MADOCA II Data Logging System Using NoSQL Database for SPring-8

A.Yamashita and M.Kago
SPring-8/JASRI, Japan
NoSQL
OR: How I Learned to Stop Worrying and Love Cassandra
Outline

- SPring-8 logging database
- Why NoSQL, why Cassandra
- Implementation
- Production Run
SPring-8 Logging database

- Relational database system (RDBMS) has been used since 1997
- Grows from 871 signals to 27,626 signals (end of 2014)
- 7,000 signal inserts per seconds
- 4TB raw data at end of 2014
- SACLA (X-ray FEL) is also using it
SPRing-8 Logging database

• What made the system live long?
  – Uniform data store
  – Simple access
SPring-8 Logging database

- What made the system live long?
  - Uniform data store
    - Every data
    - Every time
    - in one database
  - Simple access
**SPring-8 Logging database**

- What made the system live long?
  - Uniform data store
  - Simple access
    - Just Key + time range access
      ```
      get ("sr_mag_ps_b/current_adc",
           "2014/10/10 19:24:12",
           "2014/10/10 22:00:00")
      ```
RDBMS to NoSQL

• For the next generation SPring-8-II
• We changed logging database for SPring-8 from RDBMS to a NoSQL database; Cassandra
• Why NoSQL, Why Cassandra?
RDBMS is great

• We are currently using RDBMS for
  – Configuration management
  – Parameter management
  – Alarm record
  – Etc

• But,
RDBMS limitation in logging

- Performance
- Scalability
- Availability
- Flexibility
Logging in Accelerator Control

- Time series data
Logging in Accelerator Control

- Time series data
- Write many and rare read
Logging in Accelerator Control

- Time series data
- Write many and rare read
- Is RDBMS is suitable for this task?
Logging in Accelerator Control

- Time series data
- Write many and rare read
- Is RDBMS suitable for this task?
- Looking for new database
  - Keeping advantage of the old system
  - Make up for its shortcomings
NoSQL (Not only SQL)

- Simplicity of design, simpler "horizontal" scaling to clusters of machines, which is a problem for relational databases, and finer control over availability. (Wikipedia)
NoSQL (Not only SQL)

- Simplicity of design, simpler "horizontal" scaling to clusters of machines, which is a problem for relational databases, and finer control over availability. (Wikipedia)
NoSQL variations

- Key-value
- Graph
- Document

• No solutions for time-series data in above NoSQL
• Wide-column
Wide-column database

- One type of NoSQL (Not only database)

<table>
<thead>
<tr>
<th>Row Key</th>
<th>Column key</th>
<th>Column value</th>
</tr>
</thead>
<tbody>
<tr>
<td>sig1:20130504</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sig2:20130504</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sig3:20130504</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Wide-column database

• Columns are added when data added.

<table>
<thead>
<tr>
<th>Row Key</th>
<th>Column</th>
</tr>
</thead>
<tbody>
<tr>
<td>sig1:20130504</td>
<td>t0</td>
</tr>
<tr>
<td></td>
<td>value0</td>
</tr>
<tr>
<td>sig2:20130504</td>
<td></td>
</tr>
<tr>
<td>sig3:20130504</td>
<td></td>
</tr>
</tbody>
</table>
**Wide-column database**

- **Columns are added when data added**

<table>
<thead>
<tr>
<th>Row Key</th>
<th>Column</th>
<th>Column key</th>
<th>Column value</th>
</tr>
</thead>
<tbody>
<tr>
<td>sig1:20130504</td>
<td>t0</td>
<td>t1</td>
<td>value0</td>
</tr>
<tr>
<td></td>
<td>value0</td>
<td>value1</td>
<td></td>
</tr>
<tr>
<td>sig2:20130504</td>
<td>T'0</td>
<td></td>
<td>Value'0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sig3:20130504</td>
<td>T&quot;0</td>
<td></td>
<td>Value&quot;0</td>
</tr>
</tbody>
</table>
## Wide-column database

- **Row is added at any time**

<table>
<thead>
<tr>
<th>Row Key</th>
<th>Column</th>
<th>Column Key</th>
<th>Column Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>sig1:20130504</td>
<td>t0</td>
<td>value0</td>
<td>t1 value1</td>
</tr>
<tr>
<td></td>
<td>t2</td>
<td>value2</td>
<td></td>
</tr>
<tr>
<td>sig2:20130504</td>
<td>T'0</td>
<td>Value'0</td>
<td>T'1 Value'1</td>
</tr>
<tr>
<td>sig3:20130504</td>
<td>T''0</td>
<td>Value''0</td>
<td></td>
</tr>
<tr>
<td>sig4:20130510</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Wide-column database

