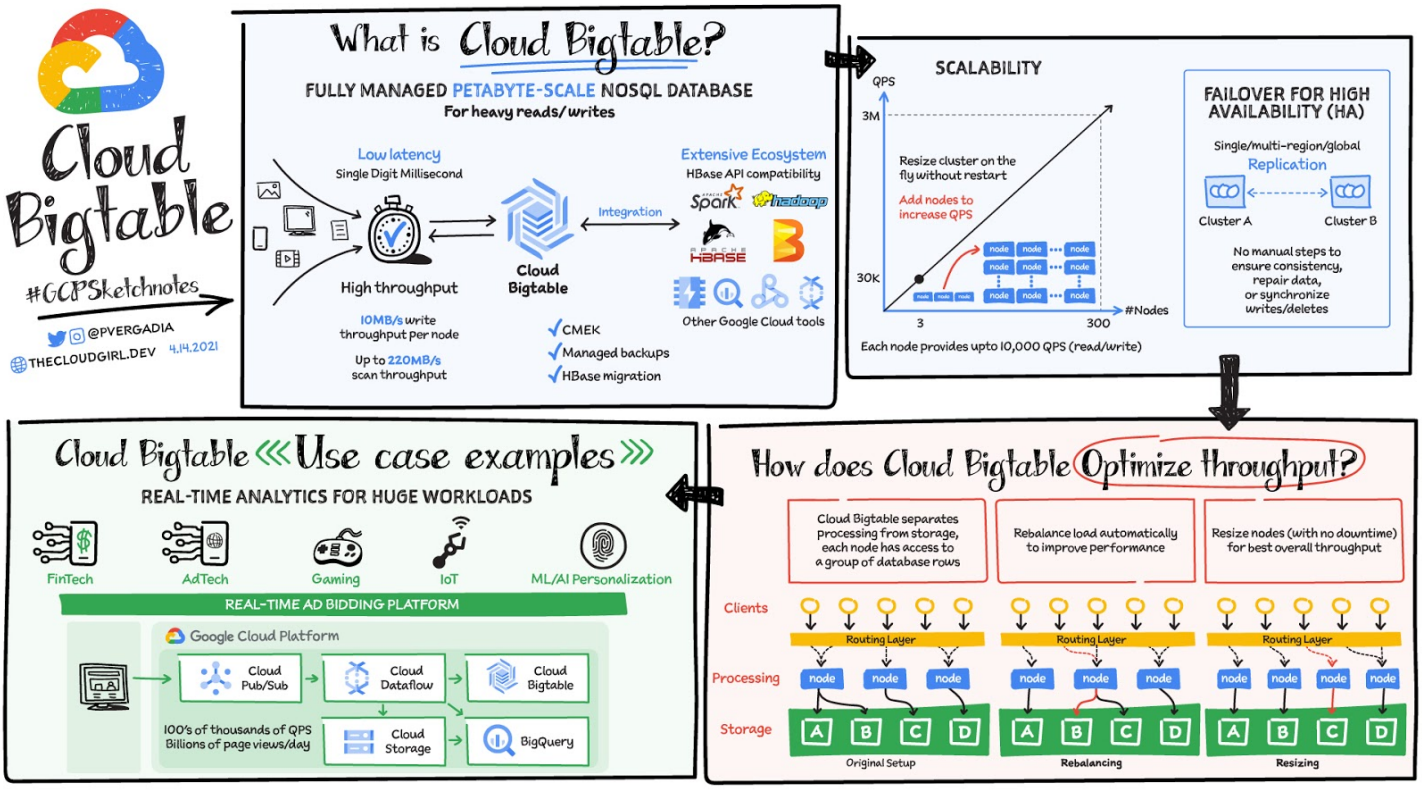
History of BigTable

- BigTable was among the early attempts Google made to manage big data.
- Apache HBase and Cassandra are some of the best known open source projects that were modeled after Bigtable.
- It is one of the three components Google built for managing big data (the other two are Google File System and MapReduce).
- These three components focus on different aspects of big data:
    1. **Google File System** is a reliable distributed file system that the other two build upon.
    2. **MapReduce** is a distributed data processing framework.
    3. **BigTable** is a distributed storage system.
- Google Map, Google Earth, Gmail and YouTube fare all powered by Bigtable's. Google published it as paper in 2006.
- As of January 2022, Bigtable manages over 10 Exabytes of data and serves more than 5 billion requests per second.
- On May 6, 2015, a public version of Bigtable was made available as a part of Google Cloud under the name Cloud Bigtable.
- Google announced a number of updates to Bigtable, including automated scalability.

