An Investigation Into Performance Efficiencies In AWS DynamoDB Configurations For Various Serverless Application Workloads

MSc Research Project
Cloud Computing

Vijayanand Somavaram
Student ID: x22147802

School of Computing
National College of Ireland

Supervisor: Aqeel Kazmi

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Vijayanand Somavaram
Student ID:	x22147802
Programme:	Cloud Computing
Year:	2023
Module:	MSc Research Project
Supervisor:	Aqeel Kazmi
Submission Due Date:	31/01/2024
Project Title:	An Investigation Into Performance Efficiencies In AWS DynamoDB Configurations For Various Serverless Application Workloads
Word Count:	7969
Page Count:	25

Signature:	S.Vijayanand
Date:	30th January 2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

An Investigation Into Performance Efficiencies In AWS DynamoDB Configurations For Various Serverless Application Workloads

Vijayanand Somavaram
x22147802

Abstract

NoSQL database systems are being widely used and undoubtedly gained a lot of popularity in the last decade or so. Many factors affect the performance of the NoSQL database system. Among these factors, the database system's throughput settings, including the read and write capacity units, can affect its performance. In many NoSQL databases, especially in cloud-based systems like Amazon DynamoDB, Azure Cosmos DB, or Google Cloud Big-table, users must provision the read and write capacity units. These systems allow users to manually provision the capacity units and provide services to auto-manage the throughput settings. This research studies how these capacity units affect the performance of database systems and focuses on identifying the optimal provisioned throughput settings considering the incoming traffic requests with predictable payloads, to eventually optimize the performance of severless environments that use these databases. If these values are under-provisioned it can lead to increased latency, throttling, and poor performance during high-traffic periods. If over-provisioned, it can lead to unnecessary costs. As part of this research, AWS DynamoDB which is one of the prominent NoSQL databases was considered. This study is about evaluating the performance of DynamoDB under the default settings provided by AWS and investigating whether these settings could be improved further to assess if latencies can be reduced even further. The experimental results indicate that read-intensive workloads showed improvement in latencies through additional capacity provisioning. However, write-intensive workloads did not exhibit improvement with additional capacity provisioning. This research utilized data collected from tests conducted and developed a machine-learning model that predicts the optimal read capacity units for requests under predictable workloads.

1 Introduction

As the web generates a variety of data that in nature is not structured, it becomes hard to manage it with relational databases. To solve this problem, the NoSQL databases have emerged. Over the past few decades, NoSQL database systems have gained an exponential rise in popularity, because of their ability to handle the demands of modern, data-intensive applications. Now, many factors affect the performance of the NoSQL databases. Factors like the type of Data Model, Scalability, Data Distribution, Query Complexity, Caching, Replication and Redundancy, Hardware, etc. One of the factors that influences the performance of a NoSQL database is its provisioned throughput settings, which include provisioning its read and write capacity units.

1.1 Background And Motivation

This research has been largely motivated by the work conducted by Palepu et al. (2022). The authors conduct experiments to Benchmark the Data Layer across Serverless Platforms. The paper talks about how the decoupling of computing or in this scenario, the serverless environments and their corresponding storage layers results in latencies and performance degradation. The paper also talks about how it is important to accelerate the to and fro data transfer between serverless environments and storage environments for building high-performance applications. Another key point the paper talks about is that the data transfer rate between serverless platforms and storage services depends on multiple factors such as the computing power of the resources, network bandwidths, design schema of the storage being used, etc.

It is important to note that one of the key factors that influence the performance of a Serverless application is the data storage layer from which all the necessary data transactions are done. From the complete application end-to-end environment, that is, both the serverless environment and the data layer, this research will focus on the data storage layer. The paper Idziorek et al. (2023) says that the response time for the process of storing or retrieving data based on the type of request results in delay as there are operations that have to be performed. Now many factors influence the time taken to respond to an incoming request by a NoSQL database, some of them are the data model, data distribution, data replication and redundancy, and so on. One of the factors is the provisioned read and write capacity units or the throughput capacity of the NoSQL database. The throughput capacity of a database refers to how many operations (reads or writes) the database can handle and process effectively to ensure a smooth application performance, within a given time frame.

A table in AWS DynamoDB can be created using two capacity modes, one of them is Provisioned Capacity mode where the read and write capacity units have to be provisioned manually by the user, and the other mode is On-demand capacity where the read and write capacity units are automatically managed by AWS. In either case, it is important to note that these settings directly impact the database's ability to handle concurrent read and write operations which will impact its performance. If these values are under-provisioned it can lead to increased latency, throttling, and poor performance during high-traffic periods and will result in bad user experience. If over-provisioned, it can lead to unnecessary costs. So, it is crucial to provision the system with appropriate values for optimum performance. This research uses AWS DynamoDB's provisioned capacity mode, to provide the ideal capacity units that optimize performance.

1.2 Research Question

Based on all the information presented in the above section, the following research question arises:

How can cloud database infrastructures be further improved through novel configurations for optimal performance in a serverless environment ?

1.3 Objectives

Optimizing with Novel Configurations:

- Determine the ideal read and write capacity units for AWS DynamoDB tables, which is a cloud and serverless database infrastructure.
- Develop and implement methodologies, elaborated in upcoming sections, to configure DynamoDB with novel settings, aiming for improved response times when receiving requests from a serverless Lambda function which is also an application serverless environment.

Machine Learning Model Development:

- Construct a machine learning model that is capable of predicting the ideal capacity units needed for various workloads within a Serverless Application environment.
- With the experimental data and methodologies developed in this study, enhance the accuracy and application of the machine learning model under various workloads.

1.4 Contribution

As a developer or a general user, the information to provision the DynamoDB table with the ideal and efficient capacity units is very little. It would be very beneficial to have a set of bench-marked guidelines and data that is derived from experimental evidence. The motive behind this research is to fill in this gap and serve as a helping guide to users who would like to incorporate DynamoDB services into their applications and want to understand the impact, provisioning capacity units have on the performance of its tables.

2 Related Work

This research has acquired a lot of information from many research papers and studies. The following sections will explore how these studies have had an impact on the current research.

2.1 Basis of the Research Idea

The research paper by (Palepu et al.; 2022) is on benchmarking the performance of different storage systems that can be used with serverless platforms, across multiple cloud vendors. The paper discusses that high scalability in serverless environments can be achieved by segregating the compute infrastructure from the storage layer. However, this decoupling of the compute layer from the storage layer results in latency. It also addresses the latency in serverless platforms resulting from using storage systems for data transfers. Several cloud service providers provide storage for serverless platforms, one of them is AWS DynamoDB. AWS DynamoDB is considered to be a low-latency, key-value store database system. The paper also discusses how adjusting various cloud service settings can enhance data transfer speed between serverless platforms and storage systems. The research paper Palepu et al. (2022) talks about storage systems from various cloud vendors, but this current research focuses on AWS serverless applications and AWS DynamoDB.

In the last decade or so, studies have shown the gain in popularity of NoSQL databases driven by the explosion of data in volume and size. (Filip and Čegan; 2020) shows the drop in the performance of relational databases when compared with NoSQL databases. Research paper by (Gomes et al.; 2019) talks about how NoSQL databases are the best at handling huge amounts of data in very little time, but it also mentions some trade-offs like on a few occasions data across different database replicas may not have the same consistent data. From (Kumar et al.; 2018) efficiency of a database system has been calculated by the amount of time it takes to perform write, read, and delete operations. From the research Seghier and Kazar (2021), on conducting read and write related tests on different NoSQL systems, the results show the performance of query execution times change with read and write-intensive workloads. In (Elhemali et al.; 2022) the authors, mention that for applications requiring strongly consistent reads, double the read capacity units are necessary, and also that write capacity units impact the speed of writing and replicating data across the number of database replicas created. In these cases, insufficient RCUs and WCUs may lead to request throttling ultimately affecting the application performance.

