**CSP554 – Research Paper**

**Paper Topic:** Research Cloud NoSQL databases (AWS DynamoDB ,Google BigTable and Microsoft Azure CosmosDB).

**Paper subject area:** Compare and evaluate cloud NoSQL databases: AWS DynamoDB, Microsoft Azure CosmosDB and Google BigTable.

**Abstract:**

A NoSQL database is exactly the type of database that can handle the sort of unstructured, messy and unpredictable data that many application requires.NoSQL is particularly useful for storing unstructured data, which is growing far more rapidly than structured data and does not fit the relational schemas of RDBMS.NoSQL databases are widely recognized for their ease of development, functionality, and performance at scale. They use a variety of data models, including document, graph, key-value, in-memory, and search. NoSQL databases offer enterprises important advantages over traditional RDBMS, including scalability,performance,high avaibility,global avaibility and flexible data modeling.With NoSQL, data can be stored in a schema-less or free-form fashion. Any data can be stored in any record. There is various NoSQL database available like AWS DynamoDB,Google BigTable and Microsoft Azure Cosmos DB.

* Amazon DynamoDB is a fully managed NoSQL database service that provides fast and predictable performance with seamless scalability.Dynamodb is 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. Dynamo is used to manage the state of services that have very high reliability requirements and need tight control over the tradeoffs between availability, consistency, cost-effectiveness and performance.Dynamo uses a variant of consistent hashing instead of basic consistent hashing algorithm.Dynamo uses the concept of “virtual nodes” in it’s algorithm.To achieve high availability and durability, Dynamo replicates its data on multiple hosts.Dynamo uses Merkle trees for anti-entropy. DynamoDB lets you offload the administrative burdens of operating and scaling a distributed database, so that you don't have to worry about hardware provisioning, setup and configuration, replication, software patching, or cluster scaling.
* Azure Cosmos DB is Microsoft’s globally distributed, horizontally partitioned, multi-model database service. The service is designed to allow customers to elastically and independently scale throughput and storage across any number of geographical regions.With the click of a button, Azure Cosmos DB enables you to elastically and independently scale throughput and storage across any number of Azure's geographic region.It offers throughput, latency,availability,and consistency guarantee.Azure Cosmos DB contains a write optimized, resource governed, schema-agnostic databaseenginethat natively supports multiple data models:key-value, documents, graphs,and columnar.It is mainly used for large-scale OLTP solutions.APIs for the following data models are supported with SDKs available in multiple languages. APIs for the following data models are supported with SDKs available in multiple languages: SQL API, MongoDB API, Cassandra API, Gremlin API, Table API.It is financially backed, comprehensive service level agreements (SLAs) for availability, latency, throughput, and consistency for your mission-critical data.Azure Cosmos DB is up to two to threetimes cheaper than DynamoDB for high volume workloads.
* Bigtable is a distributed storage system for managing structured data that is designed to scale to a very large size: petabytes of data across thousands of commodity servers.Bigtable has achieved several goals: wide applicability, scalability, high performance, and high availability.Bigtable is a sparse, distributed, persistent multidimensional sorted map. The map is indexed by a row key, column key, and a timestamp.The Bigtable API provides functions for creating and deleting tables and column families.Bigtable uses the distributed Google File System (GFS) [17] to store log and data files.The Google SSTable file format is used internally to store Bigtable data.Bigtable relies on a highly-available and persistent distributed lock service called Chubby.The Bigtable implementation has three major components: a library that is linked into every client, one master server, and many tablet servers
* This paper includes resources to help you better understand various NoSQL databases like AWS DynamoDB,Microsoft Azure CosmosDB and Google’s Bigtable and to get started.

**NoSQL databases**

NoSQL is an approach to databases that represents a shift away from traditional relational database management systems (RDBMS). NoSQL can mean “not SQL” or “not only SQL.” As RDBMS have increasingly failed to meet the performance, scalability, and flexibility needs that next-generation, data-intensive applications require, NoSQL databases have been adopted by mainstream enterprises. NoSQL is particularly useful for storing unstructured data, which is growing far more rapidly than structured data and does not fit the relational schemas of RDBMS. Common types of unstructured data include: user and session data; chat, messaging, and log data; time series data such as IoT and device data; and large objects such as video and images. NoSQL systems store and manage data in ways that allow for high operational speed and great flexibility on the part of the developers. Many were developed by companies like Google, Amazon and Microsoft that sought better ways to store content or process data for massive websites. Unlike SQL databases, many NoSQL databases can be scaled horizontally across hundreds or thousands of servers.

