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, 27. November 2024

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

Abstract

Cloud computing has grown rapidly, driving the demand for scalable and flexible infrastructure solutions. Amazon Web Services (AWS) leads the market, offering diverse compute and datastore services for various applications. Despite their importance, empirical data on latency dynamics between AWS compute and datastore services remains scarce. This thesis addresses this knowledge gap by systematically benchmarking access latency between EC2 and Lambda paired with RDS, DynamoDB, and S3. We evaluate these service pairs under constant and burst workloads. The results show that EC2 pairs consistently achieve lower mean latency than Lambda pairs. This study acknowledges certain limitations, including constraints imposed by AWS Free Tier, the focus on a single region, and the exclusion of mixed workload scenarios. This work provides foundational insights for designing latency-sensitive cloud applications. It emphasizes the need for further research on multi-region setups, advanced configurations, and mixed workload scenarios to deepen the understanding of AWS service interactions.

Kurzfassung

Cloud Computing hat sich schnell entwickelt und führt zu einer steigenden Nachfrage nach skalierbaren und flexiblen Infrastruktur-Lösungen. Amazon Web Services (AWS) führt den Markt an und bietet eine Vielzahl von Compute- und Datenspeicher-Diensten für unterschiedlichste Anwendungen. Trotz ihrer Bedeutung fehlen jedoch empirische Daten zu den Latenz-Dynamiken zwischen den Compute- und Datenspeicher-Diensten von AWS. Diese Arbeit schließt diese Wissenslücke, indem sie die Zugriffs-Latenz zwischen EC2 und Lambda in Kombination mit RDS, DynamoDB und S3 systematisch benchmarkt. Diese Dienst-Paare werden unter konstanten und burstigen Arbeitslasten evaluiert. Die Ergebnisse zeigen, dass EC2-Paare durchgehend eine niedrigere durchschnittliche Latenz aufweisen als Lambda-Paare. Die Arbeit erkennt bestimmte Einschränkungen an, darunter die Begrenzungen durch das AWS Free Tier, die Fokussierung auf eine einzelne Region und die Ausschließung gemischter Arbeitslastszenarien. Diese Arbeit liefert grundlegende Erkenntnisse für das Design latenzempfindlicher Cloud-Anwendungen und betont die Notwendigkeit weiterer Forschung zu Multi-Region-Setups, erweiterten Konfigurationen und gemischten Arbeitslastszenarien, um das Verständnis der Interaktionen zwischen AWS-Diensten zu vertiefen.

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 and Amazon Web Services (AWS), has emerged as a leader and pioneer of various cloud models. AWS maintains over 30% market share 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, therefore latency between these services significantly influence overall performance and user experience of the application [<**empty citation**>]. AWS provides performance insights for its datastore services from both the service-side and client-side perspectives. The service-side perspective evaluates how efficiently the datastore processes queries internally, while the client-side perspective focuses on the time taken for a client to receive a response after submitting a query. However, it remains unclear how client-side latency varies across different AWS compute services.

Prominent AWS compute services include (1) Elastic Cloud Compute (EC2), offers customizable virtual machines for consistent performance and long-running workloads, and (2) Lambda, a serverless, event-driven service ideal for short tasks and automatic scaling. Key datastore services are (1) Relational Database Service (RDS), provides managed SQL databases for structured data, (2) DynamoDB, a serverless NoSQL database for high-scale, low-latency use cases, and (3) Simple Storage Service (S3), a distributed object storage service for unstructured data [<**empty citation**>].

In this thesis, we benchmark access latency between pairs of the above AWS compute and datastore services to examine how the choice of compute service affects latency dynamics with datastores, such as whether an EC2-RDS pair outperforms a Lambda-RDS pair. The results consistently show that EC2-based pairs outperform Lambda-based pairs by small margins. These findings can guide cloud and application developers in making informed deployment decisions, especially for latency-sensitive applications.

We therefore make the following contributions in this thesis:

1. We address the identified knowledge gap by proposing a benchmark in Section 3.
2. We evaluate the proposed approach through experimentation in Section 4.
3. We present and interpret our results in Section 5.
4. We discuss our observations, possible causes, and limitations in Section 6.

