Compare Cloud NoSql Solution

Ming Lei

lei.m.ming@gmail.com

Many NoSql Solutions out there:

1. Concepts of Distributed Programming Architecture.
2. Application Programming Model.
3. Detailed Comparison of a few NoSql open-source projects
   - Cassandra
   - Hbase
   - MongoDB
   - MemCache

Common Architectures

Highly Coupled (clustered):

refers typically to a cluster of machines that closely work together, running a shared process in parallel. The task is subdivided in parts that are made individually by each one and then put back together to make the final result.

Space Based (virtual single address space)

refers to an infrastructure that creates the illusion (virtualization) of one single address-space. Data are transparently replicated according to application needs. Decoupling in time, space and reference is achieved.

Common Architecture (2)

Peer-to-Peer vs. Client-to-Server

No special machine or machines that provide a service or manage the network resources. Instead all responsibilities are uniformly divided among all machines, known as peers.

vs.

Machines are assigned different tasks and responsibilities in a cluster. They depend on each other as client to server.

Common Architecture (3)

Inter-process Co-ordination vs. Database Centric

Method of communicating and coordinating work among concurrent processes on different machines.

vs.

Enable distributed computing to be done without any form of direct inter-process communication, by utilizing a shared database

Comparison Metrics

- Storage Type
- CAP
- ACID
- Read/Write Performance
- Replication and Sharding
- Indexed Query and Secondary Index
- High Availability, Fault Tolerance, Failover
- Scalability on data and request

Storage Types

- Document: Jackrabbit, MongoDb, ArangoDb, CouchBase, CouchDb
- Columns: Hbase, Cassandra
- Key-Values: Riak, MemCacheDb

CAP Theorem

- Consistency
  - data is the same across replications
- Availability
  - ability to access the cluster even if a node in the cluster goes down
- Partition Tolerance
  - cluster continues to function even if there is a "partition" (communications break) between nodes

CAP Theorem

- A distributed system can not satisfy all three.
- CA
  - data is consistent between all nodes but may become out of sync if there is partition in cluster.
- CP
  - data is consistent between all nodes, and maintains partition tolerance by becoming unavailable when a node goes down.
- AP
  - nodes remain online and unsynced during a partition

CAP and Eventual Consistency

- Eventual Consistency
  - Data may be temporarily out of sync on different nodes and will eventually be brought to the same version.
  - Can be implemented with a background process that updates out-of-sync nodes.
- A distributed system can be eCAP
  - This is how many noSql stores works.

Clustering: Sharding/Replication

- Sharding – distributes a single logical database system across a cluster of machines.
- Replication – replicates data among a group of machines within a cluster to ensure redundancy, backup, and automatic failover.
  - Master-master
  - Master-slave

ACID

- Atomic. Everything in a transaction succeeds or the entire transaction is rolled back.
- Consistent. A transaction cannot leave the database in an inconsistent state.
- Isolated. Transactions cannot interfere with each other.
- Durable. Completed transactions persist in the event of crashes or server failure.

The Scope of ACID

- Traditional SQL: ACID for the whole database across all tables.
- Cloud based Lightweight SQL: Logically partition data and support ACID within a logical partition. E.g.,
  - Website hosting service partition data by different website
- NoSql: partial ACID or ACID at row level, (eg, HBase)


MEMCACHED

MemCache

“

Free & open source, high-performance, distributed memory object caching system, generic in nature, but intended for use in speeding up dynamic web applications by alleviating database load.

Memcached is an in-memory key-value store for small chunks of arbitrary data (strings, objects) from results of database calls, API calls, or page rendering.

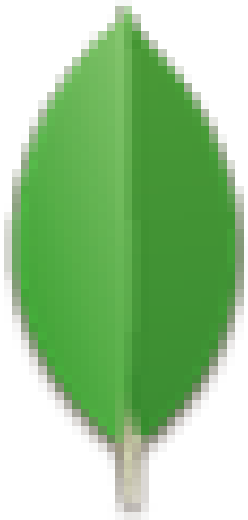
“

MemCache – Architecture

- Sharding in client code to select server.
- Peer-to-Peer Server instances.
- Server uses in-mem storage.
- Potentially expand to persistent store.

MemCache – Usage Characteristics

- Object-level Consistency, Isolation and Atomicity.
- No persistent storage
- No replication for load-balancing or failover
- Consistency + Partition-tolerance in CAP



mongoDB

{name: "mongo", type:"DB"}

MongoDb

- Document-oriented.
  - Think of MySQL but with JSON-like objects comprising the data model, rather than RDBMS tables.
- Supports neither joins nor transactions.
- secondary indexes
- atomic writes on a per-document level, and fully-consistent reads.
- master-slave replication with automated failover and built-in horizontal scaling via automated range-based partitioning.

MongoDb – Document Model

- Collection: collection of document of the same type.
- Document: a set of name-value pairs (properties)
- Flexible and dynamic schema of documents
- Secondary indexes can be built on document properties.
- Usage: map objects/relational tables to MongoDb documents.