The advantages of NoSQL don’t come without a cost, though. NoSQL systems don’t generally provide the same level of data consistency as SQL databases. In fact, while SQL databases have traditionally sacrificed performance and scalability for the ACID properties behind reliable transactions, NoSQL databases have largely ditched those ACID guarantees for speed and scalability.

Among the NoSQL databases, you will find four common models for storing data, which lead to four common types of NoSQL systems:

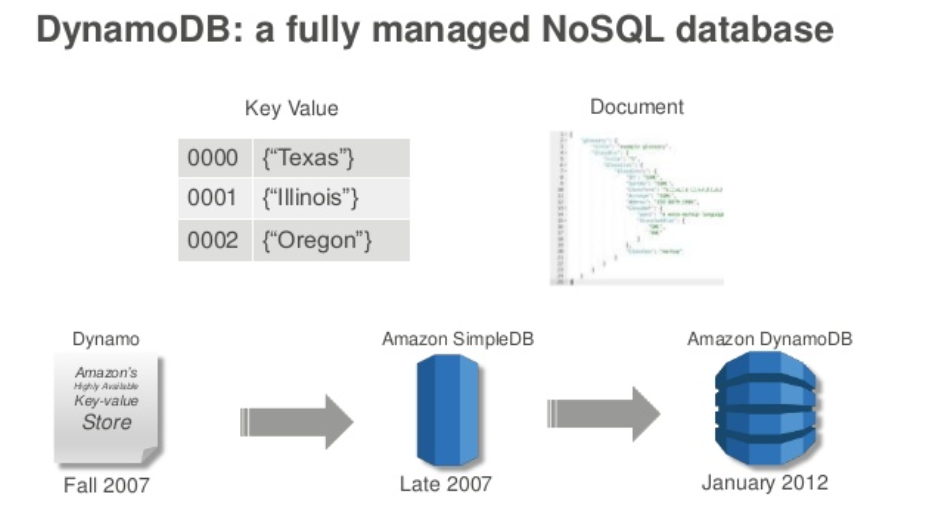
1. **Document databases** (e.g. CouchDB, MongoDB). Inserted data is stored in the form of free-form JSON structures or “documents,” where the data could be anything from integers to strings to freeform text. There is no inherent need to specify what fields, if any, a document will contain. Examples: MongoDB, CouchDB, OrientDB, and RavenDB .
2. **Key-value stores** (e.g. Redis, Riak). Free-form values—from simple integers or strings to complex JSON documents—are accessed in the database by way of keys. Riak , Tokyo Cabinet, Redis server, Memcached, and Scalaris are examples of a key-value store.
3. **Wide column stores** (e.g. HBase, Cassandra). Data is stored in columns instead of rows as in a conventional SQL system. Any number of columns (and therefore many different types of data) can be grouped or aggregated as needed for queries or data views. Examples: BigTable, Cassandra, Hbase, and Hypertable.
4. **Graph databases** (e.g. Neo4j). Data is represented as a network or graph of entities and their relationships, with each node in the graph a free-form chunk of data. Examples are Neo4J, InfoGrid, Infinite Graph, and FlockDB

**AWS DynamoDB**

Amazon DynamoDB is a key-value and document database that delivers single-digit millisecond performance at any scale. It's a fully managed, multiregion, multimaster database with built-in security, backup and restore, and in-memory caching for internet-scale applications. DynamoDB can handle more than 10 trillion requests per day and support peaks of more than 20 million requests per second.

Many of the world's fastest growing businesses such as Lyft, Airbnb, and Redfin as well as enterprises such as Samsung, Toyota, and Capital One depend on the scale and performance of DynamoDB to support their mission-critical workloads.

More than 100,000 AWS customers have chosen DynamoDB as their key-value and document database for mobile, web, gaming, ad tech, IoT, and other applications that need low-latency data access at any scale. Create a new table for your application and let DynamoDB handle the rest.



**Features of Amazon DynamoDB**

1. **Performance at scale**

* **Key-value and document data models**

The first step in designing your DynamoDB application is to identify the specific query patterns that the system must satisfy. It is important to understand three fundamental properties of your application's access patterns before we start using dynamo db. They are data size, data shape and data velocity.