2 Background

As outlined in Section 1, in this thesis we benchmark access latency between AWS compute and datastore services. Access latency refers to the duration between a client request and its corresponding response, influenced by factors such as network delays, processing overhead, and caching mechanisms. Benchmarking, in this context, involves systematically evaluating and comparing IT components or systems against predefined performance metrics.

In this section, we summarize general requirements and key elements for benchmarking cloud services, drawing on insights from [BWT17; Bin+09; Coo+10; Fol+13]. Furthermore, we discuss challenges and provide context for the following sections.

2.1 Requirements of Cloud Benchmarking

Cloud benchmarking generally requires relevance to the problem domain, affordability, and simplicity to ensure usability and trust. Benchmarks must allow fair comparisons, be repeatable under similar conditions, and realistically reflect typical workloads. Configurability and scalability are essential for adapting benchmarks to diverse scenarios while maintaining alignment with real-world applications. Additionally, meaningful metrics are critical for interpreting system performance effectively. However, according to use-case and limitations, tradeoffs between these requirements might be required.

Benchmarks should measure deviations from an ideal case, incorporate failure scenarios, and use meaningful cost metrics. They must account for geo-distribution, stress tests, and scalability under variable loads. Long-running experiments and adaptable configurations ensure relevance as applications evolve.

A major challenge for this thesis is to conduct the benchmarking under limitations of AWS Free Tier[1], hence affordability is a major requirement for us.

2.2 Elements of a Cloud Benchmark

define Workload Generator, SUT, k6, AWS services, refer instance with service name

2.3 Challenges

define, ddb provisioned capacity, lambda concurrency,

What can be controlled by client and what not. –> BACKGROUND: 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. <– What is a benchmarking run? What is open vs. closed workload model?

[1] **https://aws.amazon.com/free/**.

3 Benchmark Design

In this section, we present our approach to benchmark AWS compute and datastore service pairs, while addressing requirements and challenges discussed in Section 2. Affordability is a major requirement in this thesis, which influences design and implementation decisions while necessitating trade-offs with other requirements.

Sections 1 and 2 highlight the importance of access latency in applications and it is the primary metric of interest in this thesis. To isolate access latency, we focus on read-only operations between compute and datastore services, to avoid locking and transactional overheads in writes. Queries are kept simple, database tables, and object files minimal to reduce querying and data transmission overhead. While mixed read-write workloads, larger databases, and complex queries reflect real-world scenarios, simple read-only benchmarks can provide initial baseline latency performance. Additionally, focusing on reads is cost-effective, as services like S3 offer significantly more free-tier reads compared to writes.

We evaluate and compare six pairs of prominent AWS compute and datastore services for access latency: (1) EC2-RDS, (2) EC2-DynamoDB, (3) EC2-S3, (4) Lambda-RDS, (5) Lambda-DynamoDB, and (6) Lambda-S3. Each benchmarking run involves two primary components: (1) a pre-loaded datastore serving as the System Under Test (SUT) and (2) a compute instance which generates workload on the SUT and collects access latency data. For simplicity, we refer to the datastore as SUT and the compute instance as workload generator in this thesis. However, in reality, a compute instance and a datastore together constitute one SUT, as we are not particularly testing the datastores.

To address relevance, we employ open workload model, due to the reasons discussed in Section 2.2 and evaluate each pair under two distinct workload types: constant and burst. Under constant workload, the reads per second (RPS) remain stable throughout the benchmarking run. A burst workload, on the other hand, simulates sudden spikes in RPS, reflecting real-world scenarios where workload can be unpredictable. This allows us to observe how each pair performs under steady load and how they respond to fluctuating, high-demand conditions. To address repeatability we evaluate each pair under both workload types twice, resulting in total of 24 runs.

As discussed in Section 2.3, the fluctuating underlying conditions of the cloud can impact performance. To address this challenge, we extend the durations of benchmarking runs as much as possible and repeat them at different times and days, while balancing affordability and relevance. Fairness is ensured by maintaining identical workloads, resource allocation, network conditions, and datastore configurations across service pairs.

4 Benchmark Implementation

This section outlines the implementation details of the benchmark, focusing on three main aspects: (1) service instance configurations, (2) workload generation, and (3) data analysis.

