International Journal of
INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING
ISSN:2147-6799 www.ijisae.org Original Research Paper

DM-DATA Model with onsite Oracle system and AWS to Migrate Web Services through Oracle Database

Varun Varma Sangaraju

Submitted:13/05/2024 **Revised**: 28/06/2024 **Accepted**: 05/07/2024

Abstract: AWS, or Amazon Web Services, is a cloud computing platform that is adaptable, affordable, and simple to use. Relational database management systems, or RDBMS, are frequently used in the Amazon cloud. Derive how to set up Oracle Database on AW and Oracle Database may be operated on Relational Database Service (Amazon RDS). To show how you can operate Oracle Database on Amazon RDS, as well as to inform you of the benefits of each strategy and how to deploy and monitor your Oracle database, as well as how to handle scalability, performance, backup and recovery, high availability, and security in Amazon RDS. In this paper, proposed the DM-DATA Model to establish an Emergency Recovery solution with an onsite Oracle system and AWS and to migrate your existing Oracle database to AWS. We provide a strategy for designing an architecture that protects you against hardware failures, datacenter issues, and disasters by using replication technologies stock market data. In the performance analysis, there are several alternatives are choose to optimize the performance of the propose infrastructure with Oracle database based on certain metrics like, disk I/O management, sizing, database replicas, etc.

Keywords*: sizing, Oracle, architecture, AWS, DM-DATA*

I. Introduction

In the 1600s, the East India Company founded the world's first stock market [1]. A stock exchange was a place where current and potential investors would gather to swap shares. On a trading floor, open outcry was the primary form of communication. It involves screaming and hand gestures to convey information about the directives. For centuries, that model remained mostly unaltered. With the advent of internet access and more potent computers in the late 1980s and early 1990s, the drive towards automation supplanted the remaining vestiges of open outcry. With the advent of computerized trading, the trading landscape shifted dramatically in the early 2000s. About 80% of the cash stock markets were entirely electronic by the end of 2019 [2].

Modern computer technology has made it possible to process orders more quickly, reduce human error, and conduct more thorough market research. Modern trading relies on constant and incredibly rapid analysis of extremely large quantities of data [3], which is frequently in time series consisting of a date, a unique identifier (such as a stock symbol), and values observed for an entity on that day. One such variety is OHLC data, which captures the Open, High, Low, and Close prices of an instrument over a specified period.

To derive patterns and trading signals from market data, OHLC data are particularly crucial. The rationale for recording these prices rather than all intra-daily prices is that they have a higher informative value. OHLC data may be used to define and anticipate asset price volatility [4] and are frequently less expensive to get and work with than high-frequency tick data, which consists of bid and ask prices aggregated from many exchanges. In fact, [5] demonstrates that volatility models constructed using daily OHLC time series data may offer accuracy comparable to that of models constructed using high-frequency data. As a result, investors continue to buy and sell by precise forecasts of OHLC data [6].

To undertake such studies, a time series management system or a time series database is required. Because OHLC data are typically generated in the application layer by a programmer (or script) that processes measurements of stock price movements [7], it is critical to store this data in a time series database where it is easy and fast to store, query, and perform operations such as sum, mean, and median on multiple records of data. Furthermore, because financial time series databases may quickly become quite big — there are several

Independent researcher and senior engineer, Dallas, TX, USA.
varunvarma93@yahoo.com

thousand equities listed on the New York Stock Exchange alone — it is vital to have an efficient database system capable of doing the needed large-scale analytics processing. Because standard database systems, such as Relational Database Management Systems, are frequently suboptimal and generally unsuitable for time series data [8].

These time series databases could appear promising, but they lack the widespread support of SQL (Structured Query Language) or NoSQL (Not Only SQL) databases and are sometimes quite challenging to move to.

In order to overcome the problem of hardware failures, datacenter issues, and disasters in the database, data migration is performed to handle scalability, performance, backup and recovery, high availability, and security among the data. The work contribution is summarized as below,

- DM-DATA framework is establishing an Emergency Recovery solution with an onsite Oracle system and AWS and to migrate your existing Oracle database to AWS, which can handle scalability, performance, backup and recovery, high availability, and security in Amazon RDS.
- Transfer an Oracle database from a local installation to Amazon RDS for Oracle, which can migration data with fulfilled business goals, such as decreased downtime migration.
- The automation of file transfers from on-premises to Amazon RDS for Oracle databases using Amazon S3, Lambda, and AWS Secrets Manager
- To optimize the performance of the propose infrastructure with Oracle database based on certain metrics like, disk I/O management, sizing, database replicas, etc.

