Understanding NoSQL Database Implementations

Sadalage and Fowler, Chapters 7 - 11

Foreword

- NoSQL is a broad and diverse collection of technologies.
- It is not possible to provide in-depth coverage of the entire spectrum of NoSQL technology implementations in an introductory course such as this one.
- This topic provides a high level overview of the implementation considerations for representative NoSQL technologies.

Hadoop

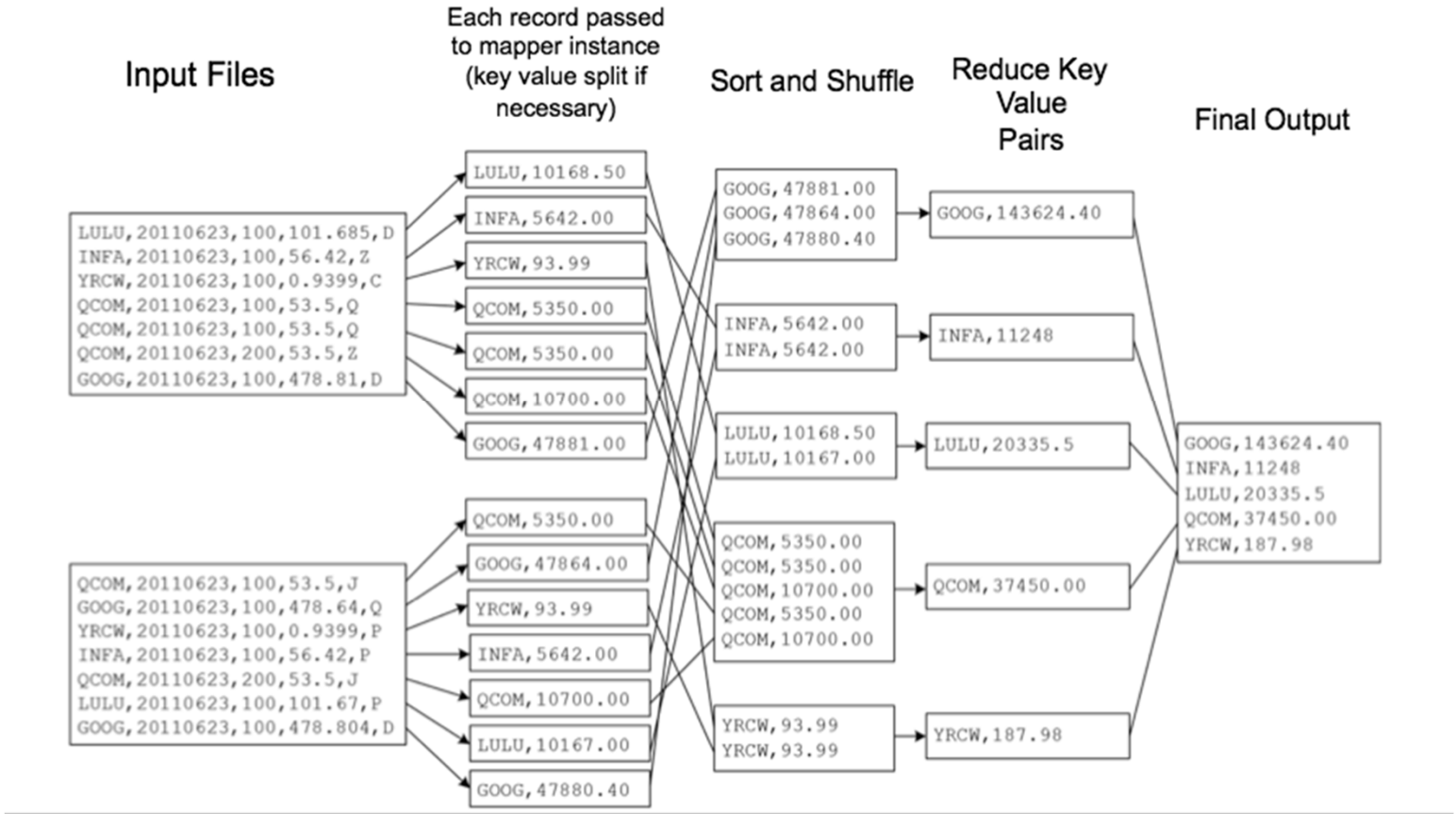
Hadoop Implementation

- Apache Hadoop is an open-source framework for distributed storage and processing of unstructured data on computer clusters constructed using commodity hardware.
- Core components:
  - Hadoop Distributed File System (HDFS)
    - Manages the distribution and storage of data files on the cluster server in a fault-tolerant manner
    - Other file systems such as Amazon S3 are also supported
  - MapReduce (next slide)

