

# Comparative analysis of Amazon Web Services: Deciding right database for your Applications

Preeti Bhatt\*

*\*D.C.S.A. Panjab University sector-14 Chandigarh  
Preeti.gc@gmail.com*

**Abstract-** The idea in most databases that were ever developed, built, invented were built for a purpose. And when we find a database which was built for the purpose that our application really needs then we have the best match. In this paper we are focusing on choosing the right databases for your applications and exploring Amazon web services (AWS) - DynamoDB, Athena, Redshift, and Kinesis, Relational Database Service (RDS), DynamoDB Accelerator (DAX) and Amazon Neptune.

**Keywords-** Non-Relational, NoSQL, DynamoDB, Amazon Web Services (AWS), Scalability, Distributive Databases, Redshift, Athena, kinesis, Elastic Search.

## I. INTRODUCTION

Database is all about storing retrieving and processing data. We look at the application with respect to data - the shape of data, size of the data and computational requirement for the data. Data considerations are Shape, Size and Compute. And these are the three important considerations that helps you decide which database is the right database for your application.

### 1.1 Shape:

When it comes to database, relational database is around us for a long time since the 70s and it's build around a row. So, the concept of row is the fundamental component of an RDBMS. Most of the applications that were built for payroll, HR, Finance and so on, models the data in the form of row which were have good performance when you want to deal with a record. But as time goes, people have lots of data in their database and they need to do lots of queries processing. So doing an analytic workloads on row store become an expensive and inefficient so that's why a column store was invented. Columns stores the data in the form of columns. It takes the rows, breaks it up, it stores each rows separately and this gives us great performance and efficiency. Key value store are invented to store keys and values where we are doing queries based on the key. The idea was we need to do lots of key value look ups and we need them to be really fast regardless of the size of data and that are the reason the key value store are used in so many workloads that have vast amount of partionable data where you are able to get the data based on a key as fast as possible. Whether I have billion, hundred billion or trillion items in my table, I can get that consistent performance with key store.

*Document Store:* Document store are invented to store document. There is a lot of data which fits very nicely in the form of document. For example, patient record.

*Graph store:* When you need to persist relationship, graph store is really good thing so it's very fast in retrieving and processing relationship data. So social graph, recommendations fits really well in a graph store.

*Time series Store:* It is a data which is a sequence in time, you almost never updated, always in an insert workload. For example, stock data where set of computations applied to a range of data.

*Unstructured store:* For storing unstructured data where you impose the structure on the data yourself.

### 1.2 Size:

Size at limit is far more interesting than the current size of your database. Size at limit means there are some workloads where size of a data is fairly well bounded. For example, if you build an application to track an employee of a company.

### 1.3 Compute:

We have stores that are really good at retrieving and storing data but they don't do a lot of computation for it. Computing functions, throughput, latency, change rate, rate of ingestions are the things that you have to consider while choosing a database.

#### 1.4 Can we choose a single database that can do everything?

If we have very small workload and not a lot of data for example 50GB of data so we can probably use any brand of database for example relational database. But when stuff gets big and efficiency at scale really matters, operational efficiency is important, dealing with hundred terabytes of data then choice becomes really important.

#### 1.5 Managed database services vs unmanaged database services

Managed database services matters a lot because it changes the cost and convenience and developers capabilities. If we are going to build an application 30 years ago then we would go and buy an enterprise class servers, would buy some database, manage those servers. But when we move to the cloud and start using managed databases, a lot of those inconvenience go away. So everything is fast that an end of API call. So whether we make an API call to put an image in S3 (the BLOB store) or we make an API call to insert a date or item into DynamoDB or we do an insert into Postgres database like Aurora, no need to manage those servers to do backups on them or fix the memory when they breaks or replace the disk that dies. So developer now have a choice to choose a right set of API calls to build an applications.

#### 1.6 Which database to use when?

Our strategy is: - *Purpose-built* -> *managed* -> *open* -> *customer obsessed*

We need a purpose built databases to satisfy particular workload for the best performance price, convince and programmability. Developer should not choose only one database because we can have miss-match workload and miss-match databases.

#### 1.7 Workloads classification

Most workloads we look at is a kind of split between analytics and operational workloads.