II. Literature Survey

This section first reviews the development of both relational and non-relational databases and highlights their respective strengths. Second, it analyses the outcomes of experiments conducted on various case studies to assess their individual performance. Finally, as the main area of interest of this study is the usage of databases to store and process large volumes of financial trading data, usually on a hybrid cloud architecture, this review explores then their storage costs and the characteristics of their cloud implementation.

A. Existing Data Modeling:

A new model of data called relational database where all data are represented in terms of tuples and attributes, formally described using tabulated [9]. The platforms used to manage these databases are known as Relational Database Management Systems (RDBMS). Most of them employ SQL (Structured Query Language) as their query language [10]. Relational databases rely on the ACID (Atomic, Consistent, Isolated, and Durable) properties to operate efficiently and correctly. This guarantees data validity despite errors, power failures and other mishaps.

Relational databases perform best with structured data, but they have a limited or restricted ability to represent complex semi-structured or unstructured data. A study has shown that it is difficult to store clinical visit data in an RDBMS due to their semi-structured information and dynamic changing properties [11]. Indeed, usage of relational databases for such data leads to creating fields that are mostly empty resulting in inefficient storage and poor performance. Moreover, another limitation of relational databases is their inability to store increasing volumes of real-time data [12]. As in the cases of national votes and fingerprints data, the amount collected increases drastically both in terms of volumes (Terabytes of data) and velocity (rate of data generated, in Gigabytes/day), which eventually requires a large number of tables to accommodate the growth in data. Actually, the usage of a relational database in such scenarios becomes inappropriate because of its inability to scale with the ever-growing real-time data [83]. Finally, relational databases cannot take advantage of modern advancements in distributed computing as they are not designed to function with data partitioning [13].

The non-relational databases were created as a means to offer high performance (both in terms of speed and size) and high availability at a price of losing the ACID trait of relational databases and instead offering the weaker BASE (Basic Availability, Soft state, Eventual consistency) feature [14]. These databases store semi-structured and structured data in a non-complex data model such as key-value pairs, which consists of two parts, a string which represents the key and the actual data which is referred to as value. These keys are then used as indices, making the query process faster than the RDBMS [15]. Non-relational databases started becoming popular with the internet boom in the mid-

1990s as relational databases could not handle the flow of information demanded by users [16]. Since then, numerous companies and organisations have developed their own non-relational databases. Many studies have shown that non-relational databases enable better performance in terms of speed and flexibility. Indeed, availability, real-time response, advanced data analysis, and the ability to manage bigdata remain weaknesses which are displayed by relational databases [17]. Moreover, these shortcomings are overcome by the latest NoSQL systems which have been designed to address the challenges associated with dealing with large amounts of data. As a consequence, they have become the option of choice for applications involving geographically distributed data, large amounts of data, or scalability requirements [18]. This is particularly the case for services relying on Internet of Things (IoT) technology. For example, in a recent case study where IoT enabled sensors provide measurements to monitor manufacturing defects in the automobile industry, usage of a NoSQL database allowed real-time data processing and, thus, the detection of faults at early stages of the manufacturing process [19].

Unlike relational databases that can only scale vertically by adding more resources to the current server, non-relational databases also support and embrace horizontal scaling. This is achieved by adding more machines to the network and then dividing the workload or in this case distributing the data among them [20].

Despite this, the latest Database Engine rankings [21] (based on top searches on various search engines, Stack Overflow, Google trends, job offers or number of mentions in social networks) reveals that relational databases remain prevalent: there are only three non-relational databases in the top ten and none of them are in the top four! This is probably because relational systems have been used extensively for many decades and are trusted for maintaining accurate transactional records, legacy data sources, and many other use cases within organizations of all sizes. In addition, non-relational databases lack a standard query language: there are more than 200 implementations, each providing its own language and interface that developers and users must learn. Finally, a major challenge of non-relational databases is their weak security mechanisms. Indeed, they were initially designed without security being considered as an essential feature. Thus, there have been growing concerns related to data privacy in NoSQL systems which results from compromises made for better performance and scalability [32]. Whereas relational databases have inbuilt authentication instead of relying of a middleware application for authentication or authorization of the data source, by design, non-relational databases offer limited security and place more emphasis on data handling. Indeed, the feature of distributed data, termed as 'sharding' [22], which is considered the key of their success, is associated with a concern on how the confidentiality and privacy of data is maintained across systems.

