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, 19. 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

With the rapid expansion of cloud computing, organizations increasingly rely on cloud providers like Amazon Web Services (AWS) for scalable and flexible infrastructure solutions. Key AWS services, such as EC2, Lambda, DynamoDB, RDS, and S3, support diverse applications across industries, from real-time data processing to large-scale data storage. For cloud-based systems, latency between services is critical; it directly affects application responsiveness, user experience, and operational costs, particularly in latency-sensitive applications like financial services, e-commerce, and gaming.

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 (e.g., Availability Zones), caching strategies, and service-specific characteristics (e.g., the underlying storage mechanisms in S3 vs. RDS). As organizations seek efficient and performant cloud architectures, understanding the latency dynamics of these services is essential for making informed architectural decisions.

Despite AWS's built-in optimizations, there is limited publicly available, empirical data on the exact latency characteristics between compute and data services, especially across different compute paradigms like EC2 and Lambda. EC2 provides traditional virtual machine-based computing, while Lambda represents serverless, event-driven computing. The latency between these compute options and data storage solutions—specifically DynamoDB, RDS, and S3—affects application performance and scalability but is not thoroughly documented in AWS's standard benchmarks.

This study aims to bridge this knowledge gap by providing detailed benchmarking results on access latency between EC2-Lambda and three AWS data services: DynamoDB, RDS, and S3. The study includes configurations such as VPC settings and instance specifications, which influence latency outcomes.

The main objectives of this thesis is to conduct benchmarking, measuring and comparing access latency between EC2 instances and Lambda functions interacting with DynamoDB, RDS, and S3 under controlled configurations.

This thesis focuses on access latency benchmarks within the AWS EU-Central-1 (Frankfurt) region to maintain a controlled environment. While benchmarking different configurations within AWS, the results may not be universally applicable across other cloud providers or AWS regions. Additionally, the focus remains on synchronous interactions between services, excluding asynchronous and multi-region setups.

A primary limitation of this thesis lies in the requirement to design and execute the entire benchmarking process within the constraints of the AWS Free Tier. This imposed restrictions on the selection of components, the duration of the benchmarking activities, and the number of repetitions conducted.

The benchmarks were conducted with tools such as k6 and JavaScript, with latency metrics recorded and analyzed using Python's data analysis libraries. These choices align with industry-standard benchmarking practices but may introduce minor tool-based variances that are acknowledged and accounted for in the analysis.

The remainder of this thesis is structured as follows:

1. Section II Background

2. Section III Approach
3. Section IV Evaluation
4. Section V Discussion
5. Section VI Related Work
6. Section VII Conclusion

2 Background

TODO (Define RDS, DynamoDB, S3, EC2, Lambda, k6, terraform)

3 Benchmark Design

To address the main question, 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 2, 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 session 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 load 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 Lambda 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 Lambda.

The benchmark evaluates each pair under two load modes, with each configuration tested twice, resulting in a total of 24 sessions. We use Terraform and shell scripts for resource provisioning and de-provisioning in AWS, ensuring a high level of automation to minimize human error. All sessions are conducted in the same AWS region.

Post-session, data is exported to a dedicated S3 bucket for data collection. Finally, we transform collected data and perform analysis to gain insights.

4 Benchmark Implementation

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 none-VPC Node.js 20 function with 1.65GiB RAM, 1 vCPU, and concurrency limit of 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 load patterns: (1) constant load and (2) bursty load. An open workload model was employed, meaning requests per second (RPS) is 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 Load: For the constant load scenario, the objective is to simulate a stable, continuous load over an extended period to observe baseline latency behavior for each datastore. The configurations is as follows:

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

Bursty Load: For the bursty load scenario, the benchmark incorporated multiple spikes to simulate unpredictable load surges. The configuration were as following:

- **RDS and DynamoDB**: The session 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 session concludes with an additional 2 hours at the 2 RPS baseline load, resulting in a total test duration of approximately 7 hours.
- **S3**: The sessions 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 session 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 low (20,000 GET requests per month), which resulted in short sessions for S3.

4.3 Data Analysis

As discussed, for each targeted datastore service, we compare the latency performance of EC2 and Lambda. Averages alone are insufficient to fully capture the performance distribution, therefore, we present aggregate in bar plots and additionally time-series graphs for comprehensive view.

Prior to analysis, we remove 2.5% of data from the start and end of the datasets to remove effects of warm-up and shut-down behavior. Additionally, outliers are identified and removed.