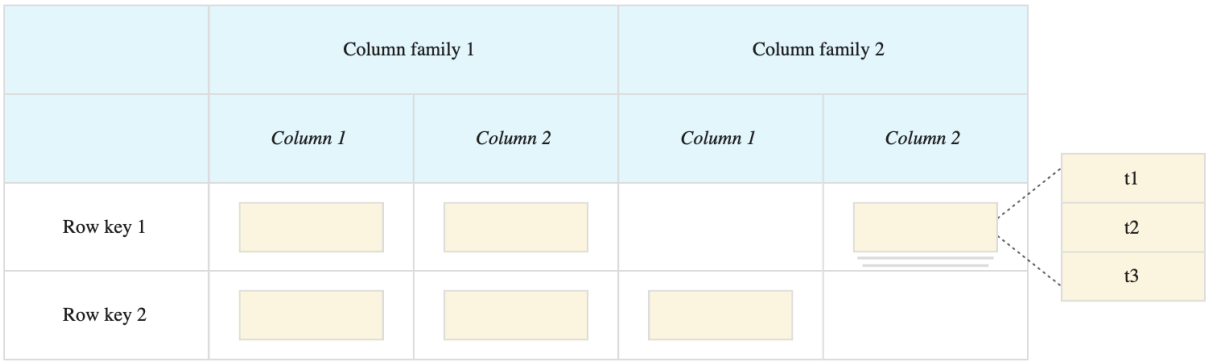


Overview on Cloud BigTable

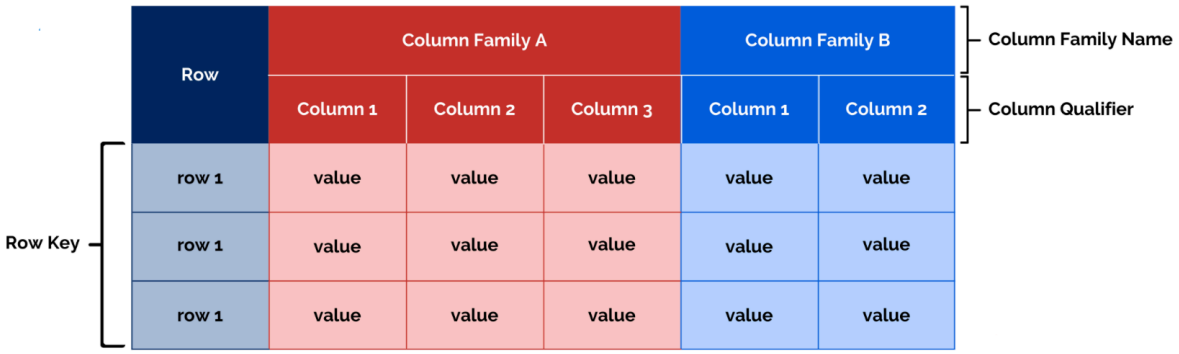
- Cloud Bigtable is Google's NoSQL Big Data database service. It falls into the Wide-Column based family.
- HBase, an open-source implementation of the Bigtable model. It uses a data model very similar to that of Bigtable. HBase was then adopted as a top-level Apache project and became part of the Hadoop Ecosystem of open-source Big Data products. HBase-compatible, enterprise-grade NoSQL database service with single-digit millisecond latency, limitless scale, and 99.999% availability for large analytical and operational workloads.
- It's 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.
- Cloud Bigtable is a sparsely populated table that can scale to billions of rows and thousands of columns, enabling you to store terabytes or even petabytes of data.
- A single value in each row is indexed; this value is known as the row key.
- Bigtable is ideal for storing large amounts of single-keyed data with low latency.
- Bigtable is also designed to scale linearly. BigTable doesn’t support RDBMS concept.
- Not serverless, Milli second latency and Handles millions of request per second.
- Bigtable also provides out of the box high availability with cross-cluster replication.
- Bigtable's powerful backend servers offer several key advantages over a self-managed HBase installation:
    - Incredible scalability. Bigtable scales in direct proportion to the number of machines in your cluster.
    - Simple administration. Bigtable handles upgrades and restarts transparently, and it automatically maintains high data durability.
    - Cluster resizing without downtime. You can increase the size of a Bigtable cluster for a few hours to handle a large load, then reduce the size of the cluster again—all without any downtime.