- And it grows

<table>
<thead>
<tr>
<th>Row Key</th>
<th>Column</th>
<th>Column key</th>
<th>Column value</th>
</tr>
</thead>
<tbody>
<tr>
<td>sig1:20130504</td>
<td>t0</td>
<td>value0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>t1</td>
<td>value1</td>
<td>t2</td>
</tr>
<tr>
<td></td>
<td>t2</td>
<td>value2</td>
<td>t3</td>
</tr>
<tr>
<td>sig2:20130504</td>
<td>T'0</td>
<td>value'0</td>
<td>T'1</td>
</tr>
<tr>
<td></td>
<td>T'1</td>
<td>value'1</td>
<td>T'2</td>
</tr>
<tr>
<td></td>
<td>Value'0</td>
<td>Value'1</td>
<td>Value'2</td>
</tr>
<tr>
<td>sig3:20130504</td>
<td>T''0</td>
<td>Value''0</td>
<td></td>
</tr>
<tr>
<td>sig4:20130510</td>
<td>T''0</td>
<td>Value'''0</td>
<td></td>
</tr>
</tbody>
</table>
Wide-column database

• Suitable for time-series data logging
  - Each data has its own time-stamp
    • cyclic data + event driven data in same place
  - Access
    • Key+ column range
      - same as current access method
Which Wide-column DB?

- Major wide-column database
  - Apache Cassandra
  - Apache Hbase
  - Hypertable
Apache Cassandra

- We select by its availability
- Every node has the same role
  - No master node
    - No single point of failure (SPOF)
    - HBase and Hypertable have masternode: SPOF
Apache Cassandra

• Our criteria
  – Reliability
  – Scalability
  – Flexibility

• Consistency is covered by the other DB
Reliability

• Most essential
Reliability

• Most essential

• Cassandra
  – No master node, no single point of failure
  – Data redundancy
    • 3 data replicas
Scalability

• Just add nodes when you need more power
  − No cluster reboot is needed
• Apple is operating 100,000 node cluster for iTunes
Flexibility

- Insert at any time
Flexibility

- Insert at any time
  - Signal by signal
Flexibility

- Insert at any time
- Data type
Flexibility

• Insert at any time

• Data type
  - Store data using object serialization
    • Not using cassandra's data type
    • blob type column only
Flexibility

- Insert at any time
- Data type
  - Store data using object serialization: MessagePack
    - Very fast
    - Low overhead 8 Byte float -> 9 Byte string
    - Self described
      - NO Interface Definition Language like Protocol Buffer
Consistency

- Cassandra does not guarantee consistency
- In our cluster, it takes about 1 second after insert to obtain consistent value.
  - No real-time access
Redis

- Covers Cassandra's inconsistency
Redis

- Covers Cassandra's inconsistency
- Stores newest data only
Redis

• Covers Cassandra's inconsistency
• Stores newest data only
• In-memory key-value database
Redis

- Covers Cassandra's inconsistency
- Stores newest data only
- In-memory key-value database
  - Very fast by key access
  - Newest value + meta data only
  - Data packed by MessagePack
Redis

- Covers Cassandra's inconsistency
- Stores newest data only
- In-memory key-value database
- Two redis servers are running in parallel for redundancy
Implementation

• Data acquisition system
• Cassandra structure
• Performance
Entire system

Embedded CPU
- Collector server
  - Collector client
  - SHM poller
  - Collector client
- Collector server
  - SHM poller
  - Collector client
- Po2m2db
- VME
  - Collector server
  - Collector client
  - SHM poller
  - Collector client
- Po2m2db
- BPM etc.

RPC
- cc2m2db
- Relay server
- ØMQ

ØMQ
- Redis
- Cassandra
- stdout

RDB
Entire system

Tightly coupled by RPC

RPC

Embedded CPU
- Collector server
- SHM

Collector server

Collector client

Collector client

VME
- Collector server
- SHM
- po2m2db

RPC

cc2m2db

Relay server

ØMQ

BPM etc.

device

poller

SHM

device

poller

SHM

po2m2db

Relay server

W

W

W

Redis

stdout

ZMQ

Relay server

ZMQ

RPC
Entire system

Loosely coupled by 0MQ
Just throw messages to relay server
Message creation

Key: LGsr_mag_ps_b/current_adc:

Metadata: 
```
{"tm":1444814445053727,"tl":1296000000,"cy":1000}
```

Data: 419.6774238114116
Message structure

• 3 part message

<table>
<thead>
<tr>
<th>Key</th>
<th>LGsr_mag_ps_b/current_adc:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Metadata</td>
<td>{&quot;tm&quot;:1444814445053727,&quot;tl&quot;:1296000000,&quot;cy&quot;:1000}</td>
</tr>
<tr>
<td>Data</td>
<td>419.6774238114116</td>
</tr>
</tbody>
</table>

• Key: raw string

• Metadata and data are packed by MessagePack
Convert to Cassandra command

```
insert('LGsr_mag_b/current_adc:201510',
{1444814445053727:419.674238114116},
1296000000)
```

Blue means MessagePacked string
Convert to Redis Command

```
insert('LGsr_mag_b/current_adc:',
'{"tm:1444814445053727":419.6774238114116'}
```
Writer

- Converts messages to insert commands
- Plug-in structure using 0MQ's in-process pub/sub
Writer

- Converts messages to insert commands
- Plug-in structure using 0MQ's in-process pub/sub
  - Other DB engine or anything may be added in the future
**Structure of Cassandra**

