Hypertable Distilled by edydkim.github.com

Preface

NoSQL is the most interesting parts in database system recently. There are two reason why use that - Scalability and Productivity. Many NoSQL Database have those features and difficult to consider to choose one. Here is very powerful and sophisticate one, Hypertable based on Google BigTable, like to recommend to all. I will focus to how to use it in real time application, not only logging. Is it possible mission critical state something like user table no one believe that a big NoSQL database used to retrieve single row in CRUD. I hope this will be helpful to choose a database out of them when you wonder.

Note: This paper approaches how to programm and implement in a real application, if I get a chance next time will see to benchmark performance of distributed environments.

Tip: What is the DBMS and How to store data shortly?

- RDBMS & ORDBMS : structured and object-oriented data
- Document-oriented DBMS : semi-structured data like xml, json based on documents
- Graph DBMS : edges and nodes which have properties data represented graph
- Column-oriented DBMS : atttribute data
- Typically row-oriented (=row based) database is mentioned to Relational DBMS & ORDBMS and column-oriented (= column based) & document-oriented is mentioned to NoSQL database.

Contents

Abstract

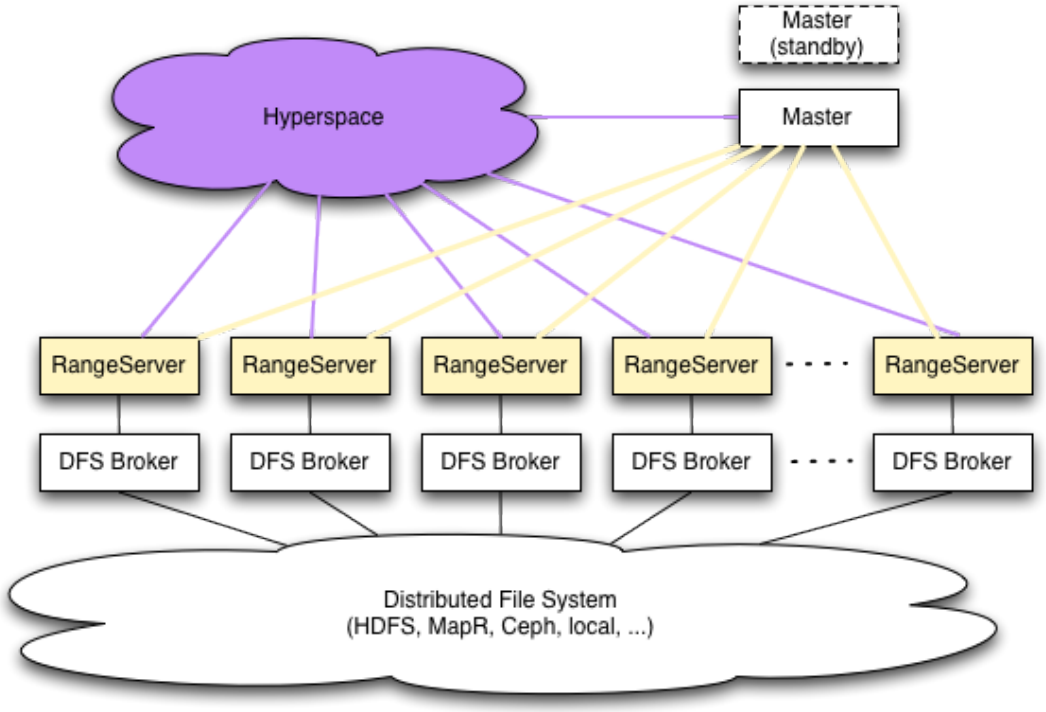
Hypertable is a high-performance alternative to HBase on Hadoop. The essential characteristics of Hypertable are quite similar to HBase, which in turn is a clone of the Google Bigtable. Hypertable is actually not a new project. It started around the same time as

HBase in 2007. Hypertable runs on top of a distributed filesystem like HDFS.

In HBase, column-family-centric data is stored in a row-key sorted and ordered manner by Map/Reduce. The each cell of data maintains multiple versions of data. In Hypertable all version information is appended to the row-keys. The version information is identified via timestamps. All data for all versions for each row-key is stored in a sorted manner for each column-family and is supported multiple indices.
*1 reference source : Professional NoSQL by Shashank Tiwari

