

Cassandra

MORNINGSTAR[®]

Operational Data Store

Initial Requirements

(Late 2007)

- On big data security aggregator from multiple sources using Morningstar global security identifier
- Highly scalable both horizontally and vertically
- Easy to distribute computation processing
- Easy to store various types of data

MySQL

Initial Implementation (2008)

- One database on one big database server
- Very simple data model - one table per source with a simple key (Morningstar ID and date)
- Tables were manually replicated with complicated logic
- Tables stored data as binary blobs
- No indexing on the tables other than the primary key(s)

MySQL Tables

	id	date	DataUnit5	DataUnit5 updt_time	DataUnit6	DataUnit6 updt_time	
F0	id	date	DataUnit3	DataUnit3 updt_time	DataUnit4	DataUnit4 updt_time	
F0	S0	id	date	DataUnit1	DataUnit1 updt_time	DataUnit2	DataUnit2 updt_time
F0	S0	F0000001	2011-01-31	BLOB	2011-02-15 12:03:01	BLOB	2011-02-08 12:05:02
F0	S0	F0000002	2011-01-31	BLOB	2011-02-14 10:17:14	BLOB	2011-02-01 14:35:10
		F0000003	2010-12-31	BLOB	2011-01-15 22:17:11	BLOB	2011-03-15 18:36:24

What worked?

- Great interface to query the data
- Very stable system
- Simple data model meant high efficiency for queries
- Great memory usage

What did not work

- Hard to implement Map-Reduce
- Hard to increase capacity with data growth
- Multi-site replication slow and somewhat complicated
- Limited number of columns and rows per table
 - Did manual table partitioning to keep under 2 million records per table
 - Table per source to keep column count down, and to not have sparsely populated rows

Cassandra

Current Implementation (2010)

- 5 Machine Cluster
 - In house VMs on blade farm
 - 4 cores, 8 GB ram per node
- Column families based on access type not source
- Manual indexing of data unit type to key(s)

Cassandra Column Families

Data

key	columns					
F0000001	DataUnit7		DataUnit8		DataUnit9	
	BLOB	2011-02-15T12:03:01	BLOB	2011-02-08T12:05:02	BLOB	2011-02-08T12:05:02
F0000002	DataUnit7		DataUnit9		DataUnit10	
	BLOB	2011-02-14T10:17:14	BLOB	2011-02-05T14:35:10	BLOB	2011-02-05T14:35:10
S0000001	DataUnit11		DataUnit12		DataUnit13	
	BLOB	2011-02-15T12:03:01	BLOB	2011-02-15T12:03:01	BLOB	2011-02-15T12:03:01
S0000002	DataUnit12		DataUnit13		DataUnit14	
	BLOB	2011-02-15T12:03:01	BLOB	2011-02-15T12:03:01	BLOB	2011-02-15T12:03:01

Cassandra Column Families

Time Series Data

key	super columnn	columns			
F0000001	2011-01-31	DataUnit1		DataUnit2	
		BLOB	2011-02-15T12:03:01	BLOB	2011-02-08T12:05:02
	2011-02-05	DataUnit1		DataUnit5	
		BLOB	2011-02-14T10:17:14	BLOB	2011-02-05T14:35:10
F0000002	2011-01-31	DataUnit2		DataUnit6	
		BLOB	2011-01-31T12:03:01	BLOB	2011-02-01T12:05:01
	2011-02-04	DataUnit5		DataUnit6	
		BLOB	2011-02-04T10:03:15	BLOB	2011-02-04T15:03:21
S0000002	2011-01-31	DataUnit3		DataUnit4	
		BLOB	2011-02-15T12:07:01	BLOB	2011-02-15T12:03:31
	2011-02-04	DataUnit3		DataUnit4	
		BLOB	2011-02-04T16:13:01	BLOB	2011-02-05T18:03:01

What works?

- Very easy to query when the keys are known (normal use)
- Very scalable, just add more nodes, even at a later point in time.
- Multi-site replication is easy
- Basically unlimited number of columns per column family
- Unlimited number of rows per column family
- Sparse rows don't waste space
- Disaster recovery automatically taken care of by multi-site redundancy

What is hard

- Arbitrary queries are difficult.
 - Had to create our own indexes to go from data unit type back to key (can't select where != NULL)
 - Need to add extra indexes and/or de-normalized column families when we think of a new way that we want to query the data
- Monitoring a cluster is harder than one server
- Getting memory usage settings correct so that nodes don't die with OOM errors

Future Plans

- Upgrade to 0.7
- Expand cluster to multiple data centers around the globe