TECHNISCHE UNIVERSITÄT BERLIN

Exploring Datastore Access Latency across AWS Compute Services

Bachelor's Thesis

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

Technische Universität Berlin
Fakultät Elektrotechnik und Informatik
Fachgebiet Scalable Software Systems

2024

Eigenständigkeitserklärung

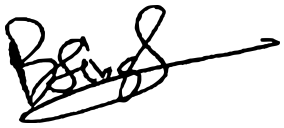
Hiermit versichere ich, dass ich die vorliegende Arbeit eigenständig ohne Hilfe Dritter und ausschließlich unter Verwendung der aufgeführten Quellen und Hilfsmittel angefertigt habe. Alle Stellen die den benutzten Quellen und Hilfsmitteln unverändert oder sinngemäß entnommen sind, habe ich als solche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

Sofern generative 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 15. Februar 2023* habe ich zur Kenntnis genommen.

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

Berlin, den 9. Dezember 2024



(Unterschrift) Bhupendra Dev Singh

*https://www.static.tu.berlin/fileadmin/www/10002457/K3-AMBl/Amtsblatt_2023/Amtliches_Mitteilungsblatt_Nr._16_vom_30.05.2023.pdf

Abstract

Cloud computing has become essential for modern applications, offering scalable and flexible services. Amazon Web Services (AWS) is a leading provider, yet little empirical data exists on how its compute and datastore services interact in terms of access latency. This thesis benchmarks access latency between EC2 and Lambda compute services paired with RDS, DynamoDB, and S3 datastores under constant and burst workloads. The results reveal that EC2-based pairs consistently deliver lower latency and more stable performance compared to Lambda-based pairs, which show greater variability. These insights can guide cloud architects in designing latency-sensitive applications. However, limitations such as simplified configurations, AWS Free Tier constraints, and single-region testing highlight the need for further studies to explore more complex and distributed scenarios. This thesis provides initial foundational insights that can serve as a basis for further exploration and development.

Kurzfassung

Cloud-Computing ist unverzichtbar für moderne Anwendungen geworden, da es skalierbare und flexible Dienste bietet. Amazon Web Services (AWS) ist ein führender Anbieter, jedoch es gibt nur wenige empirische Daten darüber, wie seine Compute- und Datenspeicherdienste in Bezug auf Zugriffslatenz zusammenwirken. Diese Arbeit benchmarkt die Zugriffslatenz zwischen den Compute-Diensten EC2 und Lambda in Kombination mit den Datenspeicherdiensten RDS, DynamoDB und S3 unter konstanten und variierenden Arbeitslasten. Die Ergebnisse zeigen, dass EC2-basierte Paare konsistent niedrigere Latenzen und eine stabilere Leistung liefern als Lambda-basierte Paare, die größere Schwankungen aufweisen. Diese Erkenntnisse können Cloud-Architekten dabei unterstützen, latenzsensible Anwendungen zu entwickeln. Einschränkungen wie vereinfachte Konfigurationen, die Beschränkungen des AWS Free Tiers und die Begrenzung auf eine einzige Region verdeutlichen jedoch die Notwendigkeit weiterer Studien, um komplexere und verteilte Szenarien zu untersuchen. Diese Arbeit liefert erste grundlegende Einblicke, die als Basis für weitere Untersuchungen und Entwicklungen dienen können.

Acknowledgements

I would like to express my sincere gratitude to my supervisor, Trever Schirmer (Research Associate and PhD student at the Scalable Software Systems TU Berlin), for his invaluable guidance, constructive feedback, and continuous support throughout the development of this thesis. His expertise and encouragement have been instrumental in shaping my work and deepening my understanding of the subject matter.

I acknowledge the use of ChatGPT (GPT-4o and GPT-4o mini, OpenAI, https://chatgpt.com/) and Grammarly (https://grammarly.com/) for paraphrasing, language improvement, and text correction in this thesis.

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 31% market share driven by its innovation and broad service portfolio [Ric24].