Design

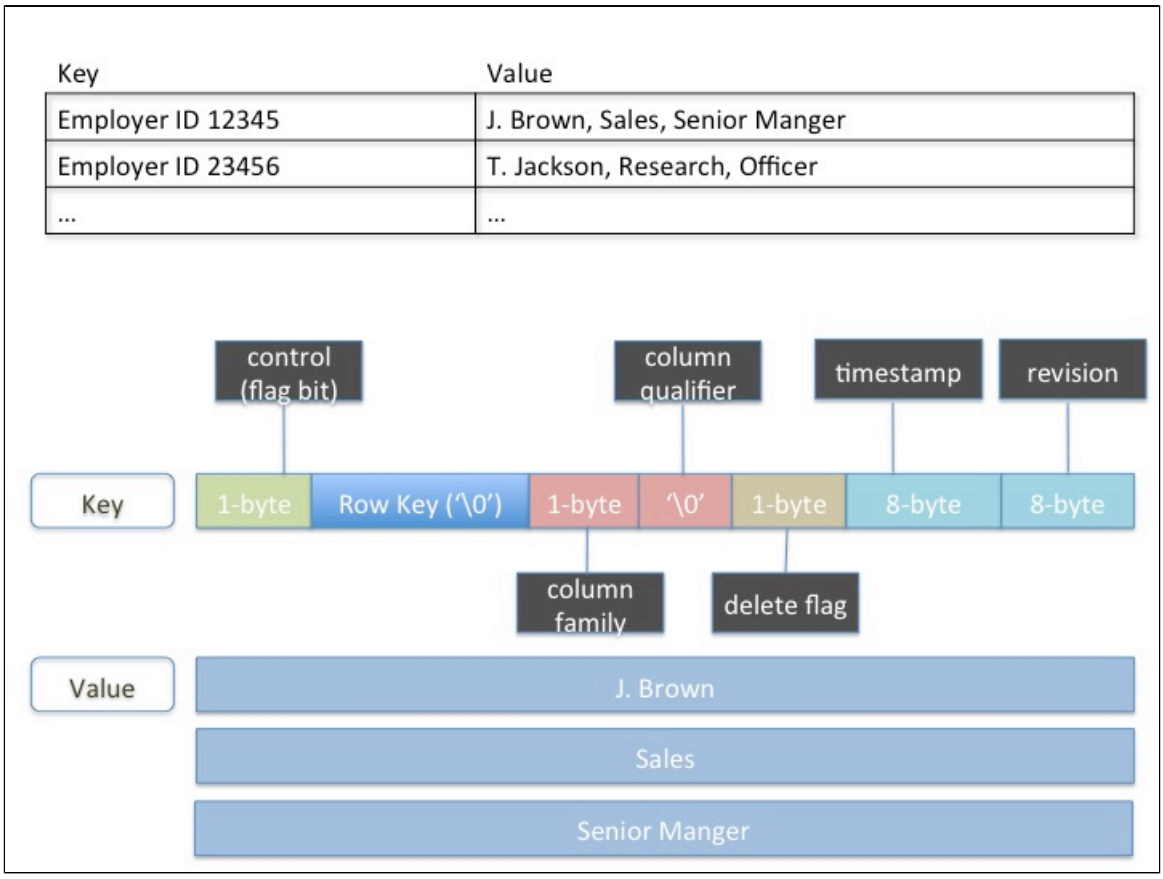
System overview



*2 reference source : http://hypertable.com

Hypersapce is a filesystem for locking and storing small amounts of metadata means the root of all distributed data structures. Master has all meta information of operations such as creating and deleting tables. Master has not a single point of failure as client data does not move through it for short periods of time when switch standby to primary. Master is also responsible for range server load balancing. Range server manage ranges of table data, all reading and writing data. DFS Broker has the interface to Hypersapce. it translates normalized filesystem requests into native filesystem requests for HDFS, MapReduce, local and so on and vice-versa.

Key and Value



Here are some check points to need to know.
Control describes the format of remaining fields. ex) HAVE REVISION = 0x80, HAVE TIMESTAMP = 0x40, AUTO TIMESTAMP = 0x20, SHARED = 0x10, TS_CHRONOLOGICAL = 0x01
flag is a logical delete flag to be garbage collected.

Access group

Would you have head about Partitioning in some RDBMS? The Access group provide the all data defined the access group to store the same disk together. It can be limited Disk I/O and the frequency by reducing the amount of data transferred from disk during query execution.

```
create table User (
name
, department
, position
access group default(name, position)
, access group department(department)
)
```

Note: The column family 'department' can belong to only one access group. If the column family belong to one more access group HYPERTABLE HQL parse error is occured.

Run a query below then, check elaspsed time and throughput.

```
select department from User;
```

Secondary indices

It is very important that the table can have one more indices or not about each a single column family. Hypertable has two types of indices: a cell value index and qualifier index. A cell value index scans on a single column family that do an exact match or prefix match of the value, and qualifier index scans on same as a cell valud index but of the column qualifier.

Note: The indices is stored index table in same namespace as the primary table. You need to consider additional disk storage for

indexing data and cause a very small performance impact of side effect for inserting data.

How to define and see below:

```
create table User (
name
, department
, position
, index name
, index department
, qualifier index department
, index position
);
```

```
hypertable> create table User (
-> name
-> , department
-> , position
-> , index name
-> , index department
-> , qualifier index department
-> , index position
-> );

Elapsed time: 35.21 s
hypertable> insert into User values ('A00', 'name', 'I am'), ('A00', 'department', 'Sales'), ('A00', 'position',
'Manager');

Elapsed time: 0.05 s
Avg value size: 5.33 bytes
Total cells: 3
Throughput: 59.39 cells/s
Resends: 0
hypertable> insert into User values ('B00', 'name', 'You are'), ('B00', 'department:old', 'Human Resource'),
('B00', 'position', 'Senior Manager');

Elapsed time: 0.12 s
Avg value size: 11.67 bytes
Avg key size: 1.33 bytes
Throughput: 319.36 bytes/s
Total cells: 3
Throughput: 24.57 cells/s
Resends: 0
hypertable> select name from User where name = 'I am';
A00 name I am

Elapsed time: 0.07 s
Avg value size: 4.00 bytes
Avg key size: 4.00 bytes
Throughput: 121.82 bytes/s
Total cells: 1
Throughput: 15.23 cells/s
hypertable> select name from User where name =^ 'I';
A00 name I am

Elapsed time: 0.00 s
Avg value size: 4.00 bytes
Avg key size: 4.00 bytes
Throughput: 15009.38 bytes/s
Total cells: 1
```

```
Throughput: 1876.17 cells/s

hypertable> select department:old from User where department = 'Human Resource';
B00 department:old Human Resource

Elapsed time: 0.00 s
Avg value size: 14.00 bytes
Avg key size: 7.00 bytes
Throughput: 8183.94 bytes/s
Total cells: 1
Throughput: 389.71 cells/s
hypertable> select department:old from User;
B00 department:old Human Resource

Elapsed time: 0.00 s
Avg value size: 14.00 bytes
Avg key size: 7.00 bytes
Throughput: 17632.24 bytes/s
Total cells: 1
Throughput: 839.63 cells/s
hypertable> select department:^o from User;
B00 department:old Human Resource

Elapsed time: 0.00 s
Avg value size: 14.00 bytes
Avg key size: 7.00 bytes
Throughput: 39325.84 bytes/s
```

```
Total cells: 1
Throughput: 1872.66 cells/s
```