From Gunawan et al. (2019), a study was conducted to determine the estimated query response time of various NoSQL databases. As part of this research, different NoSQL database services were selected, and experiments were executed by repeatedly processing various types of queries, which included the CRUD operations. The queries were repeated multiple times, specifically 1000, 2000, 3000, 4000, and 5000 repetitions. After each query execution, the response time was recorded to analyze and understand the time taken by each database system to handle the requests. This process aimed to provide valuable insights into the abilities of each system.

To summarize the research papers mentioned in this section,

- NoSQL databases have performance advantages over traditional databases.
- NoSQL databases excel at handling large volumes of data with some trade-offs.
- The efficiency of a NoSQL database can be measured in terms of its ability to perform read and write operations in less time.
- The time taken by a database system to resolve a request is different for read-intensive tasks and different for write-intensive tasks.
- Under-provisioning read and write capacity units to a database system may lead to request throttling which is nothing but denial or delay of service.
- The capacity of different database systems was estimated by recording the query execution times.

Even though these articles provide only an overview of various aspects of NoSQL database systems relevant to this current research and do not dive deeper into issues like optimization, and configuring DynamoDB database systems to enhance efficiency, they provide a deeper understanding of the underlying structure of Serverless applications and database systems, especially DynamoDB, which serves as a basis for the current research undertaken.

2.2 Influence of Database Performance on Serverless Applications

The Journal by (Eismann et al.; 2021), talks about the increase in the adoption of serverless applications, its ease of use and efficiency are considered to be two of the most desirable properties of the services. It also hides usage of the server from the users and runs code on-demand which can be automatically scaled and billed only for the time the function is used. Serverless platforms relieve users from bothering about operational concerns like deployment, scaling, monitoring, and focus on development. In the research article by (Mvondo et al.; 2021) the authors, talk about how cloud-based applications that use the "Functions as a Service" (FaaS) paradigm have become very popular. Applications based on the Faas paradigm are designed to be stateless, which means they do not retain information about previous interactions or transactions. Each function execution is independent and doesn't store any data between executions. To maintain or retrieve data between these stateless executions, FaaS applications have to communicate with an external data store. This, in turn, leads to delays in Serverless Platforms. Another research article by (Filip and Čegan; 2020), compares the performance of SQL and NoSQL databases. It discusses the delay between requests to the database system and its fulfillment response during operations involving storing and retrieving data. Based on these research articles, It can be said that the performance of the external data store becomes crucial to minimize the latencies in Serverless Applications.

2.3 DynamoDB capacity units and their impact

In the research article by (Pelle et al.; 2019), the authors say that applications on stateless, serverless platforms like AWS Lambda require external data stores. Since applications in such scenarios rely on external data stores, the performance of all activities related to data store operations becomes very crucial. The research talks about DynamoDB along with other AWS services. They mention that the read and write capacity units of DynamoDB can be configured to achieve maximum performance and avoid request throttling. They however do not offer any further insights on how DynamoDB can be provisioned to achieve better performance.

In (Tantiphuwanart et al.; 2023) the authors say that for their application, the efficiency has been enhanced by Using DynamoDB in terms of both read and write operations for a large number of users. The research reveals that using DynamoDB for operations, especially for read operations, the response time decreased significantly. In this research, DynamoDB was used in an on-demand mode, however, the authors suggest that further enhancements could be achieved by switching to provisioned mode, a consideration left for their future work. Their future work also includes predicting the optimal provisioned throughput for DynamoDB, ultimately optimizing response times to surpass the performance of the on-demand mode. The current research undertaken aims to find the optimal provisioned capacity units under varying workloads to achieve lower response times.

From the experiments conducted in (Preuveneers and Joosen; 2020), results show the impact of the application workloads on the NoSQL database's latency. The workloads in the experiments consisted of different distributions of read and update operations. The results concluded that the variance in the latencies with write operations, and less variance with read operations. This means that a table in DynamoDB has to be configured according to the incoming varying workload traffic.

So what are Default settings, item size, RCUs, WCUs, On-demand Mode, and Provisioned Mode that are being talked about? Generally, Capacity Unit is the measure of the number of resources that are allocated to a system to perform a certain task.

- Default Settings: These are the RCUs and WCUs suggested by the AWS DynamoDB capacity calculator, how they are calculated will be discussed in further sections.
- Item Size: Item size represents the "memory of an element" or "size" or "Byte size" of an item involved in the operations. When it is said that 4 KB of item size is being read, It is referring to the size of the data(4KB) present in the DynamoDB table that is being read. Similarly, in the case of write operations, it is the size of the data that is being written to the DynamoDB Table.
- Read Capacity Units (RCUs): This represents the number of reads per second that a table can perform. One read capacity unit corresponds to one strongly consistent read per second for items up to 4 KB in size, or two eventually consistent reads per second for items up to 4 KB.
- Write Capacity Units (WCUs): This represents the number of writes per second that a table can perform. One write capacity unit corresponds to one write per second for items up to 1 KB in size.
- On-demand Mode: In DynamoDB's on-demand mode, the user does not have to specify how much read and write capacity a table needs beforehand. Instead, DynamoDB automatically scales to handle workloads and charges the user for the actual read and write requests made.

2.3.1 Provisioned Mode:

The primary focus of this research is on the Provisioned Capacity mode of DynamoDB. The article by (Andreoli et al.; 2023) mentions that DynamoDB allows its customers to specify the throughput requirements for a table. Depending on the provisioned values the AWS service allocates sufficient resources to the table. (Tantiphuwanart et al.; 2023) also mentions that provisioned throughput settings improve response times. (Chawathe; 2019) discusses the pricing for the two available modes of DynamoDB and Provisioned Mode. It is generally considered that the On-demand provisioned mode is several times costlier than the provisioned-capacity mode, and also that if the incoming workloads are known and if the tables are provisioned accordingly the response times are expected to be faster than the On-demand mode.

2.3.2 Impact of Read and Write Capacity Units:

The following section explains how RCUs and WCUs affect the performance of an application:

In the research by (Dineva and Atanasova; 2021), the application system being used is based on IOT to enhance agricultural processes. According to the application requirement details mentioned, the optimal throughput for the system is 50 requests per second, and each operation carries a data payload of 24 KB per item. For this, they have configured 2400 Write Capacity Units and 300 Read Capacity Units. After utilizing the DynamoDB system as part of their application, the system was found to handle the requirement efficiently. Thus configuring the DynamoDB table with optimum throughput settings will result in efficient performance in applications. However, the research does not further investigate improving the DynamoDB latencies.

(Wang et al.; 2019), investigates the cost-performance trade-offs for an IoT-based application. As part of the research, under the worst practices scenario, DynamoDB which was being used as part of the application system was poorly configured or under-provisioned. From the results, it was observed that many of the requests were being throttled or rejected and some cloud architectures under this scenario were prone to delays, which confirms that the throughout settings of DynamoDB impact the performance of applications.

2.4 Evolution of DynamoDB

Before releasing DynamoDB to the cloud in 2012, it was used internally for Amazon's e-commerce platform. (DeCandia et al.; 2007), is a very old research article published by amazon.com. Even though this article is very old, it is being discussed to provide an idea about how DynamoDB has evolved over the past decade. At the time of its initial release, AWS Dynamo was built for latency-sensitive applications that provided at least 99.9 percent of read-and-write operations to be performed within a few hundred milliseconds. Despite the commitment made by Amazon to deliver a 99 percent reliability rate with provisioned throughput settings for DynamoDB tables, user feedback has played a crucial role in addressing operational challenges. Also as mentioned in (Idziorek et al.; 2023), in the year 2018, AWS released On-demand provision mode. It is also mentioned in (Andreoli and Cucinotta; 2021), that DynamoDB is considered to be a solution with the capability of providing guaranteed levels of read and write operations per second, and keeping the latency lower than 10ms. Based on this observation of constant evolution and room for development, this research aims to find if cloud database infrastructures be further improved through novel configurations for optimal performance in a serverless environment.

