Available Online at www.ijcsmc.com

International Journal of Computer Science and Mobile Computing

A Monthly Journal of Computer Science and Information Technology

ISSN 2320–088X

IJCSMC, Vol. 3, Issue. 10, October 2014, pg.268 – 279

RESEARCH ARTICLE

The Research Study on DynamoDB – NoSQL Database Service

Niranjanamurthy M[1], Archana U L[2], Niveditha K T[3], Abdul Jafar S[4], Shravan N S[5]

[1]Assistant Professor, Department of MCA, MSRIT, Bangalore-54, INDIA

[2]Student, Department of MCA, MSRIT, Bangalore-54, INDIA

[3]Student, Department of MCA, MSRIT, Bangalore-54, INDIA

[4]Student, Department of MCA, MSRIT, Bangalore-54, INDIA

[5]Student, Department of MCA, MSRIT, Bangalore-54, INDIA

[1] niruhsd@gmail.com; [2] archanalonakadi@gmail.com; [3] niveditha.kt@gmail.com; [4] jabdul63@gmail.com; [5] shravan4000@gmail.com

Abstract -- Reliability at massive scale is one of the biggest challenges we face as the operations in the e-commerce world keeps on expanding; even the slightest outage has significant financial consequences and impacts customer trust. The amazon.com platform provides services for many web sites worldwide. It is implemented on top of an infrastructure of tens of thousands of servers and network components located in many datacenters around the world. At this scale, small and large components fail continuously and the persistent state is managed in the face of these failures. This drives the reliability and scalability of the software systems. DynamoDB is a fast, fully managed NoSQL database service that makes it simple and cost-effective to store and retrieve any amount of data, and serve any level of request traffic. It has guaranteed throughput and low latency which makes it a great fit for gaming, ad technology, mobile and many other applications. This paper presents the design and implementation of DynamoDB, a highly available key-value *storage system that some of amazon's core services use to provide an "always-on" experience. To achieve this* level of availability, DynamoDB sacrifices consistency under certain failure scenarios. It makes extensive use of object versioning and application-assisted conflict resolution in a manner that provides a novel interface for developers to use. We also compare and contrast the DynamoDB services with other leading database services available for same purposes.

Keywords— DynamoDB, Reliability, Amazon Dynamo DB, Features, Challenges faced by DRAMAFEVER, MongoDB

I. INTRODUCTION

DynamoDB is a fully managed NoSQL database service that provides fast and predictable performance with seamless scalability. If you are a developer, you can use DynamoDB to create a database table that can store and retrieve any amount of data, and serve any level of request traffic. DynamoDB automatically spreads the data and traffic for the table over a sufficient number of servers to handle the request capacity specified by the customer and the amount of data stored, while maintaining consistent and fast performance. All data items are stored on solid state disks (SSDs) and are automatically replicated across multiple Availability Zones in a Region to provide built-in high availability and data durability.

If you are a database administrator, you can create a new DynamoDB database table, scale up or down your request capacity for the table without downtime or performance degradation,

and gain visibility into resource utilization and performance metrics, all through the AWS Management Console. With DynamoDB, you can offload the administrative burdens of operating and scaling distributed databases to AWS, so you don't have to worry about hardware provisioning, setup and configuration, replication, software patching, or cluster scaling.

This paper describes Dynamo, a highly available data storage technology that addresses the needs of these important classes of services. Dynamo has a simple key/value interface, is highly available with a clearly defined consistency window, is efficient in its resource usage, and has a simple scale out scheme to address growth in data set size or request rates. Each service that uses Dynamo runs its own Dynamo instances. I am also suggesting the readers with the pros and cons of the use of this database security as compared to the HBase (Apache Proprietary) and MongoDB (Open source) and Cassandra (Apache software foundation Proprietary).

II. AIM OF THE STUDY

- Exploring DynamoDB and understanding its feature
- Understanding the challenges faced by current DynamoDB users.
- Comparisons with other Database providing similar services
- Choosing perfect database for our application considering all the pros and cons of each services.