Installation

Hypertable standalone can downlaod here: http://hypertable.com/download/0971/
(this paper is tried Hypertable Binary Version 0.9.7.1)
Before follow steps below, you better to make a user as owner.

Install Filesystem Hierarchy Standard the directories.

```
$ sudo mkdir /etc/opt/hypertable /var/opt/hypertable
$ sudo chown hypertable:hypertable /etc/opt/hypertable /var/opt/hypertable

$ /opt/hypertable/$VERSION/bin/fhsize.sh
```

Set current link.

```
$ cd /opt/hypertable
$ ln -s $VERSION current
```

Edit hypertable configuration file.
My local configuration file for experiment is below :

```
cat /opt/hypertable/current/conf/hypertable.cfg [~]
#
# hypertable.cfg
#

# HDFS Broker
HdfsBroker.Hadoop.ConfDir=/etc/hadoop/conf

# Ceph Broker
CephBroker.MonAddr=10.0.1.245:6789

# Local Broker
DfsBroker.Local.Root=fs/local

# DFS Broker - for clients
DfsBroker.Port=38030

# Hyperspace
# for one instance only
Hyperspace.Replica.Host=localhost
# for multi instance repli
# Hyperspace.Replica.Hostt=PC-7370.local or pc-2473.cyberagent.co.jp
Hyperspace.Replica.Port=38040
Hyperspace.Replica.Dir=hyperspace

# Hypertable.Master
Hypertable.Master.Port=38050

# Hypertable.RangeServer
Hypertable.RangeServer.Port=38060

Hyperspace.KeepAlive.Interval=30000
Hyperspace.Lease.Interval=1000000
Hyperspace.GracePeriod=200000

# ThriftBroker
ThriftBroker.Port=38080
```

make a rc script like this:

```
#!/bin/sh /usr/local/etc/hypertable.sh

HYPERTABLE_BASE_DIR="/opt/hypertable/current/bin/"
__start( ) {
${HYPERTABLE_BASE_DIR}start-all-servers.sh local
}

__stop( ) {
${HYPERTABLE_BASE_DIR}stop-servers.sh
}


__restart( ) {
${HYPERTABLE_BASE_DIR}stop-servers.sh && ${HYPERTABLE_BASE_DIR}start-all-servers.sh local
}

##
# :: main ::
case "${1}" in
start|stop|restart)
# [ -x ${HYPERTABLE_DAEMON} ] || exit 2
__${1}
;;
*)
;;
esac
```

set environment:

```
# for hypertable
export PATH=/opt/hypertable/current/lib/java/:$PATH

# aliases
alias ht="/opt/hypertable/current/bin/ht"
```

then, start and test:

```
$ /usr/local/etc/hypertable.sh start [~]
DFS broker: available file descriptors: 256
Started DFS Broker (local)
Started Hyperspace
Started Hypertable.Master
Started Hypertable.RangeServer
Started ThriftBroker

$ ht shell
Welcome to the hypertable command interpreter.
For information about Hypertable, visit http://hypertable.com

Type 'help' for a list of commands, or 'help shell' for a
list of shell meta commands.

hypertable> use '/';

Elapsed time: 0.00 s
hypertable> get listing;
Tutorial (namespace)
sys (namespace)
test (namespace)
tmp (namespace)

Elapsed time: 0.00 s
hypertable> use test;

Elapsed time: 0.01 s
hypertable> show tables;
```

finally, shutdown it:

```
$ /usr/local/etc/hypertable.sh stop [~]
Killing ThriftBroker.pid 30946
Shutdown master complete
Sending shutdown command
Shutdown range server complete
Sending shutdown command to DFS broker
Killing DfsBroker.local.pid 30729
Killing Hyperspace.pid 30725
Shutdown thrift broker complete
Shutdown hypertable master complete
Shutdown DFS broker complete
Shutdown hyperspace complete
```

Implementation

Look at source first. You, engineer, love it as nothing better than understanding by source.
So, how to use this for work on actual fields.

Let's try each methods of retreiving data below:

0. Prerequisite

1. HQL Query

2. Shared Mutex

3. Asynchronization

4. Put them into one table togather

5. Delete all or each value

Let's try all of them.