Data size helps knowing how much data will be stored and requested at one time will help determine the most effective way to partition the data. Data shape is a key factor in increasing speed and scalability.

NoSQL database organizes data so that its shape in the database corresponds with what will be queried. Data velocity helps in determining how to partition data to best use I/O capacity. DynamoDB scales by increasing the number of physical partitions that are available to process queries, and by efficiently distributing data across those partitions.

* **Microsecond latency with DynamoDB Accelerator**

Amazon DynamoDB is designed for scale and performance. In most cases, the DynamoDB response times can be measured in single-digit milliseconds. However, there are certain use cases that require response times in microseconds. For these use cases, DynamoDB Accelerator (DAX) delivers fast response times for accessing eventually consistent data. DAX is a DynamoDB-compatible caching service that enables you to benefit from fast in-memory performance for demanding applications. DAX addresses three core scenarios:

* + As an in-memory cache, DAX reduces the response times of eventually-consistent read workloads by an order of magnitude, from single-digit milliseconds to microseconds.
  + DAX reduces operational and application complexity by providing a managed service that is API-compatible with Amazon DynamoDB, and thus requires only minimal functional changes to use with an existing application.
  + For read-heavy or bursty workloads, DAX provides increased throughput and potential operational cost savings by reducing the need to over-provision read capacity units. This is especially beneficial for applications that require repeated reads for individual keys.
* **Automated global replication with global tables**

Amazon DynamoDB global tables provide a fully managed solutionfor deploying a multi-region, multi-master database, without having to build and maintain your own replication solution. When you create a global table, you specify the AWS regions where you want the table to be available. DynamoDB performs all of the necessary tasks to create identical tables in these regions, and propagate ongoing data changes to all of them.

DynamoDB global tables are ideal for massively scaled applications, with globally dispersed users. In such an environment, users expect very fast application performance. Global tables provide automatic multi-master replication to AWS regions world-wide, so you can deliver low-latency data access to your users no matter where they are located.

* **Real-time data processing with DynamoDB Streams**

DynamoDB Streams captures a time-ordered sequence of item-level modifications in any DynamoDB table, and stores this information in a log for up to 24 hours. Applications can access this log and view the data items as they appeared before and after they were modified, in near real time. A DynamoDB stream is an ordered flow of information about changes to items in an Amazon DynamoDB table. When you enable a stream on a table, DynamoDB captures information about every modification to data items in the table. Whenever an application creates, updates, or deletes items in the table, DynamoDB Streams writes a stream record with the primary key attribute(s) of the items that were modified. A stream record contains information about a data modification to a single item in a DynamoDB table.

1. **Serverless**

Amazon DynamoDB has two read/write capacity modes for processing reads and writes on your tables:

* On-demand
* Provisioned (default, free-tier eligible)

The read/write capacity mode controls how you are charged for read and write throughput and how you manage capacity. You can set the read/write capacity mode when creating a table or you can change it later.

* On-Demand Mode

Amazon DynamoDB on-demand is a flexible billing option capable of serving thousands of requests per second without capacity planning. DynamoDB on-demand offers pay-per-request pricing for read and write requests so that you pay only for what you use.

When you choose on-demand mode, DynamoDB instantly accommodates your workloads as they ramp up or down to any previously reached traffic level. If a workload’s traffic level hits a new peak, DynamoDB adapts rapidly to accommodate the workload. Tables that use on-demand mode deliver the same single-digit millisecond latency, service-level agreement (SLA) commitment, and security that DynamoDB already offers. You can choose on-demand for both new and existing tables and you can continue using the existing DynamoDB APIs without changing code.

On-demand mode is a good option if any of the following are true:

* You create new tables with unknown workloads.
* You have unpredictable application traffic.
* You prefer the ease of paying for only what you use.

For on-demand mode tables, you don't need to specify how much read and write throughput you expect your application to perform. DynamoDB charges you for the reads and writes that your application performs on your tables in terms of read request units and write request units.

* Provisioned Mode

If you choose provisioned mode, you specify the number of reads and writes per second that you require for your application. You can use auto scaling to adjust your table’s provisioned capacity automatically in response to traffic changes. This helps you govern your DynamoDB use to stay at or below a defined request rate in order to obtain cost predictability.