III.RELATED WORK

A.CAP continuum

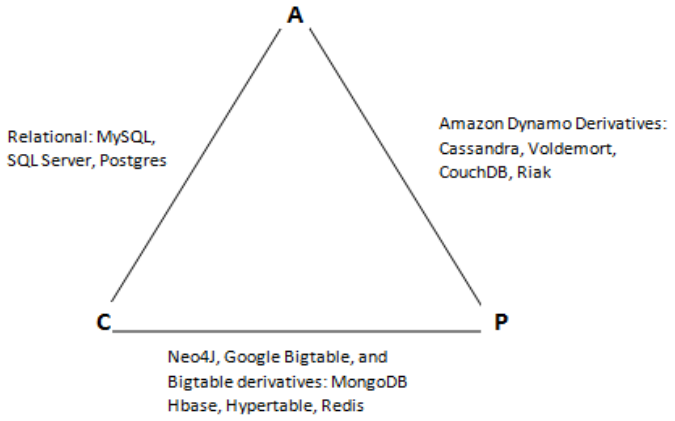

Fig.1 CAP Continuum

All database developers know ACID acronym. It says that database transactions should be:

Atomic: Everything in a transaction succeeds or the entire transaction is rolled back.
Consistent: A transaction cannot leave the database in an inconsistent state.
Isolated: Transactions cannot interfere with each other.
Durable: Completed transactions persist, even when servers restart etc.
These qualities are indispensable, and yet they are incompatible with availability and performance in very large systems.

Eric Brewer's CAP theorem says that if you want consistency, availability, and partition tolerance, you have to settle for two out of three. (For a distributed system, partition tolerance means the system will continue to work unless there is a total network failure. A few nodes can fail and the system keeps going.)

An alternative to ACID is BASE:

- Basic Availability
- Soft-state
- Eventual consistency

Rather than requiring consistency after every transaction, it is enough for the database to eventually be in a consistent state. (Accounting systems do this all the time. It's called "closing out the books.") It's OK to use stale data, and it's OK to give approximate answers.

It's harder to develop software in the fault-tolerant BASE world compared to the fastidious ACID world, but Brewer's CAP theorem says you have no choice if you want to scale up. However, as Brewer points out, there is a continuum between ACID and BASE. You can decide how close you want to be to one end of the continuum or the other according to your priorities.

B. Features of DynamoDB:

Provisioned Throughput:
When creating a table, simply specify how much throughput capacity you require. DynamoDB allocates dedicated resources to your table to meet your performance requirements and automatically partitions data over a sufficient number of servers to meet your request capacity. If your application requirements change, simply update your table throughput capacity using the AWS Management Console or the DynamoDB APIs. You are still able to achieve in your prior throughput levels while scaling is underway.

Automated Storage Scaling:
There is no limit to the amount of data you can store in a DynamoDB table, and the service automatically allocates more storage, as you store more data using the DynamoDB write APIs.
Fully Distributed, Shared Nothing Architecture:
DynamoDB scales horizontally and seamlessly scales a single table over hundreds of servers. DynamoDB is designed for seamless throughput and storage scaling.

Easy Administration:
DynamoDB is a fully managed service – you simply create a database table and let the service handle the rest. You don't need to worry about hardware or software provisioning, setup and configuration, software patching, operating a reliable, distributed database cluster, or partitioning data over multiple instances as you scale.

Efficient Indexing:
Every item in a DynamoDB table is identified by a primary key, allowing you to access data items quickly and efficiently. You can also define secondary indexes on non-key attributes, and query your data using an alternate key.

Strong Consistency, Atomic Counters:
Unlike many non-relational databases, DynamoDB makes development easier by allowing you to use strong consistency on reads to ensure you are always reading the latest values. DynamoDB supports multiple native data types (numbers, strings, binaries, and multi-valued

attributes). The service also natively supports atomic counters, allowing you to atomically increment or decrement numerical attributes with a single API call.

Cost Effective:
DynamoDB is designed to be extremely cost-efficient for workloads of any scale. You can get started with a free tier that allows more than 40 million database operations per month, and pay low hourly rates only for the resources you consume above that limit. With easy administration and efficient request pricing, DynamoDB can offer significantly lower total cost of ownership (TCO) for your workload compared to operating a relational or non-relational database on your own.