0. Prerequisite

0-1. Create namespace

```
hypertable> user '/';
hypertable> CREATE NAMESPACE JustDoIt;
```

0-2. Required libraries

```
<component name="libraryTable">
<library name="cdh3">
<CLASSES>
<root url="file:///opt/hypertable/current/lib/java/cdh3" />
</CLASSES>
<JAVADOC />
<SOURCES />
<jarDirectory url="file:///opt/hypertable/current/lib/java/cdh3" recursive="false" />
</library>
</component>
<component name="libraryTable">
<library name="cdh4">
<CLASSES>
<root url="file:///opt/hypertable/current/lib/java/cdh4" />
</CLASSES>
<JAVADOC />
<SOURCES />
<jarDirectory url="file:///opt/hypertable/current/lib/java/cdh4" recursive="false" />
</library>
</component>
<component name="libraryTable">
<library name="org.apache.commons:com.springsource.org.apache.commons.cli:1.2.0"
type="repository">
<properties maven-id="org.apache.commons:com.springsource.org.apache.commons.cli:1.2.0" />
<CLASSES>
<root url="jar://$PROJECT_DIR$/lib/com.springsource.org.apache.commons.cli-1.2.0.jar!/" />
</CLASSES>
<JAVADOC />
<SOURCES />
</library>
</component>
<component name="libraryTable">
<library name="org.apache.commons:com.springsource.org.apache.commons.httpclient:3.1.0"
type="repository">
<properties maven-id="org.apache.commons:com.springsource.org.apache.commons.httpclient:3.1.0" />
<CLASSES>
<root url="jar://$PROJECT_DIR$/lib/com.springsource.org.apache.commons.httpclient-3.1.0.jar!/" />
<root url="jar://$PROJECT_DIR$/lib/com.springsource.org.apache.commons.codec-1.3.0.jar!/" />
<root url="jar://$PROJECT_DIR$/lib/com.springsource.org.apache.commons.logging-1.1.1.jar!/" />
</CLASSES>
<JAVADOC />
```

```
<SOURCES />
</library>
</component>
<component name="libraryTable">
<library name="org.apache.directory.studio:org.slf4j.log4j12:1.7.1" type="repository">
<properties maven-id="org.apache.directory.studio:org.slf4j.log4j12:1.7.1" />
<CLASSES>
<root url="jar://$PROJECT_DIR$/lib/org.slf4j.log4j12-1.7.1.jar!/" />
<root url="jar://$PROJECT_DIR$/lib/slf4j-log4j12-1.7.1.jar!/" />
<root url="jar://$PROJECT_DIR$/lib/org.slf4j.api-1.7.1.jar!/" />
<root url="jar://$PROJECT_DIR$/lib/slf4j-api-1.7.1.jar!/" />
<root url="jar://$PROJECT_DIR$/lib/org.apache.logging.log4j-1.2.17.jar!/" />
<root url="jar://$PROJECT_DIR$/lib/log4j-1.2.17.jar!/" />
<root url="jar://$PROJECT_DIR$/lib/annotations-1.0.0.jar!/" />
</CLASSES>
<JAVADOC>
<root url="jar://$PROJECT_DIR$/lib/slf4j-log4j12-1.7.1-javadoc.jar!/" />
<root url="jar://$PROJECT_DIR$/lib/slf4j-api-1.7.1-javadoc.jar!/" />
<root url="jar://$PROJECT_DIR$/lib/log4j-1.2.17-javadoc.jar!/" />
</JAVADOC>
<SOURCES />
</library>
</component>
<component name="libraryTable">
<library name="org.apache.log4j:com.springsource.org.apache.log4j:1.2.16" type="repository">
<properties maven-id="org.apache.log4j:com.springsource.org.apache.log4j:1.2.16" />
<CLASSES>
<root url="jar://$PROJECT_DIR$/lib/com.springsource.org.apache.log4j-1.2.16.jar!/" />
<root url="jar://$PROJECT_DIR$/lib/com.springsource.javax.jms-1.1.0.jar!/" />
</CLASSES>
<JAVADOC />
<SOURCES />
</library>
</component>
<component name="libraryTable">
<library name="org.apache.thrift:libthrift:0.9.0" type="repository">
<properties maven-id="org.apache.thrift:libthrift:0.9.0" />
<CLASSES>
<root url="jar://$PROJECT_DIR$/lib/libthrift-0.9.0.jar!/" />
<root url="jar://$PROJECT_DIR$/lib/slf4j-api-1.5.8.jar!/" />
<root url="jar://$PROJECT_DIR$/lib/commons-lang-2.5.jar!/" />
<root url="jar://$PROJECT_DIR$/lib/servlet-api-2.5.jar!/" />
<root url="jar://$PROJECT_DIR$/lib/httpclient-4.1.3.jar!/" />
<root url="jar://$PROJECT_DIR$/lib/httpcore-4.1.3.jar!/" />
<root url="jar://$PROJECT_DIR$/lib/commons-logging-1.1.1.jar!/" />
<root url="jar://$PROJECT_DIR$/lib/commons-codec-1.4.jar!/" />
</CLASSES>
<JAVADOC />
<SOURCES />
</library>
</component>
<component name="libraryTable">
<library name="org.codehaus.jackson:com.springsource.org.codehaus.jackson.mapper:1.4.3"
type="repository">
<properties maven-id="org.codehaus.jackson:com.springsource.org.codehaus.jackson.mapper:1.4.3" />
<CLASSES>
<root url="jar://$PROJECT_DIR$/lib/com.springsource.org.codehaus.jackson.mapper-1.4.3.jar!/" />
<root url="jar://$PROJECT_DIR$/lib/com.springsource.org.codehaus.jackson-1.4.3.jar!/" />
</CLASSES>
<JAVADOC />
<SOURCES />
```

```
</library>
</component>
```

0-3. DDL included a source

```
drop table if exists sample;
create table (col);
```