Applications in cloud environments typically consist of compute and datastore service instances that interact frequently, therefore latency between these services significantly influences the overall performance and user experience of the application [DB13; PZM17]. 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 performs internally. In contrast, the client-side perspective focuses on the time it takes 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), which offers customizable virtual machines (VMs) called EC2 instances, for consistent performance and long-running workloads, and (2) Lambda, a serverless, event-driven service ideal for short tasks and automatic scaling. Popular AWS datastore services include (1) Relational Database Service (RDS), which 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.

In this thesis, we benchmark access latency between 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. This thesis provides insights for cloud architects and application developers to optimize AWS compute and datastore service selection based on access latency, supporting decision-making for latency-sensitive applications and improving cloud architecture and cost-efficiency.

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

Access latency refers to the time required for data to travel between two points, often termed latency or Round-Trip Time (RTT). In the context of this thesis, it specifically refers to the time a compute service instance requires to obtain a response from the datastore. It is a well-known fact, that even small amounts of delay, even on a microsecond scale, may lead to a negative impact on the performance of an application [DB13; PZM17], which implies the importance of latency. From a cloud user's perspective, latency between cloud services can be influenced by certain controllable factors, such as service configuration, choice of region and availability zone (AZ), the proximity of resources, selecting instance types or service tiers optimized for low latency, network configuration, caching strategies, and application code optimization. However, underlying cloud infrastructure, especially network conditions remains uncontrollable.

Benchmarking carries different definitions across domains, however, in this thesis, it specifically refers to systematically evaluating and comparing access latency between AWS compute and datastore services. Cloud benchmarking is a well-researched field, therefore the following subsections summarize the key elements and requirements of cloud benchmarking, drawing from existing literature [BWT17; Bin+09; Coo+10; Fol+13]. Additionally, we define relevant terms to provide clarity for subsequent sections.

2.1 Elements of Cloud Benchmarking

Cloud benchmarking typically involves a System Under Test (SUT) and a workload generator. The SUT may represent a single cloud service or an entire application containing components of interest. The workload generator applies an artificial load on the SUT while tracking the metric of interest. In this thesis, an AWS service pair consisting of a compute instance and a datastore constitutes the SUT. However, when the compute instance itself generates load on the datastore, the tool or executable responsible for load generation is considered the workload generator.

Workload generation is a critical element of cloud benchmarking, as it directly impacts SUT. Two common workload models are open and closed models. In a closed model, a fixed number of concurrent threads independently execute a predefined sequence of tasks iteratively. In contrast, the open model specifies a rate of arrival, such as request per second. The closed model has a fundamental limitation: it ties load generation to task completion. New tasks are only scheduled once a thread completes the previous one, allowing the SUT to self-regulate. If the SUT slows or stalls, incoming load decreases, enabling recovery and potentially skewed results.

A benchmarking run refers to a single-timed execution of a predefined set of workload to evaluate the performance of an SUT under specific conditions. It involves provisioning the resources, initiating the workload, collecting relevant performance data, and storing the results for further analysis.

2.2 Requirements of Cloud Benchmarking

Cloud benchmarking requires several general considerations: **Relevance** ensures that the benchmarks reflect real-world scenarios and test conditions that align with the application's needs. **Repeatability** ensures that results can be consistently reproduced under similar conditions, enhancing the reliability of the findings. **Fairness** is vital to ensure that all SUTs are compared equitably, with identical configurations, resources, and network

conditions to avoid bias. **Affordability** focuses on minimizing the costs of conducting the benchmark, particularly when using cloud resources with strict budget constraints. Additionally, **simplicity** is important for creating benchmarks that are easy to understand, interpret, and build trust.

To meet advanced cloud benchmarking requirements, benchmarks should account for **failure scenarios**, evaluating how services handle failure conditions. They must also consider the **geo-distribution** of both measurement clients and the SUT, reflecting real-world, geographically dispersed applications. Additionally, benchmarks should provoke **stress situations** to measure qualities like **scalability and elasticity**, using variable load patterns to evaluate service limits. The design must ensure **detailed data capture**, avoiding reliance solely on aggregate values, and should track resource consumption and costs across various components using monitoring tools. Finally, benchmarks should be **long-running** and executed across different times of day and days of the week to capture stabilized behavior, short-term effects, and seasonal variations [BWT17].

