Hypertable

Doug Judd

www.hypertable.org

Background

- Zvents plan is to become the “Google” of local search
- Identified the need for a scalable DB
- No solutions existed
- Bigtable was the logical choice
- Project started February 2007

Zvents Deployment

- Traffic Reports
- Change Log
- Writing 1 Billion cells/day

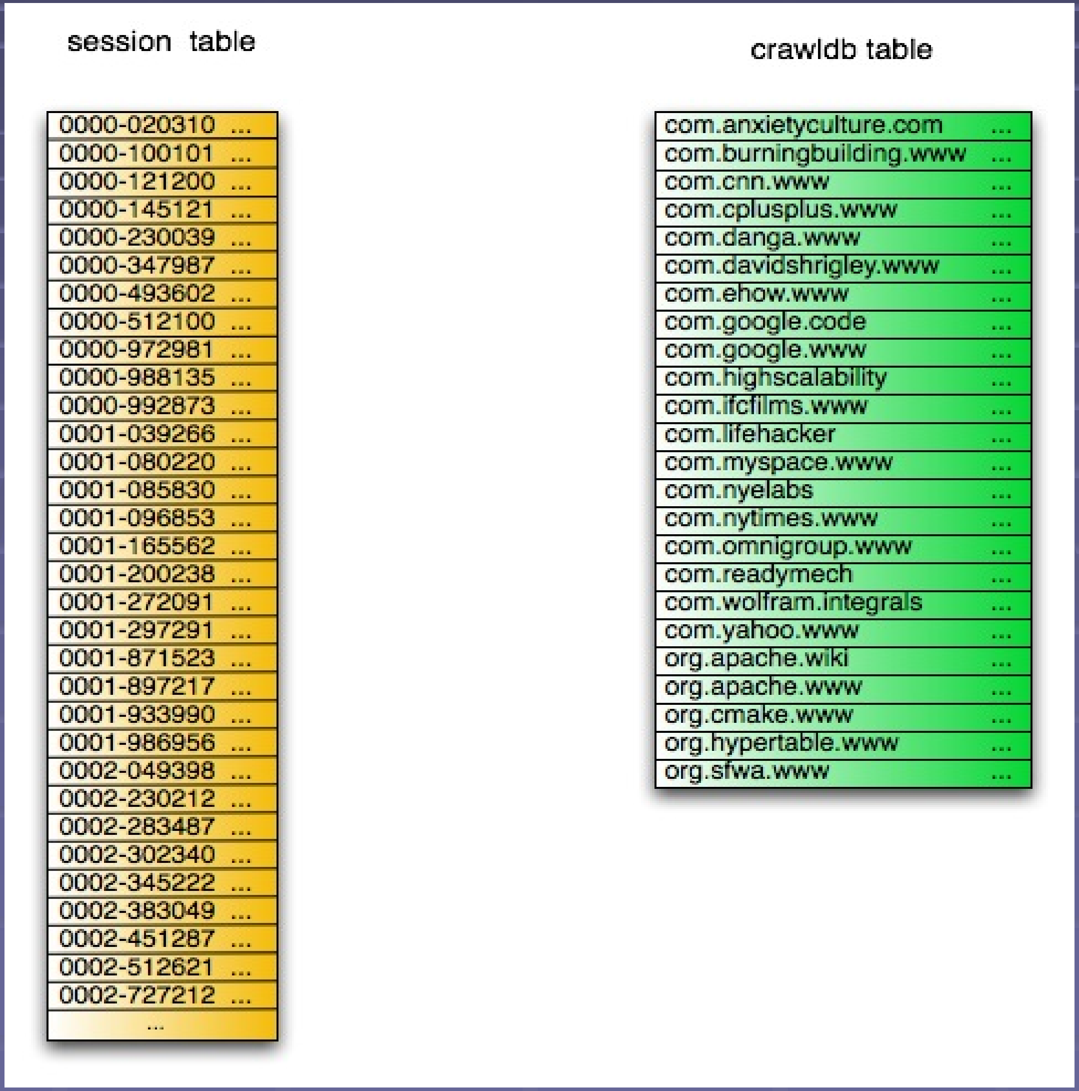
Baidu Deployment

- Log processing/viewing app injecting approximately 500GB of data per day
- 120-node cluster running Hypertable and HDFS
  - 16GB RAM
  - 4x dual core Xeon
  - 8TB storage
- Developed in-house fork with modifications for scale
- Working on a new crawl DB to store up to 1 petabyte of crawl data

