# Amazon's DynamoDB

# (Key-Value Database) and Its Ecosystem

University of Illinois at Chicago

IDS 521 Advance Database Management

Dr. Yann Chang


**Team members:** Vanisa Achakulvisut, CJ All, Alex Timperman, Anas Sayoury

**Executive Summary**

To start off the term paper, let's introduce the DynamoDB Database: a fully managed NoSQL database service that provides fast and predictable performance with seamless scalability for its users. The introduction serves as a starting point for the user to be able to diagnose this paper with a wide array of knowledge regarding what DynamoDB is, and what's going to be explained about it. It wouldn't be fair to assume that all people within data analytics, big data, and database management, completely understand everything that has to do with this database service. In short, there's a lot to go over, and the intention is to make going over everything more manageable for the user.

The next section of the paper goes over the comparison between DynamoDB and a relational database system (RDBMS for short). There are several distinctions made between both of these types of databases: one of them is that RDBMS requires a schema where data is normalized while DynamoDB contains no schema. Another is that RDBMS uses SQL for storing or retrieving data while DynamoDB uses AWS Management Console or AWS Command Line Interface. In short, an RDBMS is optimized for storage, while DynamoDB is optimized for throughput. These are just a few of the comparisons and comparable traits made between both databases. After distinguishing the difference between DynamoDB with the RDBMS, what can a user take away from both the positives and negatives of the database? That question is answered in the next section.

After comparing DynamoDB with an RDBMS, the reader is then led onto the advantages and limitations of the database. Some of the advantages that are gone over include its scalability, performance, and ability to support many different data types along with a few others. A few of the limitations include the 64KB limit on each row size of the data, 1MB limit of querying, and

the limited querying abilities (querying non-indexed data, and inability to run complex queries just to name a couple). Each point is gone over in detail, and leads the user to form his/her own viewpoints on whether or not the database is for him/her from both the pros and cons. How would the user know about any of these advantages and limitations if they haven't used the database before? That then leads the reader into the next section.

How does one use DynamoDB? That question is addressed in the section "How to use DynamoDB", which goes over an 8-step tutorial on how a user can get started with the database. The tutorial explains basic functions such as how to create a table, write data into a table, and update data, which are all important for understanding how to use the database. A user should not only have information regarding what DynamoDB is conceptually, but should also understand how to use the software to really experience the functionality and importance of the database.

Now that there's an idea about how to use DynamoDB, what about use cases and evidence regarding how successful the database service is for a few of its users? "Use cases of DynamoDB" has a full list of examples filled with examples of how companies and users are implementing this database into their businesses and lives. Companies such as Netflix, Amazon, and Lyft are mentioned along with the creation of real-time dashboards and data recreation, just to name a few names and types of use cases for different situations. The phrase "The proof is in the pudding" could not be more true when looking into this section.

When it comes to the pricing of DynamoDB, the main idea that lies within this section is that DynamoDB charges for reading, writing, and storing data in DynamoDB tables and extra features depends on user selections. It has two capacity modes;  Pricing for on-demand capacity mode and Pricing for provisioned capacity mode. The charges for each data storage option, both

on-demand and provisioned capacity modes, and service costs are also brought up, detailing how each of them gets priced for DynamoDB. Lastly, there is a full table explaining different features, what each of those features do, and the Billing Unit for each of them (Ex. $ per GB) just as an added bonus for pricing granularity.

Last but certainly not least, is the conclusion of our paper, along with the sources cited. The conclusion rounds up every section from the Introduction up until the Pricing section, and serves as helpful backtracking for re-emphasizing the main points of the paper. If you're a reader and are reading this paper for the first time, the conclusion is ideal for getting an idea of what to take away from DynamoDB and its main selling points. Without further ado, let's now dive into the introduction:

**Introduction**

One of the biggest obstacles faced in the world of E-commerce today, is the rapidly expanding quantity of data being created every day. Each day humans are creating 2.5 quintillion tons of data, and this rapid data production is the reason that 90% of the world's data has been created in the last two years (Bulao, 2021). This drastic increase in data illustrates the importance of a fast and scalable database to process and store this incredible quantity of data. DynamoDB, developed by Amazon in 2007, was created for exactly this purpose. DynamoDB is a NoSQL Non-Relational database that was built out of necessity because of the rapidly growing quantity of data being produced by their online marketplace. It allows any user to store and retrieve any amount of data, and at the same time support any amount of server traffic (Niranjanamurthy et al., 2014, p. 269). DynamoDB today has become popular across all industries and is now used by

other major companies such as Nike, Netflix, Capital One, Samsung, and the U.S Census Bureau (Introduction to DynamoDB, n.d.).