For EC2-Pairs, the benchmarking process is straightforward, with the EC2 instance generating workload on datastores using k6 and storing the collected data locally. For Lambda-Pairs, each Lambda invocation corresponds to a single read operation on the datastore. To invoke the Lambda function programmatically and collect results, an additional EC2 instance, termed as "Lambda-Helper," is provisioned with k6 and the necessary test scripts, as shown in **??**.

All benchmarking runs are executed in the *eu-central-1* AWS region. All components of a run are provisioned and de-provisioned using Infrastructure-as-Code combined with shell scripting. This approach ensures a fresh environment for every run and minimizes the potential for human error. Upon successful completion of a run, the collected data is exported to a dedicated S3 bucket. Additionally, to keep network conditions and resource allocation identical, no benchmarking runs are executed simultaneously.

–> Diagram here <–

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.

DynamoDB: A DynamoDB table with 10 items, read capacity of 25 Read Capacity Units (RCUs), write capacity of 5 Write Capacity Units (WCUs), and configured with *Provisioned* capacity mode.

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 (HVM-based) and is equipped with the k6 and corresponding test scripts.

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

4.2 Workload Generation

With following workload configuration we specifically address relevance, affordability, repeatability, and cloud performance fluctuations. However, for relevance, there is no standard rate or load pattern at which, for instance, a backend server reads from a datastore, and depends highly on the application's use-case and demand. Instead, we focus on extending the run duration to a point where it can be repeated twice while remaining within AWS Free Tier. Time limitations are also considered.

Constant Workload:

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

Burst Workload:

- **RDS and DynamoDB**: The run starts with a baseline of 2 RPS for the first 2 hours, followed by 30 load spikes. Each spike lasts 3 minutes, with RPS increasing linearly from 2 to 20 in 90 seconds and then decreasing back to the baseline of 2 in 90 seconds. Between spikes, the load remains at 2 RPS for 3 minutes. The run concludes with an additional 2 hours at the 2 RPS, resulting in a total test duration of approximately 7 hours.
- **S3**: The run starts with a baseline of 0.2 RPS for the first 20 minutes, followed by 6 load spikes. Each spike lasts 30 seconds, with RPS increasing linearly from 0.2 to 16 in 15 seconds and then decreasing back to the baseline of 0.2 in 15 seconds. Between spikes, the load remains at 0.2 RPS for 6 minutes. The run concludes with an additional 15 minutes at the 0.2 RPS, resulting in a total test duration of approximately 70 minutes.

As noted, the monthly limits for S3 are relatively low, specifically 20000 GET requests per month, which resulted in shorter runs with S3.

4.3 Data Analysis

As discussed, for each targeted datastore service, we compare latency performance across EC2 and Lambda. Mean values alone cannot fully capture the behavior of access latency throughout the experiment, so we use bar plots for aggregated metrics and time-series graphs to provide a comprehensive view of the data.

In our context, outliers in latency values may indicate potential network issues between services and can distort analysis, particularly when calculating averages [BWT17]. To address this, we identify and remove outliers using the 3-sigma rule, which defines data points beyond three standard deviations from the mean as outliers [<**empty citation**>]. However, the number and extent of these outliers provide valuable insights into a service pair's performance and reliability. Therefore, we also analyze and report outliers to understand their implications and highlight variations between pairs.