Amazon Elastic MapReduce Integration:
DynamoDB also integrates with Amazon Elastic MapReduce (Amazon EMR). Amazon EMR allows businesses to perform complex analytics of their large datasets using a hosted pay-as-you-go Hadoop framework on AWS. With the launch of DynamoDB, it is easy for customers to use Amazon EMR to analyze datasets stored in DynamoDB and archive the results in Amazon Simple Storage Service (Amazon S3), while keeping the original dataset in DynamoDB intact. Businesses can also use Amazon EMR to access data in multiple stores (i.e. DynamoDB and Amazon RDS), perform complex analysis over this combined dataset, and store the results of this work

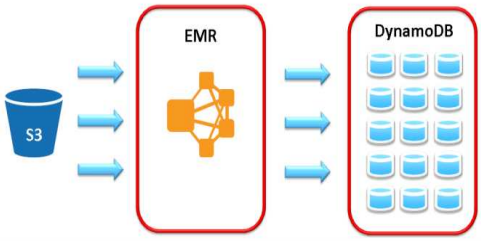

Fig2: Amazon Elastic MapReduce Integration

IV. SOME OF THE CASE STUDIES ANALYZED

Washington Post implemented DynamoDB to power their App(Social Reader app) and their website(socialreader.com),

Werner Hans Peter Vogels is the chief technology officer and Vice President of Amazon.com on 17 April 2013 says:

"Our customers have been asking us to expand the query capabilities of DynamoDB and we're excited to see how they use LSI."

Milo Milovanovic, Washington Post Principal Systems Architect reports that "database performance and scalability are critical for delivering new services to our 34+ million readers on any device. For this reason, we chose DynamoDB to power our popular Social Reader app and site experience on socialreader.com. The fast and flexible query performance that local secondary indexes provide will allow us to further optimize our social intelligence, and continue to improve our readers' experiences."

AdRoll Retargetting Platform using DynamoDB:

Valentino Volonghi, Chief Architect of retargeting platform AdRoll, says "we use DynamoDB to bid on more than 7 billion impressions per day on the Web and FBX. AdRoll's bidding system accesses more than a billion cookie profiles stored in DynamoDB,

and sees uniform low-latency response. In addition, the availability of DynamoDB in all AWS regions allows our lean team to meet the rigorous low latency demands of real-time bidding in countries across the world without having to worry about infrastructure management."

Scopely social gaming platform uses DynamoDB:

Ankur Bulsara, CTO of the Scopely social gaming platform, says "LSI will enable his team to deploy DynamoDB even more broadly. We default to DynamoDB wherever we can, and also use MySQL for some query types," he says. "We're very excited that local secondary indexes will allow us to further remove traditional RDMSes from our ever-growing stack. DynamoDB is the future, and with LSI, the future is very bright."

V. DAILYCRED FACING CHALLANGES USING DYNAMODB

DailyCred is a CRM and backoffice tool:

At DailyCred they use DynamoDB as a data store, and they find these prospects highly useful:

- Scalability and simplicity of NoSQL
- Consistent performance
- Low learning curve
- It comes with a decent object mapping facility for Java Language.

It's a great service considering how new it is, but it definitely still has some rough edges that make some things harder than they expected.

Fig4:DailyCred is a CRM and backoffice tool

DynamoDB is not ideal for storing events:

Like most websites, they store a variety of user events. A typical event has an event ID, a user ID, an event type and other attributes that describe actions performed by users. At DailyCred, they needed an event storage that is optimized for reads. For the dashboard they need to quickly filter events by type, sort events by time and group events by user. However, they don't need to record events as soon as they happen. A second of delay is fine.

Using a relational database, we can store events in a de-normalized table. Add indices to columns that answer query predicates. In this setup, writes are not very fast, but reads are extremely fast. We can use the richness of SQL to query events easily.

A big limitation of DynamoDB and other non-relational database is the lack of multiple indices. In an events table, we could use the event ID as the hash key, the event time as the range key. This schema would enable us to retrieve most recent events, but we can't filter event by type without doing a full table scan. Scans are expensive in DynamoDB. You could store events in many tables, partitioned by event type or by user ID. Perhaps for each predicate, create an index table with the predicate's value as the key and the event ID as an attribute. Is the added complexity worth it? Probably not. DynamoDB is great for lookups by key, not so good for queries, and abysmal for queries with multiple predicates.SQL was a better tool in this case, so we decided not to use DynamoDB at all for storing events.

