A Review of Column-Oriented Datastores

By: Zach Pratt

Spring 2011

Introduction

- The problem: Internet-scale applications have additional requirements over and above those of lesser scale applications.
  - Reliably storing session state and application data for millions of users as apposed to thousands of users.
  - Handle high volumes of data by using dynamic partitioning
  - Replication for high availability
  - No single point of failure

RDBMS Limitations

- Traditional relational database systems aren’t designed with the requirements of Internet-scale applications in mind.
  - Partitioning must be part of the initial logical data model, if its even supported.
  - Replicating to many nodes while maintaining strong consistency limits availability.
  - Single points of failure

Solution

- Use a non-relational datastore (NoSQL) that is designed with the following in mind:
  - Dynamic partitioning
  - Large scale replication
  - Totally decentralization
- There are many; I will cover a few column-oriented versions. See http://nosql-database.org/ for a big list of other implementations.

Background

- There are many differences between relational databases and column-oriented datastores
  - Data models
  - CRUD operations
  - Support for ACID properties

Data Models

- Relational vs Column-Oriented
  - Relational:
    - DDL to define tables and constraints
    - Schema contained within the database
    - Rows of columns make up tables
  - Column-Oriented: Groupings of key-value pairs
    - Schema is primarily procedural
    - Groups of key-value pairs instead of tables

Example Data Model

BigTable and Cassandra

```
row_key: {
    key_name1:key_value1
    , ...
    , key_nameN:key_valueN
}
```

```
edu.wiu.appserver: {
    architecture:x86_64
    num_cpus:4
    memory:16GB
    ip_address:10.200.15.4
}
```

Dynamo

```
key_name1:blob1
...
key_nameN:blobN
```

CRUD Operations

- Relational
  - DML for inserts, updates, deletes
  - SQL for queries
  - Both are fairly standard across RDBMS's.
- Column-Oriented
  - Each datastore has its own client-side API
  - get() for queries
  - put() for inserts and updates

ACID Properties

- ACID: **A**tomicity, **C**onsistency, **I**solation, **D**urability
- Atomicity: all or nothing transactions
- Consistency: ensure the data is valid
- Isolation: handling concurrent operations
- Durability: recover committed data after a failure

Theory

- Important column-oriented principles:
  - CAP Theorem
  - Eventual consistency
  - Consistent hashing

CAP Theorem

- Initially proposed by Eric Brewer
- Distributed systems of sufficient size cannot maintain:
  - **C**onsistency
  - **A**vailability
  - **P**artition tolerance
- Trade-offs must be made between consistency and availability in order to maintain high availability.