3 Methodology

The main objective of this research is to study if some performance or efficiency aspects of Serverless platforms that use DynamoDB as a Data Store be further improved. The performance of a Serverless platform depends on a wide variety of factors, one of them being the efficiency of its data store. As mentioned in the above sections. A table in AWS DynamoDB can be set up using On-demand or Provisioned capacity mode, the first is a fully managed AWS service, while the other is manually providing the necessary throughput settings.

3.1 How are throughput settings calculated:

While creating a table on DynamoDb, a throughput settings calculator is provided. The theory behind this calculation as mentioned in the study by (Dineva and Atanasova; 2021) is as follows:

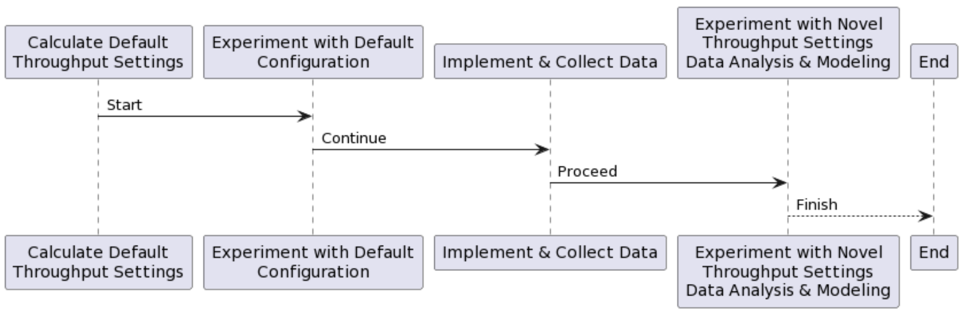

Calculate Default Throughput Settings
Experiment with Default Configuration
Implement & Collect Data
Experiment with Novel Throughput Settings Data Analysis & Modeling
End
Start
Continue
Proceed
Finish

Figure 1: Overview of the Methodology

Read Consistency	Capacity Units(RCU)	Read Requests per second (r)	Item Size(i)
Eventually consistent	1	2	4KB
Strongly consistent	1	1	4KB

Table 1: Read Capacity Units.

3.1.1 For Read Capacity units:

- There are two consistency models supported by DynamoDB, eventually consistent (default) and strongly consistent.
- Eventually consistent reads give higher throughput at the cost of not always returning the latest data.
- In contrast, strongly consistent reads always return the latest data at the cost of reduced throughput.

Both of these consistency models work with items of up to 4KB in size, the difference being eventually consistent reads provide 2 x 4KB reads per 1 RCU, whereas strongly consistent reads only provide 1 x 4KB reads per RCU. Table 1 provides a clear understanding.

3.1.2 For Write Capacity Units:

As the name suggests this controls the write throughput to your DynamoDB table. Calculating the required write capacity units is simpler because a consistency model does not apply here. One write capacity unit will give you one write per second of an item up to 1KB in size. Table 2 provides a clear understanding.

With the references from AWS Documentation and (Dineva and Atanasova; 2021), we arrive at the following formulae:

Capacity Units (WCU)	Requests per second (r)	Item Size(i)
1	1	1 KB

Table 2: Write Capacity Units.

$$RCU = [x * (y/i)]/r \quad (1)$$

Where x is the expected number of reads per second and y is the size of the item in KB.

$$WCU = x * y \quad (2)$$

Where x is the expected number of writes per second and y is the size of the item in KB.

3.2 Scenario With Theoretical Calculation

Let's say RCU should be calculated for an application that uses a DynamoDB table as its Datastore. The application requires an average read of 2000 per minute with an item size of 24KB, the reads can be eventually consistent to maximize throughput. Applying the formula from Equation 1, we get

$$RCU = [(2000/60) * (24/4)]/2 \quad (3)$$

The output for the expression will be 99 RCUs, for the same requirement the data obtained from the AWS DynamoDB capacity calculator is also the same.

In the same way, calculating the write capacity units for an application, that has a payload size of 24 KB, for 50 standard write requests per second. After applying the formula from Equation 2, we get

$$RCU = [24 * 50] \quad (4)$$

The output result of the expression will be 1200 WCUs, which is the same recommendation setting suggested by the AWS DynamoDB capacity calculator.

3.3 Data Collection Procedure

As discussed in the related work section about the study conducted from (Gunawan et al.; 2019), recording the time taken to fulfill an operation requirement or latency times was done as part of this current research.

Now to carry out the experiments, for DynamoDB to handle a certain number of read or write requests per minute, it has been configured by the default values obtained from AWS DynamoDB calculator. To test this as part of this research, the following environment was set up:

- A DynamoDB table was created in Provisioned Capacity mode, and the auto-scaling setting was turned off. While creating the table AWS provides a throughput capacity calculator. For an item size of 24KB, if entered in the throughput capacity calculator, DynamoDB will suggest read and write capacity units, these are recommended capacity units, and in this analysis report, these settings are being addressed as Default settings.

- A Serverless Lambda function was created, whose job is to read or write data from or into DynamoDB. DynamoDB client can be imported, using the "boto3" python module, into the Lambda Function to perform various operations.

- An AWS API Gateway was configured which acts as the front runner to access the Lambda function. The API gateway was configured to accept and handle all POST, and GET requests.

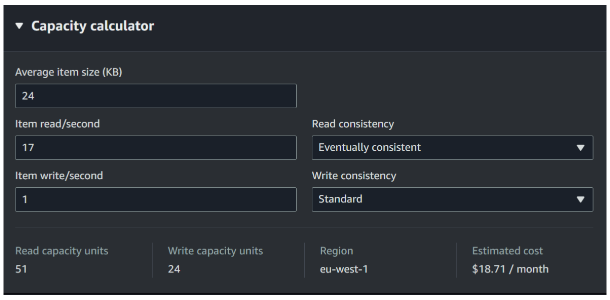
- The Web application developed will send requests to the API gateway which will in turn trigger the lambda function. The application can be used to send read-related requests and write-related requests. The UI of the application clearly distinguishes between the types of operations that can be performed for different sizes of data.

- Now that the experiment setup is completed. By clicking a button on the application, a request will be sent to the API Gateway, the API gateway will trigger the lambda function, and the lambda function will perform the operations. During this time, on the lambda function, the times before beginning and after executing the query are saved. The difference between these two values is logged. The logged values can be found in the AWS CloudWatch.

- After collecting the data for the "Default settings" to test the query execution times under novel configurations, the same DynamoDB table can be edited. To test the scenarios, the read and write capacity units were increased by 10, 20, and 30 percent above the default or recommended values by the AWS throughput capacity calculator.

- The web application scripts were designed to send continuous requests to the API gateway every second for over 250 seconds under each scenario. This procedure was repeated for 24KB, 49KB, and 96KB of item size, with the DynamoDB table configured with default and novel configurations and for both read and write related operations separately.

- The AWS DynamoDB table throughput capacity calculator as shown in 2 provides the price or cost that will be incurred for setting up the table, depending on the number of RCUs and WCUs. The cost will also be updated when the table is edited for upgrading the table from default configurations to novel configurations. To calculate the cost, the data is collected manually either while setting up the table or editing the table.


▼ Capacity calculator

Average item size (KB): 24

Item read/second: 17

Read consistency: Eventually consistent

Item write/second: 1

Write consistency: Standard

Read capacity units	Write capacity units	Region	Estimated cost
51	24	eu-west-1	$18.71 / month

Figure 2: AWS DynamoDB Capacity Calculator

3.4 Further Improvement

Now after testing the suggested RCU and WCU by DynamoDB, it raises the question that also happens to be the main focus of this research, i.e., can these response times be further improved or brought down? There are many optimization techniques like query optimization, assigning the partition key and sort key most efficiently, etc. However, this research focuses on whether keeping parameters like query, the payload of the request, item read size constant, and varying capacity units, can latencies be improved.