- 1 Analytic workload is about getting insight.
  - 1.1.1 Retrospective Analytics: It may be retrospective that tells you what happened yesterday. For example how much stuff we sell last year or last five years, to look at history.
  - 1.1.2 Steaming Analytics: It could be the steaming like data coming from some devices and you need to raise an alarm when something gets too harder.
  - 1.1.3 Predictive analytic: It is used when you look at the body of data and trying to figure out what is going to happen tomorrow.
- 2 Operational workload includes
  - 2.1.1 Transactional workload for example: if Tina took \$10 out of her account and put \$10 in her account.
  - 2.1.2 System of records for example: Employee record, patient record.
  - 2.1.3 Content Management Systems (CMS) for example: Blog, WordPress etc.

#### 1.8 Relational or Not

- 1 Relational databases are really good if you need Referential integrity with strong consistency, transactions, and hardened scale. It provides complex Query support through SQL.
- 2 Key-value databases are good for Low –latency key based queries with high throughput and fast ingestion of data. It provides simple query methods with filters.
- 3 Document store databases are good for indexing and storing documents with support for query on any property. It provides simple query with filters, projections and aggregates.
- 4 Graph based databases are good for creating and navigating relations between data easily and quickly. It provides easy expressible queries in terms of relations.

#### 1.9 AMAZON RDS (Relational Database Service) [19]

RDS is a great general purpose database that has the ability to start very small and grow with your business and offers all the different engines so, RDS is a great place to start when you are building an application where you want to use common framework like ruby on rails or django or you may have a particular engine that you choose on the basis of skills in your teams, type of framework that you are using or may be third party packages that you bringing from the outside.

*For example:-* If I am building an application that have some ability to track location, so people gravitate towards **postgres** because it has post GIS-the ability to run geospatial queries in relational context. My team is really comfortable with SQL but I need the massive scale, I need ability to scale up dynamically on my storage volume and I need it to give me huge number of read and writes so in both case **Amazon Aurora** is a great choice that scale extremely high over time. For relational workload, I need relational capabilities, I like SQL as a language but there is a limit like if I want to record the location of every car in every minute and have a massive input coming then **Amazon RDS** is one of the system for this purpose. You have provision of certain amount of storage and you can grow into that storage certainly with Amazon Aurora, you can be flexible with the size of database. The ability to have completely predictable performance regardless of the size of the database then we may need to look upon something little bit more sophisticated and gives us some features that relational database traditionally have not able to support.

#### 1.10 DYNAMODB [8]

DynamoDB is going to give us the ability to break out of the boundaries that we may find from a relational database with the relational validations that's occurring with the database and give us the ability to operate much more efficiently with unbounded scale. If you have a huge amount of push down computation, you may put that against the database let's say RDS database. If you putting shopping cart as the ability to transact with really high availability but you can never run out capacity then you choose **DynamoDB**. Moving data between data sources is done by data migration services. Like, **dynamoDB update stream** which allow you to listen to changes that are occurring in your dynamo environment and then replicate those down into relational database. So there are the integration pattern to move data between these two different environments.

#### 1.11 DAX (DynamoDB Accelerator) [15]

Dynamo DB running a scale that is unbounded. DAX gives us virtually unbounded low latency. DAX allow us to create a layer with dynamoDB to implement an application with really high availability and high performance. We now add DAX that derives the latency down to micro-seconds but where we don't have to worry about managing the complexity of cache management where we don't have to think about what is the data up to date in my DynamoDB table vs my cache, the reason is that DAX we called right-through cache, so we do writes to DAX and DAX take responsibility for writing to DynamoDB. So, we gets all the benefits like, we can get API compliant with dynamoDB with massive exploration and micro-seconds performance.

#### 1.12 AMAZON ELASTIC CACHE [18]

It allow us to add caching or using different type of operation store. The real difference between DAX and elastic cache is that

DAX is a specialized caching for DynamoDB and the elastic cache is a generic caching mechanism with Redis store and you can put elastic cache in front of any of RDS database. For example, dynamo DB too. DAX is built in write-through cache while elastic cache is explicitly added into other operational systems. Elastic cache gives you memcached and Redis interface so you can have an open approach for bringing your entity management framework. For low latency read, I need DynamoDB and use DAX. If you need low latency reads and your data-sets size that you operating is fits the cache then caching is a good thing. But if your data is bigger than a cache, and every action the cache misses, then you are wasting your time on missing the cache.