DynamoDB was designed with a focus on a few key qualities to ensure the best database system for their rapidly growing online marketplace. The major areas of focus were continuous availability, Network partition tolerant, no-loss conflict resolution, efficiency, economy, and lastly incremental scalability (Harrison, 2015, p. 45). Continuous availability coupled with network partitioning assures that the data store will always be available even if there is network partition (Harrison, 2015, p. 45). This is incredibly important because with a distributed database that supports their online marketplace, an outage in the data store caused by a network partition or another source could be incredibly costly. The data needs to remain available at all times for the marketplace to be operational. Another major focus was no-loss conflict resolution. This means that a user of the marketplace should never lose an item that they added to their "shopping cart", regardless of when or where they added it from, and they should also never be blocked from making an addition to the shopping cart (Harrison, 2015, p. 45). No-loss conflict resolution is integral for the overall customer experience on the marketplace. The broad goal of Amazon as an online store is to maximize profit, and ensuring that a customer can easily browse and purchase products is one of the primary goals of their business. The last two focuses of DynamoDB were that the system needed to run on commodity hardware and that the system needed to be incrementally scalable (Harrison, 2015, p. 45). Amazon recognized that they not only needed a database for their current data but also one that could handle an immense amount of traffic as well. Because of this Amazon ensured that the distributed system could be scaled based on their needs now and in the future. With all of these focuses though Amazon did have to make sacrifices, particularly with the consistency of the database.

The Power of Amazon's DynamoDB can be witnessed in a case study with a company called FanFight. Fanight is a fantasy sports company specialized in India's most popular sport, Cricket. FanFight was initially paired with MongoDB Atlas for its cloud based database because of its availability (FanFight Case Study – Amazon Web Services, 2020). However FanFight saw massive influxes in activity spikes during big Cricket matches and needed a system that could efficiently scale to meet the demand. FanFight was able to find a solution in DynamoDB because it allowed for automatic scaling during high traffic periods. This allowed them to handle spikes in traffic and eased the burden on the small, 50 employee company. It also was able to seamlessly pair with their existing cloud based platform, AWS Lambda, and they did not have to sacrifice any availability (FanFight Case Study – Amazon Web Services, 2020). The ability for Amazon to scale up to properly handle high traffic events and scale down during low traffic periods allowed FanFight to reduce the cost of their database by half and quadruple their revenue (FanFight Case Study – Amazon Web Services, 2020). FanFight is a great example of how DyanmoDB's constant availability and scalability can drastically improve the overall business practices of a company. It also exemplifies how DynamoDB can be a solution for any company whether it has 50 or 5,000 employees.

**DynamoDB basic concepts & environment**

One of the major limitations to database architecture is the concept of the CAP theory. In a distributed database there are three major characteristics: Partition Tolerance, consistency, and availability. Partition Tolerance means that the entire distributed system will continue to operate even if there is a disturbance in communication between two nodes, consistency is the idea that all clients will see the same data regardless of node they are connected to, and lastly availability

means that a node will return a response if there is a request (IBM, n.d.). The CAP theorem, first created by Eric Brewer, says that in a distributed system you can have two of these three characteristics, but it is impossible to have all three (Niranjanamurthy et al., 2014, p. 269). What made DynamoDB so successful is that it created a distributed database that did not have to completely sacrifice any of the three major characteristics. Instead it allows some consistency to be sacrificed to allow for availability and network partitioning to be possible (Harrison, 2015, p. 42). This form of consistency is called eventually consistency because the system will not always be up to date immediately after every transaction but eventually it will reach a consistent point. This eventual consistency allows for DynamoDB to ensure availability and partition tolerance across its distributed database.

The idea behind eventual consistency has become prominent since the early 2000's with the rise of NoSQL databases (Bailis, Ghodsi, 2013, p. 55-63). When comparing the CAP theorem to the overall user experience on an e-commerce or social media website, availability and partition tolerance play a significant role in the useability of the platform. The availability and partition tolerance are integral in ensuring the website is accessible and useful at all times. Because of this eventual consistency rose to popularity with NoSQL databases such as Amazon DB. Eventual consistency means that when you read data from DynamoDB the data received in the response may not be entirely up to date but instead may be stale data. Even though the database may not always be up to date, in theory eventual consistency should reach a point of consistent data in the future, "all servers eventually "converge" to the same state; at some point in the future, servers are indistinguishable from one another" (Bailis, Ghodsi, 2013, p. 55-63). This however can be problematic because with a database that is constantly updating and receiving more data, servers will be constantly converging and never fully in sync. This can lead

to issues because users will never be certain of the accuracy or the timeliness of the data they are using (Bailis, Ghodsi, 2013, p. 55-63). However, this drawback is the necessary price to pay to ensure the availability and performance of the database.