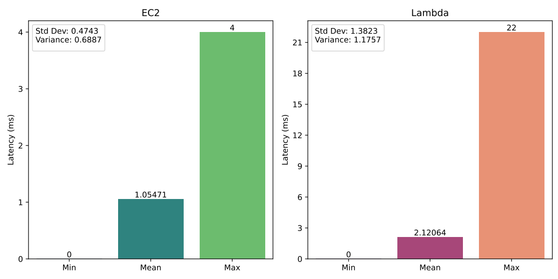
Outlier detection was performed using the 3-sigma rule for univariate data, a method commonly used for statistical outlier identification based on the principle that values lying beyond three standard deviations from the mean are considered outliers. This approach is outlined in [<**empty citation**>] and ensures that extreme values do not skew the analysis.

For visualizing aggregates in bar plots, we concatenate the two sessions for each pair and mode. For visualizing time-series data we use resampling.

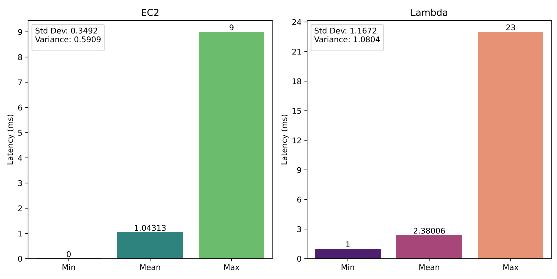
The analysis was conducted using Python Jupyter Notebook with the pandas, matplotlib, and seaborn libraries.

5 Benchmark Results

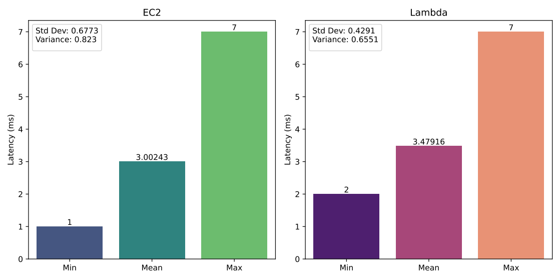
In this section, we present and compare the findings from the data analysis across each targeted datastore. Throughout the benchmarking sessions, no CPU performance issues were detected on the client-side (load generator), ensuring that client limitations did not impact latency measurements. Additionally,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



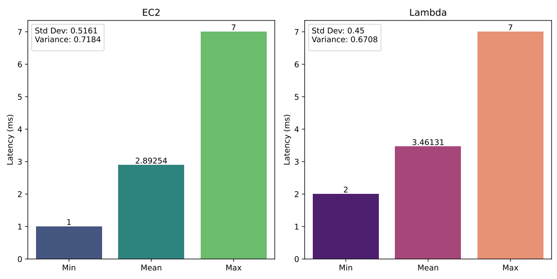
(5.1.1) Constant Load on RDS



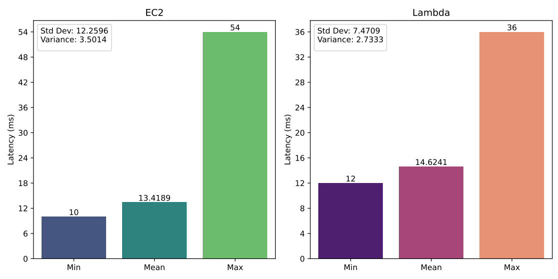
(5.1.2) Bursty Load on RDS



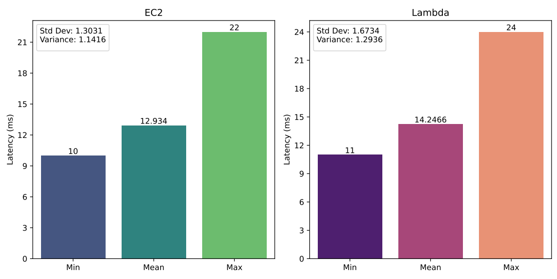
(5.1.3) Constant Load on DynamoDB



(5.1.4) Bursty Load on DynamoDB



(5.1.5) Constant Load on S3



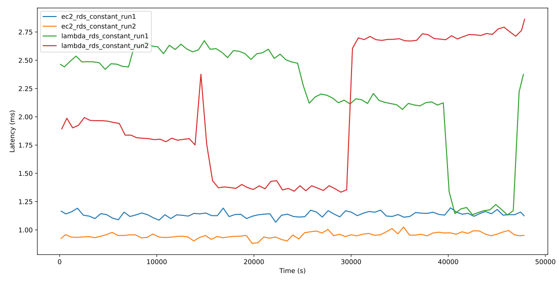
(5.1.6) Bursty Load on S3

Figure 5.1: Aggregation of Latency Metric

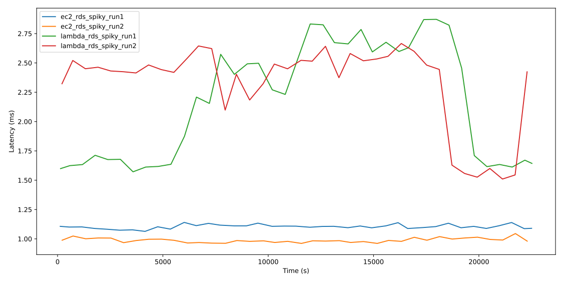
As observed in Figure 5.1, mean latencies of EC2-Pairs is less than that of Lambda-Pairs for all experiments. The absolute difference is although minimal, always less than 1.34 milliseconds.