#### 1.13 AMAZON NEPTUNE [17]

Graph is really an important thing. *Why would you use a purpose built specialized database for something like relations?* Because it is really hard to manage graphs and relations in any other type of operational store. We have some other analytical systems that supports graphs like Elastic Search but with the kind of performance, we are going to get from Neptune you really need, a specially built database. Graph databases are specialized for query the data regarding people and their behavior and the links between the data. It is very difficult to compute relations between myself and every family member at the time of query using SQL. So we need to use graph databases or graph store so you can get the relationship and walk the nodes.

#### 1.14 AMAZON ATHENA-INTERACTIVE ANALYSIS [13]

It allow us to treat data at rest, unstructured or structured data like it was a database. For example, if you are working with lots of log files and you are going to store those log files, you should compress those files every time. If you are working with much more structured data then using a format like **parquet** or **orc** its very useful because it implements that compression but once you're written it down and its sitting on a storage system Athena just gives you the ability to start working with it. Athena is not a database. It is an engine for working with data that allows you to

write SQL as though it was a database. For example, I have OLTP system and I do transactions, sell stuff and have last five years' worth of sales data and I export it out my OLTP system in a comma separated vial and I put this file on Amazon S3. So we can query this file now with SQL. We are paying for the storage on Amazon S3 for the data that you should be keeping anyway then you are able to issue a query using ANSI SQL. There is no service to provision, just pay for data that streams through Athena and you can get your answers back or you can either use it in an interactive way with the Athena console or you can use an API to save that data down to another location.

#### 1.15 AMAZON REDSHIFT [11] –Data Warehousing

If we have data warehouse and we want to issue queries, and want to do some cost optimization, so the redshift is really the counter side to where we are doing data exploration. You select redshift because you understand your data and its schema, its value. You import that data into a system that is designed for extremely fast query time at scale and there is also the ability to integrate in an Athena model with redshifts spectrum. So you don't actually have to choose either or you can use Athena to establish that some data is actually useful, you generate new data sets and immediately start querying those from inside of your enterprise data warehouse in a hybrid compute and storage model. So you get the best of both worlds of extremely fast query response times from data stored within the spectra the redshift data warehouse and then the ability to also use this data at rest

So you are modeled your data in a way that's sympathetic to the types of queries that you're going to get as opposed to Athena which is issuing a query you can expect and you might actually be modeling your data using a regular expression or a grok expression. In redshift you just do queries, don't process data. In Athena, you querying a data using regular expression.

#### 1.16 AMAZON KINESIS ANALYTICS [14]

The ability to do analytics second by second as the data comes in and again the common thread that runs across these analytics tools is that they use SQL. Athena is using ANSI SQL, redshift uses ANSI SQL and Kinesis uses streaming SQL. So it's really all about moving historical analysis into that very short term of decision making and you need to have an answer very quickly and your business needs to be able to react to that there needs to be business value associated with you making a decision quickly. For example: customer support case. You don't need to put this kind of data into data warehouse.

#### 1.17 AMAZON ELASTIC SEARCH SERVICE [16]

It does store your data in the form of specialized indices for doing a specific type of query. You might have an index that's particularly good at doing a natural language search or you might have another type of index that's really good at doing a time series expansion or interpolation or you might have another type of index that's particularly good at doing geospatial manipulations. So in these cases, you want to analysis that isn't about select star from query, it's about having a lot more flexibility to explore your data in different ways that are probably more natural. Using Elastic Search, you have the ability to create graphs and dashboards that are based upon these much more interesting types of queries and much more certain powerful type of problems. If I was storing documents in DynamoDB and I have big giant JSON and there's free text so the update stream from dynamoDB can send the changes that are happening and I can have a lambda trigger that calls elastic search and indexes the data in dynamoDB so that my application can go query Elastic Search for a particular keyword and find all the records in dynamo. To apply time series expansion on a stream of Internet of things (IOT) data that you are storing in dynamo DB because it's extremely high throughput rates and you need to be able to find anomalies.

## II. RELATED WORK

[1] Amazon S3 is a storage technology that provides reliability and scaling; meet scalability and availability needs by synthesizing various technologies. We studied the literature [2] in which the author compares NoSQL databases based on includes Replication, CAP features, Storage type, and Map Reduce and also did time comparison between insertions, deletion, updating operation. [3] Introduced a high performance key-value benchmark that can interconnect web-scale workloads. [4, 5] uses YCSB as a measurement tool to measure performance of NoSQL databases. We studied the literature [6] in which the author analyzed the performance of CouchDB and Elasticsearch based on insertion, deletion, updating and selection operations and conclude that CouchDB works efficiently on insertion, deletion and update operation but Elasticsearch works much better in case of selection operation. [7] Introduced YCSB tool for measuring read, update, insert operation time of key-value store databases [8] Describe the architecture of Couchbase servers and its evolution in detail. Couchbase internal architecture supports OLTP like workload. [9] Introduced YCSB tool to test NoSQL databases based on the factors such as data loading, reads, read –modify-write, only read, only updates and updates.

