TECHNISCHE UNIVERSITÄT BERLIN

Exploring Datastore Access Latency across AWS Compute Services

Bachelor's Thesis

Author
Bhupendra Singh
464877
b.singh@campus.tu-berlin.de

Advisor
Trever Schirmer

Examiners
Prof. Dr.-Ing. David Bermbach
Prof. Dr. habil. Odej Kao

Technische Universität Berlin, 2024
Fakultät Elektrotechnik und Informatik
Fachgebiet Scalable Software Systems

Exploring Datastore Access Latency across AWS Compute Services

Bachelor's Thesis

Submitted by:
Bhupendra Singh
464877
b.singh@campus.tu-berlin.de

Technische Universität Berlin
Fakultät Elektrotechnik und Informatik
Fachgebiet Scalable Software Systems

2024

Hiermit versichere ich, dass ich die vorliegende Arbeit eigenständig ohne Hilfe Dritter und ausschließlich unter Verwendung der aufgeführten Quellen und Hilfsmittel angefertigt habe. Alle Stellen die den benutzten Quellen und Hilfsmitteln unverändert oder sinngemäß entnommen sind, habe ich als solche kenntlich gemacht.

Sofern generische KI-Tools verwendet wurden, habe ich Produktnamen, Hersteller, die jeweils verwendete Softwareversion und die jeweiligen Einsatzzwecke (z. B. sprachliche Überprüfung und Verbesserung der Texte, systematische Recherche) benannt. Ich verantworte die Auswahl, die Übernahme und sämtliche Ergebnisse des von mir verwendeten KI-generierten Outputs vollumfänglich selbst.

Die Satzung zur Sicherung guter wissenschaftlicher Praxis an der TU Berlin vom 8. März 2017* habe ich zur Kenntnis genommen.

Ich erkläre weiterhin, dass ich die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt habe.

(Unterschrift) Bhupendra Singh, Berlin, 20. November 2024

*https://www.static.tu.berlin/fileadmin/www/10000060/FSC/Promotion___Habilitation/Dokumente/Grundsaetze_gute_wissenschaftliche_Praxis_2017.pdf

Abstract

–> TODO <–

Kurzfassung

–> TODO <–

Contents

1 Introduction

Cloud computing delivers on-demand resources like servers, storage, and databases via the Internet, eliminating the need for physical infrastructure and enabling flexibility and scalability. Its adoption has surged in the past decade due to the emergence of data-intensive applications and improved connectivity [<**empty citation**>]. AWS, a leader in cloud computing, pioneered various cloud models and controls over 30% of the global market as of 2023, driven by its innovation and broad service portfolio [<**empty citation**>].

Applications in cloud environments typically consist of compute and datastore service instances that interact frequently. For example, AWS Elastic Compute Cloud (EC2) is often used for backend processing, while AWS Relational Database Service (RDS) handles structured data storage. Communication between these services is a critical aspect of cloud application architecture, and network latencies significantly influence overall performance and user experience, particularly in latency-sensitive applications such as e-commerce, real-time analytics, and gaming [<**empty citation**>].

AWS offers latency insights for datastore services from both service-side and client-side perspectives. However, it remains unclear how client-side latency varies across different AWS compute services. For instance, does the EC2-RDS pair perform differently compared to Lambda-RDS? AWS provides numerous configurations and tools for optimizing service interactions, yet the specific latency characteristics between services can vary depending on network setup, service proximity, caching strategies, and service-specific characteristics. As organizations seek efficient and performant cloud architectures, understanding the latency dynamics between these services is essential for making informed architectural decisions.

Despite the critical role of latency in cloud-based systems, publicly available empirical data on latency characteristics between AWS compute and datastore services remains limited. This thesis aims to fill this knowledge gap by providing an initial benchmark for access latency metrics between key AWS compute and datastore service pairs, specifically EC2, and Lambda paired with RDS, DynamoDB, and Simple Storage Service (S3).

We therefore make the following contributions in this thesis:

1. We address the identified knowledge gap by proposing a benchmark design, detailed in Section 3.
2. We evaluate the proposed approach through experimentation on AWS compute and datastore service pairs, with implementation and results presented in Sections 4 and 5.