Even though DynamoDB defaults to eventual consistency it does allow tunable consistency for users who require real time consistency for their data. This can be set in the configuration of the database using specific read option commands such as GetItem and Query (Introduction to DynamoDB, n.d.). If strong consistency is set though some drawbacks with the database will arise such as a lack of guaranteed availability, high latency, and the use of more throughput capacity (Introduction to DynamoDB, n.d.). Based on business needs this option could be useful but it sacrifices some of DynamoDB's most important qualities such as consistent data availability and rapid data retrieval.

One incredibly important feature of DynamoDB is that it uses consistent hashing for data retrieval. Consistent hashing allows for quick retrieval of data and is one of the reasons that DynamoDB can be incrementally scalable. Basic Hash tables allow a database to index immense amounts of data which allows for efficient retrieval of information. The way that this is accomplished is through running a hash function on an object and then running a modelo function on the hash to receive the index values (Carzolio, 2017). This allows the database to avoid creating individual index values for every data point that would need to be iterated through for every retrieval (Carzolio, 2017). This can then be turned into a distributed system, where the objects and keys are stored across different servers, typically in specific cache servers. "To store or retrieve a key, the client first computes the hash, applies a modulo N operation, and uses the resulting index to contact the appropriate server (probably by using a lookup table of IP addresses)" (Carzolio, 2017). The system allows for efficient retrieval of data even when

multiple servers need to be contacted. However, this system can become problematic when new servers are added because each server key, that is used to direct the user to a specific server, will need to be reallocated with the addition (Carzolio, 2017). This can make scalability incredibly difficult and inefficient, and because of that makes it nearly impossible for a database to scale on this basic distributed hashing system.

One of the ways to overcome this issue though is through consistent hashing. Consistent hashing is done by creating a hashing system that is independent of the number of servers found in the system. Using this independent hashing system allows a database to scale without worrying about disrupting the overall system. Consistent hashing will assign index values to data as well as the servers around a circle. Each data point will be assigned a server based on the closest server found counter clockwise to it on the circle (Carzolio, 2017). This allows for increased flexibility when adding and removing servers because the only data points that will need to be reassigned to a server are the ones directly affected by the addition or subtraction. Figure 1 shows how each data point with a name such as John or Steve are assigned to the immediate server plotted on the circle directly counter clockwise to it. This illustrates how the server assigned to the data will stay consistent for all data points not immediately affected by a server change. All nodes not affected will continue operating with their current server. Even though the overhead of scaling is still there because affected data will need to be redistributed to new servers, it is significantly lower than a basic distributed hash database because all of the data will not be affected.

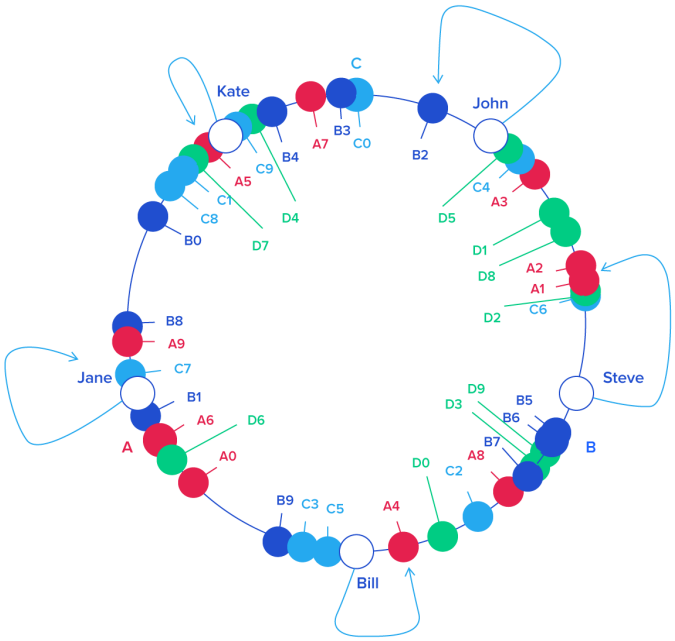


Figure 1 Consistent Hashing

Note. From A Guide to Consistent Hashing, by Carzolio, J. P. (2017, April 25) (https://www.toptal.com/big-data/consistent-hashing).Copyright Toptal.