Cloud BigTable Storage Model

- 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 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*.
  - Each cell contains a unique timestamped version of the data for that row and column.
- Storing multiple cells in a column provides a record of how the stored data for that row and column has changed over time.

- Bigtable tables are sparse; if a column is not used in a particular row, it does not take up any space.

- Cloud Bigtable is designed for high throughput where a typical well-designed workload can achieve around 10,000 reads or writes per second as well as low latency where you can expect responses in around 6 milliseconds and that's per node.
  - So as a rough guide, a cluster of 10 nodes could provide 100,000 rows per second of reads or writes.

Row Key:

- Each row has a unique key, which is a unique identifier for that row.

Column:

- Each column contains a name, a value, and timestamp.

Name:

- This is the name of the name/value pair.

Value:

- This is the value of the name/value pair.

Timestamp:

- This provides the date and time that the data was inserted. This can be used to determine the most recent version of data.

Logical Schema

EmpNo	LName	Job	Mgr
1	Smith	Clerk	3
2	Ward	Sales	8
3	Jones	Manager	12
4	Blake	Slaes	8

Row Based Storage

1	Smith	Clerk	3
2	Ward	Sales	8
3	Jones	Manager	12

Column Based Storage

Smith	Ward	Jones
Clerk	Sales	Manager
3	8	12
1	2	3

Example

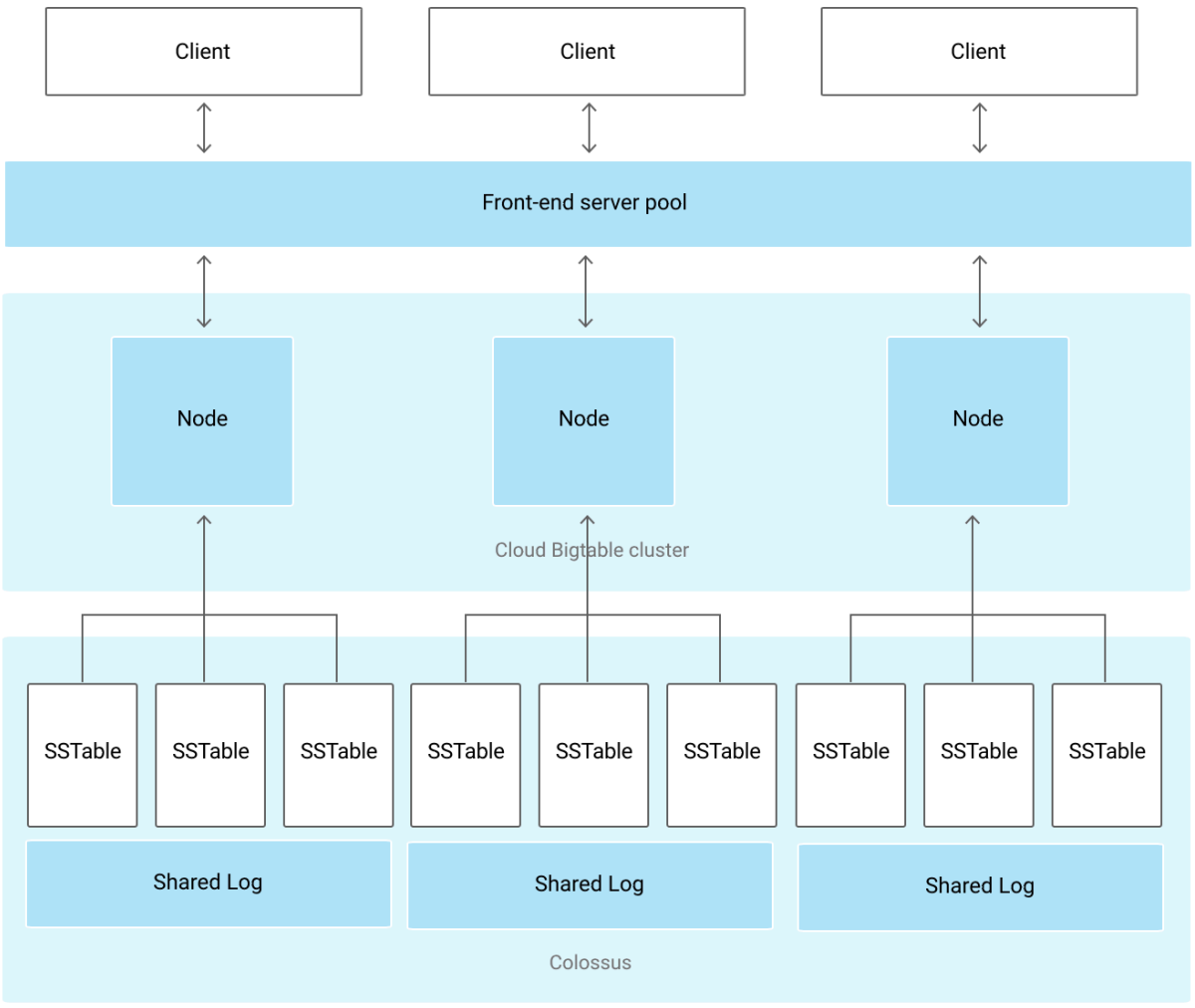
Consider a User database for example, a relational database would organize first name, last name, and address all near each other.