B. Experimental Comparison on Public Database:

Many experiments have been conducted to compare characteristics of non-relational and relational databases including their scalability, performance, flexibility, power of querying, and security. Experiments conducted a decade ago proved quite inconclusive as performance varied significantly according to the type of operation performed and the type of data used. Focusing on processing a modest amount of structured data, it was shown that MongoDB – a popular non-relational database – performed at least as well as MySQL with exceptions of aggregate functions (such as medians, modes and sums). A more recent study analysing performance of non-relational databases for spatial and aggregate functions suggests that the performance of MongoDB has since improved [3]. Focusing on applications handling large volumes of data (i.e., terabytes), it was concluded that non-relational databases were preferable because they offer flexible architectures which can accommodate a large variety of data storage needs [68, 70]. Similar results were obtained in a performance comparison of various types of non-relational databases against MySQL [23]. Focused on the storage of unstructured data of hospital patients during COVID-19, various forms (Key-value stores, Graph based, Column-oriented, Document) of non-relational databases were evaluated based on their data model, CAP (Consistency, Availability, and Partitioning) theorem, suitability for being distributed across multiple servers and other factors. The authors eventually designed an algorithm able to suggest the most suitable database type according to the hospital's needs. Also targeting a COVID-19 dataset, a recent study investigated data retrieval

from an unstructured large volume dataset, the COVID-19 Genome Sequence dataset [17]. It concluded that non-relational databases outperform SQL databases in aspect of data load time. Moreover, it indicated that non-relational queries were easier to formulate than SQL ones. This has been further supported by another study of a dataset of COVID-19 patients, where the NoSQL MongoDB database showed superior performance over other databases, demonstrating that it is more appropriate for processing large amounts of data [8].

In terms of privacy and security, not only do most non-relational databases not provide encryption mechanisms to protect user-related sensitive data, but also by default the inter-node communication is not encrypted for data in transit [24]. A recent review of advancements for these databases to improve the security reported their use of Kerberos (a computer-network authentication protocol) to authenticate clients and data nodes. It also proposed solutions to deal with remaining shortcomings such as usage of an Identity Provider to authenticate and communicate where the user needs to login using a Single Sign-on method [91]. In addition, researchers have designed a Security-as-a-Service model for NoSQL databases (SEC-NoSQL) which supports execution of query over encrypted data with guaranteed level of system performance.

C. Data storage costs and cloud implementation

Another important aspect when comparing different types of databases is the costs of running the database; this is particularly significant for large organisations which deal with large volumes of data on a daily basis. Focusing on financial trading data, four different databases were used for comparison in [25]. While MongoDB proved the fastest to read and write end-of-day OHLC (Open, High, Low, Close) data — the SQL solutions were 1.5 × to 3 × slower — in terms of costs MongoDB was definitely the most expensive due to its commercial licensing costs.

To reduce costs, more and more databases run on cloud platforms as they offer low-cost servers and high-bandwidth networks delivering better reliability, durability, scalability and accessibility of data. As mentioned before, as scalability is a particular strength of non-relational databases, their presence on Cloud allows their growth in a matter of just a few clicks. Not only do the main cloud providers support and manage a variety of relational databases (such as the popular Oracle, MySQL, and PostgreSQL), but they have also been developing their own proprietary non-relational databases to address their own needs, e.g., BigTable by Google or DynamoDB by AWS (Amazon Web Services) [25]. Indeed, for example, in 2006, Google needed a solution for its ever-growing collection of semi-structured data that was distributed across multiple data centres worldwide. As the relational model they had been using was unable to accommodate such a large pool of data efficiently enough, they created BigTable, a document-based database. Nowadays, it handles most of their infrastructure [26]. Advancements in non-relational architecture motivated Yahoo to develop criteria to quantitatively evaluate non-relational database systems. Its Cloud Serving Benchmark is the most widely used and well-known benchmarking framework for evaluating NoSQL databases with varying workloads.

In [27], the author has surveyed non-relational databases on Cloud and recorded their features in terms of the storage type (Column, Key-value, Document or Graph), the license type (Commercial or Open source) and the programming language used to develop them. He reported that, out of the 15 cloud databases surveyed, MongoDB, Cassandra and HBase were the most used.