This research examines whether keeping certain factors like the type of query used, payload size of the request, and read or write item size constant, but changing the provisioned capacity units can reduce latency. Specifically examining, if adjusting the capacity of the system can reduce the response times.

3.4.1 Test Scenarios

To begin the further improvement testing, the capacity units were increased by 10 percent, keeping the Item size to be read or written and requests per second constant.

- **Read Capacity Units results**: The response times logged did not show any significant improvement. This way data was collected by increasing the capacity units by 10, and 20 percent. The results still were found not to be very significant. However, when the capacity units were increased by 30 percent, a significant decrease in response times was observed. To verify this further, the read capacity units were increased by 30 percent of the recommended capacity units. This new adjustment was implemented across varying request volumes, specifically 1000, 2000, 3000, 4000, and 5000 requests per minute for each set of requests over 250 seconds. The difference in response was found to be significant. The highest difference in default vs Novel configuration response time was observed while testing 5000 read requests per second for 249 default capacity units vs 324 capacity units, the difference in response times logged was 444.7 milliseconds. This number may not be significant in less time-sensitive tasks, but for real-time applications, this number is very significant and has a huge impact. The following graph 3 shows how the latencies for reading 24 KB size of items for 1000, 2000, 3000, 4000, and 5000 requests per second

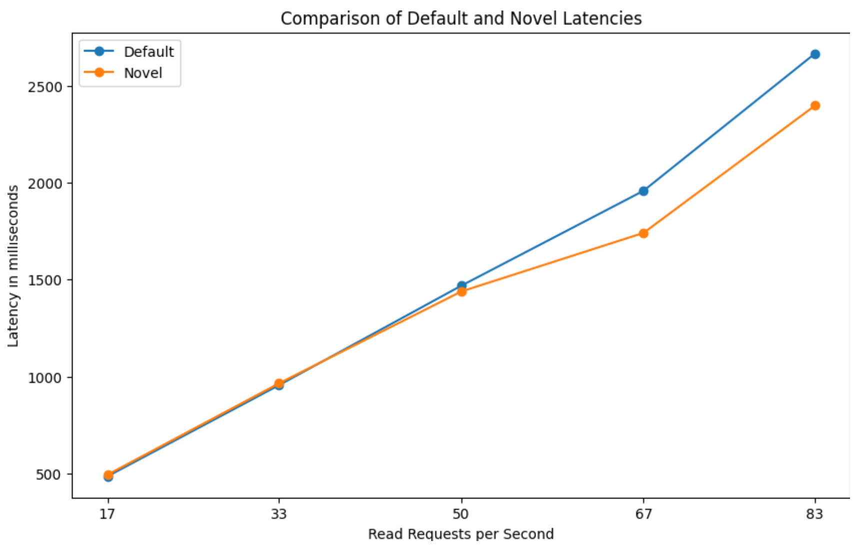


Figure 3: Default Read configurations vs Novel Read Configurations

- **Write Capacity Units results:** After recording the response times for writing data of both 24 KB and 49 KB to the DynamoDB table, it was observed that an increase of 10 percent from the default recommended settings did not result in any significant improvements, except for one instance. To investigate further the values were incremented by 10, 20, and 30 percent progressively. Unlike the RCU results, increasing the WCUs progressively did not provide evidence suggesting a drop in the response times. In fact, in some cases, the latency times of increased configuration settings show delayed responses compared to the default configuration settings. The following graph 4 shows the latencies for writing 24 KB size of items for 17, 33, 50, and 67 requests per second or 1000,2000,3000 and 4000 requests per minute. From the graph 4, it can understood that the latencies when the tables were configured with default values, the latencies were minimal when compared to the latencies of tables that were configured with novel configurations.

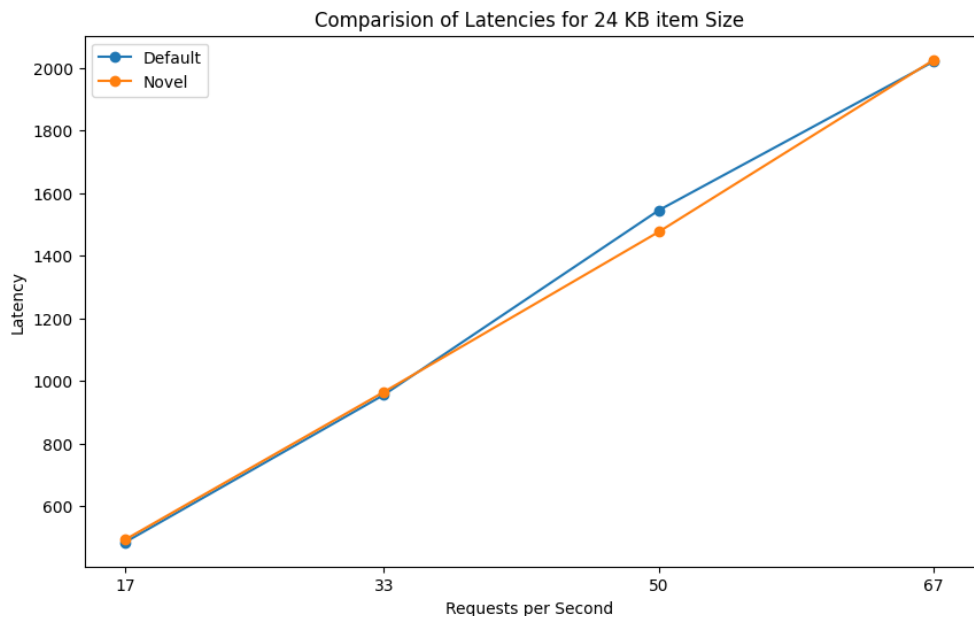


Figure 4: Default Write configurations vs Novel Write Configurations

3.5 Data Analysis

AWS Cloudwatch was used to log the response times. In every scenario i.e., 1000, 2000, 3000, 4000, and 5000 requests per minute, default configuration response times vs novel configuration response times have been collected. The difference in times was analyzed for each scenario. When the read capacity units were increased by 30 percent, the difference in log times was found to be significantly lower by 60-70 percent on average. But in the case of write capacity units, the default or recommended settings by the AWS DynamoDB capacity calculator on analysis were found to be faster than the novel configurations. Despite increasing the values up to 40 percent, no significant improvements were observed. Since write capacity units did not indicate any improvements with the novel configurations, this research's attention was directed exclusively toward the data associated with read capacity units. This data was further analyzed and used for predictions in the machine learning models.

3.5.1 Data Collected

As mentioned in the previous sections, data was collected during the experiments and prepared for model training as shown in the table 3. The nature of the data collected as shown in the table is named as the following columns, ”itemSizeinKB”, ”requestspersecond”, ”defaultConfiguration”, ”expectedDefaultLatency”, ”NovelConfiguration”, ”expectedNovelLatency”. Due to size limitation only a small amount of information is being shown, but the information was gathered for 24 KB and 1000 req/min (17 req/s), 24 KB and 2000 req/min (33 req/s), 24 KB and 3000 req/min (50 req/s), 24 KB and 1000 req/min (67 req/s), and 24 KB and 5000 req/min (83 req/s). This was repeated with 49 KB of data and 98 KB of data.

col 1	col 2	col 3	col 4	col 5	col 6
24	17	51	0.743492842	66	0.754738092
24	17	51	0.735300779	66	0.730283499
24	17	51	0.771490574	66	0.7067132
24	33	99	0.918589354	129	0.643455267
24	33	99	0.941638708	129	0.954394102
24	33	99	0.851253271	129	1.000084877
24	50	150	1.438911915	195	1.421566486
24	50	150	1.462717533	195	1.39555335
24	50	150	1.428402662	195	1.449452162

Table 3: Data Collected for 24 Kb Item size Read. Col 1: itemSizeinKB, col 2: requestspersecond, col 3: defaultConfiguration, col 4: expectedDefaultLatency, col 5: NovelConfiguration, col 6: expectedNovelLatency

3.5.2 Machine Learning Model for Predictions

In this scenario with the data that has been recorded, the idea is to predict multiple outputs i.e., ”defaultConfiguration”, ”expectedDefaultLatency”, ”NovelConfiguration”, and ”expectedNovelLatency”, with two available features ”itemSizeinKB” and ”requestspersecond”. In such scenarios, several models can be used to predict data that has many features. In this research, Multi-output Regression Models will be used. Multi-output Regression Models are well-suited for tasks where the output variables are correlated. In situations where predicting multiple outputs is interconnected, a multi-output regression model can simplify the modeling process. MultiOutputRegressor is a sci-kit-learn wrapper that extends any single-output regression model to become a multi-output model or in other words it allows the usage of traditional single-output regression models, such as RandomForestRegressor or GradientBoostingRegressor, for tasks where predicting multiple outputs simultaneously are needed. The sequential diagram 5 describes the steps implemented for the model to process the input data and generate an output.