DynamoDB overhead (compared to SQL):
DynamoDB supports transactions, but not in the traditional SQL sense. Each write operation is atomic to an item. A write operation either successfully updates all of the item's attributes or none of its attributes. There are no multi-operation transactions. For example, you have two tables, one to store orders and one to store the user-to-order mapping. When a new order comes in, you write to the order table first, then the mapping table. If the second write fails due to network outage, you are left with an orphaned item. Your application has to recognize orphaned data. Periodically, you will want to run a script to garbage collect those data, which in turn involve a full table scan. The complexity doesn't end here. Your script might need to increase the read limit temporarily. It has to wait long enough between rounds of scan to stay under the limit.

One strike, and you are out:
While DynamoDB's provisioned throughput lets you fine tune the performance of individual tables, it doesn't degrade gracefully. Once you hit the read or write limit, your requests are denied until enough time has elapsed. In a perfect world, your auto-scaling script will adjust throughput based on anticipated traffic, increasing and decreasing limits as necessary, but unexpected traffic spikes is a fact of life. Say you push up the limits as soon as DynamoDB throws a ProvisionedThroughputExceededException, the process could take a minute to complete. Until then, you are at the mercy of retries, a feature that is thankfully enabled by default by the official SDK.
At DailyCred, it's hard to accurately anticipate resource usage, due to the fact that they do not know when their clients are hit with a wave of visitors. What if the wave happens to multiple clients at once?

Backups: slow or expensive (pick one):
Another annoyance with provisioned throughput is that you can only decrease your provisioned limit once a day. Say your daily backup script temporarily increases the read limits for the duration of the job. After the limits were reduced, your site gets a burst of traffic. After increasing the limits for the second time, you are stuck with the new limits for up to a day. In that regard, DynamoDB isn't as elastic as other AWS services, where resources can be deallocated with no daily limits.

Unit tests: slow or expensive (pick one):
They also run a lot of tests that use DynamoDB, which means a lot of items are written and read very quickly. They run the tests several times a day, which means their development database tables are sitting completely idle most of the time, only to be hammered with reads and writes when we run our unit tests. From a cost perspective, this isn't ideal. However, it's even worse if a developer has to wait extra time for his unit tests to complete.

VI. CHALLENGES FACED BY DRAMAFEVER

DramaFever is a website for online destination for the Asian serials and dramas. They offer HQ videos with English subtitles.

The Firehose:
They started running into real trouble when they wanted to use DynamoDB for time series data. I'm not saying DynamoDB is unsuited for time series data, but the gotchas

(programming mistakes) start to multiply rapidly. Their use case here was their analytics "firehose"; a ping that each of their video player clients sends back every minute containing a bunch of data they need for metrics, revenue-sharing data, etc. In other words, business critical and high volume. Originally all this data was going into a giant append-only MySQL table, but with something like 70% of all their requests resulting in writes to this table the performance was getting terrible as they scaled-up. They could have sharded the MySQL database, of course. But an eventually-consistent lock-free database that supported widely-parallel writes seemed like an ideal use case for DynamoDB for them.

But if you want a whole month's worth of data for, say, cutting your monthly revenue-sharing checks to content licensors, you're going to run an EMR job. And there are two additional problems. The first is that the time it takes to run that EMR job will increase over time as the number of properties grows.

You still end up having to do a scan with EMR, but only over the data from a given day. Then you can aggregate data for series and episodes based on the attributes. You'll also need to roll-up tables as you go to reduce the amount of processing. This is a bad hack, because you end up with write throughput that looks like this:

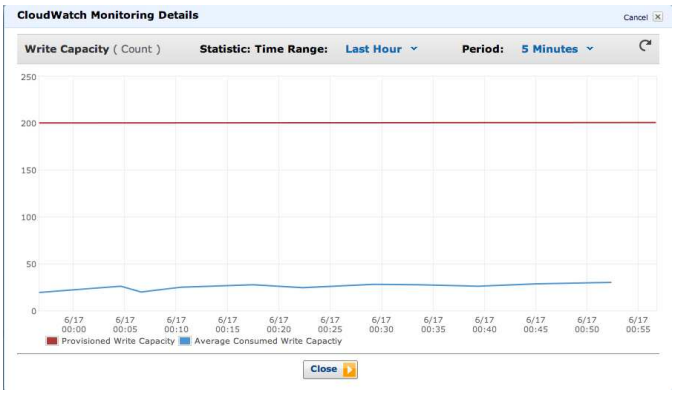

Fig.5 Throughput Graph

We see that the throughput is a tenth of what they provisioned. This is where the abstraction of the managed service starts to leak. When you provision throughput, Amazon is spinning up n instances of whatever the underlying processes are. Your provisioned throughput is distributed among these instances.