- key : one key / one signal one day
- LGsr_mag_ps_b/current_adc:20151003

### Keyspace: database

#### Column Family: Table

<table>
<thead>
<tr>
<th>Row Key</th>
<th>Column</th>
<th>Column key</th>
<th>Column Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>sig1:20130504</td>
<td>t0</td>
<td>value0</td>
<td>value1</td>
</tr>
<tr>
<td></td>
<td>t1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>t2</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>t3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sig2:20130504</td>
<td>t0</td>
<td>value0</td>
<td>value1</td>
</tr>
<tr>
<td></td>
<td>t1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sig3:20130504</td>
<td>t0</td>
<td>value0</td>
<td>value1</td>
</tr>
<tr>
<td></td>
<td>t1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>t2</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>t3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Performance; write to Cassandra

Values insert/sec (batch)

Writers

Nodes

$6 \times 10^5$

$4 \times 10^5$

$2 \times 10^5$

$0$

$0$

$20$

$40$

$60$

$80$

$100$

$120$

$140$

$160$
Read from Cassandra

- One day data = 60s * 60min * 24hour
- Done during normal writing operation

Time to get one day data (sec)

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td>0.76sec</td>
</tr>
<tr>
<td>Sigma</td>
<td>0.06sec</td>
</tr>
<tr>
<td>95%</td>
<td>0.86sec</td>
</tr>
</tbody>
</table>
Read from Redis

Online time to get one data (ms)

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td>0.77ms</td>
</tr>
<tr>
<td>Sigma</td>
<td>0.47ms</td>
</tr>
<tr>
<td>95%</td>
<td>1.4ms</td>
</tr>
</tbody>
</table>
One year Test

- Test performed about one year
- No major trouble
  - Test
    - Forced to shutdown a node and recovery
- Some modification are needed for the production run
For production run

- Data migration from RDBMS
- Structure modification
- Monitoring tools
- Client libraries
- Node added
Data migration from RDBMS

- **Data since 1997**
  - 4TB in RDBMS (logical file size, become larger in RAID disks)
  - 0.75TB/node 9TB in total in 12 nodes (3 replicas)
Structure changed

- One large column family was divided into small column families of each month
  - Cassandra's compaction operation
    - Batch operation
    - Columns that marked as “delete” are deleted at this time.
Temporary disk space for compaction

- One column family needs same size temporary disk space at compaction.
- One big column family cannot be larger than \( \frac{1}{2} \) disk space.
Structure changed

- One big column family was divided into small column families of one month

<table>
<thead>
<tr>
<th>Keyspace</th>
<th>Column Family</th>
<th>Keyspace</th>
<th>Column Family 201501</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Column Family 201502</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Column Family 201503</td>
</tr>
</tbody>
</table>

- Backup becomes easy by copying separate files
Monitoring tool

- Server system monitoring by Zabbix.
  - Not only SNMP but also JXM
    - Cassandra is written by JAVA
    - JAVA VM monitoring

- DAQ system monitoring tool
  - For experts
  - For operators
Zabbix screen
DAQ system monitoring for experts
## DAQ System monitoring for shift operators

<table>
<thead>
<tr>
<th>Status</th>
<th>Running</th>
<th>Name</th>
<th>Update Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>12/12</td>
<td></td>
<td>Archive DB (Cassandra)</td>
<td>2015/01/07 12:17:05</td>
</tr>
<tr>
<td>2/2</td>
<td></td>
<td>Online DB (Redis)</td>
<td>2015/01/07 12:17:05</td>
</tr>
<tr>
<td>2/2</td>
<td></td>
<td>Relay Process</td>
<td>2015/01/07 12:17:05</td>
</tr>
<tr>
<td>40/40</td>
<td></td>
<td>Writer Process</td>
<td>2015/01/07 12:17:05</td>
</tr>
</tbody>
</table>
Client libraries

- Mainly C and C++
  - For applications written in C
  - Same interfaces, no modification to source code of application
    - Just re-link
- Python modules
  - for Web applications
- We used
  - ZeroMQ, Messagepack, Cassandra CQL/Thrift, Redis
# Cassandra Cluster

<table>
<thead>
<tr>
<th>Specification</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of nodes</td>
<td>12</td>
</tr>
<tr>
<td>Server</td>
<td>Dell PowerEdge R420</td>
</tr>
<tr>
<td>CPU</td>
<td>Intel Xeon E5-2420 2.2GHz</td>
</tr>
<tr>
<td>Memory</td>
<td>16GB</td>
</tr>
<tr>
<td>System disk</td>
<td>600GB 15k rpm</td>
</tr>
<tr>
<td>Data disks</td>
<td>3TB 7200 rpm x3</td>
</tr>
<tr>
<td>Cassandra</td>
<td>2.0.10</td>
</tr>
<tr>
<td>JavaVM</td>
<td>JRE1.7.0-67-b01</td>
</tr>
</tbody>
</table>
Summary

- We implemented new data acquisition and store system with new technologies
- Apache Cassandra provides high-performance, reliable, scalable and flexible data store that was impossible by RDBMS
- We build supporting infrastructures for healthy operations
- The system is stably running more than one year including test run