MapReduce

- MapReduce is a framework for breaking files into smaller blocks and distributing the blocks across the cluster.
  - Enables parallel processing, -- essential for efficiently handling large volumes of data (so-called “big data”).
  - Mapping processes separate data file records into key-value pairs, and prepares the data for the reduce process.
  - Reduce process combines key-value pairs in order to derive one record (block) per key
  - Results are stored as a file that is prepared for analysis

MapReduce Process Example



MapReduce Example Details

- Input files contain historical stock trade transactions (comma-separated values):
  - Stock Symbol
  - Trade Date (YYYYMMDD format)
  - Number of Shares traded
  - Price Per Share
  - Stock Class
- Requirement: Find the total value of the shares traded (calculated by multiplying the number of shares by the price per share)
  - For each stock symbol
  - Regardless of the date or stock class.

MapReduce Example Process

1. Extracts the Stock Symbol field and Calculates the Total Value field.
   - Each record from the source files is handled by an independent mapper instance.
   - When input file records contain multiple occurrences of the desired fields, a key value split mapper task would also be required to produce multiple key-value pairs from each input record (Unnecessary in this case)

MapReduce Example Process

2. Sorts the key-value pairs by the keys, and “shuffle” keys to individual nodes (computers)
   - Ensures each node contains the key-value blocks for just one key value
   - Each source file record is handled by an independent mapper instance
   - When input file records contain multiple occurrences of the desired fields, a key value split mapper task would also be required to produce multiple key-value pairs from each input record. (Unnecessary in this case)

MapReduce Example Process

3. Reducer process performs some form of aggregation of the blocks for each key value, producing a single output record for each key.
   - In this example, we are summing the values for each key
   - Other possible reductions include finding:
     - Minimum value
     - Maximum value
     - Average value
     - Record count
     - Any other function that yields a single value per key value

Hadoop Ecosystem

- The ***Hadoop Ecosystem*** is a collection of additional software packages that can be installed on top of or alongside Hadoop
- Available packages (partial list, details follow):
  - Hive
  - Pig
  - Apache YARN
  - Apache Spark
  - JAQL

Hive

- Hive: a data warehouse infrastructure component that supports analysis of large datasets stored in Hadoop's HDFS, or compatible file systems such as Amazon S3.
  - Founded by Jeff Hammerbacher while he was working at Facebook.
  - Incudes an SQL-like query language (HiveQL) that can be used;
    - To mine large amounts of data
    - To transform queries into MapReduce, Apache Tez, and Spark jobs.

Other Features of Hive

- Indexing, primarily using bitmap indexes, although others are planned
- Support for file types beyond the text files supported by HDFS, including HBase, RCFile, ORC, and others
- Metadata storage in an RDBMS, such as MySQL or Apache Derby, to speed semantic checking of data
- Direct support for compressed data stored in HDFS using various algorithms
- Built-in user defined functions (UDFs) for supporting dates, strings, and additional data mining tools

Pig

- **Pig**: a high-level tool for creating MapReduce programs.
  - Originally developed at Yahoo Research around 2006
  - Moved into the Apache Software Foundation in 2007.
  - *Pig Latin* is the language used in Pig.
    - A pipeline language, similar to writing extract, transform, and load steps in ETL.
    - In contrast, SQL is considered a declarative language.

Other Features of Pig

- Extensions to Pig Latin with user defined functions (UDFs) written in Java, Python, JavaScript, Ruby, or Groovy.
- Ability to store data at any point in the process.
- Support for major data operations such as ordering, filtering and joins.
- A declared execution plan.
  - In SQL, the DBMS query optimizer evaluates each query and chooses an execution plan, so the SQL programmer has less control over operations such as joins.
  - In Pig Latin, the programmer declares how the joins and other operations will be executed.

Other Features of Pig (cont.)

- The ability to split data pipelines so that data need not be processed in a strictly sequential manner.
- Support for a wide range of data types that are not present in MapReduce.

Apache Yarn

- **Apache YARN** (Yet Another Resource Negotiator) is a resource-management platform used for:
  - Managing computing resources across cluster servers
  - Scheduling applications.

Apache Spark

- **Apache Spark** is a data analytics cluster computing framework
  - Originally developed in the AMPLab at UC Berkeley. Spark builds on top of Hadoop HDFS
  - Provides an alternative to Hadoop MapReduce that is easier to use
  - Offers performance that is 10 times faster than MapReduce for certain applications.