1. HQL Query : a source of HQL Query.

HQLTest.java

```
package org.hypertable.examples.justdoit;

/**
* HQLTest
* User: edydkim.github.com
* Date: 13/03/25
* Time: 20:20
*/

import java.nio.ByteBuffer;
import java.util.List;
import java.util.Iterator;

import org.hypertable.thrift.ThriftClient;
import org.hypertable.thriftgen.*;

public class HQLTest {
public static void main(String [] args) {
ThriftClient client = null;
long ns = -1;
try {
client = ThriftClient.create("localhost", 38080);
ns = client.namespace_open("JustDoIt");

Cell cell;
Key key;
String str;

// HQL examples
show(client.hql_query(ns, "show tables").toString());
show(client.hql_query(ns, "drop table if exists JustDoItHQLTest").toString());
show(client.hql_query(ns, "create table JustDoItHQLTest (col)").toString());
show(client.hql_query(ns, "select * from JustDoItHQLTest").toString());
// Schema example
Schema schema = new Schema();
schema = client.table_get_schema(ns, "JustDoItHQLTest");

Iterator ag_it = schema.access_groups.keySet().iterator();
show("Access groups:");
while (ag_it.hasNext()) {
show("\t" + ag_it.next());
}

Iterator cf_it = schema.column_families.keySet().iterator();
show("Column families:");
while (cf_it.hasNext()) {
```

```
show("\t" + cf_it.next());
}

// set mutator
long mutator = client.mutator_open(ns, "JustDoItHQLTest", 0, 0);

try {
cell = new Cell();
key = new Key();
key.setRow("000");
key.setColumn_family("col");
key.setColumn_qualifier("test");
cell.setKey(key);
str = "beer";
cell.setValue(ByteBuffer.wrap(str.getBytes()));
client.mutator_set_cell(mutator, cell);

cell = new Cell();
key = new Key();
key.setRow("000");
key.setColumn_family("col");
cell.setKey(key);
str = "カクテル";
cell.setValue(ByteBuffer.wrap(str.getBytes("UTF-8")));
client.mutator_set_cell(mutator, cell);
}
finally {
client.mutator_close(mutator);
}

HqlResult hqlResult = client.hql_query(ns, "select * from JustDoItHQLTest");
List<Cell> cells = hqlResult.cells;
int qualifier_count = 0;
show("Cells:");
for(Cell c : cells) {
byte[] array = c.value.array();
byte[] values = new byte[c.value.remaining()];
for (int i = 0; i < values.length; i++) {
values[i] = array[i + c.value.position()];
}

show("\t" + new String(values) + " : " + c.toString());
if (c.key.isSetColumn_qualifier() && c.key.column_qualifier.length() == 0)
qualifier_count++;
}

if (qualifier_count != 1) {
show("ERROR: Expected qualifier_count of 1, got " + qualifier_count);
client.namespace_close(ns);
System.exit(1);
}

client.namespace_close(ns);
} catch (Exception e) {
e.printStackTrace();
try {
if (client != null && ns != -1)
client.namespace_close(ns);
}
catch (Exception ce) {
System.err.println("Problem closing namespace \"JustDoIt\" - " + e.getMessage());
}
System.exit(1);
```

```
}
}

private static void show(String line) {
System.out.println(line);
}
```

```
}
```

The Result :

```
HqlResult(results:[JustDoItHQLTest])
HqlResult()
HqlResult()
HqlResult(cells:[])
Access groups:
default
Column families:
col
Cells:
カクテル : Cell(key:Key(row:000, column_family:col, column_qualifier:,
timestamp:1364277553876608002, revision:1364277553876608002, flag:INSERT),
value:java.nio.HeapByteBuffer[pos=99 lim=111 cap=190])
beer : Cell(key:Key(row:000, column_family:col, column_qualifier:test,
timestamp:1364277553876608001, revision:1364277553876608001, flag:INSERT),
value:java.nio.HeapByteBuffer[pos=183 lim=187 cap=190])
```

2. Shared Mutex : a source of Shared Mutex.

SharedMutexTest.java

```
package org.hypertable.examples.justdoit;

/**
* SharedMutexTest
* User: edydkim.github.com
* Date: 13/03/25
* Time: 20:20
*/

import org.hypertable.thrift.ThriftClient;
import org.hypertable.thriftgen.*;

import java.nio.ByteBuffer;
import java.util.Iterator;
import java.util.List;

public class SharedMutexTest {
public static void main(String [] args) {
ThriftClient client = null;
long ns = -1;
try {
client = ThriftClient.create("localhost", 38080);
ns = client.namespace_open("JustDoIt");

Cell cell;
Key key;
String str;

// HQL examples
show(client.hql_query(ns, "show tables").toString());
show(client.hql_query(ns, "drop table if exists JustDoItSMutexTest").toString());
show(client.hql_query(ns, "create table JustDoItSMutexTest (col)").toString());
show(client.hql_query(ns, "select * from JustDoItSMutexTest").toString());
```

```
// Schema example
Schema schema = new Schema();
schema = client.table_get_schema(ns, "JustDoItSMutexTest");

Iterator ag_it = schema.access_groups.keySet().iterator();
show("Access groups:");
while (ag_it.hasNext()) {
show("\t" + ag_it.next());
}

Iterator cf_it = schema.column_families.keySet().iterator();
show("Column families:");
while (cf_it.hasNext()) {
show("\t" + cf_it.next());
}

// shared mutator
show("Shared mutator");
MutateSpec mutate_spec = new MutateSpec();
mutate_spec.setAppname("test-java");
mutate_spec.setFlush_interval(1000);

cell = new Cell();
key = new Key();
key.setRow("v1");
key.setColumn_family("col");
cell.setKey(key);
String vtmp = "IamFirst";
cell.setValue( ByteBuffer.wrap(vtmp.getBytes()) );
client.offer_cell(ns, "JustDoItSMutexTest", mutate_spec, cell);

cell = new Cell();
key = new Key();
key.setRow("v2");
key.setColumn_family("col");
cell.setKey(key);
vtmp = "IamSecond";
cell.setValue( ByteBuffer.wrap(vtmp.getBytes()) );
client.shared_mutator_refresh(ns, "JustDoItSMutexTest", mutate_spec);
client.shared_mutator_set_cell(ns, "JustDoItSMutexTest", mutate_spec, cell);
Thread.sleep(2000);

// scan
show("Full scan");
ScanSpec scanSpec = new ScanSpec(); // empty scan spec select all
long scanner = client.scanner_open(ns, "JustDoItSMutexTest", scanSpec);

try {
List<Cell> sCells = client.scanner_get_cells(scanner);

while (sCells.size() > 0) {
for (Cell c : sCells) {
byte[] tmp = c.getValue();
String s = new String(tmp);
show("\t" + s);
}
sCells = client.scanner_get_cells(scanner);
}
}
finally {
client.scanner_close(scanner);
}
// restricted scanspec
```

```
scanSpec.addToColumns("col:/^.*$/");
scanSpec.setRow_regexp("v.*");
scanSpec.setValue_regexp("Second");
scanner = client.scanner_open(ns, "JustDoItSMutexTest", scanSpec);
show("Restricted scan");
try {
List<Cell> ssCells = client.scanner_get_cells(scanner);

while (ssCells.size() > 0) {
for (Cell c : ssCells) {
byte[] tmp = c.getValue();
String s = new String(tmp);
show("\t" + s);
}
ssCells = client.scanner_get_cells(scanner);
}
}
finally {
client.scanner_close(scanner);
}

client.namespace_close(ns);
} catch (Exception e) {
e.printStackTrace();
try {
if (client != null && ns != -1)
client.namespace_close(ns);
}
catch (Exception ce) {
System.err.println("Problem closing namespace \"JustDoIt\" - " + e.getMessage());
}
System.exit(1);
}
}

private static void show(String line) {
System.out.println(line);
```

```
}
}
```