Trade-offs between requirements are inevitable, as some are inherently conflicting. For example, prioritizing relevance may lead to complex applications that reduce simplicity, while long-running experiments can increase costs, affecting affordability. Therefore, it is essential to carefully consider all requirements and make conscious trade-offs based on the benchmarking goals and scope [BWT17].

2.3 Terms and Definitions

k6[1] k6 is an open-source and extensible workload generation tool. It uses JavaScript-based test scripts to define workloads, specifying parameters such as duration, arrival rate (or concurrent threads), load patterns, and operations on the SUT. It is written in the golang with embedded JavaScript engine[2].

Virtual Private Cloud (VPC) VPC is an isolated network within the AWS cloud that allows users to launch AWS resources, such as EC2 instances, within a defined IP address range. It provides control over network configuration, including subnets, route tables, and security settings, enabling secure and scalable deployment of cloud resources.

Lambda Lambda is a serverless computing service that automatically runs code in response to events, eliminating the need to manage the underlying infrastructure. It scales automatically based on incoming requests, with users only charged for execution time. It introduces cold starts, where a new execution environment is initialized when the function is invoked after being idle, causing slight latency. To reduce this, *provisioned concurrency* is configured to keep a predefined number of instances ready to handle requests, while *reserved concurrency* limits the maximum concurrent executions per function.

DynamoDB DynamoDB is a fully managed NoSQL database service designed for low-latency, high-throughput applications. It supports the *provisioned capacity* mode, where users allocate Read Capacity Units (RCUs) and Write Capacity Units (WCUs) to define the number of reads and writes per second the database can handle. An RCU represents one strongly consistent read per second for an item up to 4 KB, while a WCU allows one write per second for an item up to 1 KB.

[1] **https://k6.io/open-source/**.
[2] **https://github.com/grafana/k6**.

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, as the whole experiment is conducted within the limits of AWS Free Tier[3]. This 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 SUT, and (2) a compute instance that generates workload on the SUT and collects access latency data. For simplicity, we refer to the datastore as SUT and the compute instance as the workload generator in this thesis. However, in our context, a compute instance and a datastore together constitute one SUT, as we are not particularly benchmarking the datastores.

To address relevance, we employ an open workload model, due to the reasons discussed in Section 2.1 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 the 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 a total of 24 runs.

As discussed in Section 2.2, benchmarks should be long-running. To address this, 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.

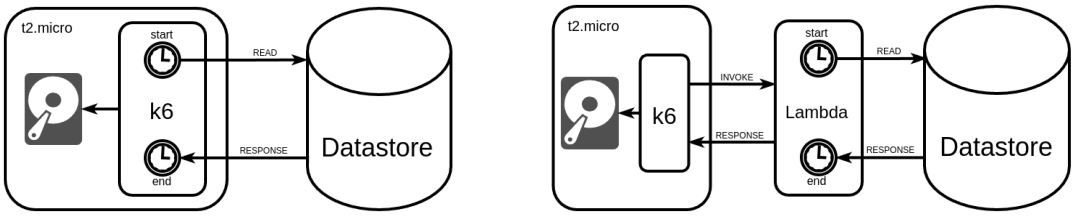
[3] **https://aws.amazon.com/free/**.

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 "Lambda-Helper", is provisioned with k6 and the necessary test scripts, as shown in Figure 4.1.

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.



(4.1.1) EC2 based Pairs (4.1.2) Lambda based Pairs

Figure 4.1: Implementation Overview of EC2 and Lambda with Datastores

4.1 Resource Configuration

RDS: A MySQL 8.0 instance on a t3.micro VM with 1GiB RAM, 2 vCPUs, and 5GB storage. The instance is located in the *eu-central-1a* AZ, 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 in *provisioned capacity* mode with read capacity of 25 RCUs, write capacity of 5 WCUs, and containing 10 items.

