Resumen 4
Estudiante: Celina Madrigal Murillo
Carné: 2020059364

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.

1.1 Overview of the Capabilities

The query language supports rich relational and hierarchical queries.The database engine is optimized to serve consistent queries in the face of sustained high volume document writes. Transactional execution of application logic provided via stored procedures and triggers, authored entirely in JavaScript and executed directly inside DocumentDB's database engine. DocumentDB offers well-defined and tunable consistency levels for developers to choose from and corresponding performance guarantees.

1.2 Resource Model

Each resource is uniquely identified by a stable and logical URI and is represented as a JSON document.

1.3 System Topology

The DocumentDB service manifests itself as an overlay network of machines, referred to as a federation which spans one or more clusters.

1.4 Design Goals for Indexing

Automatic indexing, Configurable storage/performance tradeoffs, Efficient, rich hierarchical and relational queries, Consistent queries in face of sustained volume of document writes and Multi-tenancy.

2. SCHEMA AGNOSTIC INDEXING

2.1 No Schema, No Problem!

DocumentDB exploits the simplicity of JSON and its lack of a schema specification.

2.2 Documents as Trees

The technique which helps blurring the boundary between the schema of JSON documents and their instance values, is representing documents as trees.

2.3 Index as a Document

Each update of a document to a DocumentDB collection leads to update of the structure of the index.

2.4 DocumentDB Queries

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

3. LOGICAL INDEX ORGANIZATION

The tree representation of documents and the index enables a schema-agnostic database engine.

3.1 Directed Paths as Terms

A term represents a unique path in the index tree.

3.1.1 Encoding Path Information: The choice of encoding scheme for the path information significantly influences the storage size of the terms and consequently the overall index.

3.1.2 Partial Forward Path Encoding Scheme: it involves parsing of the document from the root and selecting three suffix nodes successively to yield a distinct path consisting of exactly three segments.

3.1.3 Partial Reverse Path Encoding Scheme: it selects three suffix nodes successively to yield a distinct path consisting of exactly three segments.

3.2 Bitmaps as Postings Lists

Partitioning a Postings List: Each insertion of a new document to a DocumentDB collection is assigned a monotonically increasing document id.
Dynamic Encoding of Posting Entries: This marks the upper bound on the postings list within a bucket.

3.3 Customizing the Index

Developers can customize the trade-offs between storage, write/query performance, and query consistency, by overriding the default indexing policy on a DocumentDB collection and configuring the following aspects: Including/Excluding documents and paths to/from index, Configuring Various Index Types and Configuring Index Update Modes.

4. PHYSICAL INDEX ORGANIZATION

4.1 The “Write” Data Structure

Index update performance must be a function of the arrival rate of the index-able paths, It cannot assume any path locality among the incoming documents, for documents in a collection must be done within the CPU, memory and IOPS budget allocated per DocumentDB collection, each index update should have the least possible write amplification and each index update should incur minimal read amplification.

4.2 The Bw-Tree for DocumentDB

The Bw-Tree uses latch-free in-memory updates and log structured storage for persistence.

4.2.1 High Concurrency: The Bw-Tree operates in a latch-free manner, allowing a high degree of concurrency in a natural manner.

4.2.2 Write Optimized Storage Organization: Bw-tree storage is organized in a log-structured manner and Bw-tree pages are flushed in an incremental manner.

4.3 Index Updates

4.3.1 Document Analysis: The document analysis function A takes the document content D corresponding to a logical timestamp when it was last updated, and the indexing policy I and yields a set of paths P.

4.3.2 Efficient and Consistent Index Updates: Recall that an index stores term-to-postings list mappings, where a postings list is a set of document ids. A new document insertion requires updating of the postings lists for all terms in that document.

4.3.3 Lazy Index Updates with Invalidation Bitmap: The lazy indexing mode is performed in the background, asynchronously with the incoming writes, usually when the replica is quiescent.

4.4 Index Replication and Recovery

4.4.1 Index Replication: The primary replica receiving the writes analyzes the document and generates the terms.

4.4.2 Index Recovery: The Bw-Tree exposes an API that allows the upper layer in the indexing subsystem to indicate that all index updates below some LSN should be made stable.

4.5 Index Resource Governance

4.5.1 Index Resource Governance: All operations within a DocumentDB's database engine are performed within the quota allocated for CPU, memory, storage IOPS and the on-disk storage.

5. INSIGHTS FROM THE PRODUCTION WORKLOADS

5.1 Document Frequency Distribution

Document frequency distribution for the unique terms universally follow Zipf's Law.

5.2 Schema Variance

Recall that we represent JSON documents as trees, with schema represented by the interior nodes and the instance values represented by the leaves.

5.3 Query Performance

We define query precision in terms of the number of false positives in postings for a given term lookup. Query precision therefore is a good proxy for query performance

5.5 Blind Incremental Updates

Doing highly performant index updates within an extremely frugal memory and IOPS budget is the key reason behind the blind incremental update access method

6. Related Commercial Systems

Non-document oriented NoSQL storage systems which are designed and operated as multi-tenant cloud services with SLAs and Document databases designed as single tenant systems which run either on-premises or ondedicated VMs in the cloud.