Provisioned mode is a good option if any of the following are true:

* You have predictable application traffic.
* You run applications whose traffic is consistent or ramps gradually.
* You can forecast capacity requirements to control costs.

For provisioned mode tables, you specify throughput capacity in terms of read capacity units (RCUs) and write capacity units (WCUs):

* One read capacity unit represents one strongly consistent read per second, or two eventually consistent reads per second, for an item up to 4 KB in size. Transactional read requests require two read capacity units to perform one read per second for items up to 4 KB.
* One write capacity unit represents one write per second for an item up to 1 KB in size. If you need to write an item that is larger than 1 KB, DynamoDB must consume additional write capacity units.

1. **Enterprise ready**

DynamoDB is built for mission-critical workloads, including support for ACID transactions for a broad set of applications that require complex business logic. DynamoDB helps secure your data with encryption and continuously backs up your data for protection, with guaranteed reliability through a service level agreement.

* **ACID transactions**

Amazon DynamoDB transactions simplify the developer experience of making coordinated, all-or-nothing changes to multiple items both within and across tables. Transactions provide atomicity, consistency, isolation, and durability (ACID) in DynamoDB, enabling you to maintain data correctness in your applications easily.

You can use the DynamoDB transactional read and write APIs to manage complex business workflows that require adding, updating, or deleting multiple items as a single, all-or-nothing operation.

* **Encryption at rest**

Amazon DynamoDB offers fully managed encryption at rest. DynamoDB encryption at rest provides enhanced security by encrypting all your data at rest using encryption keys stored in AWS Key Management Service (AWS KMS). This functionality helps reduce the operational burden and complexity involved in protecting sensitive data. With encryption at rest, you can build security-sensitive applications that meet strict encryption compliance and regulatory requirements.

DynamoDB encryption at rest provides an additional layer of data protection by securing your data in the encrypted table, including its primary key, local and global secondary indexes, streams, global tables, backups, and DynamoDB Accelerator (DAX) clusters whenever the data is stored in durable media. Organizational policies, industry or government regulations, and compliance requirements often require the use of encryption at rest to increase the data security of your applications.

* **Point-in-time recovery**

Point-in-time recovery helps protect your Amazon DynamoDB tables from accidental write or delete operations. With point in time recovery, you don't have to worry about creating, maintaining, or scheduling on-demand backups. For example, suppose that a test script writes accidentally to a production DynamoDB table. With point-in-time recovery, you can restore that table to any point in time during the last 35 days. DynamoDB maintains incremental backups of your table.

In addition, point-in-time operations don't affect performance or API latencies. DynamoDB table con be restored to a point in time using the console, the AWS CLI, or the DynamoDB API.

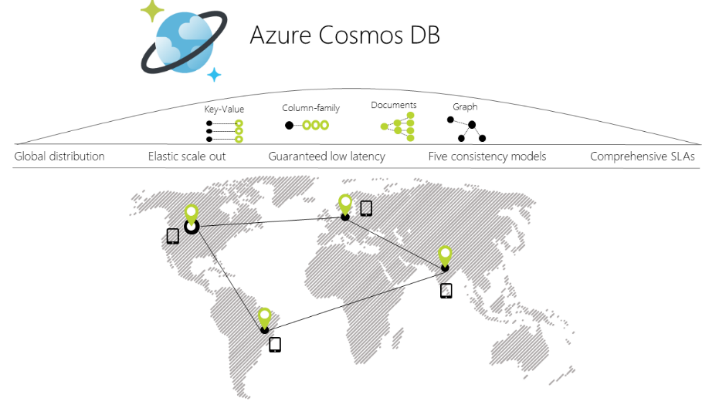
* **On-demand backup and restore**

Amazon DynamoDB provides on-demand backup capability. It allows you to create full backups of your tables for long-term retention and archival for regulatory compliance needs. You can back up and restore your DynamoDB table data anytime with a single click in the AWS Management Console or with a single API call. Backup and restore actions execute with zero impact on table performance or availability.

On-demand backup and restore scales without degrading the performance or availability of your applications. It uses a new and unique distributed technology that allows you to complete backups in seconds regardless of table size. You can create backups that are consistent within seconds across thousands of partitions without worrying about schedules or long-running backup processes. All backups are cataloged, easily discoverable, and retained until explicitly deleted.