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* AZ. 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 reserved concurrency set to 990. The configuration of Lambda-Helper is identical to that of EC2.

4.2 Workload Generation

With the following workload configuration, we specifically address relevance, affordability, and repeatability. 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 the 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, hence 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 [HKP11]. 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

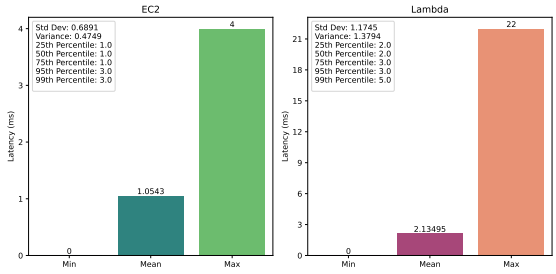
In this section, we present and compare the findings from the data analysis for each targeted datastore, reporting values on a millisecond (ms) scale. Throughout the benchmarking runs, no CPU performance issues are observed on the client side (workload generator), ensuring that latency measurements remain 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 Figures 5.2.1 and 5.2.2 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. Figures 5.3.1 and 5.3.2 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 Figures 5.2.3 and 5.2.4 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, Figures 5.3.3 and 5.3.4 reveal that Lambda-DynamoDB exhibits sudden, temporary latency spikes under both workloads, with peaks significantly exceeding those of EC2-DynamoDB. As with EC2-RDS, the second runs of EC2-DynamoDB under both workloads outperform the initial runs, particularly during the 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 $209ms$.

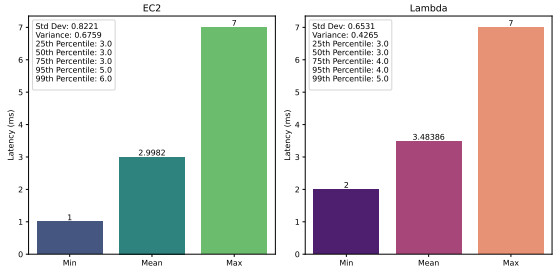
EC2-S3 vs. Lambda-S3 Figures 5.2.5 and 5.2.6 show that EC2-S3 outperforms Lambda-S3 by approximately 9% under both constant and burst workload. Lambda-S3 shows a 20% lower standard deviation under constant but 15% higher under burst workload. The maximum latency value of Lambda-S3 is 32% lower under constant but 11% higher under burst workload. Unlike RDS and DynamoDB, Figures 5.3.5 and 5.3.6 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 absolute difference between EC2- and Lambda-based pair's mean latencies 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.