5 Benchmark Results

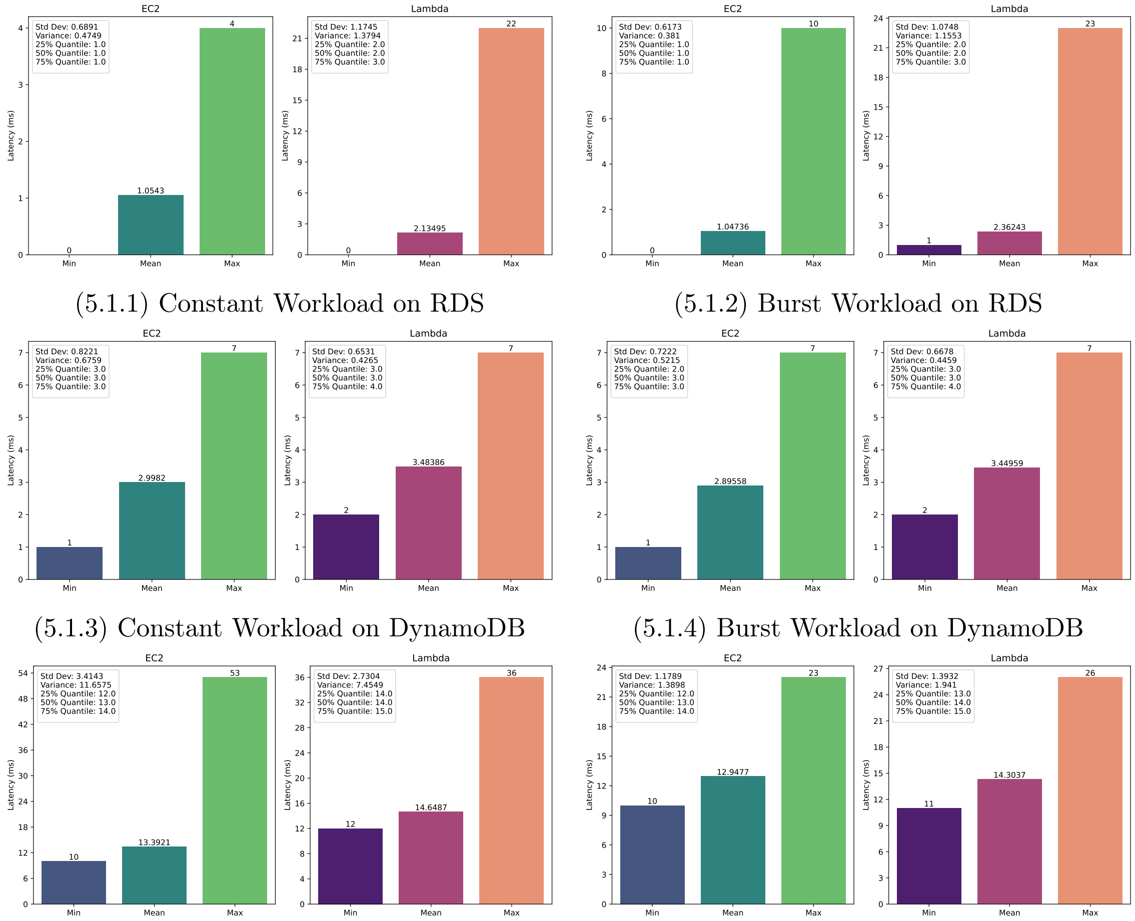
In this section, we present and compare the findings from the data analysis for each targeted datastore, reporting values on a millisecond (ms) scale. Throughout the benchmarking runs, no CPU performance issues are observed on the client side (workload generator), ensuring that latency measurements remain unaffected by client limitations. Furthermore, the error rate consistently registers zero across all runs, confirming that every read request is successfully executed.

EC2-RDS vs. Lambda-RDS Figure 5.1.1 and 5.1.2 show that EC2-RDS outperforms Lambda-RDS by approximately 51% under constant and 56% under burst workload. EC2-RDS shows 41% and 43% lower standard deviation under constant and burst workload, respectively. Maximum latency values of EC2-RDS are 82% and 56% lower under constant and burst workload, respectively. Figure 5.2.1 and 5.2.2 show that Lambda-RDS latency often experiences abrupt increases or decreases and persists at such levels for several hours, but in contrast EC2-RDS latency remains comparatively stable. The second runs of EC2-RDS under both workload patterns outperform the initial runs. Across both runs and workloads, EC2-RDS records 962 outliers with latencies between 5ms and 464ms, while Lambda-RDS records 1879 outliers, ranging from 11ms to 818ms.