In Figure 4.1.5, we can obeserve that EC2-S3 under constant load shows approximately 3x variance compared to EC2-S3 under bursty load in Figure 4.1.6, which intuitively seems contradicting. Similar is observed in Figure 5.2, especially in Figure 4.2.1, where latency with lambda suddenly rises or sinks and continues at that rate for multiple hours. Possible reason is that the underlying conditions of the cloud vary depending on day and time, leading to variations in latency performance [<**empty citation**>].

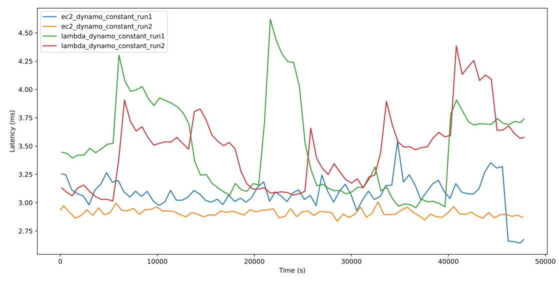
The reason behind EC2 outperforming Lambda possibly lies in the fact that, Lambdas reside inside Firecracker-VMs, which are deployed on multi-tenant EC2s. The Lamb-



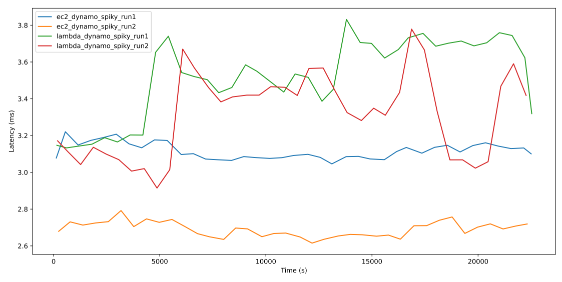
(5.2.1) Constant Load on RDS



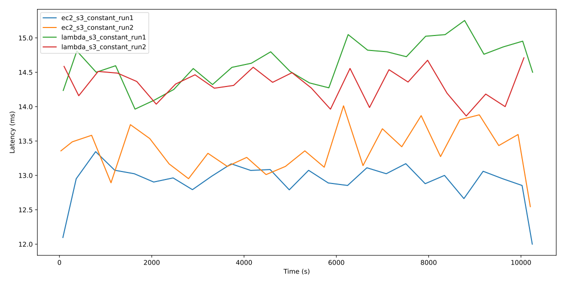
(5.2.2) Bursty Load on RDS



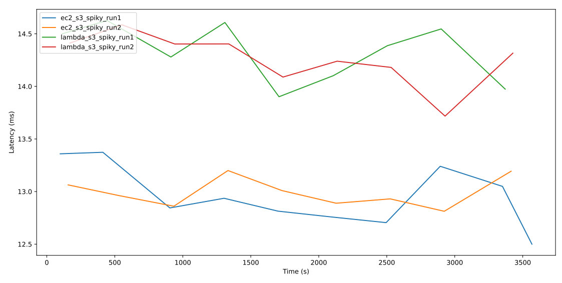
(5.2.3) Constant Load on DynamoDB



(5.2.4) Bursty Load on DynamoDB



(5.2.5) Constant Load on S3



(5.2.6) Bursty Load on S3

Figure 5.2: Time-Series Representation of Latency Metric

das communicate with the host EC2s via istio-based network interface. This introduces networking overhead which might impact latency performance [<**empty citation**>].

EC2-RDS vs. Lambda-RDS

EC2-DynamoDB vs. Lambda-DynamoDB

EC2-S3 vs. Lambda-S3

Outliers

6 Discussion

This study evaluated latency performance between AWS compute services (EC2 and Lambda) and three datastore options (RDS, DynamoDB, and S3). The findings indicate latency trends influenced by compute-service architecture 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.

The study's findings have important implications for designing cloud-based systems. EC2 consistently demonstrated lower latency, making it a reliable choice for applications with stringent performance requirements. However, the relatively small differences in latency observed between Lambda and EC2, 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, and multi-region setups to better understand the dynamics of latency in diverse scenarios.

7 Related Work

–> TODO <–

8 Conclusion

TODO