A Brief Introduction to HBase & Cassandra

曲直
quzhi@net.pku.edu.cn
HDFS

- OpenSource GFS
- Typically 64M block
- NameNode: In memory metadata
- 3 replication
- Immutable
Hadoop database, a distributed, scalable, big data store.
HBASE定位

• HBASE是存储
• 基于HDFS
• 实时随机读写
HBASE特性

• KV存储
  • Get Put Delete Scan
• 水平扩展
• 行操作的强一致性
• 自动分表
• 支持MapReduce
• 基于HDFS
• Java, Thrift, REST-ful 接口
谁在用

- Yahoo! 960 nodes
- Facebook 180B/month message

... and others
HBase 数据模型

- Table
- Region
- ColumnFamily
- Row
- Column
- Version
- Value
SortedMap<RowKey, List<SortedMap<Column, List<Value, Timestamp>>>>
物理结构

• **Challenge**
  - 可扩展性：一个Table可能很大很大，怎么办？

• **Region Split**
  - 字典序排序
HRegion

- Challenge
  - 负载均衡：适应服务器的水平扩展？
- HRegion是Hbase中分布式存储和负载均衡的最小单元
Region 定位

- **Challenge**
  - 如何查找某个key？
  - 谁来负责Region管理？

- 三级游走
  - -ROOT-.META. Region

- 增加**Master**节点
Dive into HRegion

• Challenge
  • HDFS文件只追加，不可修改，如何在这样的系统上实现增删查改？
  • 写到一半服务器崩溃了怎么办？

• 三级结构
  • HLog：保证操作不会丢失
  • MemStore：写操作缓存
  • StoreFile(HFile)：当MemStore达到一定大小，一次刷新到HDFS（排序->去重->Write）
Dive into HRegion
Dive into HRegion

• Challenge
  • 一直追加写入，一个KV在多个HFile里会有多个版本

• API
  • Put, Delete 直接追加写入
  • Get 取各StoreFile和MemStore的最新版本

• Get的性能问题
  • 定期Compact: 归并排序
  • BloomFilter
  • BlockCache
HBase as a Cluster

- 服务器是会被玩坏的
  - Master
    - 多Master，谁是Active的？
  - RegionServer
    - 如何判断RegionServer是否活着？
    - MemStore里的数据怎么办？

- Solution
  - ZooKeeper 分布式一致性服务
  - Master负责迁移Region，HLog回放恢复MemStore
    - Hlog必须放在HDFS上！
总体结构

• Master
  • Region之上的操作
  • Put/Get不经过Master

• RegionServer
  • Region之下的操作

• HDFS
  • HFile
  • HLog

• ZooKeeper
  • 状态信息
总体结构
HFile
HLog
HRegion Server
A highly scalable, eventually consistent, distributed, structured key-value store.

Cassandra
The largest production cluster has over 100 TB of data in over 150 machines.
Dynamo/Bigtable (1/3) <<related concepts & techniques
• Dynamo-like features
  • Symmetric, P2P architecture
    • No Special nodes, No Single Point Of Failure (SPF)
  • Gossip Based cluster management
  • Distributed hash table for data placement
    • Pluggable partitioning
    • Pluggable topology discovery
    • Pluggable placement strategies
  • Tunable, Eventual Consistency
Highlights

• High availability
• Incremental scalability
• Eventually consistent
• Tunable tradeoffs between consistency and latency
• Minimal administration
• No Single Point of Failure (SPF)
Super column

```json
{
  name: "address",
  value: {
    street: {name:"street",value:"north zhongshan road",timestamp:1111111},
    city:{name:"city",value:"Shanghai",timestamp:1111119},
    zip:{name:"zip",value:"200062",timestamp:1111116}
  }
}
```

Value of a super column is a map of columns while value of a column is a string. No timestamp for super column itself.