EC2-DynamoDB vs. Lambda-DynamoDB Figure 5.1.3 and 5.1.4 show that EC2-DynamoDB outperforms Lambda-DynamoDB by approximately 14% under constant and 16% under burst workload. Lambda-DynamoDB shows 21% and 7% lower standard deviation under constant and burst workload, respectively. Maximum latency values are identical across both pairs and modes. Although Lambda-DynamoDB shows lower latency variance, Figure 5.2.3 and 5.2.4 reveal that Lambda-DynamoDB experiences sudden temporary latency spikes under both workloads. These spikes reach peaks significantly higher than those observed with EC2-DynamoDB. Similar to EC2-RDS, the second runs of EC2-DynamoDB under both workloads outperform the initial runs, especially under burst workload. Across both runs and workloads, EC2-DynamoDB records 3956 outliers with latencies between 7ms and 100ms, while Lambda-DynamoDB records 3073 outliers, ranging from 8ms to 209ms.

EC2-S3 vs. Lambda-S3 Figure 5.1.5 and 5.1.6 show that EC2-S3 outperforms Lambda-S3 by approximately 9% under both constant and burst workload. Lambda-S3 shows 20% lower standard deviation under constant but 15% higher under burst workload. Maximum latency value of Lambda-S3 is 32% lower under constant but 11% higher under burst workload. Unlike RDS and DynamoDB, Figure 5.2.5 and 5.2.6 show that Lambda-S3 latency does not experience temporal performance shifts or large spikes. Additionally, for EC2-S3, the second runs under both workload patterns do not show better performance compared to the initial runs, a trend observed with RDS and DynamoDB. Across both runs and workloads, EC2-S3 records 128 outliers with latencies between 22ms and 834ms, while Lambda-S3 records 148 outliers, ranging from 26ms to 390ms.

Summary The benchmark results indicate that RDS, DynamoDB, and S3 paired with EC2 generally outperform those paired with Lambda in terms of mean latency. However, the latency difference between EC2- and Lambda-based pairs remains minimal, consistently under 1.34ms across all experiments. Standard deviation and outliers, on the other hand, do not show a consistent pattern across the pairs.



