Summary 4

> *Gabriela Gutiérrez Valverde - 2019024089*

> *Book: Schema-Agnostic Indexing with Azure DocumentDB*

Overview of the Capabilities

- Query language supports rich relational and hierarchical queries, rooted in JavaScript’s type system, expression evaluation and function invocation model.
- High volume document writes, automatically indexes all documents without requiring schema.
- Well-defined and tunable consistency levels.

Resource Model

- Users
- Permissions
- Collections
- Documents
- Stored Procedures
- Triggers
- User defined functions (UDFs)
- Attachments

System Topology

Federation: overlay network of machines, which spans one or more clusters. Each replica hosts an instance of the DocumentDB’s database engine, which manages the resources.

Design Goals for Indexing

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

SCHEMA AGNOSTIC INDEXING

Schema of a document describes the structure and the type system of the document independent of the document instance. JSON’s type system is a strict subset of the type systems of many modern programming languages. It makes no assumptions about the documents and allows documents within a DocumentDB collection to vary in schema.

- **Documents as Trees:** Representing JSON documents as trees in-turn normalizes both the structure and the instance values across documents into a unifying concept of a dynamically encoded path structure. Both the property names and their values are labels in the tree.
- **Index as a Document:** Every path in a document tree is indexed. Each update of a document leads to update of the structure of the index.
  - *Forward index mapping:* document id, path
  - *Inverted index mapping:* path, document id

    - *Index Tree:* Constructed out of the union of all of the trees representing individual documents within the collection.
- **DocumentDB Queries:** Can use queries written in SQL and JavaScript. Both get translated to an internal intermediate query language called *DocumentDB Query IL*. Supports projections, filters, aggregates, sort, flatten operators, expressions, system provided intrinsics and UDFs.

LOGICAL INDEX ORGANIZATION

Directed Paths as Terms

Forward path starting from each node in the tree to a leaf, or reverse path from leaf to the root. Associated tradeoffs: storage cost, indexing maintenance cost, cost of lookup queries.

- **Encoding Path Information:** Number of segments in each term is an important choice in terms of the trade-off between query functionality, performance and indexing cost. Default: three segments.
- **Partial Forward Path Encoding Scheme:** For non-numeric values, each of the three segments are encoded based on all the characters. The least significant byte of the resultant hash is assigned for the first and second segments. For numbers is based on the IEEE754.
- **Partial Reverse Path Encoding Scheme:** In the reverse order, with the leaf having higher number of bits in the term, placed first.

Bitmaps as Postings Lists

Captures the document ids of all the documents which contain the given term. Dynamic, compact, capable of computing fast set operations.

- **Partitioning a Postings List:** To avoid static reservation of id space partition the postings list into postings entries. Helps, to determine the maximum size of pages and split policy of the B+-tree.
- **Dynamic Encoding of Posting Entries:** postings words within a postings entry are encoded dynamically using a set of encoding schemes including various bitmap encoding schemes.

Customizing the Index

- Including/Excluding documents and paths to/from index
- Configuring Various Index Types (hash, range, spatial, and text)
- Configuring Index Update Modes (Consistent -> updated synchronously, Lazy -> update asynchronously, or None -> no index associated)

PHYSICAL INDEX ORGANIZATION

The “Write” Data Structure

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

- **The Bw-Tree for DocumentDB:** The BwTree uses latch-free in-memory updates and log structured storage for persistence. Multicore processors with multi-level memory/cache hierarchy, and flash memory based SSDs with fast random reads. Completely decouples the logical pages from their physical counterparts. Supports dynamic resizing of its secondary storage file.
- **High Concurrency:** The Bw-Tree operates in a latch-free manner, allowing a high degree of concurrency in a natural manner. A modification to a page is done by appending a delta record on top of the existing portion of the page. Starting location of the page to change after every update.
- **Write Optimized Storage Organization:** Bw-tree storage is organized in a log-structured manner; and pages are flushed in an incremental manner.

Index Updates

- **Document Analysis:** The document analyzer vastly simplifies the processing of delete and replace operations in consistent indexing.
- **Efficient and Consistent Index Updates:** The Bw-Tree in DocumentDB was extended to support a new blind incremental update operation. When a lookup comes to a Bw-Tree page on a given key k, the whole page is read from storage and the multiple fragments of the page describing base value and updates to key k are combined using a merge callback function.
- **Lazy Index Updates with Invalidation Bitmap:** performed in the background, asynchronously with the incoming writes.

Index Replication and Recovery

- **Index Replication:** the primary replica receiving the writes analyzes the document and generates the terms, then applies it to its database engine instance as well as sending the stream containing the terms to the secondaries. Each secondary applies the terms to its local database instance.
- **Index Recovery:** an index checkpointing is optimized to ensure that the crash recovery requires minimum amount of index to be rebuilt. Limits the amount of the log that needs to be scanned during Bw-Tree recovery.

Index Resource Governance

- CPU resources: designed to be fully asynchronous and written to never block a thread.
- Memory resources: operates within a given memory budget that can be adjusted dynamically.
- Storage IOPS resources: maintains a running average of IOPS usage over time.
- On-disk storage: Size is given by -> number of bytes per term, the PES, document frecuency, and compression factor.

INSIGHTS FROM THE PRODUCTION WORKLOADS

- **Document Frequency Distribution:** document frequency distribution for the unique terms universally follow Zipf’s Law.
- **Schema Variance:** regardless of the workload, document or collection size, the number of unique leaf nodes completely dwarf the number of interior nodes.
- **Query Performance:** precision is good proxy for query performance in terms of the number of false positives in postings for a given term lookup.
- **Blind Incremental Updates:** highly performant index updates within an extremely frugal memory and IOPS budget.