Because the throughput is divided by the number of instances, you end up with not just reduce throughput when you're writing to a single hash but diminishing returns on the throughput you provision. This was also the second problem with using series/episode as a hash key. There is plenty of key space given the size of their library, but too many hot writes because the latest episodes tend to be the most popular.

Another thing to keep in mind here is that writes are at least 5x as expensive as reads. A unit of read capacity gets you 1 consistent read up to 4KB (or 2 eventually consistent reads), whereas the write unit is for 1KB writes. This doesn't include secondary indexes, which add another increment in cost for each index. So writes can be significantly more expensive.

Key design impacts throughput for both reads and writes, and the size of rows and number of secondary indexes impacts the ratio of writes vs reads. And because provisioning for DynamoDB is pay-as-you-go, this means there is a direct relationship between key schema design and operational cost.

Time Series Data:

Design makes the writes simple and reduces the cost of doing ingest, but adds operational complications. In order to reduce the time it takes to do post-processing, you're going to want to roll-off data that you've processed by rotating tables. For them this meant doing a monthly rotation of tables, but the time it took to do a month's worth of data was impractically long and we wanted to eventually be able to shrink the processing window down so that their management team could use this for actionable BI (i.e. no more than 24 hours old).

You are much better off using a secondary index on an attribute which is a timestamp. Your row-writes will double in cost, but it'll be worth the greatly reduced complication and cost of your post-processing in EMR.

Throttling:

One of the other problems they ran into with DynamoDB is dealing with throttling. Estimating the required provisioned throughput was pretty easy, but the load is also spiky. Their content team might post a new episode of a popular series and then announce it via social channels in the early evening, for example, and this will result in a sharp increase in new streams as those notifications arrive.

Semi-Homemade Autoscaling:

Amazon doesn't provide an autoscaling API for DynamoDB. The API for provisioning has a couple of important quirks. You can only increase the provisioning by up to +100% per API call, and another API request to increase will fail until the provisioning change has been completed (presumably this is because Amazon is spinning up DynamoDB instances). You can decrease the provisioning down to 1 read/write unit with a single call, but you are allowed only 2 decreases in provisioning per table per day. They have a large daily swing in load because "prime time TV" still exists on the web if you have a predominantly North American audience. Because this is a predictable swing in load, they have a cron job that fires off increases and decreases in provisioning. The job fires every 15 minutes. Starting in the early AM it checks if the current throughput is within 80% of provisioned throughput and if so steps up in 20% increments over the course of the day.

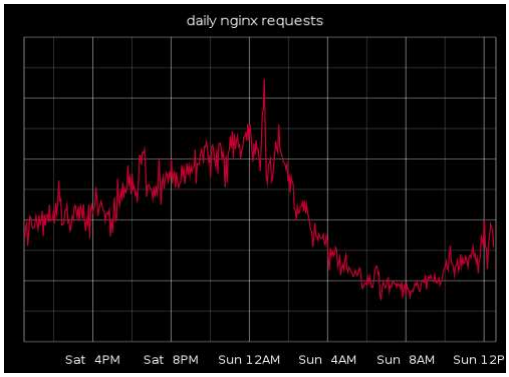

Fig 6:Semi-Homemade Autoscaling

VII. COMPARISONS WITH SOME IMPORTANT TOOLS

HBase: Apache HBase (HBase) is the Hadoop database. It is a distributed, scalable, big data store. HBase is a sub-project of the Apache Hadoop project and is used to provide real-time read and write access to your big data. According to The Apache Software Foundation, the primary objective of Apache HBase is the hosting of very large tables (billions of rows X

millions of columns) atop clusters of commodity hardware.

