CS535 Big Data

4/22/2019 Week 14-A Sangmi Lee Pallickara

FAQs

- General approach VS. customized approach
- Quiz 6: 4/24
- Workshop III
- Quiz 7: 5/1
- Workshop IV

Today's topics

- Cassandra
  - Partitioning data over 1,000s of machines

In the Lambda architecture

NoSQL Storage Systems
NoSQL databases

- Basic idea
  - Operates without a schema
  - Allows users to add fields without having to define any changes in structure first
  - Useful when dealing with nonuniform data and custom fields
- Stands for “Not Only SQL”
- Handles data access with size and performance that demand a cluster
- Improves the productivity of application development by using a more convenient data interaction style

Polyglot persistence

- Using different data stores in different circumstances
  - Without picking a particular database for all situations
  - Most organizations have a mix of data storage technologies for different circumstances

NoSQL Storage Data Model: (1) Key-Value Store

- Simple hash table
  - All access to the storage is via primary key
  - Get the value for the key
  - Put a value for a key
  - Delete a key
  - Add a key
- “value” is stored as a blob
  - Without caring or knowing what’s inside
  - Application is responsible for understanding data

NoSQL Storage Data Model: (2) Document Storage Model

- Documents
  - Self-describing
  - Data structure
    - Maps, collections, tree, and scalar values
    - Stores documents in the value part of the key-value store
- MongoDB, CouchDB, OrientDB, RavenDB, etc.
- Users can query the data inside the document
  - without having to retrieve the whole document

NoSQL Storage Data Model: (3) Column-Family Stores

- Cassandra, Hbase, Hypertable, and Amazon SimpleDB
- Stores data in column family as rows
  - Have many columns associated with a row key
- Column families
  - Groups of related data that is often accessed together

GEAR Workshop IV | Scalable Data Storage, Retrieval, and Analytics
Column Family NoSQL Storage system: Introduction to Apache Cassandra

http://www.cs.colostate.edu/~cs535

Spring 2019 Colorado State University, page 2
This material is built based on,

- Datastax Documentation: Apache Cassandra
- Now, Apache’s open source project,
  - http://cassandra.apache.org

If you design a data storage system for,

- Facebook to store data for the Facebook content, how would you prioritize properties: Consistency, Availability, or Partition tolerance? And Why?

Facebook’s operational requirements

- Performance
- Reliability
- Failures are norm
- Efficiency
- Scalability
- Support continuous growth of the platform

Inbox search problem

- A feature that allows users to search through all of their messages
  - By name of the person who sent it
  - By a keyword that shows up in the text
  - Search through all the previous messages
- In order to solve this problem,
  - System should handle a very high write throughput
  - Billions of writes per day
  - Large number of users

Now,

- Cassandra is in use at,
  - Apple
  - CERN
  - Easou
  - Comcast
  - eBay
  - GitHub
  - Hulu
  - Instagram
  - Netflix
  - Reddit
  - The Weather Channel
  - And over 1500 more companies
Data Model (1/2)
- Distributed multidimensional map indexed by a key

- Row key
  - String with no size restrictions
  - Typically 16 ~ 36 bytes long
  - Every operation under a single row key is atomic

- Value is an object
  - Highly structured

Data Model (2/2)
- Columns are grouped into column families
  - Similar to Bigtable

- Columns are sorted within a simple column or super columns
  - Sorted by time or by name

Super column family vs. Simple column family
- Simple column family
  - Some uses require more dimensions
  - Family of values
e.g. messages

- Cassandra’s native data model is two-dimensional
  - Rows and columns.
  - Columns that contain columns

API
- `insert(table, key, rowMutation)`
- `get(table, key, columnName)`
- `delete(table, key, columnName)`

Non-consistent hashing vs. consistent hashing
- When a hash table is resized
  - Non-consistent hashing algorithm requires re-hash of the complete table
  - Consistent hashing algorithm requires only partial rehash of the table
Consistent hashing

Consistent hash function assigns each node and key an m-bit identifier using a hashing function. m-bit identifier: 2^m identifiers. m has to be big enough to make the probability of two nodes or keys hashing to the same identifier negligible.

Consistent hashing assigns keys to nodes:
- Key k will be assigned to the first node whose identifier is equal to or follows k in the identifier space.

Example:
- Key 2 will be stored in machine C, successor(2) = 5.
- Machine B is the successor node of key 1, successor(1) = 1.

If machine C leaves circle, successor(5) will point to A.
If machine N joins circle, successor(2) will point to N.

Scalable Key location

- In consistent hashing:
  - Each node need only be aware of its successor node on the circle.
  - Queries can be passed around the circle via these successor pointers until it finds the resource.

- What is the disadvantage of this scheme?
  - It may require traversing all N nodes to find the appropriate mapping.

This material is built based on


GEAR Workshop IV | Scalable Data Storage, Retrieval, and Analytics
Apache Cassandra
Data Partitioning: CHORD

http://www.cs.colostate.edu/~cs535

Spring 2019 Colorado State University, page 5
Example of use
- Apache Cassandra’s partitioning scheme
- Couchbase
- Openstack’s object storage service Swift
- Akamai Content delivery network
- Data partitioning in Voldemort
- Partitioning component of Amazon’s storage system Dynamo

Scalable Key location in Chord
- Let \( m \) be the number of bits in the key/node identifiers
- Each node \( n \) maintains:
  - A routing table with (at most \( m \)) entries
  - Called the finger table
- The \( i \)th entry in the table at node \( n \) contains the identity of the first node, \( s \), that succeeds \( n \) by at least \( 2^{i-1} \) on the identifier circle
  - i.e. \( s = \text{successor}(n+2^{i-1}) \), where \( \text{successor}(x) \) (and all arithmetic is modulo \( 2^m \))

Definition of variables for node \( n \), using \( m \)-bit identifiers
- \( \text{finger}[i].\text{start} = (n+2^{i-1}) \mod 2^m \), \( 1 \leq k \leq m \)
- \( \text{finger}[i].\text{interval} = [\text{finger}[i].\text{start}, \text{finger}[i+1].\text{start}) \)
- \( \text{finger}[i].\text{node} = \text{first node} \geq n.\text{finger}[i].\text{start} \)
- \( \text{successor} = \text{the next node of the identifier circle} \)
- \( \text{predecessor} = \text{the previous node on the identifier circle} \)

Finger tables

http://www.cs.colostate.edu/~cs535

Spring 2019 Colorado State University, page 6
Lookup process

1. Go clockwise
2. Never overshoot

- First, check the data is stored in $n$
  - If it is, return the data
- Otherwise,
  - $n$ searches its finger table for the node $j$
    - Whose ID most immediately precedes $k$
  - Ask $j$ for the node it knows whose ID is closest to $k$
  - Do not overshoot!