**Comparison between DynamoDB and relational database system**

There are several characteristics in Relational Database Management System and DynamoDB. In terms of workloads, RDBMS optimally performs ad hoc queries, which are non-standard and not predefined. It is also used for data warehousing and Online Analytical Processing, which can perform complex calculations. DynamoDB, on the other hand, can be used for web-scale applications and the Internet of Things, which are physical devices connected to the internet.

RDBMS requires a schema where data is normalized while DynamoDB contains no schema. All tables contain a primary key but there are no constraints on non-key attributes. In terms of data access, RDBMS uses SQL for storing or retrieving data while DynamoDB uses

AWS Management Console or AWS Command Line Interface (Amazon DynamoDB - Developer Guide, 2012, p. 25).

Although relational databases have high flexibility query syntaxes and can easily handle OLTP and OLAP patterns in a single database, their performance is unpredictable. The reasoning is because there are many factors that impact on query performance. Additionally, when the data grows larger, the query performance drops. The performance of your relational database is subject to a number of factors, table size, index size, concurrent queries that are difficult to test accurately ahead of time and difficult to determine the proper instance size (# of CPUs + RAM) for handling the request (Amazon DynamoDB - Developer Guide, 2012, p. 25).

RDBMS is optimized for storage. Therefore, performance is dependent on how well optimized table structures, queries, and indexes are. DynamoDB is optimized for computation and can be used to build high-performance applications. Scaling up is easiest with faster hardware with RDBMS. Database tables can span across multiple hosts. DynamoDB uses distributed clusters of hardware to scale out, which increases performance without increasing latency. (Debrie, 2020)

RDBMS has problems with horizontal scaling due to difficulties with splitting up the data. SQL queries with RDBMS are unbounded which make the amount of data scanned in a query limitless. This has the potential to consume a large amount of CPU, memory, or disk usage. In addition, RDBMS allows join operations, which consumes data when comparing different tables. This is done by necessitating data reading and writing methods prior to designing the data model. DynamoDB has a resolution to all issues mentioned. It firstly, does not allow join statements. It also makes horizontal scaling much easier by forcing users to segment

their data. In order to avoid limitless potential data consumption, DynamoDB puts bounds on the queries. (Debrie, 2020)

BigTable is an alternative database system for high performance applications. It manages structured data and contains big data scalability. BigTable uses Google File System, which is a highly scalable storage platform, in which files are divided into chunks. While BigTable manages structured data, it does not manage unstructured data like DynamoDB does. However, DynamoDB only supports key-value API. Additionally, DynamoDB uses peer-to-peer storage systems, which use the combined storage of peers as a pool for storage space to share or store content (S. Khalid, 2017, p. 89). On the other hand, BigTable uses hierarchical-namespace, and contains a resource to identify a parent namespace. This establishes ownership across namespaces.

Cassandra is a highly scalable NoSQL database that is much more flexible than DynamoDB in regards to managing wide range data as well as fault handling. In addition, performance is not sacrificed while maintaining its scalability and availability. The primary functions of Cassandra include peer-to-peer systems, a decentralized storage system, and linear scalability. DynamoDB is similar to Cassandra in that both systems handle structured and unstructured data sets as shown in Figure 2. They also use a consistent hashing data partition technique. However, while Cassandra has linear scalability, the data scalability of DynamoDB is incremental (S. Kalid, 2017).

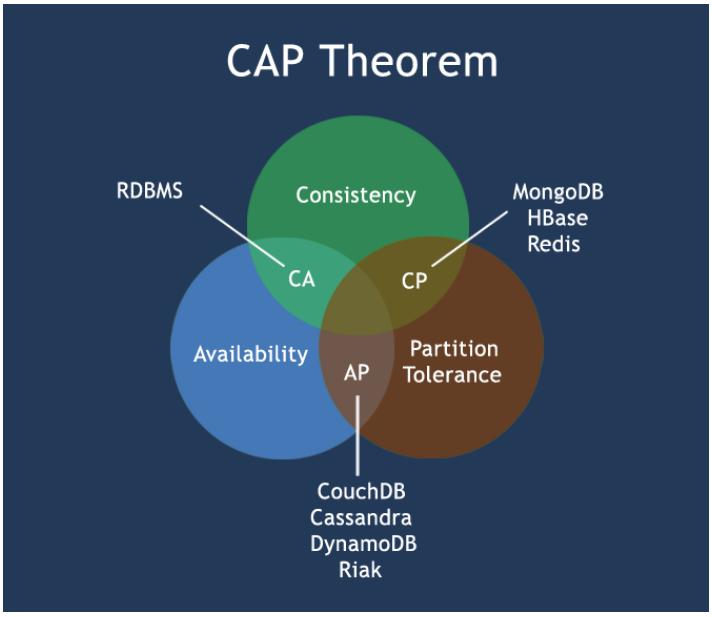


Figure 2 CAP Theorem

Note. From GeeksforGeeks, 2017 (https://practice.geeksforgeeks.org/problems/cap-theorem).



