CS535 BIG DATA

PART 2. SCALABLE FRAMEWORKS FOR REAL-TIME BIG DATA ANALYTICS

2. SERVING LAYER: CASE STUDY—CASSANDRA

Sangmi Lee Pallickara
Computer Science, Colorado State University
http://www.cs.colostate.edu/~cs535

Today’s topics

- Cassandra
  - Partitioning

RandomPartitioner

- RandomPartitioner was the default partitioner prior to Cassandra 2.1
  - Uses MD5
    - 0 to 2127-1

ByteOrderPartitioner

- This partitioner orders rows lexically by key bytes
- The ordered partitioner allows ordered scans by primary key
  - If your application has user names as the partition key, you can scan rows for users whose names fall between Jake and Joe

- Disadvantage of this partitioner
  - Difficult load balancing
  - Sequential writes can cause hot spots
  - Uneven load balancing for multiple tables

FAQs

- Midterm (Nov. 16)
  - 9:30AM–10:45AM  CSB425

- PA2 demo: 11/16–11/17
  - www.SignUpGenius.com/go/30E0E4EACAE29ABFD4-pa2
### Replication
- Provides high availability and durability
- For a replication factor (replication degree) of N
  - The coordinator replicates these keys at N-1 nodes
  - Client can specify the replication scheme
    - Rack-aware/Rack-unaware/Datacenter-aware
- There is no master or primary replica
- Two replication strategies are available
  - SimpleStrategy
    - Use for a single data center only
  - NetworkTopologyStrategy
    - Multi-data center setup

### SimpleStrategy
- Used only for a single data center
- Places the first replica on a node determined by the partitioner
- Places additional replicas on the next nodes clockwise in the ring without considering topology
- Does not consider rack or data center location

### NetworkTopologyStrategy
- For the data cluster deployed across multiple data centers
- This strategy specifies how many replicas you want in each data center
- Places replicas in the same data center by walking the ring clockwise until it reaches the first node in another rack
- Attempts to place replicas on distinct racks
- Nodes in the same rack (or similar physical grouping) often fail at the same time due to power, cooling, or network issues.

### Write Consistency Levels
- **ALL**
  - A write must be written to the commit log and memtable on all replica nodes in the cluster for that partition key
- **EACH_QUORUM**
  - Strong consistency. A write must be written to the commit log and memtable on a quorum of replica nodes in all data center
  - This configuration tolerates the failure of a single node per replication group and still allows local reads at a consistency level of ONE.
- **QUORUM**
  - A write must be written to the commit log and memtable on a quorum of replica nodes
  - This configuration tolerates the failure of a single node per replication group at a strong consistency level of LOCAL_QUORUM
- **ONE**
  - A write must be written to the commit log and memtable of at least one replica node
  - This configuration tolerates the failure of one node per replication group at a strong consistency level of ONE
- **TWO**
  - A write must be written to the commit log and memtable of at least two replica nodes
  - This configuration tolerates the failure of one node per replication group at a strong consistency level of LOCAL_QUORUM
- **THREE**
  - A write must be written to the commit log and memtable of at least three replica nodes
  - This configuration tolerates the failure of one node per replication group at a strong consistency level of LOCAL_QUORUM
What are Vnodes?

- With consistent hashing, a node owns exactly one contiguous range in the ring-space
- Vnodes change from one token or range per node, to many per node
- Within a cluster these can be randomly selected and be non-contiguous, giving us many smaller ranges that belong to each node

Advantages of Vnodes

- Example
  - 30 nodes and replication factor of 3
  - A node dies completely, and we need to bring up a replacement
  - A replica for 3 different ranges to reconstitute
  - 1 set of the first natural replica
  - 2 sets of replica for replication factor of 3
  - Since our RF is 3 and we lost a node, we logically only have 2 replicas left, which for 3 ranges means there are up to 6 nodes we can stream from
  - With the setup of RF3, data will be streamed from 3 other nodes total

Restoring a new disk with Vnodes

- Process of restoring a disk
  - Validating all the data and generating a Merkle tree
  - This might take an hour
  - Streaming when the actual data that is needed is sent
  - This phase takes a few minutes

- Advantage of using Vnodes
  - Since the ranges are smaller, data will be sent to the damaged node in a more incremental fashion
  - Instead of waiting until the end of a large validation phase
  - The validation phase will be parallelized across more machines, causing it to complete faster

- If vnodes are spread throughout the entire cluster
  - Data transfers will be distributed on more machines
The use of heterogeneous machines with Vnodes

- Newer nodes might be able to bear more load immediately
  
  - You just assign a proportional number of vnodes to the machines with more capacity
  
  - e.g. If you started your older machines with 64 vnodes per node and the new machines are twice as powerful, give them 128 vnodes each and the cluster remains balanced even during transition

Use of Gossip in Cassandra

- Peer-to-peer communication protocol
  
  - Periodically exchange state information about nodes themselves and about other nodes they know about
  
  - Every node talks to up to three other nodes in the cluster

  - A gossip message has a version associated with it
  
  - During a gossip exchange, older information is overwritten with the most current state for a particular node

What is gossip?

- Broadcast protocol for disseminating data
  
  - Decentralized, peer-to-peer networks
  
  - 'epidemic'
  
  - Fault tolerant

  - Epidemic broadcast protocol provides a resilient and efficient mechanism for data dissemination

  - Cassandra uses gossip for peer discovery and metadata propagation

Why gossip for Cassandra?