Keyed by the column’s name.
Super Column family

Columns are attributes of super

No super column is allowed in a standard column family while no standard column is allowed in a super column family.
Storage model (3/4)

Key (CF1, CF2, CF3)

Commit Log
Binary serialized
Key (CF1, CF2, CF3)

Memtable (CF1)

Memtable (CF2)

Memtable (CF2)

• out of space
• too many keys (128 is default)
• timeduration (client provided, no cluster clock)

Dedicated Disk

Data file on disk

<Key name><Size of key Data><Index of columns/supercolumns><Serialized column family>
---
---
---

K_{128} Offset
K_{256} Offset
K_{384} Offset
Bloom Filter

(Index in memory)

BLOCK Index <Key Name> Offset, <Key Name> Offset
---
---

<Key name><Size of key Data><Index of columns/supercolumns><Serialized column family>
Storage model (4/4)

- K1 < Serialized data >
- K2 < Serialized data >
- K3 < Serialized data >

- K4 < Serialized data >
- K5 < Serialized data >
- K10 < Serialized data >

- K1 < Serialized data >
- K2 < Serialized data >
- K3 < Serialized data >
- K4 < Serialized data >
- K5 < Serialized data >
- K10 < Serialized data >

- K1 Offset
- K5 Offset
- K30 Offset
- Bloom Filter

Index File
Loaded in memory

MERGE SORT

Data File
CAP (2/3) <<related concepts & techniques
• Why AP?
  – CA-corruption possible if live nodes can’t communicate
  – CP-completely inaccessible if any nodes are dead
  – AP-always available, but may not always read most recent

Cassandra trade-off strong C in favor of high A choose AP but allows them to be tunable to have more C.

• eventual consistency (weak consistency)
  – when no updates occur for a long period of time, eventually all updates will propagate through the system and all the replicas will be consistent
partitioning & replication

N=3, rackunaware

h(key2)

h(key1)

1/2

32/63
### Consistency level (1/2)  <<related concepts & techniques

#### write

<table>
<thead>
<tr>
<th>Level</th>
<th>Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZERO</td>
<td>Ensure nothing. A write happens asynchronously in background. Until <a href="https://issues.apache.org/jira/browse/CASSANDRA-685">CASSANDRA-685</a> is fixed: If too many of these queue up, buffers will explode and bad things will happen.</td>
</tr>
<tr>
<td>ANY</td>
<td>(Requires 0.6) Ensure that the write has been written to at least 1 node, including <a href="https://issues.apache.org/jira/browse/CASSANDRA-492">HintedHandoff</a> recipients.</td>
</tr>
<tr>
<td>ONE</td>
<td>Ensure that the write has been written to at least 1 replica's commit log and memory table before responding to the client.</td>
</tr>
<tr>
<td>QUORUM</td>
<td>Ensure that the write has been written to (N / 2 + 1) replicas before responding to the client.</td>
</tr>
<tr>
<td>DCQUORUM</td>
<td>As above but takes into account the rack aware placement strategy. See <a href="https://issues.apache.org/jira/browse/CASSANDRA-492">https://issues.apache.org/jira/browse/CASSANDRA-492</a></td>
</tr>
<tr>
<td>ALL</td>
<td>Ensure that the write is written to all (N) replicas before responding to the client. Any unresponsive replicas will fail the operation.</td>
</tr>
</tbody>
</table>
Write operation <<storage model
## Consistency level

### read

<table>
<thead>
<tr>
<th>Level</th>
<th>Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZERO /ANY</td>
<td>Not supported, because it doesn't make sense.</td>
</tr>
<tr>
<td>ONE</td>
<td>Will return the record returned by the first replica to respond. A consistency check is always done in a background thread to fix any consistency issues when ConsistencyLevel.ONE is used. This means subsequent calls will have correct data even if the initial read gets an older value. (<a href="#">ReadRepair</a>)</td>
</tr>
<tr>
<td>QUORUM</td>
<td>Will query all replicas and return the record with the most recent timestamp once it has at least a majority of replicas (N / 2 + 1) reported. Again, the remaining replicas will be checked in the background.</td>
</tr>
<tr>
<td>DCQUORUM</td>
<td>When using rack aware placement strategy reads are kept within a data center. See <a href="https://issues.apache.org/jira/browse/CASSANDRA-492">https://issues.apache.org/jira/browse/CASSANDRA-492</a></td>
</tr>
<tr>
<td>ALL</td>
<td>Will query all replicas and return the record with the most recent timestamp once all replicas have replied. Any unresponsive replicas will fail the operation.</td>
</tr>
</tbody>
</table>
Read operation