If you wanted to access the state and zip of many users, the database would have to jump around to pull all the fields. A column based database on the other hand is optimized for accessing data by column instead of row. So in our Users database example it would store all the names together, all the states together, all the zip codes together and so on. This makes reads much more efficient. To scan all the states, the database can stay within the same area on disk.

A Wide-Column datastore looks similar however they often group the columns into Column Families, a set of columns that are typically used together. These Column Families are further optimized on disk to ensure fast access.

Cloud BigTable Architecture

- As the diagram illustrates, all client requests go through a frontend server before they are sent to a Bigtable node.



Note: The diagram shows an instance with a single cluster. You can also add clusters to replicate your data, which improves data availability and durability.

- The nodes are organized into a Bigtable cluster, which belongs to a Bigtable instance, a container for the cluster.
- Each node in the cluster handles a subset of the requests to the cluster. By adding nodes to a cluster, you can increase the number of simultaneous requests that the cluster can handle. Adding nodes also increases the maximum throughput for the cluster.
- Then if one cluster becomes unavailable, you can fail over to another cluster.Cluster can support Single zone, regional and cross regional as well.
- A Bigtable table is sharded into blocks of contiguous rows, called *tablets*, to help balance the workload of queries. (Tablets are similar to HBase regions.)
- Tablets are stored on Colossus, Google's file system, in SSTable format. (Colossus - Google's internal global file system)
    - An SSTable provides a persistent, ordered immutable map from keys to values, where both keys and values are arbitrary byte strings. Each tablet is associated with a specific Bigtable node.
- In addition to the SSTable files, all writes are stored in Colossus's shared log as soon as they are acknowledged by Bigtable, providing increased durability.
- Importantly, data is never stored in Bigtable nodes themselves; each node has pointers to a set of tablets that are stored on Colossus. As a result:
    - Rebalancing tablets from one node to another happens quickly, because the actual data is not copied. Bigtable simply updates the pointers for each node.
    - Recovery from the failure of a Bigtable node is fast, because only metadata must be migrated to the replacement node.
    - When a Bigtable node fails, no data is lost.

An instance has a few important properties that you need to know about:

- Bigtable instance
- Storage type (SSD or HDD)
- Application profiles

Instances

The following sections describe these properties.

- There are 2 types of Bigtable instance.
  - **Production instance:**
    - contains 1 or more clusters with a minimum of 3 nodes per cluster
  - **Development instance:**
    - a single node cluster, purely designed for development work.
    - A development instance cannot use replication, does not have an SLA, and has severe performance limitations.

Storage types

- **SSD**
  - SSD is almost always the right choice. There may be initial cost savings in using HDD disks but the impact to performance means you will often end up spending more money to overcome this.
  - SSD disks are the fastest and most predictable option in terms of performance, offering 6 millisecond latencies for 99% of read and write operations.
  - Even though your cluster nodes don't store data, they need to be able to process data. So, there is a calculation for how many nodes you need based on how much data you have.
  - With SSD instances, each node can comfortably process 2.5 terabytes of data.
- **HDD**

- With HDD instances, each node can process 8 terabytes of data.
- This seems like you would pay for fewer nodes but this is not usually the case due to the fact that throughput of HDD disks is severely limited.
- HDD instances might be a cheaper option.This would work perhaps for archival data or data that is analyzed in batches where latency is not considered.