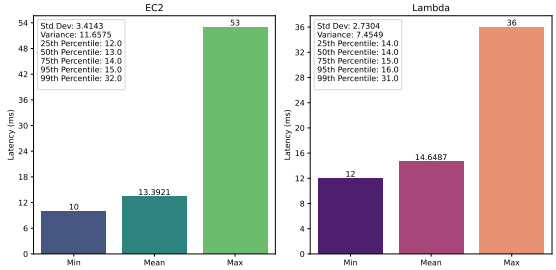
(5.2.1) Constant Workload on RDS

(5.2.2) Burst Workload on RDS



(5.2.3) Constant Workload on DynamoDB

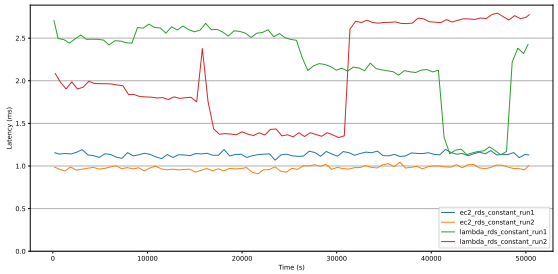
(5.2.4) Burst Workload on DynamoDB



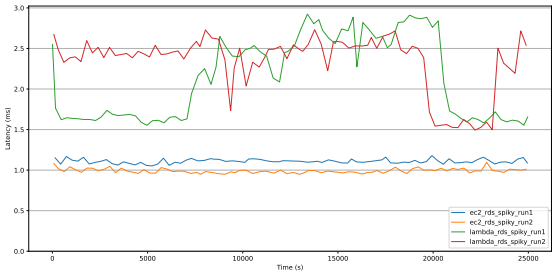
(5.2.5) Constant Workload on S3

(5.2.6) Burst Workload on S3

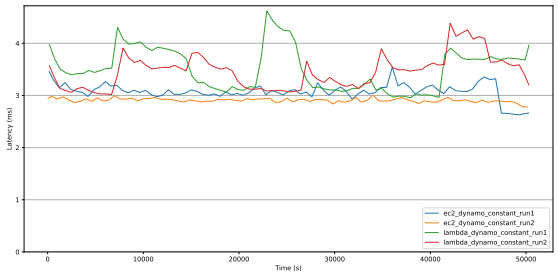
Figure 5.2: Aggregation of Latency Measurements



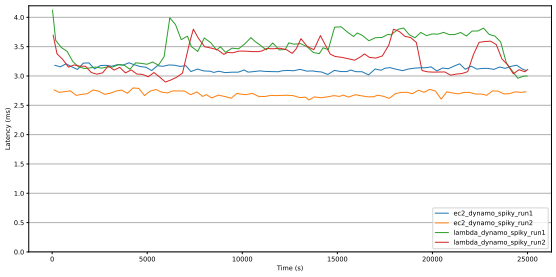
(5.3.1) Constant Workload on RDS



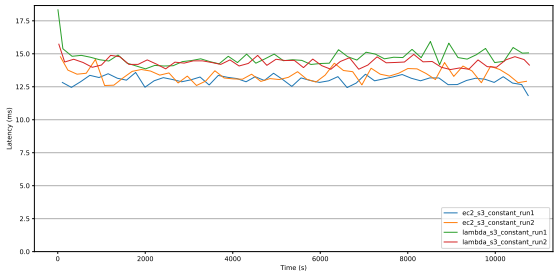
(5.3.2) Burst Workload on RDS



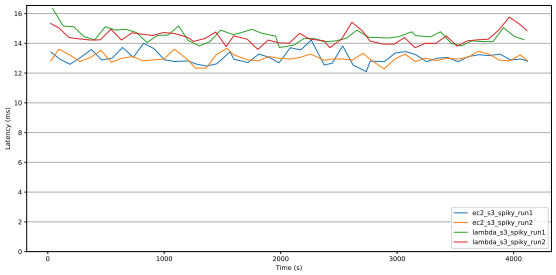
(5.3.3) Constant Workload on DynamoDB



(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

Section 5 shows that datastore access latency is influenced by the choice of compute service, with EC2 consistently outperforming Lambda across all experiments. This suggests an inherent variance and overhead associated with Lambda. In this section, we explore possible reasons for these observations and also assess our approach and its limitations.

In Figure 5.3, we observe temporal shifts in the latency performance, even under constant workload. Temporal performance variability in serverless platforms, including AWS Lambda, is well-documented [Eis+22; GF21; Sch+23]. Ginzburg et al. [GF21] report significant performance shifts in AWS Lambda during a one-week continuous benchmark, even suggesting that these variations could be exploited to reduce costs. It is possible that our benchmarking runs, specifically Lambda-RDS (Figure 5.3.1 and 5.3.2) and Lambda-Dynamo (Figure 5.3.3 and 5.3.4), were affected by such temporal shifts, impacting the results. Similar is also applicable for EC2 [IYE11; SDQR10], however, Dancheva et al. [DAB24] suggest that recent advances are able to improve networking performance in EC2.