2 Background

–> TODO <–

3 Benchmark Design

To address the main purpose of this thesis, we adopt a benchmarking approach to evaluate the pairs and compare their results. This section presents the benchmark design, which is based on the guidelines outlined in [<**empty citation**>]. As described in Section 1, the study examines six pairs of AWS compute and datastore services:

- EC2 paired with RDS, DynamoDB, and S3
- Lambda paired with RDS, DynamoDB, and S3

Each benchmarking run consists of two main components: (1) a pre-loaded datastore serving as the System Under Test (SUT) and (2) a compute instance functioning as the workload generator. The compute instance performs read operations on the datastore and records latency metrics.

For Lambda-Pairs, each invocation corresponds to a single read operation on the datastore. To invoke the Lambda function in a scripted manner and collect results, an additional EC2 instance, referred to as the "Lambda-Helper", is provisioned with k6 and the necessary scripts, as shown in **??**. This instance sends HTTP requests to invoke the Lambda function and collects response data, which includes timestamps recorded by the Lambda function.

The benchmark evaluates each pair under two workload types, with each configuration repeated twice, resulting in of 24 runs. We use infrastructure-as-code and shell scripting for resource provisioning and de-provisioning in AWS, ensuring a high level of automation to minimize human error. All runs are executed in the same AWS region.

After each run is successfully completed, data is exported to a dedicated S3 bucket for collection. Finally, we transform the collected data and perform analysis to gain insights.

–> TODO: Add diagram here <–

4 Benchmark Implementation

This section describes the implementation of the benchmark design used to evaluate access latency. It provides detailed information on three key aspects: (1) the configuration of service instances, (2) the methods for load generation and workload types, and (3) the approach to data analysis.

4.1 Resource Configuration

RDS: A MySQL 8.0 instance on a t3.micro virtual machine (VM) with 1GiB RAM, 2 vCPUs, and 5GB storage. The instance is located in the eu-central-1a availability zone, with multi-AZ failover disabled to maintain a single-zone configuration. It hosts a single database containing a table with 1000 rows and four columns: *id*, *name*, *address*, and *email*.

DynamoDB: A DynamoDB table configured with *Provisioned* capacity mode, a read capacity of 25 Read Capacity Units (RCUs), and a write capacity of 5 Write Capacity Units (WCUs). The table contains 10 items, each with an *id* (hash key) and an *email* attribute.

S3: A single S3 bucket with a 20-byte text file.

EC2: An EC2 instance of t2.micro type with 1GiB RAM, and 1vCPU, located in the eu-central-1a availability zone. It is operated by Ubuntu Server 24.04 (64-bit, HVM-based) and is equipped with the k6 and corresponding scripts.

Lambda: A non-VPC Node.js 20 function with 1.65GiB RAM, 1 vCPU, and a concurrency limit set to 990. The configuration of Lambda-Helper is identical to that of EC2.

4.2 Load Generation and Types

The benchmark evaluates performance under two distinct workload types: (1) constant workload and (2) burst workload. An open workload model was employed, meaning requests per second (RPS) are configured rather than the number of client threads. Both compute services were tested using identical benchmarking scripts to ensure consistency in test duration and load configuration.

Constant Workload: For the constant workload scenario, the objective is to simulate a stable, continuous load over an extended period to observe baseline latency behavior for each datastore. The configurations are as follows:

- **RDS and DynamoDB**: 2 RPS for 14 hours.
- **S3**: 0.2 RPS (1 read every 5 seconds) for 3 hours.

Burst Workload: For the burst workload scenario, the benchmark incorporated multiple spikes to simulate unpredictable load surges. The configurations are as follows:

- **RDS and DynamoDB**: The run starts with a baseline rate of 2 RPS for the first 2 hours, followed by 30 load spikes. Each spike lasts 3 minutes, with RPS increasing linearly to 20 RPS and then decreasing back to the baseline of 2 RPS. Between spikes, the load remains at the baseline rate for 3 minutes. The run concludes with an additional 2 hours at the 2 RPS baseline load, resulting in a total test duration of approximately 7 hours.