Application profiles

- After you create an instance, Bigtable uses the instance to store application profiles, or app profiles.
- These are custom preferences created for each application that will be making connections to Bigtable.
- It's good practice to create a profile for each individual application as this will help with viewing connection metrics.Additionally it also allows you to define a couple of policies regarding the application's connection.
- The first, is whether your application should use single or multi-cluster routing.
- In single cluster routing, an application's requests will only ever route to a single cluster that you define, even if you have multiple clusters in your instance. If that cluster becomes unavailable, you'll need to update your profile to point to a different cluster.
- With multi-cluster routing, requests will route to the nearest available Bigtable cluster. If that cluster becomes unavailable, connections will automatically fail over to the next available cluster.

Clusters

- A cluster represents the Bigtable service in a specific location. A single instance can run up to 4 clusters and an instance can have clusters in up to 8 regions.
- Production clusters must run a minimum of 3 nodes each and the default quota allows up to 30 nodes per GCP project, not just per instance.
- Each instance also supports a maximum of 1000 tables.
- When your application sends requests to a Bigtable instance, those requests are handled by one of the clusters in the instance.

- Bigtable instances that have only 1 cluster do not use replication. If you add a second cluster to an instance, Bigtable automatically starts replicating your data by keeping separate copies of the data in each of the clusters' zones and synchronizing updates between the copies.

Nodes

- Each cluster in an instance has 1 or more nodes, which are compute resources that Bigtable uses to manage your data.
- Behind the scenes, Bigtable splits all of the data in a table into separate tablets.Tablets are stored on disk, separate from the nodes but in the same zone as the nodes. A tablet is associated with a single node.
- Each node is responsible for:
    - Keeping track of specific tablets on disk.
    - Handling incoming reads and writes for its tablets.
    - Performing maintenance tasks on its tablets, such as periodic compactions.
- A cluster must have enough nodes to support its current workload and the amount of data it stores.

Features Of Cloud BigTable

- High throughput and low latency at any scale
- Cluster resizing without downtime
- Flexible, automated replication to optimize any workload
- Easy migrations from Apache HBase and Cassandra to Bigtable
- Enterprise-grade security and controls
- It can efficiently handle large scale of data: petabytes and across thousands of commodities.
- No schema database.
- Suitable for handling semi structure data.
- Self-managed and handle massive workload with consistent low latency and high throughput.
- **Persistent**:
    - Persistence means data created in Bigtable will persist after the program which created it finished
- **Distributed**:

- BigTable uses the Distributed file system, so that underlying file system spread in arrays among on multiple nodes.
- Data is also replicate across the number of participating nodes. Which saves us against the cluster failing.

- **Sorted**:
  - Map implementations in BigTable keep the key/value pairs in strict alphabetical order.
- **Garbage Collection**
  - Given you may not want to store every version ever created, Bigtable offers the ability to trash cell versions with a feature called Garbage Collection.

Load balancing

- Bigtable manages the splitting, merging, and rebalancing automatically, saving you the effort of manually administering your tablets.
- To get the best write performance from Bigtable, it's important to distribute writes as evenly as possible across nodes.

Supported data types

- Bigtable treats all data as raw byte strings for most purposes.

Memory and disk usage

- The following sections describe how several components of Bigtable affect memory and disk usage for your instance.

1. Unused columns

- Columns that are not used in a Bigtable row do not take up any space in that row.
- Each row is essentially a collection of key/value entries, where the key is a combination of the column family, column qualifier and timestamp.

2. Column qualifiers

- Column qualifiers take up space in a row, since each column qualifier used in a row is stored in that row.

3. Compactions

- Bigtable periodically rewrites your tables to remove deleted entries, and to reorganize your data so that reads and writes are more efficient. This process is known as a *compaction*.
- Bigtable compacts your data automatically.

4. Mutations and deletions

- *Mutations*, or changes, to a row take up extra storage space, because Bigtable stores mutations sequentially and compacts them only periodically.
- Deletions also take up extra storage space, because deletions are actually a specialized type of mutation.
- Until the table is compacted, a deletion uses extra storage rather than freeing up space.