**Microsoft Azure CosmosDB**

Azure Cosmos DB is a database for building blazing fast, planet scale applications with native support for NoSQL. It is a globally distributed, multi-model database service that supports document, key-value, wide-column, and graph databases. Azure Cosmos DB was built from the ground up with global distribution and horizontal scale at its core. It offers turnkey global distribution across any number of Azure regions by transparently scaling and replicating your data wherever your users are. Elastically scale your writes and reads all around the globe, and pay only for what you need.



Azure Cosmos DB provides native support for NoSQL and OSS APIs including MongoDB, Cassandra, Gremlin and SQL, offers multiple well-defined consistency models, guarantees single-digit-millisecond read and write latencies at the 99th percentile, and guarantees 99.999 high availability with multi-homing anywhere in the world—all backed by industry-leading, comprehensive service level agreements (SLAs).

**Features of Microsoft Azure Cosmosdb**

As a globally distributed, multi-model database service, Azure Cosmos DB makes it easy to build scalable, highly responsive applications at global scale:

* **Turnkey global distribution.**

You can distribute your data to any number of Azure regions, with the click of a button. This enables you to put your data where your users are, ensuring the lowest possible latency to your customers.

Using Azure Cosmos DB's multi-homing APIs, the app always knows where the nearest region is and sends requests to the nearest data center. All of this is possible with no config changes. You set your write-region and as many read-regions as you want, and the rest is handled for you.

As you add and remove regions to your Azure Cosmos DB database, your application does not need to be redeployed and continues to be highly available thanks to the multi-homing API capability.

* **Multiple data models and popular APIs for accessing and querying data.**

The atom-record-sequence (ARS) based data model that Azure Cosmos DB is built on natively supports multiple data models, including but not limited to document, graph, key-value, table, and column-family data models.

APIs for the following data models are supported with SDKs available in multiple languages:

* **SQL API:** A schema-less JSON database engine with rich SQL querying capabilities.
* **MongoDB API:** A massively scalable MongoDB-as-a-Service powered by Azure Cosmos DB platform. Compatible with existing MongoDB libraries, drivers, tools, and applications.
* **Cassandra API:** A globally distributed Cassandra-as-a-Service powered by Azure Cosmos DB platform. Compatible with existing Apache Cassandra libraries, drivers, tools, and applications.
* **Gremlin API:** A fully managed, horizontally scalable graph database service that makes it easy to build and run applications that work with highly connected datasets supporting Open Gremlin APIs (based on the Apache TinkerPop specification, Apache Gremlin).
* **Table API:** A key-value database service built to provide premium capabilities (for example, automatic indexing, guaranteed low latency, global distribution) to existing Azure Table storage applications without making any app changes.
* **Elastically and independently scale throughput and storage on demand and worldwide.**

Easily scale database throughput at a per-second granularity, and change it anytime you want. Scale storage size transparently and automatically to handle your size requirements now and forever.

* **Build highly responsive and mission-critical applications.**

Your application can be easily designed to perform near real-time reads and writes. It can use single-digit millisecond latencies against all the regions you chose for your database. Azure Cosmos DB internally handles the data replication between regions. As a result, the consistency level selected for the Azure Cosmos DB account is guaranteed.

Many applications benefit from the performance enhancements that come with the ability to perform multi-region (local) writes. Some applications that require strong consistency prefer to funnel all writes to a single region. For these applications, Azure Cosmos DB supports single region and multi-region configurations.

Azure Cosmos DB guarantees end-to-end low latency at the 99th percentile to its customers. For a typical 1KB item, Cosmos DB guarantees end-to-end latency of reads under 10 ms and indexed writes under 10 ms at the 99th percentile, within the same Azure region. The median latencies are significantly lower (under 5 ms).

* **Ensure "always on" availability**

You can dynamically associate "priorities" to the regions associated with your Azure Cosmos DB database

account. Priorities are used to direct the requests to specific regions in the event of regional failures. In an unlikely

event of a regional disaster, Cosmos DB automatically failovers in the order of priority.

To test the end-to-end availability of the application, you can manually trigger failover (rate limited to two

operations within an hour). Cosmos DB guarantees zero data loss during manual regional failovers. In case a

regional disaster occurs, Cosmos DB guarantees an upper-bound on data loss during the system-initiated

automatic failover. You do not have to redeploy your application after a regional failover, and availability SLAs are

maintained by Azure Cosmos DB.