- **S3**: The run starts with a baseline rate of 0.2 RPS for the first 20 minutes, followed by 6 load spikes. Each spike lasts 30 seconds, with RPS increasing linearly to 16 RPS and then decreasing back to the baseline of 0.2 RPS. Between spikes, the load remains at the baseline rate for 6 minutes. The run concludes with an additional 15 minutes at the 0.2 RPS baseline load, resulting in a total test duration of approximately 70 minutes.

This configuration allowed us to reach the monthly limits of the AWS Free Tier while keeping a small buffer. As noted, the monthly limits for S3 are relatively low, namely 20000 GET requests per month, which resulted in short-running benchmarking runs for S3.

4.3 Data Analysis

As discussed, for each targeted datastore service, we compare the latency performance across EC2 and Lambda. Since averages alone cannot fully represent the data distribution, we provide both bar plots for aggregated metrics and time-series graphs for a detailed view.

To ensure data integrity before analysis, two preprocessing steps are applied. First, we exclude the initial and final 2.5% of data points to remove the effects of warm-up and shut-down behavior. Second, outliers are identified and removed using the 3-sigma rule, which considers data points beyond three standard deviations from the mean as outliers. This statistical approach, outlined in [<**empty citation**>], prevents extreme values from skewing the results.

Bar plots visualizing aggregates are constructed by concatenating data from two runs for each pair and workload type. For time-series visualizations, resampling is applied to enhance clarity.

Data analysis and visualization are conducted using Python's Jupyter Notebook with pandas, matplotlib, and seaborn libraries.

5 Benchmark Results

In this section, we present and compare the findings from the data analysis across each targeted datastore. Throughout the benchmarking sessions, no CPU performance issues were detected on the client side (workload generator), ensuring that client limitations did not impact latency measurements. Additionally, the error rate consistently remained zero for all runs, indicating that every read request was completed.

EC2-RDS vs. Lambda-RDS EC2-RDS outperforms Lambda-RDS by approximately 51% under constant load and 56% under burst workload. Furthermore, EC2-RDS exhibits 41% and 45% less standard deviation than Lambda-RDS under constant and burst workload, respectively. As shown in Figure 4.2.1, the latency for Lambda-RDS often experiences abrupt increases or decreases, persisting at such levels for several hours. This behavior makes Lambda-RDS less stable and predictable compared to EC2-RDS. A possible cause is that the underlying cloud infrastructure conditions fluctuate, leading to inconsistent latency performance [<**empty citation**>].

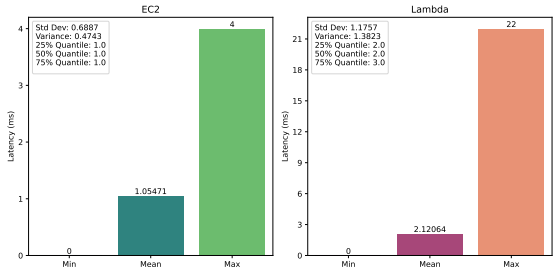
EC2-DynamoDB vs. Lambda-DynamoDB EC2-DynamoDB outperforms Lambda-DynamoDB by approximately 14% under constant load and 16% under burst workload. Furthermore, Lambda-DynamoDB exhibits 20% and 7% less standard deviation than EC2-DynamoDB under constant and burst workload, respectively.

EC2-S3 vs. Lambda-S3 EC2-S3 outperforms Lambda-S3 by approximately 8% under constant load and 10% under burst workload. Additionally, Lambda-S3 exhibits 22% less standard deviation than EC2-S3 under constant load but 12% more under burst workload. As illustrated in Figure 4.1.5, the variance of EC2-S3 under constant load is approximately three times higher than under burst workload, as shown in Figure 4.1.6. This observation appears counterintuitive. Due to the constraints of AWS Free Tier, experiments involving S3 were shorter, as discussed in Section 4, which may have amplified the impact of fluctuations in the underlying cloud infrastructure [<**empty citation**>].

We observe that EC2 pairs consistently outperform Lambda pairs across all experiments. However, the absolute difference in mean latencies remains below 1.34 milliseconds. We know that Lambda functions run within Firecracker VMs, which are deployed on multi-tenant EC2 instances. Communication between Lambda functions and their host EC2 instances occurs via a virtio-based interface, introducing additional networking overhead of approximately $0.06ms$ [<**empty citation**>]. However, this overhead appears negligible compared to the observed differences, suggesting that additional factors contribute to the latency variations.