4 Design Specification

In this research approach, two paradigms are being utilized. The initial paradigm involves collecting or gathering data, and the next involves utilizing the collected data for predictions involving user inputs.

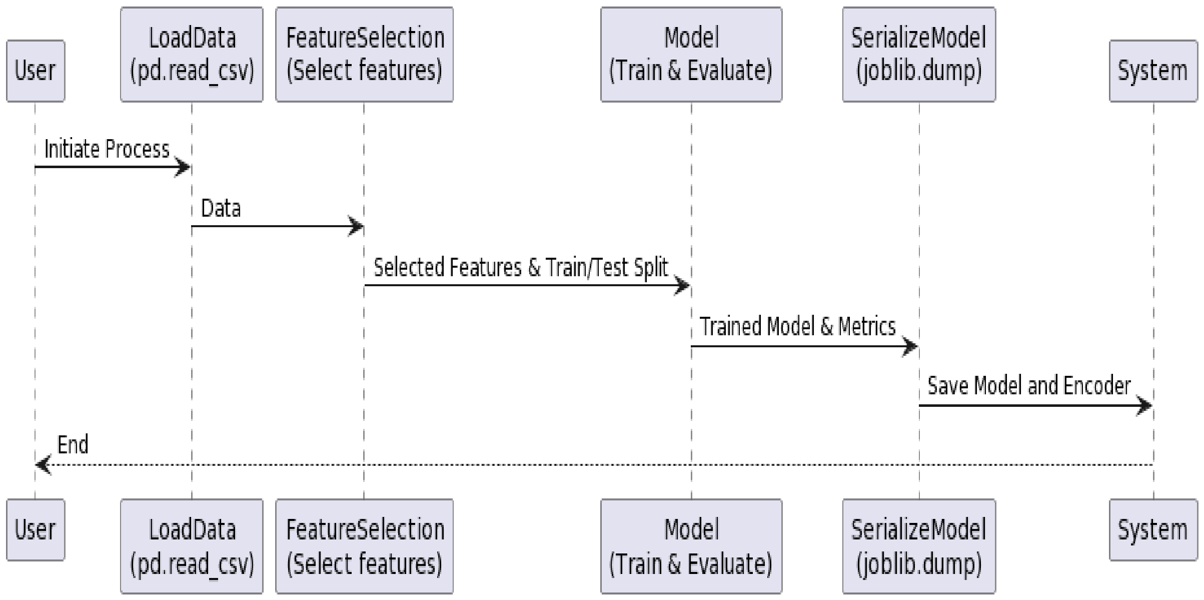


Figure 5: Sequential steps of the model training and integration

4.1 Collecting Data

The process of collecting data involves conducting various tests to gather latency-related information on DynamoDB. The entire process of data collection is made simple through a web application that was developed, whose primary job is to send requests to DynamoDB. The requests were sent through AWS API Gateway and Lambda functions to DynamoDB. The query execution time taken by DynamoDB was logged on AWS CloudWatch. The image 6 gives an overview of the architecture used in this phase.

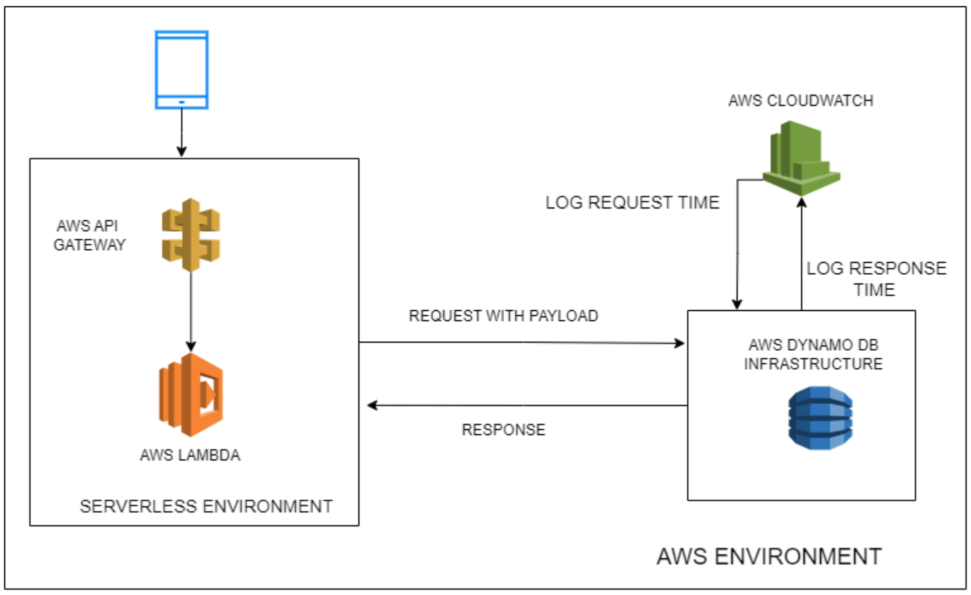


Figure 6: Collecting Data paradigm architecture overview

4.1.1 Web Application:

The web application serves as the primary interface for initiating tests and collecting data. It is designed to interact with DynamoDB through API Gateway, using serverless AWS Lambda functions to handle the requests. The application architecture includes the following components:

- API Gateway: It acts as a gateway for communication between the web application and Lambda functions.
- Lambda Functions: Handle incoming requests, perform queries on DynamoDB, and log execution time taken by DynamoDB on CloudWatch
- Web Frontend: User interface for initiating tests and either posting payload or retrieving information.

4.1.2 Latency Data Logging:

The main objective of this phase is to capture latency-related information. The process involves

- Query Execution: The web application initiates queries to DynamoDB to simulate real-world scenarios.
- CloudWatch Logging: The execution times of these queries are logged in CloudWatch for subsequent analysis.

4.1.3 Requirements:

The data collection process should meet the following requirements:

- Efficiency and Consistency: Ensure efficient execution of queries on DynamoDB. Closely monitor the size of the data that is being retrieved or posted. Closely monitor the number of requests being sent at a period.
- Logging: Accurately logging execution times for latency analysis.

4.2 Predicting Capacity Units:

This phase also has another web application that serves as an important component in the second paradigm of the project. It utilizes machine learning models trained on the collected data to predict appropriate throughput settings for DynamoDB based on user input.

4.2.1 Functionality:

The web application in this phase has been developed using the Streamlit framework. The application is designed to predict and recommend suitable throughput settings for DynamoDB based on user-defined input parameters. The functionality includes:

- User Input: Users select relevant parameters, such as expected workload requests and the size of the items.

- Machine Learning Prediction: Using pre-trained machine learning models to predict optimal DynamoDB throughput settings.
- Output: Displaying the predicted settings and corresponding latencies, to the user for future consideration and adjustments.

4.2.2 Requirements:

This phase must meet the following requirements

- User-Friendly Interface: Develop a simple interface for easy input and output-related interactions.
- Real-time Prediction: Ensure real-time prediction capabilities by integrating with pre-trained machine learning models.