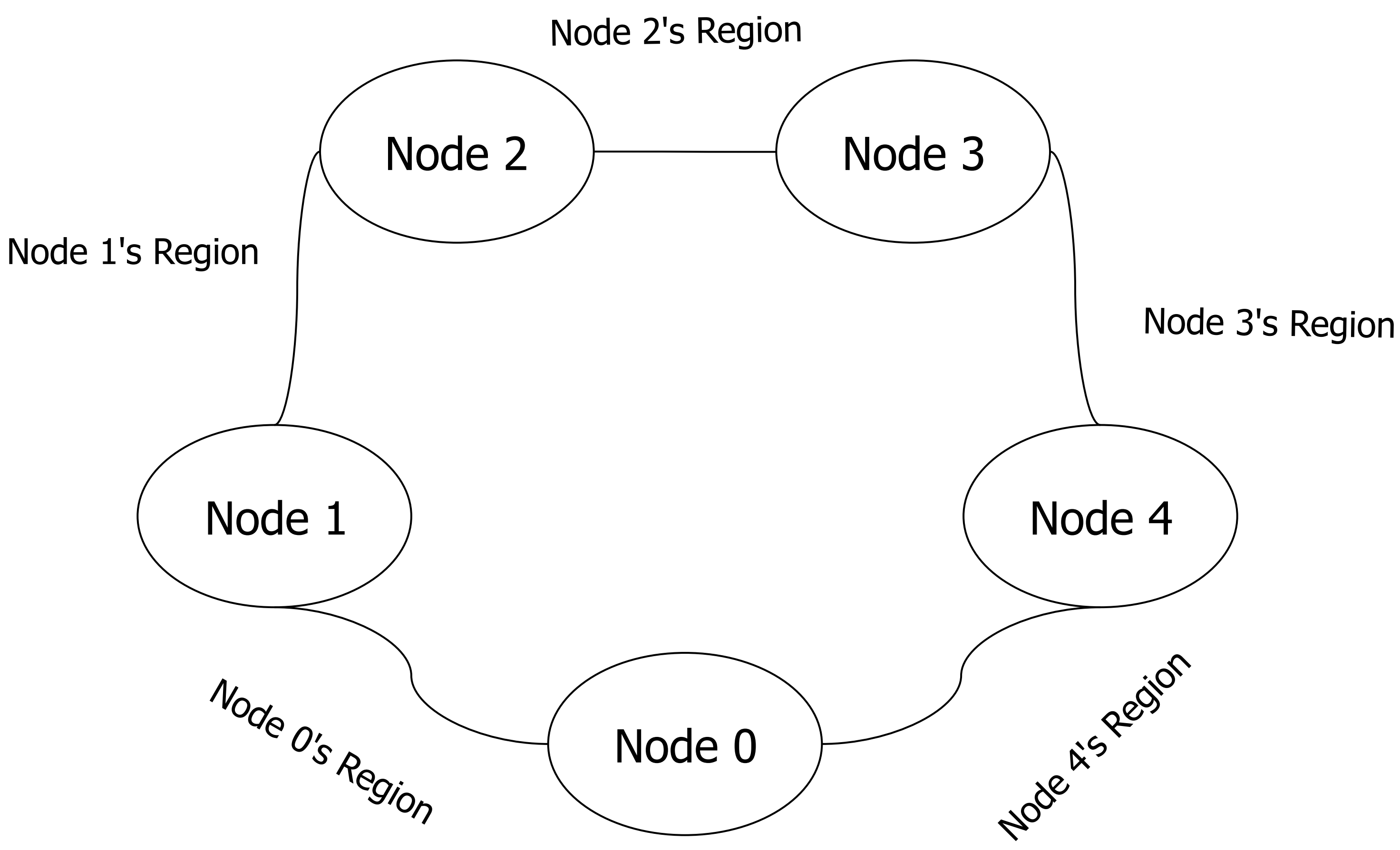
Eventual Consistency

- Strong consistency limits availability
  - Especially synchronous replication
- Eventual consistency
  - Data on all nodes will be consistent... eventually
  - Applications can specify their required level of consistency for reads and writes.

Consistent Hashing

- Used to support dynamic partitioning
  - Each node is assigned a token, which represents a range within the hash space for which it is responsible.
  - Keys are hashed and assigned to the node handling the range in which the hashed key falls under

Conceptual Hash Space



Implementations

- Google's BigTable
- Amazon's Dynamo
- Facebook's Cassandra

Google's BigTable

- Focuses on dynamically partitioning data
  - Does not focus on fault tolerance, therefore, replication is not a feature
- Data Model: Column Keys, Column Families, Rows, Timestamps
  - Semi-structured data
- Partitions data by row key range amongst a cluster of <u>tablet</u> servers.

Amazon's Dynamo

- Totally Decentralized
- Focuses on dynamic partitioning, large scale replication, fault tolerance
- Data Model: Single namespace for key-value pairs which results in an unstructured data model.
- Eventually consistent
- Has no security.

Facebook's Cassandra

- Combines parts of BigTable and Dynamo
- BigTable components:
  - Data Model: Column Keys, Column Families: Simple and Super, Rows
  - Durability: Memtables and SSTables with a commit log
- Dynamo components: eventual consistency, dynamic partitioning using consistent hashing.

Real World Use Case

- Storing IPTables log files from multiple servers
  - Log messages are often formatted as key-value pairs.
  - This solution would require a high write throughput.
  - Store and analyze large volumes of data

Use Case Data Model

```
192.168.1.2_Apr-23-18:33:12 {
    FROMHOST-IP: 192.168.1.2
    IN: eth0
    OUT:
    MAC: 00:00:00:00:00:00
    SRC: 192.168.1.8
    DST: 192.168.1.2
    LEN: 200
    TOS: 0x00
    PREC: 0x00
    TTL: 54
    ID: 3000
    PROTO: UDP
    SPT: 53
    DPT: 31232
    }
```

```
192.168.1.23_Apr-23-18:32:08 {
    FROMHOST-IP: 192.168.1.23
    IN: eth0
    OUT:
    MAC: 00:00:00:00:00:00
    SRC: 192.168.1.8
    DST: 192.168.1.23
    LEN: 40
    TOS: 0x00
    PREC: 0x00
    TTL: 32
    ID: 561
    PROTO: TCP
    SPT: 8080
    DPT: 30215
    WINDOW: 65535
    RES: 0x00
    FLAG: SYN
    }
```

Conclusion

- Non-relational column-oriented systems have specific use cases.
- They are not meant to replace relational database systems
  - Some applications require transaction handling and strong consistency
    - Banking Applications
- They are often used together