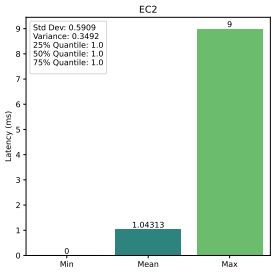
Outliers –> TODO <–

Figure 5.1: –> TODO: Add latency CDF graphs here <–

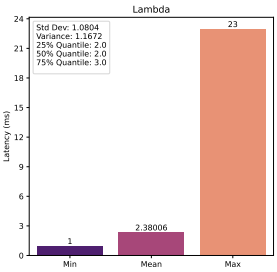


(5.2.1) Constant Workload on RDS

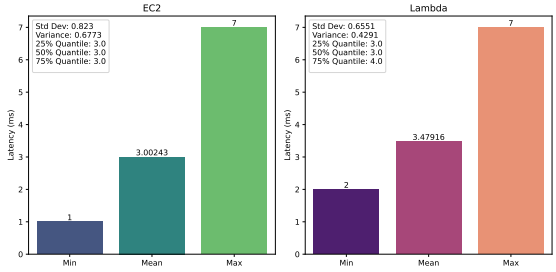
(5.2.2) Burst Workload on RDS



(5.2.3) Constant Workload on DynamoDB

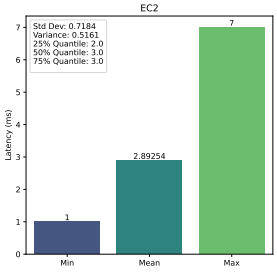


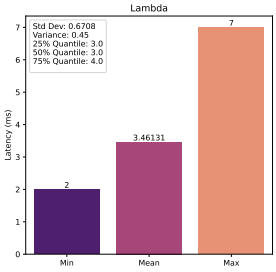
(5.2.4) Burst Workload on DynamoDB

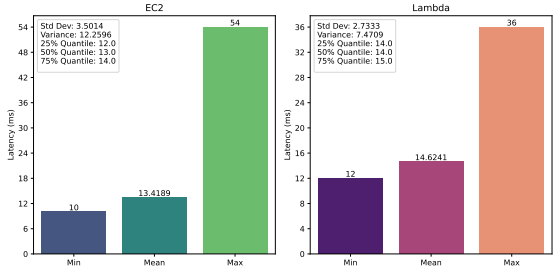


(5.2.5) Constant Workload on S3

(5.2.6) Burst Workload on S3

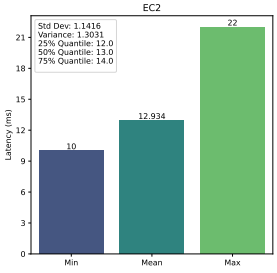






(5.2.7) Constant Workload on S3

(5.2.8) Burst Workload on S3



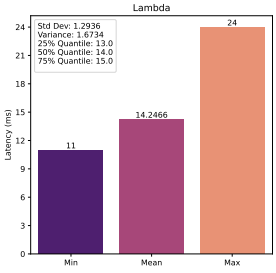
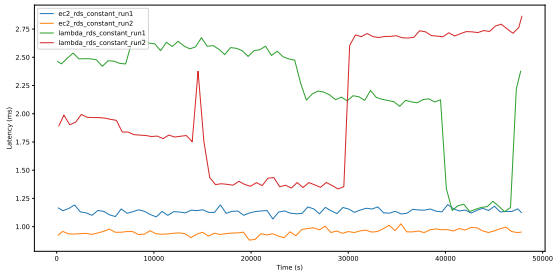
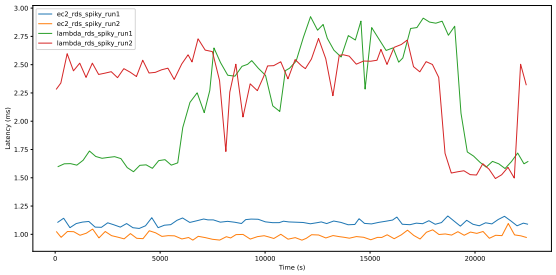