5 Implementation

In this study, the main aim was to check if the latencies of a serverless application be further improved through novel DynamoDB configurations. The process involved Data Collection and phase and Predicting Configurations by training the models on data collected from the first phase.

5.1 Data Collection:

This phase included the development of a web application hosted on AWS infrastructure, including EC2, Lambda, API Gateway, and CloudWatch. The web application was designed using HTML, CSS, and JavaScript, while Python was utilized for AWS Lambda services. AWS API Gateway was used as a front-runner for accepting and routing incoming requests from the web application to Lambda functions. The web application has been hosted on AWS EC2, and AWS CloudWatch has been used to log essential metrics. Depending on the type of request either read or write, DynamoDB tables have been created and configured. For each test scenario, DynamoDB tables were provisioned accordingly.

The primary job of the web application is to send work requests to AWS Serverless environments, where AWS API Gateway and Lambda functions manage the incoming requests. Lambda, connected to DynamoDB, forwards requests based on their type. One key factor affecting Lambda's performance, measured by the time taken to fulfill requests, is DynamoDB's ability to handle these requests. To measure DynamoDB's response time, logs are recorded on AWS CloudWatch. An initial log is made when the query starts, and another is recorded when the response is sent back. To enhance the overall request-response time and improve the performance of serverless Lambda, DynamoDB was configured with different settings, and the data collected during this process forms the output for this phase.

Languages Used:

- HTML, CSS, JavaScript for the web application
- Python for utilizing AWS Lambda services.

Tools Used:

- AWS API Gateway.
- AWS EC2 to host the web application.
- AWS Lambda.
- AWS DynamoDB.
- AWS CloudWatch.

Outputs Produced: Raw latency data collected from AWS CloudWatch logs were analyzed to understand query execution times and patterns for training machine learning models. The research initially aimed to enhance latencies in both read and write-intensive workloads. However, through insights and data analysis, it has been demonstrated that write-based workloads did not exhibit any improvement in latency. This finding influenced the outcome of the research, shifting the focus from predicting throughput settings for both read and write to exclusively addressing read-intensive workloads.

5.2 Prediction Phase:

Libraries, Tools, and Frameworks Used:

- Python Streamlit
- sci-kit-learn
- Pandas
- AWS EC2 to host the streamlit application
- Google Colab to train and test the machine learning model

Outputs Produced: A deployed Streamlit application accessible to users. Real-time predictions are presented to users based on input parameters.

5.3 Data set, Model and Implementation:

The data set used in this research originates from latency tests conducted on DynamoDB through a web application. It contains information about query execution times and relevant input parameters, providing a deeper understanding of the performance of DynamoDB interactions. This information is collected from CloudWatch logs, this data serves as the foundation for training machine learning models and for providing insights into DynamoDB performance. The training process included

- Data Splitting
- Algorithm Selection (Random Forest Regressor)
- Model evaluation using metrics such as Mean Squared Error and R-squared

The trained machine learning models have been integrated into the Streamlit application, providing users with a user-friendly interface for throughput predictions. Users input parameters through the Streamlit interface, and the application responds with predictions based on the information generated by trained models.

6 Evaluation

The evaluation of the results obtained from the research are as follows:

6.1 Experiment 1: Write-Intensive workloads

The research aimed to study and enhance performance in terms of latency for both Read and write-intensive workloads. However write-intensive workloads did not show any latency improvement with additional provisioning.

6.1.1 Test Case:

A write-intensive workload test case was designed with a 24 KB item size. The test scenario involves writing data for 1000 requests per minute or 17 requests per second. To handle this request AWS DynamoDB was configured with the values suggested by the AWS DynamoDB capacity calculator, which in this case was 408 WCU. The requests were sent and the relevant information was logged on the AWS CloudWatch. To investigate if increasing the provisioned throughput resulted in improved latency, the default setting was increased by 10, 20, 30, and 40 percent, but no improvement was noticed. After the data analysis, it was found that the requests were handled smoothly and none of the requests were throttled but the data showed no signs of improvement in terms of latencies. After analyzing the collected data, represented in 4 and 5 the difference between the latencies of the two sets of configurations was not significant and sometimes negative.

Requests/min	Default Config Avg latency	Novel Config Avg latency	Difference
1000	484.87 ms	493.16 ms	-8.29 ms
2000	955.47 ms	964.78 ms	-9.31 ms
3000	1544.84 ms	1476.01 ms	68.83 ms
4000	2020.09 ms	2024.88 ms	-4.79 ms

Table 4: Write 24 KB Workload.

Requests/min	Default Config Avg latency	Novel Config Avg latency	Difference
1000	543.40 ms	535.31 ms	8.09 ms
2000	1103.31 ms	1063.75 ms	39.56 ms
3000	1655 ms	1641.34 ms	13.66 ms
4000	2061 ms	2100.84 ms	-39.80 ms

Table 5: Write 49 KB Workload.

The graphs 7 and 8 represent that there is no significant improvement in latencies by using the novel configurations.

The experiment was continued with 24 KB and 49 KB of payload size with, 2000, 3000, and 4000 requests per minute, but no improvement was evident. Hence it was concluded that latencies of write-intensive workloads cannot be made better by provisioning AWS DynamoDB with higher throughput settings.

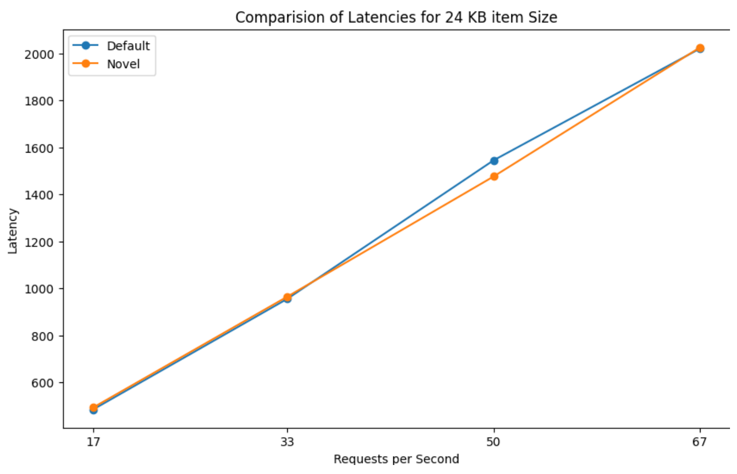


Figure 7: Write intensive workload

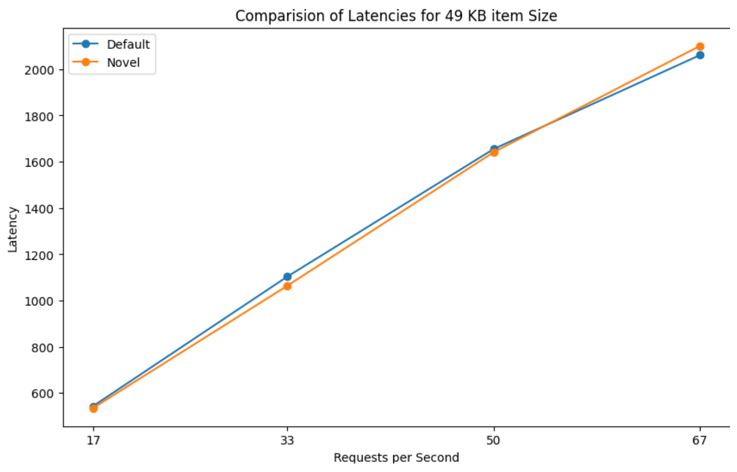


Figure 8: Write intensive workload

6.2 Experiment 2: Read-Intensive workloads

A read-intensive workload test case was designed, focusing on a specific item size of 24 KB. This test scenario involved the task of reading the data at 1000 requests per minute, which is nothing but a throughput requirement of 17 requests per second. To handle this demanding workload, the AWS DynamoDB table was configured with the precise values recommended by the AWS DynamoDB capacity calculator. In this specific case, the calculated provisioned capacity was 51 Read Capacity Units (RCU). Throughout the test phase, a detailed record of the test results was maintained by logging the information on the AWS CloudWatch. This helps in the post-analysis of the test results. The primary focus of this analysis was to find out whether the provisioned throughput would result in latency improvement. For this purpose, the initial default setting was progressively increased by 10, 20, and 30 percent. The reasoning behind this approach was to find patterns in latency improvements with the increase in provisioned throughput capacity settings. The data analysis showed that all incoming requests were handled efficiently, and none of the requests were throttled or denied the service. The procedural increase of the throughput capacity settings by 10 and 20 percent did exhibit noticeable improvements in latencies, which indicates a relation between throughput enhancement and response time. However, the most significant impact on latency was observed when the throughput settings were increased by 30 percent. The data obtained from this experiment not only validates the efficiency of the default configuration suggested by the AWS capacity calculator but also shows the potential for optimizing system performance in terms of latency by strategically adjusting the throughput settings. The graph 9 depicts the decrease in latency under 17, 33, 50, 63, and 87 requests per second with default vs novel configurations.