Show how financial markets have evolved in the last decade and have become more complex and interconnected than ever before. One cannot get a comprehensive view of a portfolio with one source of data. In the financial markets the volume of the data grows exponentially: with the growing capabilities of computers, many companies have used a fast-increasing amount of historical data to feed predictive models, forecasts, and trading impacts. Advances in big storage and processing frameworks combined with the cloud capabilities have helped financial services firms to unlock the value of data, improve their volumes and, commissions, and reduce the cost-of-trades [28]. Moreover, a recent survey has shown the value of ‘alternative data’, i.e., data originating from non-financial sources such as social media, GPS, or sensor data, for predicting stock prices and discovering new price movement indicators. Consequently, capital firms need to store and stream, in various formats, enormous amount of data, and effectively link the data together to get an actionable insight. Big data processing frameworks,

which offer parallel and distributed algorithms running on clusters of servers such as Map Reduce, Hadoop, Spark, have fulfilled their requirements at least in terms of carrying out their batch processing tasks [29] and [30]. With the increase in computing power and decrease in data storage costs, collecting and processing large amounts of data has become an increasingly viable and exercised routine in the financial industry. Still, it is important for such organizations to select their database carefully so that it can, not only store and process big data, but also handle their growth in the long term.

III. Problem Definition

As previous studies have shown, no database system provides best performance in all scenarios. On one hand, relational databases deliver accuracy and redundancy by following the ACID properties. On the other hand, non-relational databases support large and distributed datasets with frequently changing schemas providing better performance and flexibility, which makes them particularly attractive for industries requiring high-performance analytics capabilities and distributed large data scalability. Currently, efforts are being made to merge the two database systems to offer the best of both worlds, where, for example, a hybrid model would provide the flexibility that is prevented by the rigid relational database framework. Most recently, a hybrid database was implemented where simple requests (read, insert) were served by MongoDB, while complex operations, such as joins with filtering the requests, were forwarded to PostgreSQL. These hybrid models integrate SQL and NoSQL databases in one system to eliminate the limitations of individual systems. Even though they have produced promising results, their adoption has hardly started. Indeed, not only do they make maintenance more complex as two different databases must be handled, but also their associated costs are added. Moreover, a hybrid interface must be written to bridge the two databases together. Finally, there is no readily available solution that an organisation can install and run like any other database system.

Considering all the limitations of database systems when dealing with big time-series data and the requirement to use a system that can scale on-demand, in the next section we will be proposing a set of criteria to consider when selecting a database. We will then use a custom benchmarking tool for recording the results of our experiments and rating each database against the criteria to propose the best performing database.

IV. DM-DATA Model

A. Integrating the Amazon RDS with Oracle Database:

A relational database in the cloud may be set up, run, and scaled more easily with the help of Amazon RDS, a web service. Installation, disc provisioning and maintenance, patching, minor version updates, unsuccessful instance replacement, as well as backup and recovery of your Oracle database are all automated by Amazon RDS. Amazon RDS also supports automatic Multi-AZ (Availability Zone) synchronous replication, allowing you to create a highly available environment controlled entirely by AWS. If you want Amazon to administer your Oracle database on a daily basis, Amazon RDS is the best option. This allows you to concentrate on higher-level activities like schema optimization, query tuning, and application development as in Figure 1.