For this scenario, Cosmos DB allows you to interact with resources using either logical (region-agnostic) or

physical (region-specific) endpoints. The former ensures that the application can transparently be multi-homed in

case of failover. The latter provides fine-grained control to the application to redirect reads and writes to specific

regions. Cosmos DB guarantees 99.99% availability SLA for every database account. The availability guarantees

are agnostic of the scale(provisioned throughput and storage), number of regions, or geographical distance

between regions associated with a given database. Deploy to any number of Azure regions for higher availability and better performance.

Dynamically set priorities to regions and simulate a failure of one or more regions with zero-data loss guarantees to test the end-to-end availability for the entire app (beyond just the database).

* **Write globally distributed applications, the right way**

Five well-defined, practical, and intuitive consistency models provide a spectrum of strong SQL-like consistency all the way to the relaxed NoSQL-like eventual consistency, and everything in-between.

* **Money back guarantees**

Industry-leading, financially backed, comprehensive service level agreements (SLAs) for availability, latency, throughput, and consistency for your mission-critical data.

* **No database schema/index management**

Rapidly iterate the schema of your application without worrying about database schema and/or index management.

Azure Cosmos DB’s database engine is fully schema-agnostic – it automatically indexes all the data it ingests without requiring any schema or indexes and serves blazing fast queries.

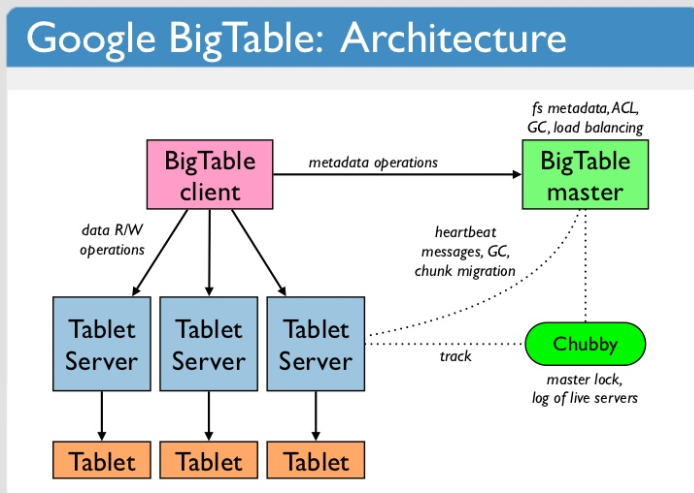
* **Low cost of ownership**

When all total cost of ownership (TCO) considerations taken into account, managed cloud services like Azure Cosmos DB can be five to ten times more cost effective than their OSS counter-parts running on-premises or virtual machines. A Five to ten times more cost effective than a non-managed solution or an on-prem NoSQL solution.Three times cheaper than AWS DynamoDB or Google Spanner.

* **Guaranteed service level agreements** Schema-free Low cost of ownership Next steps You can configure the default consistency level on your Cosmos DB account (and later override the consistency on a specific read request). Internally, the default consistency level applies to data within the partition sets which may be span regions. Cosmos DB is the first managed database service to offer 99.99% SLA guarantees for availability, throughput, low latency, and consistency. Availability: 99.99% uptime availability SLA for each of the data and control plane operations. Throughput: 99.99% of requests complete successfully. Latency: 99.99% of <10 ms latencies at the 99th percentile. Consistency: 100% of read requests will meet the consistency guarantee for the consistency level requested by you.

**Google Bigtable**

Bigtable is designed to reliably scale to petabytes of data and thousands of machines. Bigtable has achieved several goals: wide applicability, scalability, high performance, and high availability. Bigtable is used by more than sixty Google products and projects, including Google Analytics, Google Finance, Orkut, Personalized Search, Writely, and Google Earth. These products use Bigtable for a variety of demanding workloads, which range from throughput-oriented batch-processing jobs to latency-sensitive serving of data to end users. The Bigtable clusters used by these products span a wide range of configurations, from a handful to thousands of servers, and store up to several hundred terabytes of data. In many ways, Bigtable resembles a database: it shares many implementation strategies with databases. Parallel databases and main-memory databases have achieved scalability and high performance, but Bigtable provides a different interface than such systems. Bigtable does not support a full relational data model; instead, it provides clients with a simple data model that supports dynamic control over data layout and format, and allows clients to reason about the locality properties of the data represented in the underlying storage. Data is indexed using row and column names that can be arbitrary strings. Bigtable also treats data as uninterpreted strings, although clients often serialize various forms of structured and semi-structured data into these strings. Clients can control the locality of their data through careful choices in their schemas. Finally, Bigtable schema parameters let clients dynamically control whether to serve data out of memory or from disk.



