discovered a feature of Google Spanner which really made it stand out from DynamoDB and namely - Spanner offers batch inserts. Putting this feature into use, made Spanner's pre-loading phase from hundred to a thousand times shorter than the pre-loading phase of DynamoDB.

Warmup Phase:
The warmup phase we used was rather simplistic. Instead of running the experiment with 50000 operations as for the full-fledged experiment, we scaled the request count down to 1000 operations. As suggested by the authors of the "Cloud Service Benchmarking" book in chapter 9 [3], we executed the actual experiment immediately after the warmup was over, so that we can avoid periods of idleness in-between the two phases.

Experiment Phase:
The only noteworthy detail for the experiment phase were the settings used with each individual run. For the purposes of answering the research questions introduced in section "Benchmark Goals and Design", we used the following configuration:

- Every experiment included 30 000 operations
- Update heavy workload with 80/20 write-to-read ratio
- Duration of each individual experiment - 25 minutes
- The size of the database record fields varied from experiment to experiment (from 100 bytes up to 1000 bytes)

Also, to ensure that DynamoDB and Google Spanner behave similarly during different times of the day we ran the experiments in the morning, in the afternoon and late at night. The results of these experiments as well as the behavior of the two stores are included in the "Results" section.

Lastly, outside of the experiments directly related to the research questions, we collected additional measurements that reveal an interesting behavior from the two stores. Although these results are indirectly related to the benchmarking goals for the project, we decided to include some of the more interesting observations at the end of the "Results" section.

Data Collection Phase:
Since we ran the benchmarking client on a local machine, this phase was relatively effortless. We used the most simplistic data collection strategy and stored the collected measurements in a local file. After the experiment was over, we manually filtered the relevant data and organized it in a Excel spreadsheets for further processing.

Cleanup Phase:
To return both data stores to their original state (i.e. the state before running an experiment) we took the most unsophisticated, brutal, cruel, crude and uncivilized approach. Jokes aside, instead of overthinking it, we decided to just delete all tables, records and traces of the experiment altogether after each experiment is over. This way we go through all the phases described here (from pre-loading to cleanup) for each individual run and we end up with a "clean slate". Although this might not be the most elegant way to clean the benchmarking setup, it saved us a lot of time in dealing with automation tools and alternative fine-grained cleanup scenarios.

$$\alpha + \beta = \chi \tag{1}$$

B. Some Common Mistakes

V. Using the Template

A. Headings, etc

B. Figures and Tables

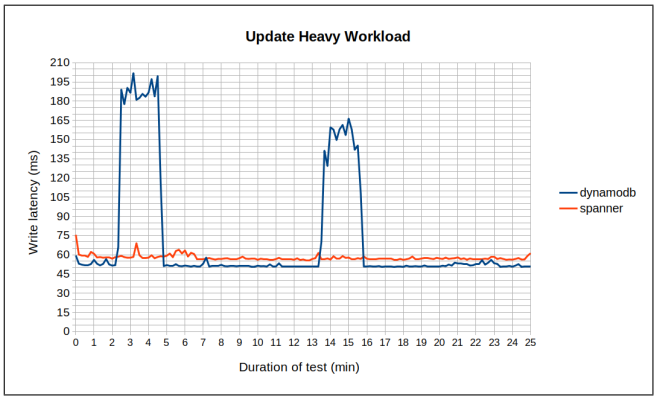

Fig. 1. Example of a parametric plot $(\sin(x), \cos(x), x)$

VI. Conclusions

Appendix

Acknowledgment

References

[1] D. T. McWherter et al. "Priority mechanisms for OLTP and transactional Web applications". In: *Proceedings. 20th International Conference on Data Engineering*. Apr. 2004, pp. 535–546. DOI: 10.1109/ICDE.2004.1320025.

[2] Brian F. Cooper et al. “Benchmarking Cloud Serving Systems with YCSB”. In: *Proceedings of the 1st ACM Symposium on Cloud Computing*. SoCC ’10. Indianapolis, Indiana, USA: ACM, 2010, pp. 143–154. ISBN: 978-1-4503-0036-0. DOI: 10.1145/1807128.1807152. URL: http://doi.acm.org/10.1145/1807128.1807152.

[3] Erik Wittern Stefan Tai David Bermbach. *Cloud Service Benchmarking. Measuring Quality of Cloud Services from a Client Perspective*. Springer, Cham, 2017. DOI: 10.1007/978-3-319-55483-9.

[4] *Amazon DynamoDB. Fast and flexible NoSQL database service for any scale*. Accessed: 11-Jan-2019. URL: https://aws.amazon.com/de/dynamodb/.

[5] *CLOUD SPANNER. The first horizontally scalable, strongly consistent, relational database service*. Accessed: 10-Jan-2019. URL: https://cloud.google.com/spanner/.

[6] *CLOUD SPANNER. Instances*. Accessed: 31-Jan-2019. URL: https://cloud.google.com/spanner/docs/instances#nodes_vs_replicas.