Lambda functions run inside microVMs based on Firecracker[4] hypervisor, deployed on bare-metal, multi-tenant EC2 instances referred as Lambda-Workers. Unlike standard EC2 instances based on the AWS Nitro[5] hypervisor, Lambda-Workers add Firecracker as an additional virtualization layer. Communication between a microVM and Firecracker occurs via optimized virtio[6] interface [Bro+23], leading to overall networking overhead of approximately $0.06ms$ under controlled, isolated conditions [Fir24]. In production, however, Lambda-Workers may host hundreds or thousands of microVMs [Aga+20], potentially increasing this overhead due to induced network noise and resource contention [DS+22; Wan+18]. Agache et al. [Aga+20] also highlight Firecracker's comparatively lower networking performance against other hypervisors. Furthermore, Lambda-Workers are hosted within a network-isolated VPC managed by Lambda in the service accounts inaccessible to customers [TBT22], leaving their precise network configuration unclear.

We observe a minimum latency of $0ms$ in Figure 5.2.1 and 5.2.2, which is due to the $1ms$ measurement accuracy, causing latencies below $1ms$ to be rounded to $0ms$. As the differences are small, on the microsecond level, high-resolution timestamps with $1\mu s$ accuracy are better suited for more precise results.

Limitations In Section 2, we outline general aspects and requirements of cloud benchmarking, present our approach in Section 3, and validate it through experimentation. However, affordability emerges as a significant limitation. AWS Free Tier constraints restrict experiment duration and scale, limiting long-term performance trend observations. The use of low-tier configurations, such as t2.micro instances and no provisioned concurrency for Lambda, hinder evaluation under realistic production conditions. Temporal variability in cloud performance poses another challenge, with short-duration benchmarks, particularly for S3, possibly failing to capture the real empirical distribution of access latency. The absence of multi-region setups excludes the analysis of geo-distributed latency behaviors.

[4] **https://firecracker-microvm.github.io/**.
[5] **https://aws.amazon.com/ec2/nitro/**.
[6] **https://libvirt.org/**.

7 Related Work

Cloud computing is a widely adopted paradigm, resulting in extensive literature exploring various aspects of its ecosystem. Given the scale of cloud applications and architectures, this study falls under the category of micro-benchmarking, where existing work remains relatively limited. Nonetheless, there are overlaps with studies focusing on different but related aspects of cloud performance.

Villamizar et al. [Vil+16] compare infrastructure costs and performance of web applications deployed using three architectures: Lambda-based microservices, EC2-based microservices, and monolithic. For data storage, they use PostgreSQL on EC2, though it is unclear if RDS is employed. Unlike this thesis, their study measures latency from a web user's perspective, factoring in variables like gateways, cold starts, load balancing, and internal communication overhead. Their findings indicate that the Lambda-based microservice architecture achieves better average response times for end users.

Klimovic et al. [Kli+18] evaluate the suitability of storage options for serverless analytics, including S3, Redis[7], and Crail-ReFlex[8], to address ephemeral data sharing requirements. They highlight the overhead of using S3 for latency-sensitive serverless applications, identifying Redis as a lower-latency alternative. Their study examines latency between AWS Lambda and S3 for both read and write operations, reporting 12.1ms read and 25.8ms write latency for 1KB requests. Additionally, they analyze aspects like I/O time, throughput, job runtime, and concurrency.

Palepu et al. [Pal+22] benchmark data transfer rates across serverless and datastore services, including AWS Lambda, S3, and DynamoDB. They identify factors affecting performance, such as memory allocation and concurrency, and highlight Lambda's limitations in scaling data transfer rates under high concurrency. The study reports that Lambda exhibits lower throughput with S3 and DynamoDB compared to EC2 but does not provide specific latency insights.

Albuquerque-Junior et al. [AJ+17] benchmark AWS Lambda against AWS Beanstalk[9], which is based on EC2 instances, to evaluate performance in microservice architectures. They use PostgreSQL for data persistence, but it is unclear whether it is RDS. Unlike our approach, they focus on end-to-end performance using a closed workload model, with experiments lasting just over 4 minutes and an unspecified number of repetitions. There are fairness concerns due to the resource configuration: Lambda is allocated 256MB of memory (0.167$vCPU$), while Beanstalk uses a t1.micro EC2 instance with 1$vCPU$ and 0.613GB of memory. They also examine the impact of resource allocation and service configuration. Their results show that EC2 with Beanstalk outperforms Lambda in terms of both read and write latencies.