(5.1.1) Constant Workload on RDS

(5.1.2) Burst Workload on RDS

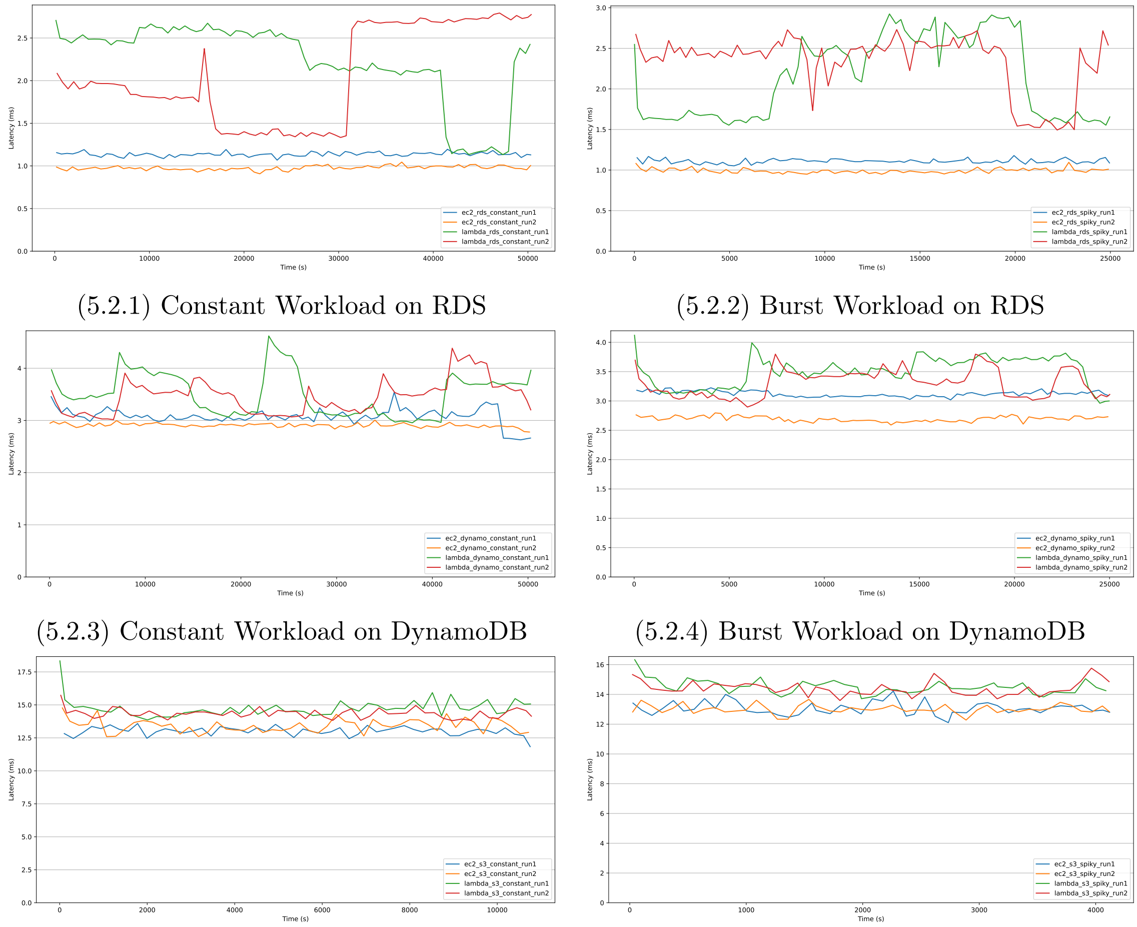
(5.1.3) Constant Workload on DynamoDB

(5.1.4) Burst Workload on DynamoDB

(5.1.5) Constant Workload on S3

(5.1.6) Burst Workload on S3

Figure 5.1: Aggregation of Latency Measurements



(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

Figure 5.2: 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.

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 $0.06ms$ cite. However, this overhead appears negligible compared to the observed differences, suggesting that additional factors contribute to the latency variations.

7 Related Work

–> TODO <–

8 Conclusion

Cloud computing has experienced rapid growth, becoming integral to industries seeking scalable and flexible solutions. AWS has emerged as a leader, offering diverse services for computing and storage. Despite its significance, limited empirical data exists on the latency characteristics of AWS compute-datastore interactions, necessitating this study to bridge the knowledge gap.

This thesis investigated the latency performance between AWS compute and datastore service pairs, specifically EC2 and Lambda paired with RDS, DynamoDB, and S3. A systematically conducted benchmark evaluated each pair under constant and burst workloads. Results indicated that EC2 pairs consistently demonstrated lower latency compared to Lambda pairs.

The study highlighted the influence of computing service choices on latency performance and provided empirical data to support architectural decisions in cloud environments. Despite limitations, such as the constraints of AWS Free Tier and the exclusion of mixed workload scenarios, the findings offer foundational insights into the dynamics of service interactions in AWS.

Future work can expand on these results by exploring higher-tier configurations, multi-region setups, and mixed read-write workloads to provide a more comprehensive understanding of cloud service latency characteristics.

References

[BWT17] David Bermbach, Erik Wittern, and Stefan Tai. *Cloud Service Benchmarking: Measuring Quality of Cloud Services from a Client Perspective.* Cham, Switzerland: Springer, 2017. ISBN: 978-3-319-85672-8.

[Bin+09] Carsten Binnig, Donald Kossmann, Tim Kraska, and Simon Loesing. "How is the weather tomorrow? Towards a benchmark for the cloud". In: *Proceedings of the Second International Workshop on Testing Database Systems.* 2009, pp. 1–6.

[Coo+10] Brian F. Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. "Benchmarking Cloud Serving Systems with YCSB". In: *Proceedings of the 1st ACM Symposium on Cloud Computing* (Indianapolis, Indiana, USA). SoCC '10. New York, NY, USA: Association for Computing Machinery, 2010, pp. 143–154. DOI: `10.1145/1807128.1807152`.

[Fol+13] Enno Folkerts, Alexander Alexandrov, Kai Sachs, Alexandru Iosup, Volker Markl, and Cafer Tosun. "Benchmarking in the cloud: What it should, can, and cannot be". In: *Selected Topics in Performance Evaluation and Benchmarking: 4th TPC Technology Conference, TPCTC 2012, Istanbul, Turkey, August 27, 2012, Revised Selected Papers 4.* Springer. 2013, pp. 173–188.