Cassandra Cluster

Client

Query -> Result

Closest replica

Result

Replica A

Digest Query

Digest Response

Replica B

Replica C

Single read

Read repair if digests differ
<table>
<thead>
<tr>
<th></th>
<th>Cassandra</th>
<th>HBase</th>
</tr>
</thead>
<tbody>
<tr>
<td>一致性</td>
<td>Quorum NRW策略，最终一致性</td>
<td>单节点，无复制，强一致性</td>
</tr>
<tr>
<td>可用性</td>
<td>1，基于Consistent Hash相邻节点复制数据，数据存在于多个节点，无单点故障。</td>
<td>1，存在单点故障，Region Server宕机后，短时间内该Server维护的Region无法访问，等待failover生效。</td>
</tr>
<tr>
<td></td>
<td>2，某节点宕机，hash到该节点的新数据自动路由到下一节点做hinted handoff，源节点恢复后，推送回源节点。</td>
<td>2，通过Master维护各Region Server健康状况和Region分布。</td>
</tr>
<tr>
<td></td>
<td>3，通过Gossip协议维护集群所有节点的健康状态，并发送同步请求，维护数据一致性。</td>
<td>3，多个Master，Master宕机有zookeeper的Paxos投票机制选取下一任Master。Master就算全宕机，也不影响Region读写。</td>
</tr>
<tr>
<td></td>
<td>4，SSTable，纯文件，单机可靠性一般。</td>
<td>4，HDFS，一备三，高可靠，0数据丢失。</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5，HDFS的Namenode是一个SPOF。</td>
</tr>
<tr>
<td>伸缩性</td>
<td>1，Consistent Hash，快速定位数据所在节点。</td>
<td>1，通过Zookeeper定位目标Region Server，最后定位Region。</td>
</tr>
<tr>
<td></td>
<td>2，扩容需在Hash Ring上多个节点间调整数据分布。</td>
<td>2，Region Server扩容，由Master均匀分布。</td>
</tr>
<tr>
<td>负载均衡</td>
<td>请求Zookeeper取得整个集群地址，然后根据Consistent Hash选择合适的节点。client会缓存集群地址。</td>
<td>请求Zookeeper取读写数据路由表定位Region Server，Master会修改这个路由表。Client自身也会缓存一部分路由信息。</td>
</tr>
<tr>
<td>数据差异比较算法</td>
<td>Merkle Tree, Bloom Filter</td>
<td>Bloom Filter</td>
</tr>
<tr>
<td>锁与事务</td>
<td>Client Timestap（Dynamo使用vector lock）</td>
<td>Optimistic Concurrency Control</td>
</tr>
<tr>
<td>读写性能</td>
<td>数据读写定位非常快。</td>
<td>数据读写定位可能要通过最多6次的网络RPC</td>
</tr>
<tr>
<td>CAP点评</td>
<td>1，最终一致性，数据可能丢失。</td>
<td>1，强一致性，0数据丢失。</td>
</tr>
<tr>
<td></td>
<td>2，可用性高。</td>
<td>2，可用性低。</td>
</tr>
<tr>
<td></td>
<td>3，扩容方便。</td>
<td>3，扩容方便。</td>
</tr>
</tbody>
</table>
Cassandra is optimized for writes, and achieves higher throughput and lower latency. Sherpa and MySQL achieve roughly comparable performance, as both are limited by MySQL’s capabilities. HBase has good write latency, because of commits to memory, and somewhat higher read latency, because of the need to reconstruct records.
• Workload b -- read heavy 95/5 Read/update

Sherpa does very well here, with better read latency – only one lookup into a Btree is needed for reads, unlike log-structured systems where records must be reconstructed. Cassandra also performs well, matching Sherpa until high throughputs. HBase does well also, although read time is higher.
YCSB (5/8) << Yahoo! Cloud Serving Benchmark

- Workload e -- short scans  Scans of 1-100 records of size 1KB

HBase and Sherpa are roughly equivalent for latency and peak throughput, even though HBase is “meant” for scans. Cassandra’s performance is poor, but the development team notes that many optimizations still need to be done.
YCSB (6/8) << Yahoo! Cloud Serving Benchmark

- Scale up - Read heavy workload with varying hardware

Sherpa and Casandra scale well, with flat latency as system size increases. HBase is very unstable; 3 servers or less performs very poorly.
Thank you!