HBase: What use cases make sense for DynamoDB v/s what make sense for HBase on EMR? Why would you choose one over the other?
DynamoDB vs. HBase on EMR:

1. Cost: Cost or pricing model is different for both offerings. Pricing of DynamoDB is based on write/read throughput and HBase on EMR is a typical of EC2 instance cost.
2. Operational management: DynamoDB is completely managed by AWS whereas HBase on EMR is also managed by AWS, but you still have some control on configuration.
3. Use case: DynamoDB seems to be a good choice for OLTP and as primary real-time store, then you run EMR on top of the data for analytics. HBase on EMR fits well only for OLAP purposes only. You can use HBase on EMR as an analytical service or for backup purposes.
4. Availability: DynamoDB is available across multiple zones whereas HBase on EMR runs in a single availability zone.
5. Durability: DynamoDB is durable whereas HBase on EMR does not guarantee durability.
6. Hardware: DynamoDB runs on SSD whereas HBase on EMR doesn't.

MongoDB: MongoDB (from "humongous") is a cross-platform document-oriented database system. Classified as a NoSQL database, MongoDB eschews the traditional table-based relational database structure in favor of JSON-like documents with dynamic schemas (MongoDB calls the format BSON), making the integration of data in certain types of applications easier and faster. Released under a combination of the GNU Affero General Public License and the Apache License, MongoDB is free and open-source software.

VIII. POINTS TO REMEMBER BEFORE CHOOSING ANY DATABASE SERVICES

Use MongoDB if your indexing fields might be altered later.
With DynamoDB, it's NOT possible to alter indexing after being created. I have to admit that there are workarounds but none of them are straight forward.

Use MongoDB if your need features of document database as your NoSQL solution.
However, DynamoDB is key-value database, and support only value or set, no sub-document supported, and no complex index and query supported. It's not possible to save sub-document {first: 'John', last: 'Backus'} to name, accordingly, not possible to query by name.first.

Use MongoDB if you are going to use Perl, Erlang, or C++.
Official AWS SDK support Java, JavaScript, Ruby, PHP, Python, and .NET, while MongoDB supports more.

Use MongoDB if you may exceed the limits of DynamoDB

Please be careful about the limits and read them carefully if you are evaluating DynamoDB. You may easy to exceed some of the limits. For example, the value you stored in an item(value of a key) cannot exceed 64k bytes. It's easy to exceed 64k bytes when you allow user to input content. User may input a 100k bytes text as article title just because of pasting it by mistake.

Use MongoDB if you are going to have data type other than string, number, and base 64 encoded binary.
In addition to string, number, binary, and array, MongoDB supports date, boolean, and a MongoDB specified type "Object ID".. Date and Boolean are quite important types. With DynamoDB you can use number as alternative, but still, need additional logic in your code to handle them. With MongoDB you can get all these data types by nature.

Use MongoDB if you are going to query by regular expression.
RegEx query might be an edge case, but in case this happens in your situation. DynamoDB provided a way to query by checking if a string or binary start with some substring, and provided the "CONTAINS" and "NOT_CONTAINS" filter when you do "scan". But you know "scan" is quite slow.

Use MongoDB if you are a big funs of document database.
10gen is the company backing MongoDB and also gave some very good suggestions on MongoDB.

Use DynamoDB if you are NOT going to have an employee to manage the database servers.
This is the top 1 reason why people migrate from MongoDB to DynamoDB.

Use DynamoDB if you didn't have budget for dedicated database servers.
Upgrading database servers will take more cost, and may still not be able to resolve the issue. There are some managed MongoDB services, but we may not be able to stand for the cost.

Use DynamoDB if you are going to integrate with other Amazon Web Services.
Amazon CloudSearch is a solution for DynamoDB full text index. Another example could be AWS Elastic MapReduce, it can be integrated with DynamoDB very easily. Also for database backup and restore, Amazon has other services to integrate with DynamoDB.

Pay for use feature of DynamoDB
Amazon lets you buy operations per second capability rather than CPU hours or storage space. This removes a whole lot of complexity for developers who would otherwise need to tune the database configuration, monitor performance levels, ramp up hardware resources when needed. This provides users a fast and reliable storage space for their needs with costs that scale in direct proportion to the demand.

Use HBase when high read and write performance is required
HBase is useful when fast, record-level queries and updates are required, but storage in HDFS is desired for use with Pig, Hive, or other MapReduce-based tools.