**Advantages and Limitations of DynamoDB**

Now that we've gone over the comparisons between DynamoDB and a regular relational database, it should serve us right to take a more granular look at the advantages and limitations of DynamoDB together:

The predictable and quick performance of DynamoDB is essential to providing a storage platform. DynamoDB is consistent and highly scalable. There is no limit for table size or throughput and it is a live repartition for changes to storage and throughput. In addition, since it is highly scalable, this allows for increased throughput without sacrificing latency (S. Khalsa, 2013).

In order to avoid flexibility, DynamoDB requires users to have preplanned decisions on data writing and reading prior to designing the database. This may require more planning on the

designer's part. Additionally, since DynamoDB does not allow join statements, data must be duplicated. This may pose a problem when data that constantly changes needs to be duplicated, which may require that records be frequently updated. DynamoDB is also less storage efficient. Although this is a setback, it is not a significant issue because computation is a much more valuable resource than storage (S. Khalsa, 2013).

**How to use DynamoDB**

Now that the user knows the advantages and limitations of DynamoDB, let's talk about how a user should be able to utilize the database service. This section follows the basic tutorial in the developer guide to help the user get started with Basic Concepts (Amazon DynamoDB - Developer Guide, 2012. p 72-87). These are steps that a beginner should follow to know more about DynamoDB:

Step 1: Create a Table

When DynamoDB is open, you can click "Create table." After that, users are enabled to fill up all the information required including table name, partition key, sort key and etc.

Step 2: Write Data to a Table Using the Console or AWS CLI

This step is to add the new data in table format to DynamoDB

Step 3: Read Data from a Table

Users can search the data using a navigation pane present on the console.

Step 4: Update Data in a Table

In this step, you can update an item from your table by changing the query drop-down list to scan and check the box the items that you want to remove and choose the update option.

Step 5: Query Data in a Table

In the console, there is an option to select when users want to query the data.

Step 6: Create a Global Secondary Index

Step 7: Query the Global Secondary Index

In step 6 and 7, a global secondary index for the table that had been created in Step 1 can be created and query.

Step 8: (Optional) Clean Up Resources

If the Amazon DynamoDB table is no longer needed, users can feel free to delete it. This step helps ensure that you aren't charged for resources that you aren't using. All of these steps can be performed in both DynamoDB console and the AWS CLI.

Considerably more about features of Dynamo's managed infrastructure, these are example of powerful features provided by DynamoDB (J. H. M. Silva, 2019);

- Automatic data replication to multiple availability zones in a single region.
- Data backed up to S3
- Pay-per-use model
- Security and access control can be done using Amazon IAM Service
- A provisioned-throughput model where read and write units can be adjusted at any time based on actual application usage.
- Integrated with other AWS services like Elastic MapReduce (EMR), Data Pipeline, and Kinesis.

**Use cases of DynamoDB**

Amazon DynamoDB is a tool in the NoSQL Database as a Service category. It's popular among Tech companies due to its high scaling and availability. 802 companies reportedly use Amazon DynamoDB in their tech stacks, including Netflix, Amazon, and Lyft. Two examples of use cases of DynamoDB are Real Time Dashboards and Data Replication.

Let's start off by going over a Real Time Dashboard case. There is a growing demand and desire to use real-time data. The Internet of Things (IoT) is one example that needs to provide instant feedback on the activities that are monitored. The traditional pattern is to build a dashboard application integrated with a NoSQL database. However, different scalable patterns need to be implemented if the data ingested volume per second is high. Amazon Web Services (AWS) is a NoSQL database that can support requirements. It is a platform that users can use the data in for higher scalability and data integration (R. Freeman, 2018).

In Figure 3, the reader observes the workflow of a near-real time dashboard on AWS. Data storage in Amazon Simple Storage Service (S3), Amazon Cognito, AWS Identity, and Access Management (IAM) is used for security.

