The Comparative study of DynamoDB - NoSQL database service

Niranjanamurthy M
Research Scholar Dept. of Computer Science Engg, JJTU, RAJASTAN
niruhsd@gmail.com

Archana U L
Student of MCA, MSRIT, BANGALORE
archanalonakadi@gmail.com

Niveditha K T
Student of MCA, MSRIT, BANGALORE
niveditha.kt@gmail.com

Jagannatha S
Associate Professor Dept. of MCA, MSRIT, BANGALORE
jagannathast@gmail.com

Abstract **-- 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. Its guaranteed throughput and single-digit millisecond latency make it a great fit for gaming, ad tech, mobile and many other applications. 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, which provides services for many web sites worldwide, 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 way persistent state is managed in the face of these failures drives the reliability and scalability of the software systems. This paper presents the design and implementation of Dynamo, 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, Dynamo 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. Compare and contrast the DynamoDB services with other leading database services available for same purposes.**

Keywords: DynamoDB, Reliability

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. 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 pro's and con's 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

The aim of this paper:

- Exploring DynamoDB and understanding its feature
- Understanding the challenges faced by current Dynamo db 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 WORKS

The architecture of a storage system that needs to operate in a production setting is complex. In addition to the actual data persistence component, the system needs to have scalable and robust solutions for load balancing, membership and failure detection, failure recovery, replica synchronization, overload

handling, state transfer, concurrency and job scheduling, request marshalling, request routing, system monitoring and alarming, and configuration management. Describing the details of each of the solutions is not possible, so this paper focuses on the core distributed systems techniques used in Dynamo: partitioning, replication, versioning, membership, failure handling and scaling.

Table1: Summary of techniques used in Dynamo and their advantages.

Problem	Technique	Advantage
Partitioning	Consistent Hashing	Incremental Scalability
High Availability for writes	Vector clocks with reconciliation during reads	Version size is decoupled from update rates.
Handling temporary failures	Sloppy Quorum and hinted handoff	Provides high availability and durability guarantee when some of the replicas are not available.
Recovering from permanent failures	Anti-entropy using Merkle trees	Synchronizes divergent replicas in the background.
Membership and failure detection	Gossip-based membership protocol and failure detection.	Preserves symmetry and avoids having a centralized registry for storing membership and node liveness information.

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. Its guaranteed throughput and single-digit millisecond latency make it a great fit for gaming, ad tech, mobile and many other applications.

DynamoDB is a distributed database in the cloud, no surprises, it's not the first such thing Amazon has in its portfolio. S3, SimpleDB, RDS and DynamoDB all provide redundant ways to store different types of data in Amazon's data centres. Currently, DynamoDB only supports the US-east region.

Let's create a table first, but make sure you wear protection, as this is yet another of Amazon's gross-looking APIs. The relevant API action is

CreateTable:

```
dynamo = Fog::AWS::DynamoDB.new(aws_access_key_id:
"YOUR KEY",
aws_secret_access_key: "YOUR SECRET")

dynamo.create_table("people",
  {HashKeyElement: {AttributeName: "username",
AttributeType: "S"}},
  {ReadCapacityUnits: 5, WriteCapacityUnits: 5})
```

Reading Data:

There's not much to reading single items. The action GetItem allows you to read entire items or select single attributes to read.

```
dynamo.get_item("people", {HashKeyElement: {S:
"roidrage"}})
```

DynamoDB is suitable for use cases where data access is by one or two dimensions of data, and usually by applications running on Amazon's EC2 service (it can accessed from outside of Amazon EC2 but for many usecases public Internet network latency makes this impractical).

E.g. For an E-commerce application, a user id could be the primary key for an DynamoDB order table, and the value for each key would be set of named fields containing order information, e.g. timestamp, amount, delivery address.

"DynamoDB technically isn't a database, it's a database service. Amazon is responsible for the availability, durability, performance, configuration, optimization and all other manner of minutia that I didn't want occupying my mind. I've never been a big fan of managing the day-to-day operations of a database, so I liked the idea of taking that task off my plate... DynamoDB only allows you to query against the primary key, or the primary key and range. There are ways to periodically index your data using a separate service like CloudSearch, but we are quickly losing the initial simplicity of it being a database service. ... However, it turns out MongoDB isn't quite as difficult as the nerds had me believe, at least not at our scale. MongoDB works as advertised and auto-shards and provides a very simple way to get up and running with replica sets.

IV. FEATURES OF DYNAMODB

Features and Benefits of DynamoDB:

* Scalable

Amazon DynamoDB is designed for seamless throughput and storage scaling.

- Automated Storage Scaling
- Provisioned Throughput
- Fully Distributed, Shared Nothing Architecture

* Easy Administration
* Flexible
* Fast, Predictable Performance
* Built-in Fault Tolerance
* Schemaless
* Strong Consistency, Atomic Counters
* Cost Effective
* Secure
* Integrated Monitoring
* Elastic MapReduce Integration

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

Some features of DynamoDB are very compelling compared to its competitors

1. Analytics

Extending the application of analytics to a database has not been as easy. Ideally, we would just want to send a request to the database and have it send back the result when ready. But every NoSQL database has failed in this aspect. MongoDB has major limitations when running map-reduce jobs.