JAQL

- **JAQL** is a declarative programming language
  - Designed for processing large volumes of structured, semi-structured, and unstructured data
  - Primarily used for handling data stored in JSON documents
  - Capable of handling other types of data such as XML, comma-separated value (CSV) data, and flat files

Columnar Databases

Columnar Databases

- Columnar databases organize and process data by columns
  - Relational databases organize and store data by row
  - Wide variety of columnar technology on the market
    - Each product has a unique set of physical characteristics.
    - Popular products:
      - Apache HBase
      - Apache Cassandra
      - Hypertable
      - HP Vertica (Clearly columnar, but perhaps not NoSQL because it uses standard SQL)

Example: Relational Table

ROWID	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	SALARY	COMMISSION_PCT
1	173	Sundita	Kumar	6100	0.1
2	172	Elizabeth	Bates	7300	0.15
3	171	William	Smith	7400	0.15
4	170	Tayler	Fox	9600	0.2
5	169	Harrison	Bloom	10000	0.2
6	168	Lisa	Ozer	11500	0.25
7	150	Peter	Tucker	10000	0.3
8	151	David	Bernstein	9500	0.25
9	149	Eleni	Zlotkey	10500	0.2
10	145	John	Russell	14000	0.4
11	147	Alberto	Errazuriz	12000	0.3
12	146	Karen	Partners	13500	0.3
13	152	Peter	Hall	9000	0.25
14	155	Oliver	Tuvault	7000	0.15
15	153	Christopher	Olsen	8000	0.2

Example: Columnar Storage

ROWID	EMPLOYEE_ID
10	145
12	146
11	147
16	148
9	149
7	150
8	151
13	152
15	153
17	154
14	155
18	156
23	157
22	158
21	159

- ROWID column is an internal identifier used by most relational DBMS products to identify each row of data.
  - Exact name of the pseudo-column varies from one product to another.
- While ROWID is used here, other methods such as RBA pointers are used by some implementations.

Columnar DBMS Queries

- With row-oriented databases, the DBMS reads rows of data, and scans rows to find the columns selected in a query.
- With columnar databases, the DBMS only needs to read the selected columns, which speeds up processing considerably, particularly for tables with many columns.
- Column stores can be ordered by the column values, eliminating the need for external indexes.

Columnar DBMS Physical Design Considerations

- **Column Grouping**: which columns to store together (as groups of columns).
  - Columns that are usually used together can be more efficiently processed if stored together.
  - However, you then must choose which column is used for ordering, and you may lose other optimization options.
- **Column Ordering**: which columns are maintained in data sequence so they can function as indexes.
  - The tradeoff is the overhead of keeping the data in order as rows are inserted, updated, and deleted.

Columnar DBMS Physical Design Considerations

- **Column Compression**: Single columns are easier to compress than entire rows, so compression works better in columnar database. Techniques include:
  - Storing repeating values once, followed by how many repetitions of the value the DBMS removed during compression.
  - For columns with uniform intervals of values (such as identifiers that usually increase by 1 for each new row), store the first value along with the interval that is to be repeated until another value appears.

Columnar DBMS Physical Design Considerations

- **Distribution**: how the column data is to be distributed across the nodes in the cluster.
  - A common method is to hash the column values. (Recall that hashing maps data of arbitrary sizes to values of a fixed size, typically in the form of binary or integer values).
  - Another common distribution method is to assign ranges of values to specific nodes.
  - The key to join performance in clustered databases is to place values that will be joined on the same node so that data does not have to be transmitted (broadcast) from one node to another in order to complete the join operation.

Document Databases

Document Databases

- MongoDB is, by many measures, the most popular document database.
- MongoDB instance can manage multiple databases and each MongoDB database can contain many collections
  - MongoDB collection is similar to a relational table in that the documents in a collection describe different instances of the same entity
  - When storing a document in MongoDB, you must specify the database and the collection to which the document belongs.

Document Databases

- Structure of documents can vary within the same collection
  - One document in a collection can have data elements that do not exist in other documents in the same collection
  - Very different compared to relational
  - Not quite the same as NULL values in a relational database
    - NULL values indicate columns for which the value is unknown
    - Documents with missing data elements carry no assumed meaning
      - Could be an older version of the document, for example

MongoDB Storage Options

- MongoDB supports replicating data by defining replica sets, typically with each replica set going to a different node in the cluster, with one replica (copy) in each set designated as the master.
  - Write consistency is controlled by specifying the number of replicas in a set that must be written before a write operation is considered complete.
- For larger databases, documents can be distributed (aplit) across cluster nodes by specifying a field within the document to be used for determining the distribution.
  - In MongoDB, the term used for distribution is ***sharding***.

MongoDB Consistency

- MongoDB does not support the concept transactions that can span multiple documents such as a series of insert, update and delete operations across multiple documents.
- The only operations that can be atomic (completely successful or having no effect) are those that are applied to the same document.