5. Data compression

- Bigtable compresses your data automatically using an intelligent algorithm.
- However, it is useful to know how to store data so that it can be compressed efficiently:
    - **Random data cannot be compressed as efficiently as patterned data.**
    - **Compression works best if identical values are near each other,** either in the same row or in adjoining rows.
    - **Compress values larger than 1 MiB before storing them in Bigtable,** It automatically turns off compression for values larger than 1 MiB.

6. Data durability

- When you use Bigtable, your data is stored on Colossus. If your instance uses replication, Bigtable maintains one copy of your data in Colossus for each cluster in the instance.

- If each cluster is in a different zone or region, durability is further improved.

7. Consistency model

- Single-cluster Bigtable instances provide strong consistency.
- By default, instances that have more than one cluster provide eventual consistency.

8. Security

- When it comes with security and access control, with all other GCP services, this is achieved for Bigtable through Cloud IAM roles.
- These can be applied to the project or instance level.

9. Encryption

- By default, all data stored within Google Cloud, including the data in Bigtable tables, is encrypted at rest using the same hardened key management systems that we use for our own encrypted data.
- If you want more control over the keys used to encrypt your Bigtable data at rest, you can use customer-managed encryption keys (CMEK).

10. Backups

- Bigtable backups let you save a copy of a table's schema and data, then restore from the backup to a new table at a later time.

CPU usage

- The values for these metrics should not exceed the following:

Configuration	Recommended maximum values1
Single-cluster routing, any number of clusters	70% average CPU utilization 90% CPU utilization of hottest node
Multi-cluster routing, autoscaling enabled, 2 or more clusters	70% average CPU utilization 90% CPU utilization of hottest node
Multi-cluster routing, autoscaling not enabled, 2 clusters	35% average CPU utilization2 45% CPU utilization of hottest node2

Multi-cluster-routing, autoscaling enabled, 3 or more clusters	Depends on your configuration.

Cloud BigTable Pricing

- Cloud Bigtable is a fast, fully managed, massively scalable NoSQL database service.
- When you use Bigtable, you are charged for the following:
  - The type of Bigtable instance and the total number of nodes in your instance's clusters.
  - The amount of storage that your tables use.
  - The amount of network bandwidth that you use.
  - Storage and bandwidth usage are measured in binary gigabytes (GB).
  - Storage and bandwidth charges accrue daily.

Pros & Cons of Cloud BigTable

Pros

- Incredible scalability
- Simple administration
- Cluster resizing without downtime

Cons

- Poor Schema Design
- Heavy Data Rows
- Multi Cells in Row
- Overload Cluster
- Choice of Disk
- Poor Network

Replication & Performance

- Replication reduces read latency but does increase write throughput.
  - Write throughput might actually go down because replication requires each cluster to do additional work.
- Replicated clusters in different regions will typically have higher replication latency than replicated clusters in the same region.

Popular Use cases

Bigtable also excels as a storage engine for batch MapReduce operations, stream processing/analytics, and machine-learning applications.

You can use 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 Vs Apache HBase - A Brief Comparison

Name	Google Cloud Bigtable	HBase
Description	Google's NoSQL Big Data database service. It's the same database that powers many core Google services, including Search, Analytics, Maps, and Gmail.	Wide-column store based on Apache Hadoop and on concepts of BigTable
Primary database model	Key-value store Wide column store	Wide column store
Developer	Google	Apache Software Foundation
Initial release	2015	2008
License	commercial	Open Source
Cloud-based only	yes	no

Data scheme	schema-free	schema-free, schema definition possible
Supported programming languages	C# C++ Go Java JavaScript (Node.js) Python	C C# C++ Groovy Java PHP Python Scala
Server-side scripts	no	yes
Triggers	no	yes
Partitioning methods	Sharding	Sharding
Replication methods	Internal replication in Colossus, and regional replication between two clusters in different zones	Multi-source replication Source-replica replication
MapReduce	yes	yes
Consistency concepts	Immediate consistency (for a single cluster), Eventual consistency (for two or more replicated clusters)	Immediate Consistency or Eventual Consistency
Foreign keys	no	no
Transaction concepts	Atomic single-row operations	Single row ACID (across millions of columns)
In-memory capabilities	no	yes
User concepts	Access rights for users, groups and roles based on Google Cloud Identity and Access Management (IAM)	Access Control Lists (ACL) for RBAC, integration with Apache Ranger for RBAC & ABAC