Instituto Tecnológico de Costa Rica

Bases de Datos - IC4302

Resumen 4

Estudiante:

Andrea María Li Hernández - 2021028783

Profesor:

Gerardo Nereo Campos Araya

Fecha de entrega: 21 de septiembre del 2022

Segundo Semestre, 2022

Schema-Agnostic Indexing with Azure DocumentDB

1. Introduction

Azure DocumentDB is Microsoft’s multi-tenant distributed database service for managing JSON documents at Internet scale. It is based on the JSON data model and JavaScript language. DocumentDB is used by many Microsoft applications as Office, Skype and Xbox. Nowadays, Azure developers have access to DocumentDB.

In this summary, you'll find a description of DocumentDB’s indexing subsystem, as well as its capabilities and architecture.

Some of DocumentDB's capabilities are the following:

- The query language supports rich relational and hierarchical queries.
- It is optimized to serve consistent queries while facing sustained high volume document writes.
- It **automatically** indexes all documents **without requiring schema** or secondary indexes from developers.
- It offers tunable consistency levels as *strong, bounded-staleness, session and eventual*.

A DocumentDB database manages a set of entities:

- Users.
- Permissions.
- Collections: It is a schema-agnostic container of arbitrary user generated documents. It also manages stored procedures, triggers, user defined functions and attachments.

Resources: The entities under the tenant's database account.

- Each resource is uniquely identified by a stable and logical URI (Uniform Resource Identifier) and is represented as a JSON document.
- Developers interact with them via HTTP using standard HTTP verbs for CRUD, queries and stored procedures.
- Tenants can scale a resource by creating new resources which get placed across resource partitions.
- A resource partition is made **highly available** by a **replica set**.

DocumentDB is deployed worldwide across multiple Azure regions. It is deployed and managed on clusters of machines each with dedicated local SSDs.

Federation: This refers to a DocumentDB service that manifests itself as an overlay network of machines.

To provide **durability and high availability**, DocumentDB persists data on local SSDs and replicates it among the database engine instances within the replica set respectively.

The indexing subsystem was designed with the following goals:

- Automatic indexing.
- Configurable storage/performance tradeoffs.
- Efficient, rich hierarchical and relational queries.

2. Schema Agnostic Indexing

This section explores the importance of making the database engine schema-agnostic.

To eliminate the impedance mismatch between the database and the application programming models, DocumentDB takes advantage of **JSON** and its lack of a schema specification.

This enables it to **automatically index documents** without requiring schema or secondary indices.

Representing JSON documents as **trees** in-turn normalizes the structure and the instance values across documents into a unifying concept of a dynamically encoded *path* structure.

- Each label in a JSON document becomes a node of the tree.

There are 2 possible mappings of a document and the paths:

1. **Forward index mapping**: Keeps a map of *(document id, path)* tuples.
2. **Inverted index mapping**: Keeps a map of *(path, document id)* tuples.

Developers can query DocumentDB collections using queries written in **SQL and JavaScript**.

---

3. Logical Index Organization

- The index is a union of all the documents and is also represented as a tree.
- Each node of the index tree contains a list of document ids corresponding to the documents containing the given label value.
- A **Term** represents a unique path in the index tree.

Encoding path information: There are 3 default segments because most of JSON docuemtns have root, a key and a values in most of the paths which fit in 3 segments.

Partial Forward Path Encoding Scheme:This involves parsing of the document form the root and selecting 3 suffix nodes successivel to yield a distinct path consisting of exaclty 3 segments. The encoding we use for numbers id based on the **IEEE754 encoding format**.

Partial Reverse Path Encoding Scheme: This is similar to the previously described, however, it works in the reverse order, with the leaf having a higher number of bits in the term, placed first.

Bitmaps as Posting Lists: A postings list captures the document ids of all documents which contain the given term. Two techniques are applied:

1. **Partitioning a Postings List**: Each insertion of a new document to a DocumentDb collection is assigned a monotonically increasing document id.

2. **Dynamix Encoding of Posting Entries**: Within a single partition (pointing by a PES), each document needs only 14 bits which can be captured with a short word.

Customizing the Index:

- There are 4 indexing types: hash, range, spatial and text.
- There are 3 indexing modes: Consistent, Lazy and None.

---

4. Physical Index Organization

These are some constraints that the index maintanance must be performed against:

1. Index update cannot assume any path locality among the incoming documents.
2. Each index update should have the least possible write amplification.

DocumentDB uses a **Bw-Tree**:

- The Bw-Tree uses a latch-free in-memory updates and loog structured storage for persistence, this ensures high degree of concurrency.
- The storage is organized in a log-structured manner.
- It supports **blind incremental update operation** that allows any record to be partially updated without accesing the existing value.

For **index updates** we need to perform document analysis: An analyzer provides basic operators to add and substract 2 document instances.

For **index replication and recovery** DocumentDB follows a single master model for writes, the primary considers the write operation successful if it is durably commited to local disk by a subset called *write quorum*, and similarly, there's a *read quorum*.

- After a failover, the new replica joining the quorum need to be rebuilt from scratch.
- The DocumentDB database engine starts a Bw-tree checkpointing procedure to make all index updates stable up to a highest LSN corresponding to the document update.

---

5. Insights from the production workloads

- **Document frequency distribution** for the unique terms universally follow Zipf's Law.
- The highest value of query precision is 1 and this is a good proxy for *query Performance**.
- **Blind incremental updates** are extremely IO efficient even in the face of index growth.

---

6. Related Commercial Systems

Currently, there are 2 types of commercial NoSQL storage systems broadly available:

1. **Non-document oriented NoSQL storage systems**: They are designed and operated as multi-tenant cloud services with SLAs.
2. **Document databases**: Single tenant systems which run either on-premises or on dedicated VMs in the cloud.

Neither of the 2 types of systems above provide a fully resource governed and schema agnostic database engine to provide automatic indexing while still serving consistent queries.

---

7. Conclusion

With the architecture previously described, DocumentDB is capable of high performance and cost effectiveness, along with performing online and in-situ index transformations. The index subsystem described is currenlty supporting production workloads for several consumer scale applications worldwide.