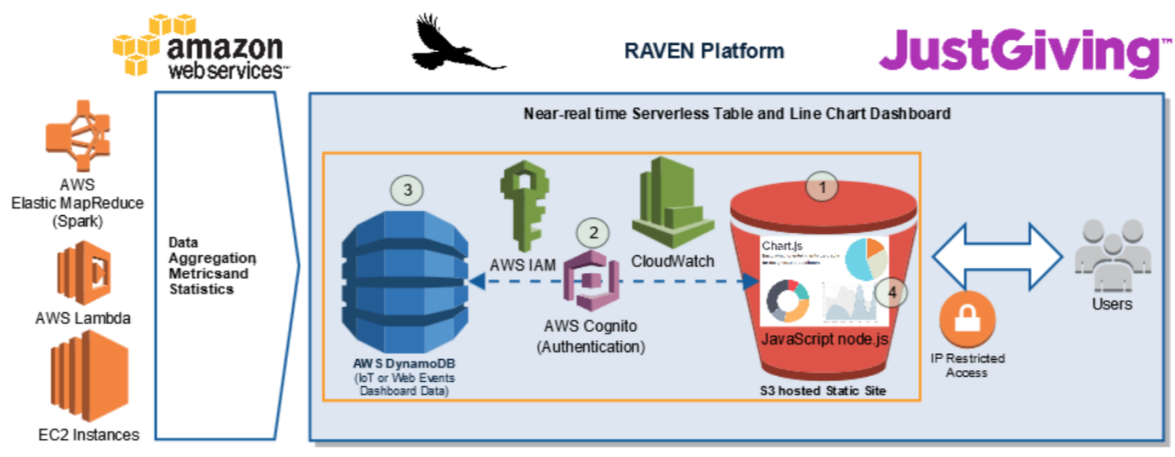


Figure 3 The near-real time dashboard in the RAVEN (Reporting, Analytics, Visualization, Experimental, Networks) platform on Amazon Web Services

Note. From Serverless Dynamic Real-Time Dashboard with AWS DynamoDB, S3 and Cognito, by Richard Freeman, 2018 (https://medium.com/@rfreeman/serverless-dynamic-real-time-dashboard-with-aws-dynamodb-a1a7f8d3bc01). Copyright Medium.

According to the figure, No.1 indicates Website and Application Hosting. S3 deals with the hosting the HTML, JavaScript and node.js code, making it highly resilient, scalable, simple to set-up and secure. No.2 is Security Layer which deals with the authentication and authorisation using Cognito and IAM roles. No.3 is Data Layer which is a persistent store for the data. Finally, No.4 shows the dashboard application is built with JavaScript and node.js.

Now that the reader knows about the Real Time Dashboard case, let's move onto Data Replication. Replicating the data is essential to keep the data safe, high availability, and accessible in the local regions. DynamoDB Global Tables is the easiest way to have your data replicated across several regions. As illustrated by Figure 4, AWS Lambda is a function that will monitor changes in DynamoDB tables, and replicate them across the regions (A. Balchunas, 2018).

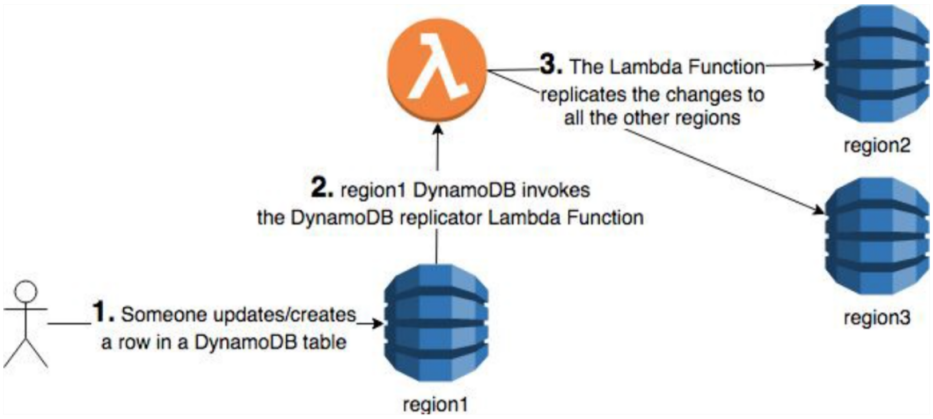


Figure 4 Data Replication using Lambda Functions

Note. From How to Easily Replicate DynamoDB across Regions, by Alexey Balchunas, 2018 (https://medium.com/@AlexeyBalchunas/how-to-easily-replicate-dynamodb-across-regions-fee349b736d7). Copyright Medium.