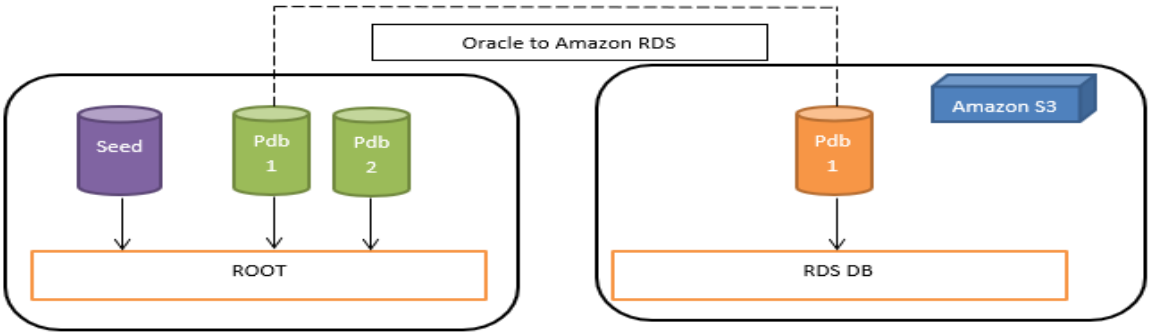

Figure 1. Database to Amazon RDS for Oracle

B. DM-DATA Model:

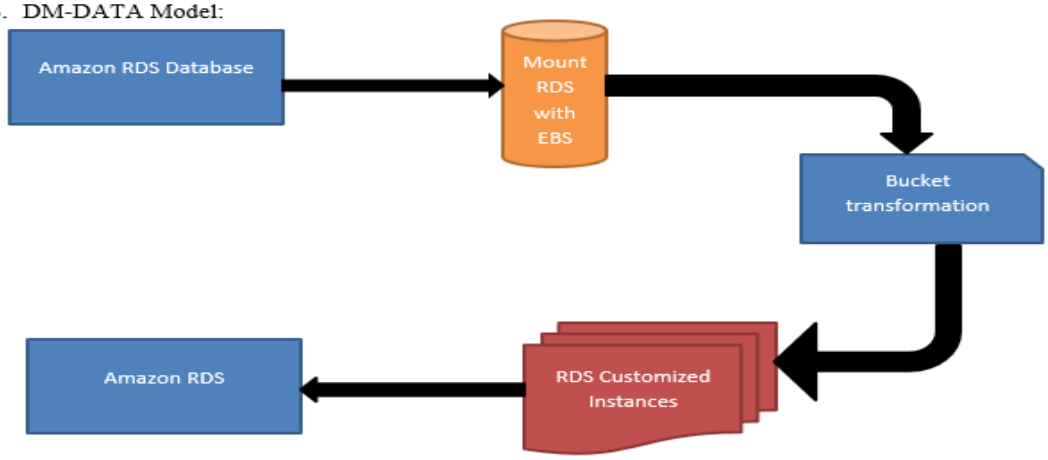

Figure 2. DM-DATA Model with data migration with Amazon S3

Either a physical migration strategy or a logical migration strategy can be used to migrate Oracle databases. Because duplicating or replicating the data at the block level provides simplicity and assurance, a physical migration technique is frequently used. A data validation exercise is often not required when using a physical transfer approach. On the other side, a logical migration method makes it easier to migrate with less downtime and between major versions. Validating data is a vital phase in the migration process and logical migration may need more time and effort to validate the procedure. You can use one of these solutions to migrate from Amazon RDS for Oracle to Amazon RDS Custom for Oracle. But in this post, we focus more on the physical migration method. The logical migration strategy is covered in detail in the Strategies for Migrating Oracle Databases to AWS report, which also applies to the transition from Amazon RDS for Oracle to Amazon RDS Custom for Oracle.

For physical migration, we utilize RMAN backup to copy the database between Amazon RDS for Oracle and Amazon RDS Custom for Oracle. Backups can be transported from source to target utilising an Amazon Simple Storage Service (Amazon S3) bucket, Amazon Elastic File System (Amazon EFS), or a database connection.

The set-up of Data Guard or automatic log shipping and application is not supported by Amazon RDS for Oracle. However, downtime for the migration can be decreased by transferring and applying archived logs from the source RDS for Oracle instance to the target RDS Custom for Oracle instance until cutover time. Figure 2 depicts the physical migration utilizing Amazon S3 integration.

You can use different tools for logical transfer, such as AWS Database transfer Service (AWS DMS), Oracle Golden Gate, and Oracle Data Pump, to load the data and replicate events. Reference architecture for logical migration employing RMAN backup for initial load and AWS DMS for replication of continuing transactions is shown in the diagram below. If a physical migration plan does not fulfill your business goals, such as decreased downtime migration, you may use a logical migration approach to move from Amazon RDS for Oracle to Amazon RDS Custom for Oracle. The procedure for logical

migration is covered in Transfer an Oracle database from a local installation to Amazon RDS for Oracle.

B. Oracle databases' file transfers to Amazon RDS:

External files are used as input in many integrated Oracle applications. Oracle databases access such files via a logical entity known as a database directory. Oracle databases employ database directories to access data pump backups, external tables, reading logs, and more, in addition to application files. The database administrator must move the files to be processed from one server to another in the conventional on-premises client-server architecture, log in to the database server to construct an Oracle database directory object, and use the aforementioned tools. With Amazon Relational Database Service (Amazon RDS) for Oracle, some of these jobs are taken care of for you, as we show throughout this post.