[7] **https://redis.io/**.
[8] **https://craillabs.github.io/**.
[9] **https://aws.amazon.com/elasticbeanstalk/**.

8 Conclusion

This thesis systematically examined datastore access latency across AWS compute services, specifically focusing on EC2 and Lambda paired with RDS, DynamoDB, and S3, under two workload types. The results indicate that EC2 consistently outperforms Lambda in terms of mean latency, however with minor differences. Lambda-based pairs exhibited greater latency susceptibility to performance shifts, likely due to inherent architectural overheads and variability. These findings emphasize EC2's stability for latency-sensitive applications while highlighting the trade-offs of Lambda's serverless model.

The study also faced limitations, including constraints of the AWS Free Tier, short duration, the use of simplified configurations, and single-region setups. These factors underscore the need for future research to incorporate multi-region benchmarks, advanced configurations, and mixed workload scenarios for a deeper understanding of cloud service interactions. The insights gained from this work provide a foundation for cloud architects and application developers to optimize AWS service selections, balancing performance, scalability, and cost-efficiency.

References

[AJ+17] Lucas F Albuquerque Jr, Felipe Silva Ferraz, RFAP Oliveira, and S Galdino. "Function-as-a-service x platform-as-a-service: Towards a comparative study on FaaS and PaaS". In: *ICSEA*. 2017, pp. 206–212.

[Aga+20] Alexandru Agache, Marc Brooker, Alexandra Iordache, Anthony Liguori, Rolf Neugebauer, Phil Piwonka, and Diana-Maria Popa. "Firecracker: Lightweight Virtualization for Serverless Applications". In: *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. Santa Clara, CA: USENIX Association, Feb. 2020, pp. 419–434. ISBN: 978-1-939133-13-7.

[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] C. Binnig, D. Kossmann, T. Kraska, and S. 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.

[Bro+23] Marc Brooker, Mike Danilov, Chris Greenwood, and Phil Piwonka. "On-demand Container Loading in AWS Lambda". In: *2023 USENIX Annual Technical Conference (USENIX ATC 23)*. Boston, MA: USENIX Association, July 2023, pp. 315–328. ISBN: 978-1-939133-35-9.

[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.

[DAB24] Tamara Dancheva, Unai Alonso, and Michael Barton. "Cloud benchmarking and performance analysis of an HPC application in Amazon EC2". In: *Cluster Computing* 27.2 (2024), pp. 2273–2290.

[DB13] Jeffrey Dean and Luiz André Barroso. "The tail at scale". In: *Communications of the ACM* 56.2 (2013), pp. 74–80.

[DS+22] Daniele De Sensi, Tiziano De Matteis, Konstantin Taranov, Salvatore Di Girolamo, Tobias Rahn, and Torsten Hoefler. "Noise in the Clouds: Influence of Network Performance Variability on Application Scalability". In: *Proc. ACM Meas. Anal. Comput. Syst.* 6.3 (Dec. 2022). DOI: 10.1145/3570609.

[Eis+22] Simon Eismann, Diego Elias Costa, Lizhi Liao, Cor-Paul Bezemer, Weiyi Shang, André van Hoorn, and Samuel Kounev. "A case study on the stability of performance tests for serverless applications". In: *Journal of Systems and Software* 189 (2022), p. 111294. ISSN: 0164-1212. DOI: https://doi.org/10.1016/j.jss.2022.111294.

[Fir24] Firecracker. *Firecracker: Secure and fast microVMs for serverless computing.* Accessed: Dec. 1, 2024. Nov. 2024. URL: https://github.com/firecracker-microvm/firecracker/blob/main/docs/network-performance.md.

[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.