### III. CONCLUSION

These days where people are trying to find insight from all kinds of data so we can have hybrid systems. You can have your operational database wired with analytics database. For example, you can have Athena or redshift in combination. The nice thing about these combinations is that you are able to pick right set of tools because you may not need servers that are up all the times when you query occasionally then Athena comes really well. But for the dashboard that you need to be active all the time, you use redshift. So it is not about choosing right database, it's about choosing right set of databases to apply the right set of data to particular applications that you have. Redshift spectrum is for data that is optimized for working within a model for example- star-schema model, because it is a data warehouse workloads and the spectrum has the ability to separate, compute from storage within an enterprise data warehouse architecture. Athena is very different. You applying a scheme either through the data catalogue or when you are issuing the query so you have a lot more flexibility in terms of the types of data that you can issue query against with an engine like Athena where a spectrum is for giving you a very well structured, well modeled data. In Athena, you can share data between clusters using spectrum.

### REFERENCES

- [1] DeCandia G, Hastorun D, Jampani M, Kakulapati G, Lakshman A, Pilchin A, Sivasubramanian S, Vosshall P, Vogels W. Dynamo: amazon's highly available key-value store. In *ACM SIGOPS operating systems review* 2007 Oct 14 (Vol. 41, No. 6, pp. 205-220). ACM.
- [2] Kumar KS, Mohanavalli S. A performance comparison of document oriented NoSQL databases. In *2017 International Conference on Computer, Communication and Signal Processing (ICCCSP) 2017 Jan 10* (pp. 1-6). IEEE.
- [3] Shankar D, Lu X, Wasi-ur-Rahman M, Islam N, Panda DK. Benchmarking key-value stores on high-performance storage and interconnects for web-scale workloads. In *2015 IEEE International Conference on Big Data (Big Data) 2015 Oct 1* (pp. 539-544). IEEE.
- [4] Tudorica BG, Bucur C. A comparison between several NoSQL databases with comments and notes. In *2011 RoEduNet international conference 10th edition: Networking in education and research 2011 Jun 23* (pp. 1-5). IEEE.
- [5] Cooper BF, Silberstein A, Tam E, Ramakrishnan R, Sears R. Benchmarking cloud serving systems with YCSB. In *Proceedings of the 1st ACM symposium on Cloud computing 2010 Jun 10* (pp. 143-154). ACM.
- [6] Gupta S, Rani R. A comparative study of elasticsearch and CouchDB document oriented databases. In *2016 International Conference on Inventive Computation Technologies (ICICT) 2016 Aug 26* (Vol. 1, pp. 1-4). IEEE.
- [7] Patil S, Polte M, Ren K, Tantisirirot W, Xiao L, López J, Gibson G, Fuchs A, Rinaldi B. YCSB++: benchmarking and performance debugging advanced features in scalable table stores. In *Proceedings of the 2nd ACM Symposium on Cloud Computing 2011 Oct 26* (p. 9). ACM.
- [8] <https://aws.amazon.com/dynamodb/>
- [9] Jayathilake D, Sooriaarachchi C, Gunawardena T, Kulasuriya B, Dayaratne T. A study into the capabilities of NoSQL databases in handling a highly heterogeneous tree. In *2012 IEEE 6th International Conference on Information and Automation for Sustainability 2012 Sep 27* (pp. 106-111). IEEE.
- [10] <https://aws.amazon.com/>
- [11] <https://aws.amazon.com/redshift/>
- [12] <https://docs.aws.amazon.com/redshift/latest/mgmt/welcome.html>
- [13] <https://aws.amazon.com/athena/>
- [14] <https://aws.amazon.com/kinesis/>
- [15] <https://aws.amazon.com/dynamodb/dax/>
- [16] <https://aws.amazon.com/elasticsearch-service/>
- [17] <https://aws.amazon.com/neptune/>
- [18] <https://aws.amazon.com/elasticache/>
- [19] <https://aws.amazon.com/rds/>