The Result :

```
HqlResult(results:[JustDoItHQLTest, JustDoItSMutexTest])
HqlResult()
HqlResult()
HqlResult(cells:[])
Access groups:
default
Column families:
col
Shared mutator
Full scan
IamFirst
IamSecond
Restricted scan
IamSecond
```

3. Asynchronization : a source of Asynchronization.

ASyncTest.java

```
package org.hypertable.examples.justdoit;

/**
* ASyncTest
* User: edydkim.github.com
* Date: 13/03/25
* Time: 20:20
*/

import org.hypertable.thrift.ThriftClient;
import org.hypertable.thriftgen.*;

import java.nio.ByteBuffer;
import java.util.Iterator;

public class ASyncTest {
public static void main(String [] args) {
ThriftClient client = null;
long ns = -1;
try {
client = ThriftClient.create("localhost", 38080);
ns = client.namespace_open("JustDoIt");

Cell cell;
Key key;
String str;

// HQL examples
show(client.hql_query(ns, "show tables").toString());
show(client.hql_query(ns, "drop table if exists JustDoItASyncTest").toString());
show(client.hql_query(ns, "create table JustDoItASyncTest (col)").toString());
show(client.hql_query(ns, "select * from JustDoItASyncTest").toString());
// Schema example
Schema schema = new Schema();
```

```
schema = client.table_get_schema(ns, "JustDoItASyncTest");

Iterator ag_it = schema.access_groups.keySet().iterator();
show("Access groups:");
while (ag_it.hasNext()) {
show("\t" + ag_it.next());
}

Iterator cf_it = schema.column_families.keySet().iterator();
show("Column families:");
while (cf_it.hasNext()) {
show("\t" + cf_it.next());
}

// asynchronous api
long future=0;
long mutator_async_1=0;
long mutator_async_2=0;
long test_scanner=0;
int expected_cells = 6;
int num_cells = 0;

try {
show("Asynchronous mutator");
future = client.future_open(0);
mutator_async_1 = client.async_mutator_open(ns, "JustDoItASyncTest", future, 0);
mutator_async_2 = client.async_mutator_open(ns, "JustDoItASyncTest", future, 0);
Result result;

cell = new Cell();
key = new Key();
key.setRow("alpha");
key.setColumn_family("col");
cell.setKey(key);
String vtmp = "winner";
cell.setValue( ByteBuffer.wrap(vtmp.getBytes()) );
client.async_mutator_set_cell(mutator_async_1, cell);

cell = new Cell();
key = new Key();
key.setRow("bravo");
key.setColumn_family("col");
cell.setKey(key);
vtmp = "looser";
cell.setValue( ByteBuffer.wrap(vtmp.getBytes()) );
client.async_mutator_set_cell(mutator_async_2, cell);

client.async_mutator_flush(mutator_async_1);
client.async_mutator_flush(mutator_async_2);

int num_flushes=0;
while (true) {
result = client.future_get_result(future, 0);
if (result.is_empty || result.is_error || result.is_scan)
break;
num_flushes++;
}
if (num_flushes > 2) {
show("Expected 2 flushes, received " + num_flushes);
System.exit(1);
}
if (client.future_is_cancelled(future) || client.future_is_full(future) ||
!client.future_is_empty(future) || client.future_has_outstanding(future)) {
```

```
show("Future object in unexpected state");
System.exit(1);
}
}
finally {
client.async_mutator_close(mutator_async_1);
client.async_mutator_close(mutator_async_2);
}

try {
show("Asynchronous scan");
ScanSpec ss = new ScanSpec();
test_scanner = client.async_scanner_open(ns, "JustDoItASyncTest", future, ss);
Result result;
while (true) {
result = client.future_get_result(future, 0);
if (result.is_empty || result.is_error || !result.is_scan)
break;
for (Cell c : result.cells) {
byte[] tmp = c.getValue();
String s = new String(tmp);
show(s);
num_cells++;
}
if (num_cells >= 2) {
client.future_cancel(future);
break;
}
}
if (!client.future_is_cancelled(future)) {
show("Expected future object to be cancelled");
System.exit(1);
}
}
finally {
client.async_scanner_close(test_scanner);
client.future_close(future);
}
if (num_cells != 2) {
show("Expected " + expected_cells + " cells got " + num_cells);
System.exit(1);
}

client.namespace_close(ns);
} catch (Exception e) {
e.printStackTrace();
try {
if (client != null && ns != -1)
client.namespace_close(ns);
}
catch (Exception ce) {
System.err.println("Problem closing namespace \"JustDoIt\" - " + e.getMessage());
}
System.exit(1);
}
}

private static void show(String line) {
System.out.println(line);
```

```
}
}
```

