Docussandra Design and Architecture

An Introduction for Developers

Theory of Operation

- Basic introduction:
  - Docussandra is a MongoDB-like document (JSON) datastore that is built on top of Cassandra and accessed via a REST API.
  - Offers better performance and scaling over MongoDB.
- Users/systems create databases and tables via REST calls. They can then upload documents to the tables they have defined. No schema design is needed.
- Data can be searched only after indexes are defined via a REST call against the table. Indexes can be deleted when no longer needed.

Technology Stack

- **Programming Language:** Java, built with Maven.
- **Database Layer:** Cassandra; any recent version (< 2.0.9) should work. The goal should be to keep Docussandra Cassandra version agnostic (within reason)
- **Application Layer:** Based on RestExpress (which in turn is based on Jetty).
- **Target OS:** The application itself should be platform independent, however, only Ubuntu packages will be offered pre-built at this time.

Deployment

- The Docussandra application can be deployed co-hosted on the same machine as the Cassandra nodes OR in a separate application layer, depending on the scaling and performance needs.
- Co-hosting with Cassandra could result in slightly lower transaction times, but would need to be balanced with more hardware and may complicate scaling.

Software Application Layers

- **Controller Layer:** Entry layer for REST calls into the application. Contains logic for parsing the request and the response. This is the “REST” sub-project.
- **Service Layer:** Receives requests from the Controller layer, performs any needed basic data verification and any rules that need to be logically enforced. Contained in the “Docussandra-Cassandra” sub-project.
- **Repository Layer:** Mainly Database Access Objects (DAOs) Receives requests from the Service Layer and persists data to Cassandra as needed. Contained in the “Docussandra-Cassandra” sub-project.
- **Command Line Interface (CLI):** Not yet started. This is the “Docussandra-CLI” sub-project.

Initial Cassandra ColumnFamilies (Tables)

- sys_meta – not presently used
- sys_db – stores databases metadata
- sys_tbl – stores table metadata
- sys_idx – stores index metadata
- sys_idx_status – contains information about the status of indexes
- sys_idx_not_done – contains a list of indexes that are presently indexing

Introduction to Indexing

- Cassandra's internal indexing is usually not a good practice; the solution is to make full copies of your data with different primary keys for each way that you will need to query the data.
- Docussandra automates the creation of these additional tables for indexing; we call them “iTables”.

Bucketing for Better Indexing

- Cassandra stores data in order based on primary key, however, most applications use a random UUID, which results in a “skatter-gather” across multiple Cassandra nodes. If you don't use a random primary key, you run the risk of “hot-spots” where one node tends to do the majority of the processing.
- Bucketing solves both of these problems; a “bucket” is assigned to each record based on the first several bytes of the first primary key in the document. This ensures that similar documents are stored, in order, on the same physical node.
- Our bucketing algorithm currently needs improvement.

Details of Docussandra Storage

- When a new Docussandra table is created:
  - An entry is added to the sys_table table.
  - A new Cassandra table is created with the naming format of: $dbName_$tableName.
    - The table has a random UUID as a primary key, has some fields for metadata, and a field that holds JSON as a blob.
    - All documents for this Docussandra table are automatically stored in this table.
    - We refer to this as the “main table” for an Docussandra table.

Details of Docussandra Storage Con't

sys_db
mydatabase

sys_table
mydatabase/mytable

sys_idx

sys_idx_status

sys_idx_not_done

mydatabase_mytable
{“mypersonrecord”:1...
{“mypersonrecord”:4...
{“mypersonrecord”:3...

Details of Docussandra Storage con't

- Using this example, let's look at what happens when an index is created on this table. Assume that an index, called “person” is created on the “mypersonrecord” field:
  - An entry is added to the sys_idx table.
  - A table called “mydatabase_mytable_person” is created with a composite primary key consisting of the bucket id and a integer field to store the value of “mypersonrecord” for all of the documents. This is the “iTable” for this index.
  - The index creation call returns to the user, then all documents currently in the mydatabase_mytable table are asynchronously *copied* over to this new table by the “IndexCreatedHandler” (ordered by the indexed field). Updates for the status of this operation are written to the sys_idx_not_done and sys_idx_status tables.

Details of Docussandra Storage Con't

sys_db
mydatabase

sys_table
mydatabase/mytable

sys_idx
person

sys_idx_status
person: active

sys_idx_not_done

mydatabase_mytable
{“mypersonrecord”:1...
{“mypersonrecord”:4...
{“mypersonrecord”:3...

mydatabase_mytable_person		
<bucketid>	1	{“mypersonrecord”:1...
<bucketid>	3	{“mypersonrecord”:3...
<bucketid>	4	{“mypersonrecord”:4...

Details of Docussandra Storage con't

- Using this example, let's look at what happens when a new record is added to a table that already has an index.
  - The record is inserted into the “main” (mydatabase_mytable) table.
  - The record is *duplicated* into the iTable, ordered by the indexed field.

Details of Docussandra Storage Con't

sys_db
mydatabase

sys_table
mydatabase/mytable

sys_idx
person

sys_idx_status
person: active

sys_idx_not_done

mydatabase_mytable
{“mypersonrecord”:1...
{“mypersonrecord”:4...
{“mypersonrecord”:3...
{“mypersonrecord”:2...

mydatabase_mytable_person		
<bucketid>	1	{“mypersonrecord”:1...
<bucketid>	2	{“mypersonrecord”:2...
<bucketid>	3	{“mypersonrecord”:3...
<bucketid>	4	{“mypersonrecord”:4...

Details of Docussandra Storage con't

- As you can see, for each additional index created, the storage used and computation needed for an insert is increased by a factor of “n”.
  - Due to this, it is advisable to not leave indexes in place if they are not presently being used. Indexes can always be recreated if needed at a later time. (Although, some lead time may be needed)

Details of Docussandra Searching

- When a query is received by Docussandra, it is parsed and it is determined if it is possible to complete the query based on the available indexes.
  - If it is not possible to complete, an error is returned to the user advising them to consider creating an index.
  - Docussandra determines which iTable it needs to query to get the data it needs
  - The bucket in which the queried data resides in is then computed.
  - Due to the fact that iTables have primary keys for the field being searched, the query is a simple select * from $iTable where bucketId = ? and $field = ?. This is extremely fast in Cassandra.

Questions?