On the other hand, DynamoDB integrates with Elastic Map Reduce and reduces the complexity of analyzing unstructured data. This is a BIG plus

2. Relaxed vs Strong consistency

With DynamoDB, there is no need to hardcode replication of values. Once you make a choice, everything works like it should. Like all Amazon product offerings, read and write units can be adjusted based on actual usage. In our opinion, nothing beats this!

3. Ease of getting started

Amazon poses a real threat to competition by offering a hosted solution. If you have an AWS account, getting started with DynamoDB is as simple as making a single API call! With other NoSQL solutions, the developer must have the right servers, installations and configurations. With DynamoDB, they just need to concentrate on the application and let AWS handle the rest.

4. Performance

Amazon, like AWS, provides amazing performance with DynamoDB. They give single digit latency even on very heavy loads. All data is synchronously replicated across all availability zones AND there is NO downtime even while there are throughput updates! The use of SSDs is a real game changer for random reads and updates

5. Pay for use

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.

With all its advantages, even if we consider a few flaws DynamoDB does have, as a product it comes closest to fulfilling the promise of NoSQL compared to its competitors: Easy-to-use structured storage without the complexity of managing SQL servers and the reliability and performance benefits of scaling out.

DATA MODEL CONCEPTS - TABLES, ITEMS, AND ATTRIBUTES

In Amazon DynamoDB, a database is a collection of tables.

A table is a collection of items and each item is a collection of attributes. In a relational database, a table has a predefined schema such as the table name, primary key, list of its column names and their data types. All records stored in the table must have the same set of columns.

Each attribute in an item is a name-value pair. An attribute can be single-valued or multi-valued set. For example, a book item can have title and authors attributes. Each book has one title but can have many authors. The multi-valued attribute is a set; duplicate values are not allowed.

When you create a table, in addition to the table name, you must specify the primary key of the table. A secondary index lets you query the data in the table using an alternate key, in addition to queries against the primary key.

Amazon DynamoDB supports the following data types:

- Scalar data types: Number, String, and Binary.
- Multi-valued types: String Set, Number Set, and Binary Set.

SUPPORTED OPERATIONS IN DYNAMODB

Table Operations:

DynamoDB provides operations to create, update and delete tables. After the table is created, you can use the Update Table operation to increase or decrease a table's provisioned throughput. DynamoDB also supports an operation to retrieve table information (the Describe Table operation) including the current status of the table, the primary key, and when the table was created. The List Tables operation enables you to get a list of tables in your account in the region of the endpoint you are using to communicate with DynamoDB.

Item Operations:

Item operations enable you to add, update and delete items from a table. The UpdateItem operation allows you to update existing attribute values, add new attributes, and delete existing attributes from an item. You can also perform conditional updates. For example, if you are updating a price value, you can set a condition so the update happens only if the current price is $20. DynamoDB provides an operation to retrieve a single item (GetItem) or multiple items (BatchGetItem). You can use the BatchGetItem operation to retrieve items from multiple tables.

SYSTEM ARCHITECTURE

Core distributed system techniques in Dynamo

- Partitioning
- Replication
- Versioning
- Execution
- Consistency
- Handling failures
- Membership and failure detection

V. SOMEOF THE OTHER CHALLENGES COMMANLY NOTICED

Provisioned throughput capacity unit sizes	• One read capacity unit = one strongly consistent read per second, or two eventually consistent reads per second, for items up 4 KB in size. • One write capacity unit = one write per second, for items up to 1 KB in size.
Provisioned throughput minimum per table	1 read capacity unit and 1 write capacity unit.
Provisioned throughput minimum per global secondary index	1 read capacity unit and 1 write capacity unit.
Maximum concurrent Control Plane API requests (includes cumulative number of tables in the CREATING, UPDATING or DELETING state)	In general, you can have up to 10 of these requests running concurrently.
Item size	• Cannot exceed 64 KB which includes both attribute name binary length (UTF-8 length) and attribute value lengths (again binary length). The attribute name counts towards the size limit. • These limits apply to items stored in tables, and also to items in secondary indexes.
Attribute values	Attribute values cannot be null or empty.
Attribute name-value pairs per item	The cumulative size of attributes per item must be under 64 KB.
Maximum number of values in an attribute set	No practical limit on the quantity of values, as long as the item containing the values fits within the 64 KB item limit.
Query	Result set limited to 1 MB per API call.
Scan	Scanned data set size maximum is 1 MB per API call.

Fig.1 Limits within Amazon DynamoDB

VIII. COMPARISONS WITH 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.

7 Reasons You Should Use MongoDB over DynamoDB

1: Use MongoDB if your indexing fields might be altered later.