**Amazon DynamoDB Pricing**

Let's now take a look at the pricing dynamic with the database service. DynamoDB charges for reading, writing, and storing data in DynamoDB tables and extra features depends on user selections. Basically, DynamoDB has two capacity modes; Pricing for on-demand capacity mode and Pricing for provisioned capacity mode. The difference between these two options is that on-demand capacity mode has higher flexibility in workloads. There is no necessity to plan ahead before running the applications. However, the trade off is customers get the higher charge compared to provisioned capacity mode which requires predictable application traffic (Balchunas, 2018).

DynamoDB features and billing overview is shown below in Table 1:

Table 1: DynamoDB features and a billing (D. Rangle, 2015)

| Feature | What it does | Billing unit |
|---|---|---|
| On-demand capacity mode | | |
| Write request | Writes data to your table | Write request unit |
| Read request | Reads data from your table | Read request unit |
| Provisioned capacity mode | | |
| Provisioned write capacity | Writes data to your table | WCU |
| Provisioned read capacity | Reads data from your table | RCU |
| Data storage | Stores data, including index values | GB-month |
| Optional features | | |
| Continuous backup | Takes continuous backups for the preceding 35 days | GB-month |
| On-demand backup | Takes snapshot backups at specified points in time | GB-month |
| Restore from backup | Restores a table to a specific snapshot or time | GB |
| Global tables | Replicates data to create a multi-Region, multi-active table | Replicated write request unit |
| Change data capture for Amazon Kinesis Data Streams | Captures item-level data changes on a table and replicates them to Kinesis Data Streams | Change data capture unit |
| Change data capture for AWS Glue Elastic Views | Captures item-level data changes on a table and replicates them to AWS Glue Elastic Views | Change data capture unit |
| Data export to Amazon S3 | Exports DynamoDB table backups from a specific point in time to Amazon S3 | GB |
| DynamoDB Accelerator (DAX) | Reduces latency through in-memory cache | Node-hour |
| DynamoDB Streams | Provides a time-ordered sequence of item-level changes on a table | Streams read request unit |
| Data transfer out | Transfers data to other AWS Regions | GB |

This Table 1 showcases DynamoDB features and a billing overview of each one. For details of prices, users should check before using the application because the prices are varying for the different regions. The service costs incurred by a DynamoDB client are measured using the amount of data used in an intermediate stage of query execution, which may be much larger than the data in the final result. The different expressions of the same logical query results in different pricing.

For financial and business reasons, the cost might be an issue. Therefore, writing the efficient query would help reduce query costs. This can be illustrated by using an extended example as follows:

Consider a DynamoDB table R(a, b, c, d, e, f) with attributes a through e of number type, attribute f of string type, partition key a, sort key b, and local secondary indexes c_idx, d_idx, and e_idx on attributes c, d, and e, respectively. (S. S. Chawathe, 2019) Consider an application that needs to retrieve items from R that satisfy DQL-like expression shown in Figure 5:

```
select *
from R
where a = 1 and b <= 101 and c >= 1001 and
    (d <= 10001 or e >= 100001)
```

Figure 5 Original Query (S. S. Chawathe, 2019)

Note: Reprinted from "Cost-Based Query-Rewriting for DynamoDB" by S. S. Chawathe, 2019, *Journal Title, IEEE 18th International Symposium on Network Computing and Applications (NCA), p. 1-3, doi: 10.1109/NCA.2019.8935057*. Reprinted IEEE.

Q1: Original Query scan all items to retrieve the data

Q2: Query without predicates other than a=1 which retrieving all items in the partition

Q3: Query using table -- i.e. index on b

Q4: Query using c_idx

Suppose that R holds $10^7$ items and, for simplicity, that all items have the same size of 100 KiB. Further, suppose $10^6$ items satisfy the predicate a = 1 and that, of these, those further satisfying the predicates b <= 101, c >= 1001, and their conjunction are, respectively, $10^5$, $10^4$, and $10^3$ in number. Finally, suppose only 10 items satisfy d <= 10001 and only 1 item satisfies e >= 100001, and all 11 of these items have a = 1 as well. The costs of the rewritings Q1, Q2, Q3, and Q4 are then, respectively, $10^7$, $10^6$, $10^5$, $10^4$ items. (S. S. Chawathe, 2019)

However the codes can be rewritten in more effective ways. The code as shown in Figure 6 is significantly lower cost which the cost of this rewriting is only 10 + 1 = 11 items.

```
(select * from R where a = 1 and b <= 101
                  and c >= 1001 and d <= 10001
    using d_idx)
union
(select * from R where a = 1 and b <= 101
                  and c >= 1001 and e >= 100001
    using e_idx)
```

Figure 6 Rewritten code

Note: Reprinted from “Cost-Based Query-Rewriting for DynamoDB” by S. S. Chawathe, 2019, *Journal Title, IEEE 18th International Symposium on Network Computing and Applications (NCA), p. 1-3, doi: 10.1109/NCA.2019.8935057*. Reprinted IEEE.