Benefit from a managed service solution with Amazon RDS for Oracle, which makes it simple to set up, run, and grow Oracle deployments in the AWS Cloud. Amazon RDS for Oracle enables you to access files using database directory objects and native tools in the same manner that you can with on-premises Oracle databases. The primary distinction between Amazon RDS for Oracle and on-premises Oracle deployments is that Amazon RDS for Oracle is a managed service, therefore access to the underlying host is restricted to provide a completely managed service. Because you can't access the underlying operating system for your database in Amazon RDS for Oracle, we need to create a solution that uses Amazon Simple Storage Service (Amazon S3) and AWS Lambda to load files into Amazon RDS for Oracle storage. If the quantity or amount of files to be moved to your Amazon RDS for Oracle database is small or infrequent, you can manually move the files to Amazon S3, download the files from Amazon S3, and then load or process the files in the database. However, when your business logic requires continuous importing and processing of a large number of files, automating this process enables IT organizations to devote their time to tasks that provide greater value to the company.

The goal of this post is to show how Amazon S3 and Lambda can be used to automatically move files from a host (on-premises or in the cloud) to an object database directory inside an Amazon RDS for Oracle database local storage.

The automation of file transfers from on-premises to Amazon RDS for Oracle databases using Amazon S3, Lambda, and AWS Secrets Manager. After the files have been posted to S3 buckets, an S3 event starts a Lambda function that gets the Amazon RDS for Oracle database keys from Secrets Manager and copies the files to the Amazon RDS for Oracle database's local storage. This procedure is depicted in the Figure 3.

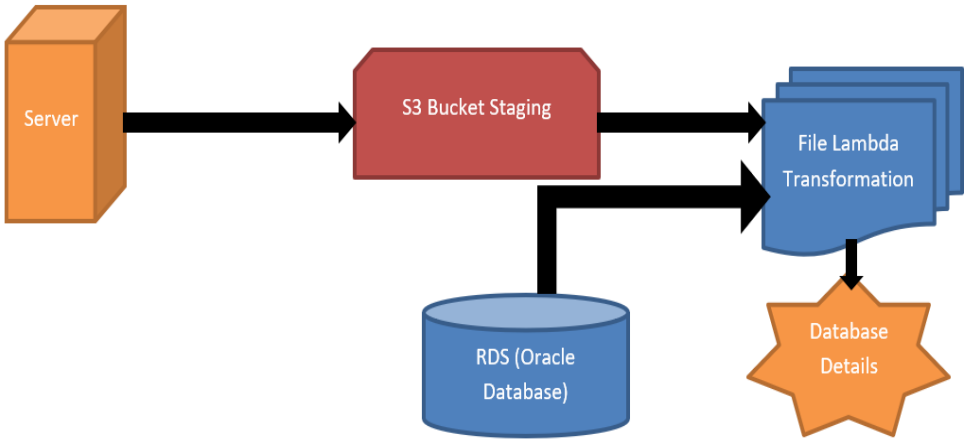

Figure. 2. Oracle databases' file transfers to Amazon RDS

Procedure to perform file transfers from Oracle databases to Amazon RDS

- Create an S3 bucket for file uploads to Amazon RDS for Oracle database local storage.
- Create a Secrets Manager secret for retrieving connection credentials to the Amazon RDS for Oracle database.

- Create AWS Identity and Access Management (IAM) rules and roles required by the solution to interface with Amazon RDS for Oracle, Secrets Manager, Lambda, and Amazon S3.
- Develop a Lambda function for automating file transfers from Amazon S3 to Amazon RDS for Oracle local storage.
- Set up S3 events to call the method whenever a new file is uploaded.
- Validate the answer.

V. Analysis The Perfromance On Dm-Data Model

There are several alternatives are choose to optimize the performance of the propose infrastructure with Oracle database based on certain metrics like, disk I/O management, sizing, database replicas, etc. Many variables influence the performance of a relational database instance on AWS, including the Amazon RDS instance type, database software setup, application workload, and, for Oracle databases operating on Amazon RDS instances, storage configuration. The choices you have to optimize the performance of the AWS infrastructure on which your Oracle database is operating are described in detail.