Cloud Bigtable's powerful back-end servers offer several key advantages:

* **Incredible scalability:** Cloud Bigtable scales in direct proportion to the number of machines in your cluster. A self-managed HBase installation has a design bottleneck that limits the performance after a certain QPS is reached. Cloud Bigtable does not have this bottleneck, and so you can scale your cluster up to handle more queries.

Horizontally scalable with no downtime to match your data throughput needs.

* **Simple administration:** Cloud Bigtable handles upgrades and restarts transparently, and it automatically maintains high data durability. To replicate your data, simply add a second cluster to your instance, and replication starts automatically. No more managing masters or regions; just design your table schemas, and Cloud Bigtable will handle the rest for you.
* **Cluster resizing without downtime:** You can increase the size of a Cloud Bigtable cluster for a few hours to handle a large load, then reduce the cluster's size again—all without any downtime. After you change a cluster's size, it typically takes just a few minutes under load for Cloud Bigtable to balance performance across all of the nodes in your cluster.
* **Fully Managed :** Cloud Bigtable is offered as a fully managed service, meaning you spend your time developing valuable applications instead of configuring and tuning your database for performance and scalability. In addition, Google’s own Bigtable operations team monitors the service to ensure issues are addressed quickly.
* **Integration:** Google Cloud Platform (GCP) integration allows easy creation of full solutions, and HBase API compatibility extends this into the open source Apache ecosystem. It integrates well with Apache Hadoop, StreamSets Data Collector, Geospatial databases, HGraphDB, JanusGraph, Pivotal Cloud Foundry, Terraform, Heroic, OpenTSDB and many more.

**What it's good for**

Cloud Bigtable is ideal for applications that need very high throughput and scalability for non-structured key/value data, where each value is typically no larger than 10 MB. Cloud Bigtable also excels as a storage engine for batch MapReduce operations, stream processing/analytics, and machine-learning applications. You can use Cloud Bigtable to store and query all of the following types of data:

* Time-series data, such as CPU and memory usage over time for multiple servers.
* Marketing data, such as purchase histories and customer preferences.
* Financial data, such as transaction histories, stock prices, and currency exchange rates.
* Internet of Things data, such as usage reports from energy meters and home appliances.
* Graph data, such as information about how users are connected to one another.

**Cloud Bigtable storage model**

Cloud Bigtable stores data in massively scalable tables, each of which is a sorted key/value map. The table is composed of rows, each of which typically describes a single entity, and columns, which contain individual values for each row. Each row is indexed by a single row key, and columns that are related to one another are typically grouped together into a column family. Each column is identified by a combination of the column family and a column qualifier, which is a unique name within the column family.

Each row/column intersection can contain multiple cells at different timestamps, providing a record of how the stored data has been altered over time. Cloud Bigtable tables are sparse; if a cell does not contain any data, it does not take up any space.

**Real Time applications of Big Table**

As of August 2006, there are 388 non-test Bigtable clusters running in various Google machine clusters, with a combined total of about 24,500 tablet servers.Bigtable is widely used by three team of google. They are :