[GF21] Samuel Ginzburg and Michael J. Freedman. "Serverless Isn't Server-Less: Measuring and Exploiting Resource Variability on Cloud FaaS Platforms". In:

WoSC '20. Delft, Netherlands: Association for Computing Machinery, 2021, 43–48. ISBN: 9781450382045. DOI: 10.1145/3429880.3430099.

[HKP11] J. Han, M. Kamber, and J. Pei. *Data Mining: Concepts and Techniques*. The Morgan Kaufmann Series in Data Management Systems. Morgan Kaufmann, 2011. ISBN: 9780123814807.

[IYE11] Alexandru Iosup, Nezih Yigitbasi, and Dick Epema. "On the performance variability of production cloud services". In: *2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. IEEE. 2011, pp. 104–113.

[Kli+18] Ana Klimovic, Yawen Wang, Christos Kozyrakis, Patrick Stuedi, Jonas Pfefferle, and Animesh Trivedi. "Understanding Ephemeral Storage for Serverless Analytics". In: *2018 USENIX Annual Technical Conference (USENIX ATC 18)*. Boston, MA: USENIX Association, July 2018, pp. 789–794. ISBN: 978-1-939133-01-4.

[PZM17] DA Popescu, Noa Zilberman, and Andrew Moore. *Characterizing the impact of network latency on cloud-based applications' performance*. 2017. DOI: 10.17863/CAM.17588.

[Pal+22] Surya Chaitanya Palepu, Dheeraj Chahal, Manju Ramesh, and Rekha Singhal. "Benchmarking the Data Layer Across Serverless Platforms". In: *Proceedings of the 2nd Workshop on High Performance Serverless Computing*. HiPS '22. Minneapolis, MN, USA: Association for Computing Machinery, 2022, 3–7. ISBN: 9781450393119. DOI: 10.1145/3526060.3535460.

[Ric24] Felix Richter. *Worldwide Market Share of Leading Cloud Infrastructure Service Providers*. Accessed: Dec. 1, 2024. Statista. Nov. 2024. URL: https://www.statista.com/chart/18819/.

[SDQR10] Jörg Schad, Jens Dittrich, and Jorge-Arnulfo Quiané-Ruiz. "Runtime measurements in the cloud: observing, analyzing, and reducing variance". In: *Proc. VLDB Endow.* 3.1–2 (Sept. 2010), 460–471. ISSN: 2150-8097. DOI: 10.14778/1920841.1920902.

[Sch+23] Trever Schirmer, Nils Japke, Sofia Greten, Tobias Pfandzelter, and David Bermbach. "The Night Shift: Understanding Performance Variability of Cloud Serverless Platforms". In: *Proceedings of the 1st Workshop on Serverless Systems, Applications and Methodologies*. SESAME '23. Rome, Italy: Association for Computing Machinery, 2023, 27–33. ISBN: 9798400701856. DOI: 10.1145/3592533.3592808.

[TBT22] Mayank Thakkar, Marc Brooker, and AWS Lambda Security Team. *Security Overview of AWS Lambda*. Accessed: Dec. 1, 2024. Amazon Web Services. Dec. 2022. URL: https://docs.aws.amazon.com/whitepapers/latest/security-overview-aws-lambda/.

[Vil+16] Mario Villamizar, Oscar Garcés, Lina Ochoa, Harold Castro, Lorena Salamanca, Mauricio Verano, Rubby Casallas, Santiago Gil, Carlos Valencia, Angee Zambrano, and Mery Lang. "Infrastructure Cost Comparison of Running Web Applications in the Cloud Using AWS Lambda and Monolithic and Microservice Architectures". In: *2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*. 2016, pp. 179–182. DOI: 10.1109/CCGrid.2016.37.

[Wan+18] Liang Wang, Mengyuan Li, Yinqian Zhang, Thomas Ristenpart, and Michael Swift. "Peeking Behind the Curtains of Serverless Platforms". In: *2018 USENIX Annual Technical Conference (USENIX ATC 18)*. Boston, MA: USENIX Association, July 2018, pp. 133–146. ISBN: ISBN 978-1-939133-01-4.