Use Hbase when
You are loading data by key, searching data by key (or range), serving data by key, querying data by key or when storing data by row that doesn’t conform well to a schema.

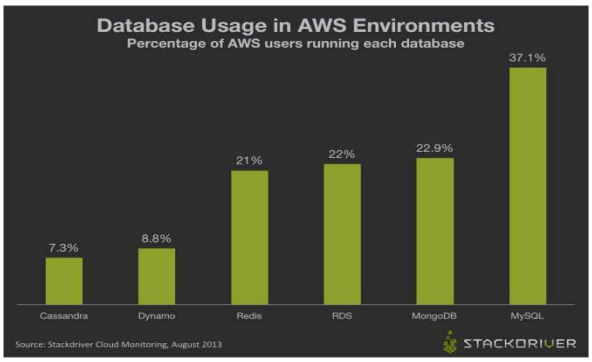

Fig10: Comparison of the databases

IX. CONCLUSION

This paper described DynamoDB, a highly available and scalable data store, used for storing state of a number of core services of Amazon.com's e-commerce platform. DynamoDB has provided the desired levels of availability and performance and has been successful in handling server failures, data center failures and network partitions. Dynamo is incrementally scalable and allows service owners to scale up and down based on their current request load. Dynamo allows service owners to customize their storage system to meet their desired performance, durability and consistency SLAs. There are two important test cases which show the challenges faced by the clients. There are success stories a swell. The paper also speaks about other trending NoSql database services like MongoDB and HBase. It compares and contrasts with them with DynamoDB and also provides the reader a better understanding regarding opting a service based on their use-case.

ACKNOWLEDGEMENT

I thank Dr. T. V. Suresh Kumar, Prof. and Head, Dept. of MCA, MSRIT, Bangalore-54. for his continuous support and encouragement for completing this research paper and also thanks to MSRIT management.

I thank Dr. Jagannatha, Associate Professor. of Dept. of MCA, MSRIT, Bangalore-54, for his valuable guidance and support for completing this paper.

REFERENCES

[1] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani - "Dynamo: Amazon's Highly Available Key-value Store "SOSP '07 Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles held on:Oct 14, 2007
[2] http://aws.amazon.com/dynamodb/
[3] http://www.paperplanes.de/2012/1/30/a-tour-of-amazons-dynamodb.html
[4] http://www.quora.com/What-are-differences-between-MongoDB-and-Amazon-DynamoDB
[5] http://developers.slashdot.org/story/13/02/22/2343212/a-tale-of-two-databases-revisited-dynamodb-and-mongodb
[6] http://blog.cloudera.com/blog/2011/04/hbase-dos-and-donts/
[7]]http://en.wikipedia.org/wiki/Apache_HBase
[8] http://www.qatar.cmu.edu/~msakr/15619-s14/lectures/Recitation11.pdf
[9] http://www.read.seas.harvard.edu/~kohler/class/cs239-w08/decandia07dynamo.pdf
[10] Ramez Elmasri, S. B. Navathe. D VLN Somayajulu, S.K Gupta, Fundamentals of Database Systems, Pearson Education 2006
[11] Abdulla, H.I.Amer, A.A. Chunk Fragmentation Level: An Effective Indicator for Read Performance Degradation in Reduplications Storage IEEE International Conference on Information Technology and e-Services (ICITeS),Page(s):1 - 7 Print ISBN:978-1-4673-1167-0, Date of Conference: 24-26 March 2012
[12] Resource Allocation and Scheduling in the Cloud

[13] Ms. Shubhangi D. Patil, Dr. S. C. Mehrotra. International Journal of Emerging Trends & Technology in Computer Science (IJETTCS),ISSN 2278-6856. Volume 1, Issue 1, May-June 2012

[14] M Mrunalini, T V Suresh Kumar, K RajaniKanth "Consistency Checking of Fragmentation before and after Database Integration" International Conference on Recent Applications of Soft Computing in Engineering & Technology", in Technical Sponsorship of IEEE Delhi Section, 22-23 December 2007.

[15] Jau-JiShena and Tzung-Liang Hungb." Minimizing Transmission Cost for Distributed Database Design" Proceedings of the17th International Conference on Advanced Information Networking and Applications (AINA'03) 0-7695-1906-7/03 Date of Conference: 27-29 March 2003