## Conclusion

To conclude, Amazon developed DynamoDB to be fast, scalable, and reliable to adapt to the changes and advancement in database management systems. It was built out of necessity, so that Amazon could handle the massive amount of online orders that its marketplace received. However, since its inception it has been one of the most prominent NoSQL distributed systems. It provides many benefits to the user including management console and APIs which allows the user to manipulate the data. It protects sensitive data by offering strong and reliable encryption. Additionally, it uses eventual consistency to allow the system to achieve constant availability of the data and network partition tolerance. It also uses cryptographic methods to prevent unauthorized access. With its many advantages, DynamoDB has some limitations in comparison to relational database management systems. It requires database architects to have a predefined method of writing and reading data prior to designing the database. Records may also need to be updated frequently if the data constantly changes. It has less storage efficiency due to its focus on computation.

## References


Amazon DynamoDB Developer Guide. (2012, August 12). Amazon Web Services. http://aws.amazon.com/dynamodb/#:~:text=Amazon%20DynamoDB%20is%20a%20key,caching%20for%20internet-scale%20applications.

Balchunas, A. (2018, October 13). How to Easily Replicate DynamoDB across Regions. Medium. https://medium.com/@AlexeyBalchunas/how-to-easily-replicate-dynamodb-across-regions-fee349b736d7

Bulao, J. (2021, April 28). How Much Data Is Created Every Day in 2021? TechJury. https://techjury.net/blog/how-much-data-is-created-every-day/#gref

Carzolio, J. P. (2017, April 25). A Guide to Consistent Hashing. Toptal Engineering Blog. https://www.toptal.com/big-data/consistent-hashing

Debrie, A. D. (2020). *SQL, NoSQL, and Scale: How DynamoDB scales where relational databases don't*. Alex Debrie. https://www.alexdebrie.com/posts/dynamodb-no-bad-queries/

Education, I. C. (2021, March 23). CAP Theorem. IBM.Com. https://www.ibm.com/cloud/learn/cap-theorem

FanFight Case Study – Amazon Web Services. (2020). Amazon Web Services, Inc. https://aws.amazon.com/solutions/case-studies/fanfight-dynamodb-case-study/

Harrison, G. (2015). Next Generation Databases. Apress.

Introduction to Amazon DynamoDB (1:01). (n.d.). Amazon Web Services, Inc. Retrieved May 4, 2021, from https://aws.amazon.com/dynamodb/#:%7E:text=Amazon%20DynamoDB%20is%20a%20key,caching%20for%20internet%2Dscale%20applications.

Kumar, P. (2020, April 24). Amazon DynamoDB Tutorial – A Complete Guide. Edureka. https://www.edureka.co/blog/amazon-dynamodb-tutorial

Niranjanamurthy, M., Archana, U. L., Niveditha, K. T., Abdul Jafar, S., & Shravan, N. S. (2014). The Research Study on DynamoDB – NoSQL Database Service. International Journal of Computer Science and Mobile Computing, 3(10), 268–279. Title: The Research Study on DynamoDB – NoSQL Database Service (1library.net)

Rangel, D. (2015). DynamoDB: everything you need to know about Amazon Web Service's NoSQL database. Amazon.

Amazon DynamoDB Pricing | NoSQL Key-Value Database | Amazon Web Services

Richard Freeman, P. D. (2018, June 2). Serverless Dynamic Real-Time Dashboard with AWS DynamoDB, S3 and Cognito. Medium. https://medium.com/@rfreeman/serverless-dynamic-real-time-dashboard-with-aws-dynamodb-a1a7f8d3bc01.

S. Kalid, A. Syed, A. Mohammad and M. N. Halgamuge, "Big-data NoSQL databases: A comparison and analysis of “Big-Table”, “DynamoDB”, and “Cassandra”," 2017 IEEE 2nd

International Conference on Big Data Analysis (ICBDA), 2017, pp. 89-93, doi: 10.1109/ICBDA.2017.8078782

Saniya Khalsa Business analyst Follow. (n.d.). Dynamo db pros and cons. SlideShare. https://www.slideshare.net/saniyakhalsa/dynamo-db-pros-and-cons.