The Result :

```
HqlResult(results:[JustDoItASyncTest, JustDoItHQLTest, JustDoItSMutexTest])
HqlResult()
HqlResult()
HqlResult(cells:[])
Access groups:
default
Column families:
col
Asynchronous mutator
Asynchronous scan
winner
looser
```

Put them into one table togather.

Try it to make one class then check how to be synchronized and asynchronized.

```
create table AllTest(hql_col, mutex_col, async_col);
```

Delete all or each value

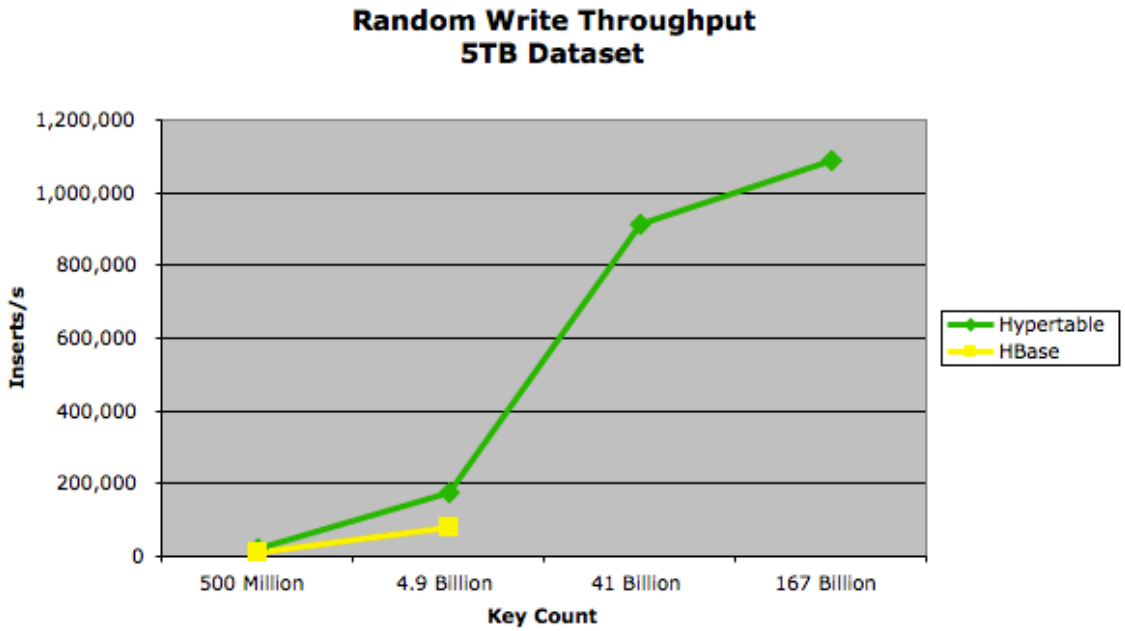
Key is necessary and column or column qualitifier is optional.

```
delete * from JustDoItHQLTest where row = '000';
or
delete col from JustDoItHQLTest where row = '000';
```

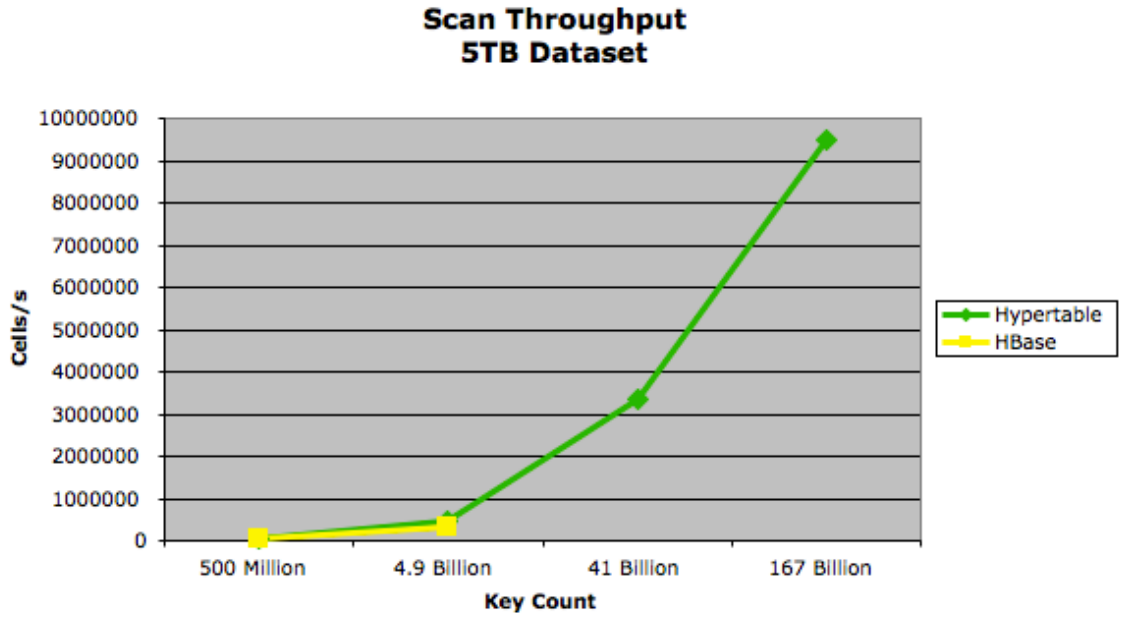
Benchmark released (Hypertable vs HBase)