A. Instance Sizing:

Increasing the performance of a database requires an awareness of which of the server's resources is the performance bottleneck. If the database performance is constrained by CPU, memory, or network throughput, you can increase memory, compute, and network performance by selecting a bigger instance type. For many clients, improving the speed of a single database instance is the simplest approach to improve the entire performance of their application. One way to do this is with vertical growth. You accomplish this by adjusting the instance size to meet the database's hardware performance requirements. Vertical scaling in the Amazon RDS and Amazon EC2 environments is relatively simple.

the capability in Amazon RDS to select the instance type that best suits your workload. Various database instance classes are supported by Amazon RDS. They now span the size spectrum from the incredibly tiny Micro to the High-Memory Quadruple Extra Large, which has eight virtual cores, 68GB of memory, and a large I/O capacity. Regarding CPU, memory, and I/O capacity, the Amazon RDS instance classes are nearly comparable to the Micro, Standard, and HighMemory Amazon EC2 instance kinds.

B. Database Replicas:

Spreading the burden of database queries over numerous instances is one way to get better performance. Scaling out or horizontal scalability are two terms used to describe this method. Amazon RDS presently does not support Oracle read-replicas. To boost database throughput, expand vertically (use bigger instance types). As an alternative, you may "shard" your database, which divides it horizontally among many Amazon RDS servers. Data might be shared based on real-world parameters (for example, product category and consumer region), or it could be distributed among different shards using a hashing technique.

VI. Conclusion

In order to overcome the problem of hardware failures, datacenter issues, and disasters in the database, data migration is performed to handle scalability, performance, backup and recovery, high availability, and security among the data. The work contribution is summarized as below, DM-DATA framework is establishing an Emergency Recovery solution with an onsite Oracle system and AWS and to migrate your existing Oracle database to AWS, which can handle scalability, performance, backup and recovery, high availability, and security in Amazon RDS. Transfer an Oracle database from a local installation to Amazon RDS for Oracle, which can migration data with fulfilled business goals, such as decreased downtime migration. The automation of file transfers from on-premises to Amazon RDS for Oracle databases using Amazon S3, Lambda, and AWS Secrets Manager

To optimize the performance of the propose infrastructure with Oracle database based on certain metrics like, disk I/O management, sizing, database replicas, etc. AWS offers two deployment options for Oracle databases as Amazon RDS. We have covered performance, high availability, monitoring, and security management for both settings in this whitepaper. You will gain from the benefits of using Amazon Web Services (AWS), including Oracle instances and storage that may be provisioned quickly and easily on AWS with no capital outlay and Oracle Database on AWS, a service provided by Amazon Web Services (AWS) as High security, durability, and availability, as well as low cost Pay-per-use pricing.

References:

[1]. Abdelhafz BM, Elhadef M (2021) January. Sharding Database for Fault Tolerance and Scalability of Data. In 2021 2nd International Conference on Computation, Automation and Knowledge Management (ICCAKM) (pp. 17–24). IEEE.

[2]. Abourezq M, Idrissi A (2016) Database-as-a-service for big data: An overview. International Journal of Advanced Computer Science and Applications (IJACSA), 7(1).

[3]. Agarwal S, Rajan KS (2017) Analyzing the performance of NoSQL vs. SQL databases for Spatial and Aggregate queries. In Free and Open Source Software for Geospatial (FOSS4G) Conference Proceedings (Vol. 17, No. 1, p. 4).

[4]. Singh et al. Journal of Cloud Computing (2022) 11:53 Page 16 of 17 4. Agarwal T, Quelle H, Ryan C (2020) Stock Trend Evolution. University of Arizona.

[5]. Ahmad AAS, Andras P (2019) Scalability analysis comparisons of cloudbased software services. Journal of Cloud Computing 8(1):1–17

[6]. Ahmad K, Alam MS, Udzir NI (2019) Security of NoSQL database against intruders. Recent Patents on Engineering 13(1):5–12

[7]. Compose, An IBM Company. Alba, L., November 2016. Building OHLC Data in PostgreSQL. Available from https://www.compose.com/articles/building-ohlc-data-in-postgresql/. Accessed 26 Oct 2021.