Hypertable

- What is it?
  - Open source Bigtable clone
  - Manages massive sparse tables with timestamped cell versions
  - Single primary key index
- What is it not?
  - No joins
  - No secondary indexes (not yet)
  - No transactions (not yet)

Scaling (part I)



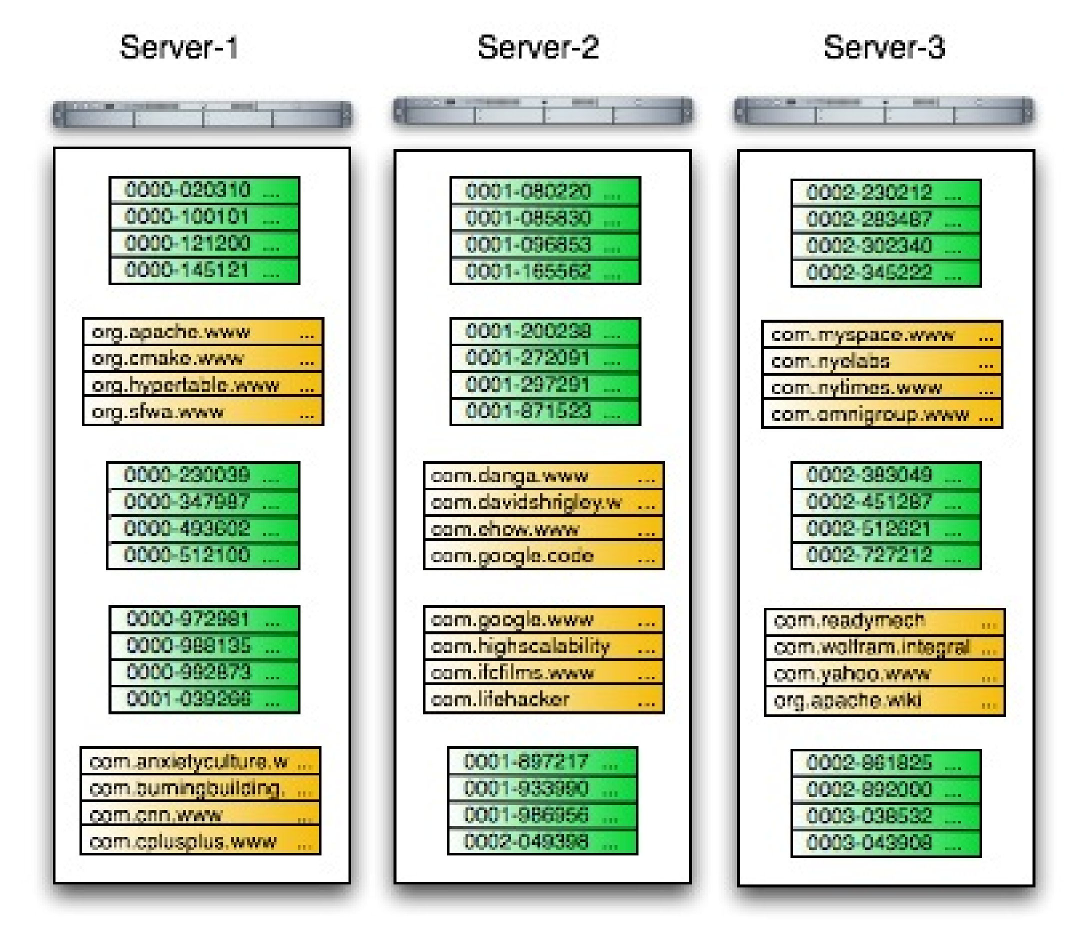
session table

0000-020310	...
0000-100101	...
0000-121200	...
0000-145121	...
0000-230039	...
0000-347987	...
0000-493602	...
0000-512100	...
0000-972981	...
0000-988135	...
0000-992873	...
0001-039266	...
0001-080220	...
0001-085830	...
0001-096853	...
0001-165562	...
0001-200238	...
0001-272091	...
0001-297291	...
0001-871523	...
0001-897217	...
0001-933990	...
0001-986956	...
0002-049398	...
0002-230212	...
0002-283487	...
0002-302340	...
0002-345222	...
0002-383049	...
0002-451287	...
0002-512621	...
0002-727212	...
...	