See details http://hypertable.com/why_hypertable/hypertable_vs_hbase_2/ for performance evaluation
*3 reference source : http://hypertable.com

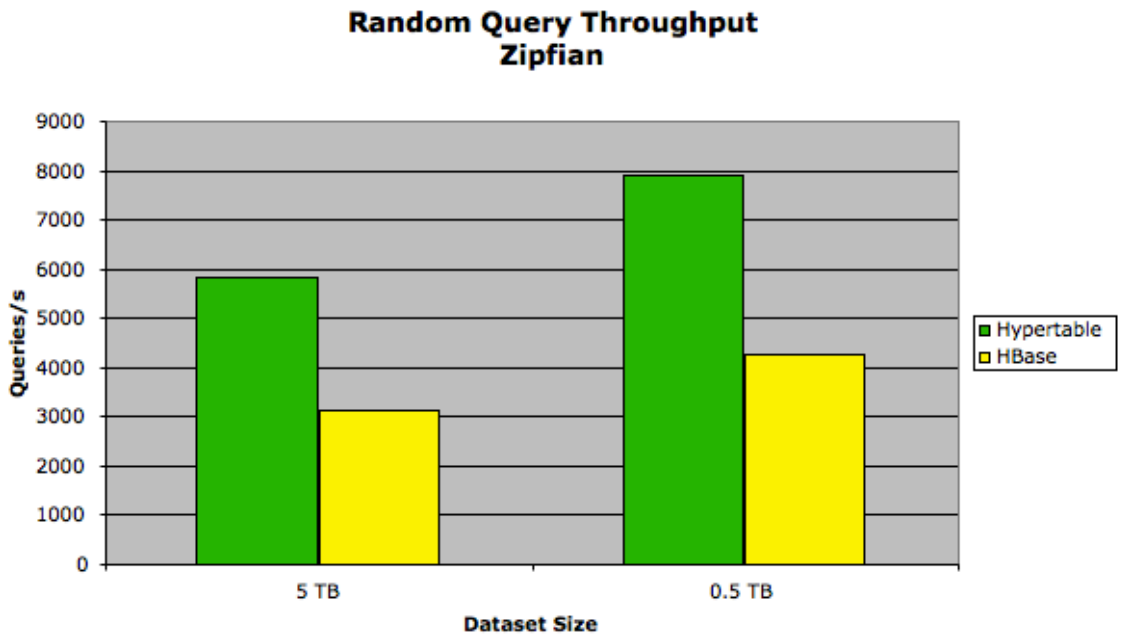
Random write



Scan



Random read



Summary

Map/Reduce model is not new concept but validated High Scalability in distributed storage systems by Google. The problem is which is best in own system. Hypertable have a various implementation method to access data and is optimized query by query caching. I believe these are very useful to make a rapid real application.

References

*1: some quotation taken from the book - Professional NoSQL by Shashank Tiwari
*2, *3: source from hypertable Inc., http://www.hypertable.com/
Github open source group: https://github.com/hypertable
Paper: HBase and Hypertable for large scale distributed storage systems by Ankur Khetrapal, Vinay Ganesh@cs.purdue.edu

Here is as Japanese.

概要

データベースの選定、特にNoSQLデータベースにおいて、一番重要視されることは拡張性と生産性である。本書では左記2点の観点からロ

Tip:DBMS属するデータの種類

- RDBMS &
  ORDBMS（関係データベース管理システムとオブジェクト関係データベース管理システム）：構造型、又はオブジェクト指向デー
- Document-DBMS（ドキュメントデータベース管理システム）：xmlやjsonのようなテキスト形の準構造型データ
- Graph-DBMS（グラフデータベース）：頂点と節点の関係性を表すデータ
- Column-oriented DBMS（列指向データベース管理システム）：非定型の類似属性データ
- 一般的にターミノロジーとして、Row-ColumnをTurple-Attribute、又はRecord-Elementと呼ぶ。数学やコンピュータサイエンスな

目次

Contents 参照。

序論

Hypertable（ハイパーテーブル）はMap/Reduceアーキテクチャ基盤のグーグルBigTableクーロンプロジェクトあり、同様のHBaseのよう

インデックス情報はプライマリーテーブル内、同ネームスペースに保持される。（OracleなどRDBMSに例えるとプライマリーテーブルはシ

内容

下記順にシステム構成図を基に、インストール方法、実装方法、ベンチマークについて説明する。
詳細はDesign、Installation、Implementation、Benchmark released (Hypertable vs HBase) 参照。

まとめ

Map/Reduceモデルより分散環境の可用性を向上する試みはGoogleなどに実検証されている。課題は如何に利便性を向上させられるかにあ

参考文献

References 参照。