[8]. Antas J, Rocha Silva R, Bernardino J (2022) Assessment of SQL and NoSQL Systems to Store and Mine COVID-19 Data. Computers 11(2):29

[9]. Bagui S, Nguyen LT (2015) Database sharding: to provide fault tolerance and scalability of big data on the cloud. International Journal of Cloud Applications and Computing (IJCAC) 5(2):36–52

[10]. BalaMurali A, Sravanthi PS, Rupa B (2020) January. Smart and Secure Voting Machine using Biometrics. In 2020 Fourth International Conference on Inventive Systems and Control (ICISC) (pp. 127–132). IEEE.

[11]. Gartner Bala R, Gill B (2021) Magic Quadrant for Cloud Infrastructure and Platform Services. Available from https://www.gartner.com/doc/reprints?id=1-271OE4VR&ct=210802&st=sb. Accessed 26 Oct 2021.

[12]. Balusamy B, Kadry S, Gandomi AH (2021) NoSQL Database. Big Data: Concepts, Technology, and Architecture, Wiley, pp. 53–81.

[13]. Beaulieu A (2009) Mary E Treseler (ed.). Learning SQL (2nd ed.).Sebastopol, O'Reilly. ISBN 978–0–596–52083–0.

[14]. GitHub Singh B (2021) Cloud based evaluation of databases. Available from https://github.com/handabaldeep/cloud-based-evaluation-of-databases. Accessed 26 Oct 2021.

[15]. Bhatti HJ, Rad BB (2017) Databases in cloud computing. Int J Inf Technol Comput Sci 9(4):9–17

[16]. Cao Z, Dong S, Vemuri S, Du DH (2020) Characterizing, modeling, and benchmarking rocksdb key-value workloads at facebook. In 18th {USENIX} Conference on File and Storage Technologies ({FAST} 20) (pp. 209–223).

[17]. Chakraborty S, Paul S, Hasan KA (2021) January. Performance Comparison for Data Retrieval from NoSQL and SQL Databases: A Case Study for COVID-19 Genome Sequence Dataset. In 2021 2nd International Conference on Robotics, Electrical and Signal Processing Techniques (ICREST) (pp. 324–328). IEEE.

[18]. Chauhan VP (2019) Google Big Table: A Change to Data Analytics. International Journal of Information Security and Software Engineering 5(1):5–9

[19]. Chawathe SS (2019) September. Cost-Based Query-Rewriting for DynamoDB: Work in Progress. In 2019 IEEE 18th International Symposium on Network Computing and Applications (NCA) (pp. 1–3). IEEE.

[20]. Chen JK, Lee WZ (2019) An Introduction of NoSQL Databases based on their categories and application industries. Algorithms 12(5):106

[21]. Codd EF (1970) A Relational Model of Data for Large Shared Data Banks. Commun ACM 13(6):377–387

[22]. Cooper BF, Silberstein A, Tam E, Ramakrishnan R, Sears R (2010) Benchmarking cloud serving systems with YCSB. In Proceedings of the 1st ACM symposium on Cloud computing (pp. 143–154).

[23]. DB-Engines. DB-Engines Ranking 2021. Available from https://db-engines.com/en/ranking. Accessed 8 Oct 2021.

[24]. Dean J, Ghemawat S (2008) MapReduce: simplifed data processing on large clusters. Commun ACM 51(1):107–113

[25]. DeCandia G, Hastorun D, Jampani M, Kakulapati G, Lakshman A, Pilchin A, Sivasubramanian S, Vosshall P, Vogels W (2007) Dynamo: Amazon's highly available key-value store. ACM SIGOPS operating systems review 41(6):205–220
[26]. Deka GC (2013) A survey of cloud database systems. It Professional 16(2):50–57
[27]. ElDahshan KA, AlHabshy AA, Abutaleb GE (2020) Data in the time of COVID-19: a general methodology to select and secure a NoSQL DBMS for medical data. PeerJ Computer Science 6:e297
[28]. Erraji A, Maizate A, Ouzzif M (2021) Toward a Smart Approach of Migration from Relational System DataBase to NoSQL System: Transformation Rules of Structure. In The Proceedings of the International Conference on Smart City Applications (pp. 783–794). Springer, Cham.
[29]. Fang B, Zhang P (2016) Big data in fnance. In Big data concepts, theories, and applications (pp. 391–412). Springer, Cham.
[30]. Fiess NM, MacDonald R (2002) Towards the fundamentals of technical analysis: analysing the information content of High. Low and Close prices Economic Modelling 19(3):353–374.