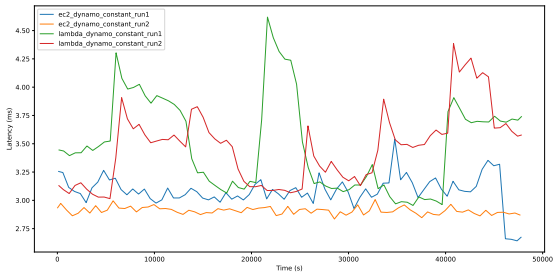
Figure 5.2: Aggregation of Latency Measurements



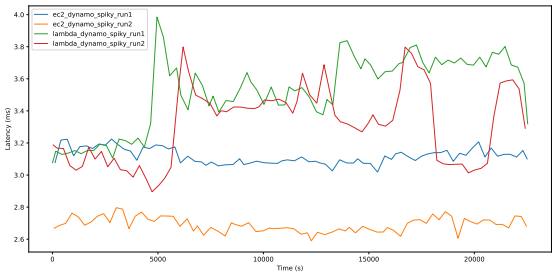
(5.3.1) Constant Workload on RDS



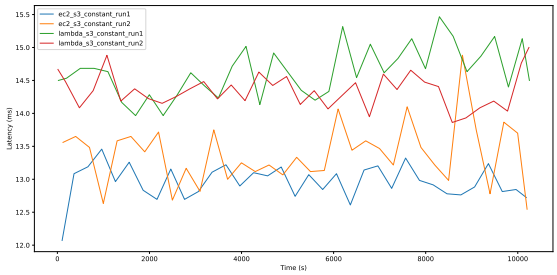
(5.3.2) Burst Workload on RDS



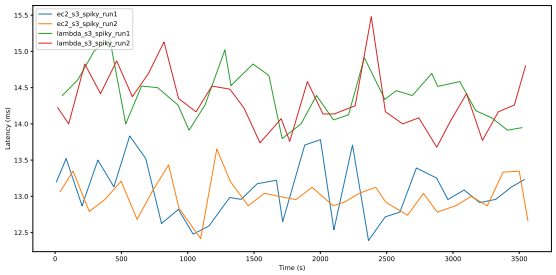
(5.3.3) Constant Workload on DynamoDB



(5.3.4) Burst Workload on DynamoDB



(5.3.5) Constant Workload on S3



(5.3.6) Burst Workload on S3

Figure 5.3: Time-Series Representation of Latency Measurements

6 Discussion

This study evaluated latency performance between AWS compute service and datastore pairs. The findings indicate latency trends influenced by compute-service choice and workload characteristics. However, several aspects of the methodology and limitations must be critically analyzed to contextualize these outcomes.

This study encountered several significant limitations. The constraints of the AWS Free Tier restricted the duration and scale of experiments, limiting the observation of long-term performance trends. The reliance on low-tier configurations, such as t3.micro instances, prevented evaluation in realistic production environments where higher-capacity instances are standard. Temporal variability in cloud performance was another challenge, with short-duration benchmarks potentially not capturing the real distribution of the latency metric caused by time-of-day effects or transient network conditions. Additionally, the absence of multi-region setups meant geo-distributed latency behaviors were not analyzed. Simplified use cases focusing on read operations excluded mixed read-write workloads and asynchronous interactions, which are typical in production systems. Lastly, while statistical methods effectively identified and excluded outliers, their underlying causes were not investigated, leaving critical insights into rare performance anomalies unaddressed.

This study's findings indicate implications for designing cloud-based systems. EC2 pairs consistently demonstrated lower access latency, making it a reliable choice for applications with high-performance requirements. However, the relatively small differences in latency observed between Lambda and EC2 pairs, particularly with DynamoDB, indicate that Lambda could be a viable option for scenarios where operational simplicity and scalability are prioritized over minimal latency. To build upon these findings, future research should consider more extensive benchmarks involving realistic production configurations, mixed workloads, long-running experiments, and multi-region setups to better understand the dynamics of latency in diverse scenarios.

7 Related Work

–> TODO <–

8 Conclusion

–> TODO <–