- Reliably disseminate node metadata to peers
  
  - Cluster membership
  
  - Heartbeat
  
  - Node status
  
  - Each node maintains a view of all peers
What gossip is not for in Cassandra?
- Streaming
- Repair
- Reads/write
- Compaction
- Hint
- CQL (Cassandra Query Language) query parsing/execution

Data structure
- HeartBeatState
- ApplicationState
- EndpointState
  - Wrapper of a heartbeat state and a set of application state

HeartBeatState
- Generation
- Heartbeat
  - Periodically update monotonically increasing value

Application state
- (enum_name, value, version)
  - Contained as a map in EndpointState per peer

ApplicationState enum
- DCRACK
  - Where you are
- SCHEMA
- LOAD
  - Updated every 60 seconds
- SEVERITY
  - I/O load
- STATUS

Status (AppState)
- Bootstrap
  - For new nodes
- Hibernate
- Normal
- Leaving/Left
- Removing/Removed
Gossip messaging

- Every second, each node starts a new round

- Peer selection (1-3 peers)
  - Live peer
  - Seed (maybe)
  - Unreachable peer (maybe)

SYN: GossipDigestSynMessage

- Initiator sends a digest of all the nodes it knows about to a peer

  - HeartBeatState
    - Generation stays the same when server is running and grows every time the node is address changes
    - Each node has one HeartBeatState associated with it

ACK: GossipDigestActMessage

- Peer receives GossipDigestSynMessage

  - Sort gossip digest list according to the difference in max version number between sender’s digest and own information in descending order
  - Handle those digests first that differ mostly in version number
  - Produces a diff and sends back an ACK
  - Diff contains
    - Map of APPStates (for any node) that the peer has which the initiator does not
    - Digest of nodes (and their corresponding metadata) which a peer needs from an initiator
ACK2: GossipDigestAct2Message

- Initiator receives ACK
- Applies any AppState and sends back an ACK
- ACK2 has a map of AppStates which the peer does not have

```plaintext
10.0.0.1:
- AppState: "load-information": 5.2, generation 1259909635, version 45
- AppState: "bootstrapping": bxLpassF3XD8Kyks, generation 1259909635, version 56
- AppState: "normal": bxLpassF3XD8Kyks, generation 1259909635, version 87
- HeartBeatState, generation 1259909635, version 325

10.0.0.3:
- AppState: "load-information": 12.0, generation 1259912238, version 3
- HeartBeatState, generation 1259912238, version 3

10.0.0.4:
- AppState: "load-information": 6.7, generation 1259912942, version 3
- AppState: "normal": bj05IVc0lvRXw2xH, generation 1259912942, version 7
- HeartBeatState, generation 1259912942, version 18
```

AppState Reconciliation

- Generation
- Heartbeat
- AppState based on comparing version

Reconciliation example

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>gen:1234</td>
<td>gen:1234</td>
</tr>
<tr>
<td></td>
<td>Hb: 994</td>
<td>Hb: 993</td>
</tr>
<tr>
<td></td>
<td>Status: normal (4)</td>
<td>Status: normal (4)</td>
</tr>
<tr>
<td>B</td>
<td>Gen:2345</td>
<td>Gen:2345</td>
</tr>
<tr>
<td></td>
<td>Hb: 10</td>
<td>Hb: 17</td>
</tr>
<tr>
<td></td>
<td>Status: bootstrap (1)</td>
<td>Status: normal (2)</td>
</tr>
<tr>
<td>C</td>
<td>Gen:5335</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Hb: 1111</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Status: normal (3)</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>Gen:2232</td>
<td>Gen:3333</td>
</tr>
<tr>
<td></td>
<td>Hb: 4444</td>
<td>Hb: 11</td>
</tr>
<tr>
<td></td>
<td>status: normal (3)</td>
<td>Status: normal (3)</td>
</tr>
</tbody>
</table>

Inter-node Communication Summary

- Each node starts a gossip round every second
- 1-3 peers per round
- 3 messages passed
- Constant amount of network traffic

Practical implications

- Who is in the cluster?
- How are peers judged UP or DOWN?
- When does a node stop sending a peer traffic?
- When is one peer preferred over another?
- When does a node leave the cluster?

Cluster membership

- Gossip with a seed upon startup
- Learn about all peers
- Gossip
- Lather, rinse, repeat
UP/DOWN?
- Local to each node
- Not shared via gossip
- Determined via heartbeat

Failure Detection
- Glorified heartbeat listener
- Records timestamp when heartbeat update is received for each peer
- Keeps backlog of timestamp intervals between updates
- Periodically checks all peers to make sure that we’ve heard from them recently

UP/DOWN affects
- Stop sending writes (hints)
- Sending reads
- Gossip
  - It is down
  - This node is treated as an unavailable node
- Repair/stream sessions are terminated

What if a peer is really slow?
- Peer is NOT marked down
  - We will try to avoid it

Dynamic “Snitch”
- Determine when to avoid a slow node
- Scoring peers based on response times
  - Scores recalculated every 100ms (default)
  - Scores reset every 10m (default)

How do nodes leave?
- STATUS = LEAVING
- Stream data
- Stream hints
- STATUS = LEFT, expiryTime
Decomission
- STATUS = LEAVING
- Stream data
- Stream hints
- STATUS = LEFT, expiryTime

Remove node
- STATUS = REMOVING
- Rebalance cluster
  - Notify coordinator
- Delete hint
- STATUS = REMOVED, expiryTime

Replace node
- Cassandra.replace_address
  - "shadow gossip"
  - Take tokens/hostID(hints)
    - Check that previous owner hasn’t gossiped
- Stream data