MongoDB Queries

- MongoDB has a query language that is expressed via JSON using constructs to specify filtering, ordering, and specifying an execution plan.
  - Sample query in SQL

```
SELECT firstName, lastName
  FROM employee
 WHERE employeeId = '147';
```

  - MongoDB equivalent:

```
db.employee.find({employeeId:"147"},
                 {firstName:1,lastName:1})
```

Key-value Databases

Key-value Databases

- Stores key-value pairs where the value is a blob (binary large object)
  - DBMS (or file system) unaware of blob structure
  - The simplest physical structures of all the NoSQL databases
  - Lends itself to in-memory databases
- Database user has only three possible operations:
  - Get (read) the value for a key
  - Put (write) the value for key
  - Delete the key from the data store

Consistency

- Concept of consistency applies only to operations on a single key because each key-value pair is stored separately from all others.
  - The database does not support relationships between different keys, so each key-value pair is independent of all others.
  - For distributed key-value databases, such as Riak, replication of key-value pairs is handled asynchronously. Therefore, at any point in time, some nodes in the cluster will be inconsistent with other nodes.
    - This consistency model if called ***eventual consistency***.

Transaction Support

- Support for transactions varies quite a bit from one product to another
  - In general, there are no guarantees on writes because of the eventual consistency model.
  - However, some products offer support for a minimum number of clusters that must be written to before a write is considered complete.
    - Riak, for example, uses the concept of quorums, where a write is considered complete when a put has been completed on a majority of the applicable nodes

Queries

- Queries are generally limited to searching (getting) values using their corresponding keys.
  - Most implementations support search based on the contents of the values, but only on the keys.
  - However, a few products do allow searching on the values.
    - Riak includes a feature known as Riak Search that permits searching based on characteristics of the stored values.

Scalability

- Most key-value stores support the concept of sharding, with the key being used to determine the node on which the key-value pair will be stored.
  - One downside of sharding occurs when a node is unavailable.
    - For example, if the node used to store keys starting with “e” goes down, we cannot retrieve any values for keys starting with “e”, nor can we write any values that have keys starting with “e”.

Graph Databases

Graph Databases

- Graph databases are radically different from other databases in that they focus more on storing the relationships (edges) between records of values (nodes) than they do on storing the values themselves
  - Don’t confuse graph database nodes with cluster nodes

Storage Considerations

- Usually store only one type of relationship, such as “which condition was this medication prescribed to treat?”.
- Adding additional relationships to an existing graph database usually leads to a lot of schema changes and movement of data.
- Relationships between nodes are created in both directions because relationships are not necessarily bi-directional.
  - For example, a medication might be prescribed to treat a medical condition, but the medication might not be the recommended treatment for the condition.

Data Consistency

- Most graph databases do not support distributing (sharding) nodes across different servers in the cluster.
  - The underlying issue is that nodes (records) are connected to many other nodes, so you cannot easily partition nodes such that all references are on the same partition.
- However, some support replication with updates to slave copies following the eventual consistency model.

Transactions

- Some graph databases support transactions.
  - Neo4J provides full transaction support.
    - Before any changes to nodes or relationships in Neo4J, a transaction must be started.
    - Transactions are completed by marking them successful, followed by marking them completed by invoking a finish method.
    - Data changes are not committed until the finish is issued. If a transaction fails, it is rolled back to the beginning, much like relational database transactions.
    - Reads, however, can be performed completely outside of transactions.

Availability

- Neo4J uses the master-slave paradigm (a master and one or more replicated slaves) to support availability.
  - Writes are committed to the master first, then one slave.
  - Additional slaves follow the eventual consistency model.
- Other graph databases, such as FlockDB and Infinite Graph, support distributing nodes across the servers in the cluster.

Queries

- Neo4J includes the Cypher query language.
  - Neo4J also supports language bindings for querying a graph to find properties of nodes, traversing a graph, and navigating relationships associated with nodes.
  - Gemini is a language specifically designed for traversing graphs, and is capable of traversing all databases that implement the Blueprints property graph, which includes not only Neo4J, but most other graph database products.

Scalability

- As mentioned earlier, sharding does not work well because records (nodes) are related to so many others
- Scaling techniques include:
  - Add sufficient memory so that active nodes and relationships will usually fit in memory
  - Add more slaves to allow slaves to share the load for reads, while restricting all write operations to the master.
  - If sharding must be done, distribute the nodes based on subsets of the nodes
    - This technique requires domain knowledge of the data as well as an understanding of how the graph database is typically used.des that always tend to be used together.