MongoDb – ACID and CAP

- Document level AID
- Data Consistency –
  tunable between read consistency and high availability/performance
  - Reads can be consistent or eventually consistent
  - To achieve complete consistency, write has to occur on all replicas synchronously.
  - CA of CAP or eCAP

MongoDb – Sharding

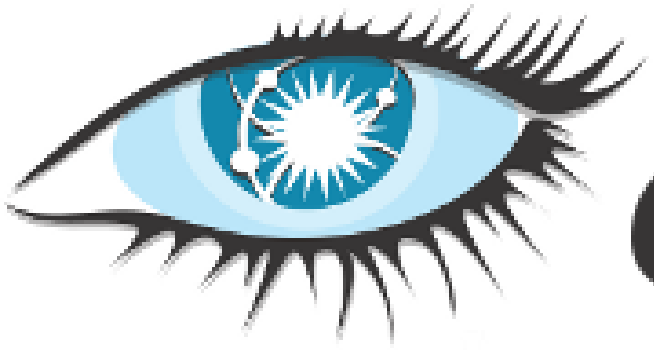
- Shard by key of document properties within a collection.
- Central mapping from key space to shards.
- Client requests are directed to different shard by service mongos.
- Dynamic balancing of shards.

MongoDb – Replication

- Primary Node and Secondary Nodes.
- Synch/Async propagation of writes from Primary to Secondaries.
- Re-elect a primary among secondaries when primary goes down.
- Roll-back of writes when a former primary rejoins as a secondary
  - Roll-back writes not propagated.
  - Human intervention is needed.

MongoDb – Read/Write

- Read
  - Consistent from primary only.
  - Eventually consistent from secondaries.
  - May lose recent updates when primary goes down.
- Write
  - Synchronously write to journal file (redo log)
  - Asynchronously write to data file
  - Asynchronously propagate to secondaries.



Cassandra

Cassandra – Peer-to-Peer Distributed Model

- All nodes are symmetric to avoid SPOF.
- Every node maintains global replication and sharding topology.
- Every node handles incoming requests and invoke other nodes to access data.
- Inter-node communication protocol:
  - Maintain data replication and sharding topology on every node.
  - Maintain replication consistency.

Cassandra – Node Storage

- SSTable – the immutable persistent base storage of a replica.
- Memtab – in memory table to accumulate updates/insertions/deletions. It serves as an addendum to the SSTable during reads.
- Redo Log – persistent queue of write ops used for failure recovery when Memtab is lost.

Cassandra – Read/Write

- Write
  - Replicates write on all replicas.
  - Write op blocks on writes of configured # of replicas.
  - Each replica updates memtab and append to redo log.
- Read
  - Replicates read on all replicas.
  - Read op blocks on reads of configured # of replicas.
  - Check consistency among read results from replicas.

Cassandra – Read/Write

- Storage Consolidation
  - Merge memtab and on-disk sstable and flush into new sstable. Then purge redo log up to the merging point.
- Node Recovery
  - All ops have sequentially increasing Ids.
  - Node detects missing ops after restart
  - Receive updates through inter-node communications

Cassandra – Index

- Data ordered by row key
- Secondary index on a column
  - Implemented as built-in table
  - No strong consistency between index and data.
- Index not suitable for conjunction clauses in query
  - Better filtering on other columns using a single index.

Cassandra – ACID

- row level AID
- Data Consistency –
  tunable between read consistency and high availability/performance
  - Reads can be inconsistent
  - Read/Write needs to go through the majority of replicas to achieve consistency


APACHE
HBASE

HBase – Data Model

- Flexible Schema
  - Predefined Column Family
  - Dynamically added columns under column family.
- 3-dimensional data cell
  - Row, column, version

HBase – ACID

- Row-level and region level ACID
  - Single row mutation
  - Single row CheckAndUpdate
  - Multi-row mutation
- Scan (involves multiple rows) does not give a consistent snapshot of the table.

HBase – Architecture

- Shard table into regions
- Store data and meta-data on HDFS
- Each region is served by 1 or more RegionServer
- MetaData – Table Directory and Region-to-Server mapping are stored as a table
  - Used by Hbase client to find regionserver
  - Used by regionservers at startup to manage regions
- HMaster
  - Monitor regionservers
  - Interface for MetaData update

	Storage Type	Secondary Index	ACID	CAP	Replication	P2P cluster	Scalability	Read/Write Performance
MongoDb	Document	yes	Document-level AID. Weak read consistency	By-default: eCAP	Yes	no	Disk Linearly with data size. CPU linearly w. requests	Default-config:Both fast. But write can be configured for full propagation for consistency.
Cassandra	Column	no	Row-level AID. Read consistency tunable.	CA and eCAP	Yes	no	Same as above	Read need to read from majority of nodes. Write can be configured to propagate to majority for read consistency.
HBase	Column	auxiliary	Row-level and region-level ACID	CA, CAP delegate to HDFS	Yes	no	Same as above	Optimized for random read and infrequent random writes.
MemCache	key-value	no	Object-level ACI	CP	no	yes	RAM Linearly w. cluster size	O(1)