ItemSize	Req/min	Default Config Avg latency	Novel Config Avg latency	Difference
24	1000	496.37 ms	483.59 ms	12.78 ms
24	2000	938.45 ms	882.81 ms	55.64 ms
24	3000	1468.70 ms	1438.90 ms	29.80 ms
24	4000	1958.55 ms	1741.22 ms	217.33 ms
24	4000	2667.19 ms	2398.64 ms	268.55 ms
49	1000	434.53 ms	433.89 ms	0.60 ms
49	2000	439.17 ms	428.47 ms	10.71 ms
49	3000	1463.91 ms	1318.56 ms	145.35 ms
49	4000	2236.82 ms	2101.71 ms	135.11 ms
49	4000	2869.23 ms	2590.59 ms	278.64 ms
96	1000	764.29 ms	573.93 ms	190.36 ms
96	2000	1275.18 ms	1140.76 ms	134.42 ms
96	3000	1854.42 ms	1727.82 ms	126.60 ms
96	4000	2453.27 ms	2334.81 ms	118.46 ms
96	4000	3007.05 ms	2933.32 ms	73.73 ms

Table 6: Read 24, 49, and 96 KB Workloads.

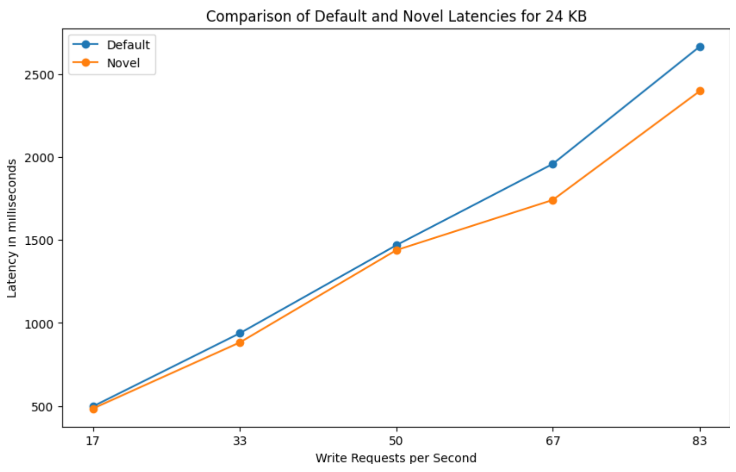


Figure 9: Read intensive workload 24 KB

Similarly, the experiment was also conducted on 49 KB and 96 KB. The results tabulated tables 6 prove that latencies were improved when the DynamoDB table was provisioned with an additional 30 percent of RCU units. The graphs 10 and 11 display the same positive results. The graph 12 shows the highest latency difference observed when comparing default configurations with novel configurations from all sets of workloads.

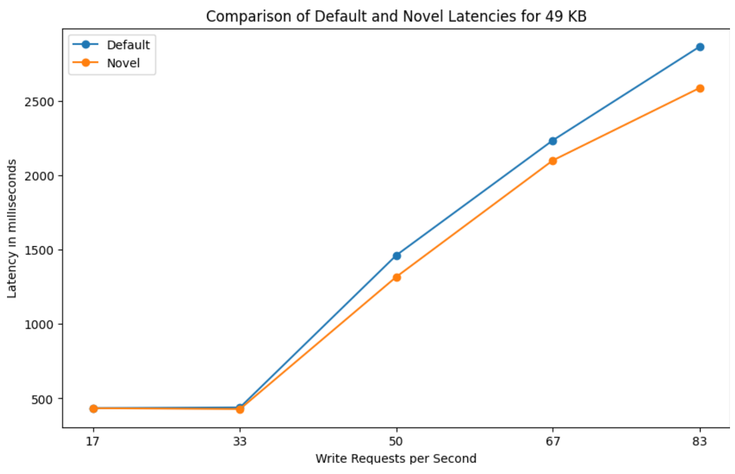


Figure 10: Read intensive workload 49 KB

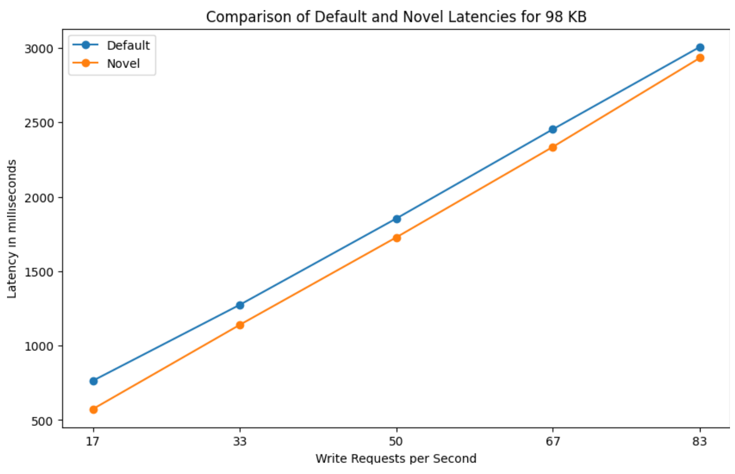


Figure 11: Read intensive workload 98 KB

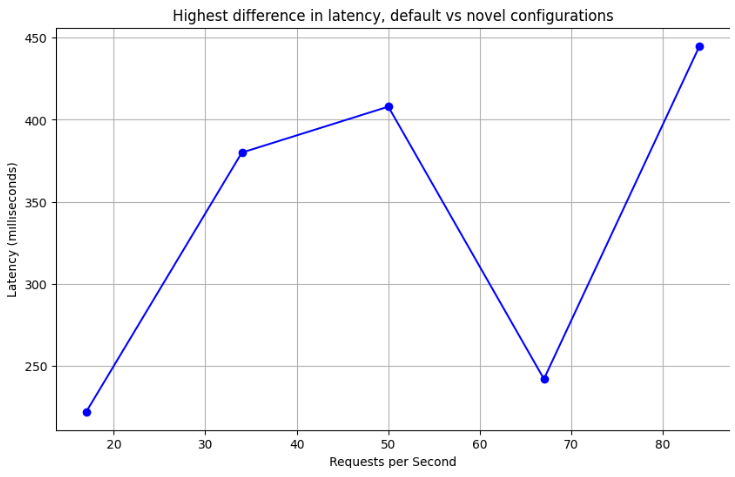


Figure 12: highest latency difference across all values

6.3 Experiment 3: Multi-output Regression Models

After collecting the information and data analysis, the Multi-output Regression Model was chosen as the appropriate one for this dataset. The focus shifted towards exploring the efficiency of multi-output regression models in predicting DynamoDB throughput settings. The model was trained on the data set. This allows the models to simultaneously provide insights into multiple facets of DynamoDB configuration. These models are capable of predicting multiple dependent variables simultaneously from the same set of independent variables. Since this is a regression model, common evaluation metrics include Mean Squared Error (MSE), Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), and R-squared (R^2). After training and testing the models, the results are shown in the table 7. The results can be interpreted as, MSE of 0 means that the model is perfect and makes no errors. The other two have small MSE values (0.00476184 and 0.00602573), suggesting that the model's predictions are quite close to the actual values for these outputs as well. RMSE values close to 0 indicate excellent model performance. MAE values close to 0, in this case (0.04088449 and 0.04984964 for the non-perfect outputs), suggest that the model is highly accurate on average. R^2 values are very high (0.99146074 and 0.98581457 for the outputs with nonzero errors), indicating that most of the variance in those outputs is explained by the model.