* Google Analytics
* Google Earth
* Personalized Speech

**Comparision between AWS DynamoDB, Microsoft Azure CosmosDB and Google BigTable.**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Dynamodb** | **Cosmodb** | **Bigtable** |
| 1. | Hosted, scalable database service by Amazon with the data stored in Amazons cloud | Globally distributed, horizontally scalable, multi-model database service. | Google's NoSQL Big Data database service. It's the same database that powers many core Google services, including Search, Analytics, Maps, and Gmail. |
| 2. | Primary database model:  1.Document store  2.Key-value store | Primary database model:  1.Document store 2.Graph DBMS 3.Key-value store 4.Wide column store | Primary database model:  1.Wide column store |
| 3. | Fully managed cloud database through AWS | Fully managed cloud database through Azure | Fully managed cloud database |
| 4. | Method for accessing API is Restful HTTP API | Method for accessing API are :  1.gRPC (using protocol buffers) API 2.HappyBase (Python library) 3.HBase compatible API (Java) | Method for accessing API is Restful HTTP API |
| 5. | Map Reduce is not used. | Map Reduce is used with Hadoop integration. | Map Reduce is used. |
| 6. | Developed by Amazon | Developed by Microsoft | Developed by Google |
| 7. | Secondary index is present | Secondary index is present | Secondary index is not present |
| 8. | There is no server side scripting | Server side scripting is done using javascript | There is no server side scripting |
| 9. | No transaction can be done | Multi-item ACID transactions with snapshot isolation within a partition | Atomic single-row operations |
| 10. | Internal replication is done using cloud service | Internal replication is done using cloud service | Internal replication in Colossus, and regional replication between two clusters in different zones |
| 11. | Access rights for users and roles can be defined via the AWS Identity and Access Management (IAM) | Access rights can be defined down to the item level | Access rights for users, groups and roles based on Google Cloud Identity and Access Management (IAM) |
| 12. | Supported Programming Language:   * .Net * ColdFusion * Erlang * Groovy * Java * JavaScript * Perl * PHP * Python * Ruby | Supported Programming Language:   * .Net * C# * Java * JavaScript * JavaScript (Node.js) * MongoDB client drivers written for various programming languages * Python | Supported Programming Language:   * C# * C++ * Go * Java * JavaScript (Node.js) * Python |
| 13. | Method to maintain Consistency :   * Eventual Consistency * Immediate Consistency | Method to maintain Consistency :   * Bounded Staleness * Consistent Prefix * Session Consistency * Eventual Consistency * Immediate Consistency | Method to maintain Consistency :   * Immediate consistency (for a single cluster) * Eventual consistency (for two or more replicated clusters) |
| 14. | Compute services used :   * EC2 * Elastic Container Service * Elastic Container Service for Kubernetes * Elastic Container Registry * Lightsail * Batch * Elastic Beanstalk * Fargate * Auto Scaling * Elastic Load Balancing * VMware Cloud on AWS | Compute services used :   * Virtual Machines * Virtual Machines Scale Sets * Azure Container Service(AKS) * Container Instances * Batch * Service Fabric * Cloud Services | Compute Service used:   * Compute Engine * Kubernetes |
| 15 | Storage Services Supported:   * Simple storage Service(S3) * Elastic Block Storage(EBS) * Elastic File System(EFS) * Storage Gateway * Snowball * Snowball Edge * Snowmobile | Storage Services Supported:   * Blob Storage * Queue Storage * File Storage * Disk Storage * Data Lake Store | Storage Services Supported:   * Cloud Storage * Persistent Disk * Transfer Appliance * Transfer Service |

**Conclusion :**

This paper gives overview of various nosql database developed by various leading companies. Through this paper I have tried to give PROa and CONs of them. Based on various property and differences given,it can help in deciding which nosql database to use under any given scenario.

Referances :

* <https://www.allthingsdistributed.com/files/amazon-dynamo-sosp2007.pdf>
* <https://docs.microsoft.com/en-us/azure/cosmos-db/introduction>
* https://www.ascent.tech/wp-content/uploads/documents/microsoft/cosmos-db/cosmos-db.pdf
* <https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/HowItWorks.html>
* [https://static.googleusercontent.com/media/research.google.com/en//archive/bigtable-osdi06.pdf](https://static.googleusercontent.com/media/research.google.com/en/archive/bigtable-osdi06.pdf)
* <https://cloud.google.com/bigtable/docs/schema-design>
* <https://www.semanticscholar.org/paper/Comparison-of-Cloud-database%3A-Amazon's-SimpleDB-and-Ramanathan-Goel/309bc0f5a4e4c099a8f0b23d15f4b4b483747757>
* <https://www.researchgate.net/publication/324228746_Saving_Large_Semantic_Data_in_Cloud_A_Survey_of_the_Main_DBaaS_Solutions>
* <https://cloud.google.com/bigtable/docs/>
* <https://docs.databricks.com/spark/latest/data-sources/azure/cosmosdb-connector.html>
* <https://www.techopedia.com/definition/29167/google-bigtable>
* <http://www.vogella.com/tutorials/Bigtable/article.html#bigtable>