2: Use MongoDB if your need features of document database as your NoSQL solution.

3: Use MongoDB if you are going to use Perl, Erlang, or C++.

4: Use MongoDB if you may exceed the limits of DynamoDB.

5: Use MongoDB if you are going to have data type other than string, number, and base 64 encoded binary.

6: Use MongoDB if you are going to query by regular expression.

7: Use MongoDB if you are a big fan of document database.

3 Reasons You Should Use DynamoDB over MongoDB

1: Use DynamoDB if you are NOT going to have an employee to manage the database servers.

2: Use DynamoDB if you didn't have budget for dedicated database servers.

3: Use DynamoDB if you are going to integrate with other Amazon Web Services.

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.

DynamoDB X	MongoDB X	
Description	Hosted, scalable database service by Amazon with the data stored in Amazons cloud	One of the most popular document stores
DB-Engines Ranking Trend Chart	Rank 33 Score 8.88	Rank 5 Score 199.99
Website	aws.amazon.com/dynamodb	www.mongodb.org
Technical documentation	aws.amazon.com/-documentation/dynamodb	www.mongodb.org/display/DOCS/-Home
Developer	Amazon	MongoDB, Inc
Initial release	2012	2009
License	n.a.	Open Source
Implementation language		C++
Server operating systems	hosted	Linux OS X Solaris Windows
Database model	Key-value store	Document store
Data scheme	schema-free	schema-free
Typing	yes	yes
Secondary indexes	no	Yes
SQL	no	No
APIs and other access methods	RESTful HTTP API	proprietary protocol using JSON
Supported programming languages	.Net ColdFusion Erlang Groovy Java JavaScript Perl PHP Python Ruby	Actionscript C C# C++ Clojure ColdFusion D Dart Delphi Erlang Go Groovy Haskell Java JavaScript Lisp Lua MatLab Perl PHP PowerShell Prolog Python R Ruby Scala Smalltalk
Server-side scripts	no	JavaScript
Triggers	no	No
Partitioning methods	Sharding	Sharding
Replication methods	yes	Master-slave replication
MapReduce	no	Yes
Consistency concepts	Eventual Consistency Immediate Consistency	Eventual Consistency Immediate Consistency
Foreign keys	no	no
Transaction concepts	no	no
Concurrency	yes	Yes
Durability	yes	yes
User concepts	Access rights for users and roles can be defined via the AWS Identity and Access Management (IAM)	Users can be defined with full access or read-only access

Fig.2 DynamoDB and MongoDB comparison

NoSQL - So many choices

- MongoDB
- Riak
- CouchDB
- Bigcouch
- Redis
- Cassandra

MongoDB
Advantages: Auto-shading, Good performance, Dynamic query interface
Disadvantages: Global write-lock, No automatic rebalancing, Master-Slave-Replication w/ Auto Failover.
Riak
Advantages: Automatic Replication and Rebalancing Secondary Indices, Large blobs, Map/Reduce.
Disadvantages: InnoDB Backend, Slow writes.
CouchDB
Advantages: Bidirectional replication, Crash only-design, Changes Feed, Views via JS Map Reduce.
Disadvantages: Serialized Writes, Sharding/Rebalancing only with BigCouch.
Redis
Advantages: Performance (in memory), Transactions.
Disadvantages: Only master-slave-replication, Limited by memory.
Cassandra
Advantages: Automatic Replication and Rebalancing, Secondary Indices.
Disadvantages: Writes are faster than reads, bad scaling for 2nd indices, Super-complex to handle,

VI. 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. I used node.js to build my backend server, both AWS SDK for node.js and mongoose SDK for MongoDB works very well. It's really amazing to use mongoose for MongoDB

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 fan of document database.
10gen is the company backing MongoDB and also gave some very good suggestions on MongoDB. This is really a very good experience.

Use DynamoDB if you are NOT going to have an employee to manage the database servers.
This is the top 1 reason I migrate from MongoDB to DynamoDB. We are launching a startup, and we have a long list of user requirements from early adoption users. We want to make them satisfied. I need to develop the Windows/Mac OS/Ubuntu software and iPhone/Android apps, also need to work on server to provide data synchronization among these apps.

Use DynamoDB if you didn't have budget for dedicated database servers.
Because I didn't have too much traffic and data records. I used 2 linode VPS as database servers, 1G RAM, 24G disk. The 2 database server is grouped as replica set, and no sharding yet. Ideally they should support my current data scale very well. However it's not true. Upgrading database servers will take more cost, and may still not be able to resolve the issue. There are some managed MongoDB services, but I 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.

VII. 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 as well. 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.

VIII ACKNOWLEDGEMENT

We 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.

IX REFERENCES

[1] Giuseppe DeCandia, DenizHastorun, MadanJampani - "Dynamo: Amazon's Highly Available Key-value Store"SOSP '07 Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles Pages 205-220

[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