Mean Squared Error for each output	0	0.00564636	0	0.0045467
Root Mean Squared Error for each output	0	0.07514225	0	0.06742924
Mean Absolute Error for each output	0	0.05191444	0	0.04371381
R-squared for each output	1	0.9907814	1	0.99323315

Table 7: Model Evaluation Metrics.

6.4 Discussion:

As mentioned in the earlier sections, this research aimed to improve latencies in both write-intensive and read-intensive workloads. After the tests, as observed in the test results, the write-intensive workloads did not show any improvement in terms of latency with additional provisioning. At the same time, the improvement in latencies observed as a result of provisioning additional read capacity units fulfills the aim of this research. Further additional provisioning to obtain higher lower latencies will have a trade-off with high costs but in this case, with only a 30 percent surplus the rise in the cost is also found to be around a 25-30 percent increase. This research did not go beyond provisioning an additional 30 percent to receive low latency. The decision to limit the experiment to 30 percent was influenced by limitations involving the resources available and was stopped at the first sight of significant improvement which is around 30 percent higher than the results obtained with default settings. Allocating excessive capacity units may result in over-provisioning and underutilized resources, leading to financial losses. The study focused on finding a balance between having low latencies and not having to spend a lot of money on underutilized resources. The model trained as part of this research has been trained on a limited number of data records. A model trained on a large data set and a variety of factors would be desirable.

7 Conclusion and Future Work

The main aim of this research was to investigate, how cloud database infrastructures can be further improved through novel configurations for optimal performance in a serverless environment. The research began with the aim to investigate and enhance latencies in both read and write-intensive workloads. However, after performing the experiments, and through the data collected and its analysis, it has been demonstrated that write-based workloads did not exhibit any improvement in latency with additional throughout provisioning. This finding had a huge impact on the research, moving the focus from predicting throughput settings for both read and write to exclusively addressing read-intensive workloads. Despite some major changes in the expected outcomes of the research, the research turned out to be insightful and successful in enhancing latencies for read-intensive workloads. Users can quickly look up this solution to understand or predict the throughput settings that they need to provision it with, or at least it equips users with information before making their judgments. Due to the limited resources, this research could only experiment with a few payload sizes, and experiment with the scenarios only in one AWS available region.

This research can be further improved by conducting experiments on various AWS regions, with different and varying workloads, by collecting huge amounts of data to enhance and diversify the accuracy of the model.

References

Andreoli, R. and Cucinotta, T. (2021). Differentiated performance in nosql database access for hybrid cloud-hpc workloads, pp. 439–449. `https://link.springer.com/chapter/10.1007/978-3-030-90539-2_30`.

Andreoli, R., Cucinotta, T. and De Oliveira, D. B. (2023). Priority-driven differentiated performance for nosql database-as-a-service, *IEEE Transactions on Cloud Computing* **11**(4): 3469–3482.
URL: *https:/doi.org/10.1109/TCC.2023.3292031*

Chawathe, S. S. (2019). Data modeling for a nosql database service, pp. 0234–0240.
URL: *https:/doi.org/10.1109/UEMCON47517.2019.8992924*

DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Vosshall, P. and Vogels, W. (2007). Dynamo: Amazon's highly available key-value store, **41**(6): 205–220.
URL: *https://doi.org/10.1145/1323293.1294281*

Dineva, K. and Atanasova, T. (2021). Design of scalable iot architecture based on aws for smart livestock, *Animals* **11**(9).
URL: *https://www.mdpi.com/2076-2615/11/9/2697*

Eismann, S., Scheuner, J., van Eyk, E., Schwinger, M., Grohmann, J., Herbst, N., Abad, C. L. and Iosup, A. (2021). Serverless applications: Why, when, and how?, *IEEE Software* **38**(1): 32–39.
URL: *https://doi.org/10.1109/MS.2020.3023302*

Elhemali, M., Gallagher, N., Tang, B., Gordon, N., Huang, H., Chen, H., Idziorek, J., Wang, M., Krog, R., Zhu, Z. et al. (2022). Amazon {DynamoDB}: A scalable, predictably performant, and fully managed {NoSQL} database service, pp. USENIX ATC '22 Full Proceedings, 1037–1048.
URL: *https://www.usenix.org/conference/atc22/technical-sessions*

Filip, P. and Čegan, L. (2020). Comparison of mysql and mongodb with focus on performance, pp. 184–187.
URL: *https://doi.org/10.1109/ICIMCIS51567.2020.9354307*

Gomes, C., Borba, E., Tavares, E. and Junior, M. N. d. O. (2019). Performability model for assessing nosql dbms consistency, pp. 1–6.
URL: *https://doi.org/10.1109/SYSCON.2019.8836757*

Gunawan, R., Rahmatulloh, A. and Darmawan, I. (2019). Performance evaluation of query response time in the document stored nosql database, pp. 1–6.
URL: *https:/doi.org/10.1109/QIR.2019.8898035*

Idziorek, J., Keyes, A., Lazier, C., Perianayagam, S., Ramanathan, P., III, J. C. S., Terry, D. and Vig, A. (2023). Distributed transactions at scale in amazon DynamoDB, pp. 705–717.
URL: *https://www.usenix.org/conference/atc23/presentation/idziorek*

Kumar, M. S. et al. (2018). Comparison of nosql database and traditional database-an emphatic analysis, *JOIV: International Journal on Informatics Visualization* **2**(2): 51–55.
URL: *https://joiv.org/index.php/joiv/article/view/58*

Mvondo, D., Bacou, M., Nguetchouang, K., Ngale, L., Pouget, S., Kouam, J., Lachaize, R., Hwang, J., Wood, T., Hagimont, D., De Palma, N., Batchakui, B. and Tchana, A. (2021). Ofc: An opportunistic caching system for faas platforms, p. 228–244.
URL: *https://doi.org/10.1145/3447786.3456239*

Palepu, S. C., Chahal, D., Ramesh, M. and Singhal, R. (2022). Benchmarking the data layer across serverless platforms.
URL: *https://doi.org/10.1145/3526060.3535460*

Pelle, I., Czentye, J., Dóka, J. and Sonkoly, B. (2019). Towards latency sensitive cloud native applications: A performance study on aws, pp. 272–280.
URL: *https:/doi.org/10.1109/CLOUD.2019.00054*

Preuveneers, D. and Joosen, W. (2020). Automated configuration of nosql performance and scalability tactics for data-intensive applications, *Informatics* **7**(3).
URL: *https://www.mdpi.com/2227-9709/7/3/29*

Seghier, N. B. and Kazar, O. (2021). Performance benchmarking and comparison of nosql databases: Redis vs mongodb vs cassandra using ycsb tool, pp. 1–6.
URL: *https:/doi.org/10.1109/ICRAMI52622.2021.9585956*

Tantiphuwanart, S., Tuaycharoen, N., Wanvarie, D., Pratanwanich, N. and Suchato, A. (2023). Performance improvement on a learning assessment web application using aws dynamodb as a cache database, pp. 303–308.
URL: *https:/doi.org/10.1109/JCSSE58229.2023.10201973*

Wang, S., Valluripally, S., Mitra, R., Nuguri, S. S., Salah, K. and Calyam, P. (2019). Cost-performance trade-offs in fog computing for iot data processing of social virtual reality, pp. 134–143.
URL: *https:/doi.org/10.1109/ICFC.2019.00025*