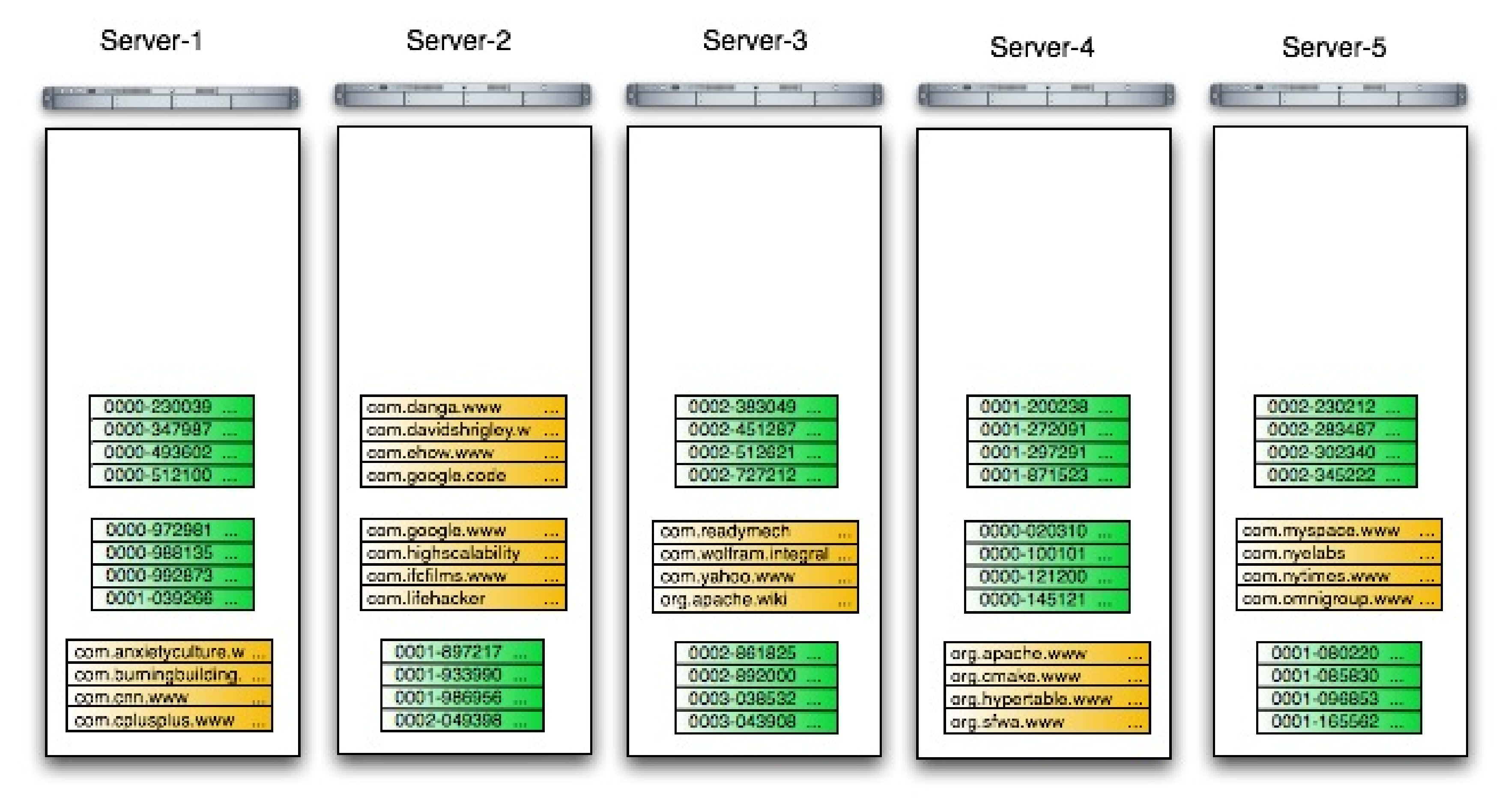
crawldb table

com.anxietyculture.com	...
com.burningbuilding.www	...
com.cnn.www	...
com.cplusplus.www	...
com.danga.www	...
com.davidshrigley.www	...
com.ehow.www	...
com.google.code	...
com.google.www	...
com.highscalability	...
com.ifcfilms.www	...
com.lifehacker	...
com.myspace.www	...
com.nyelabs	...
com.nytimes.www	...
com.omnigroup.www	...
com.readymech	...
com.wolfram.integrals	...
com.yahoo.www	...
org.apache.wiki	...
org.apache.www	...
org.cmake.www	...
org.hypertable.www	...
org.sfwa.www	...

Scaling (part II)



Scaling (part III)



System Overview

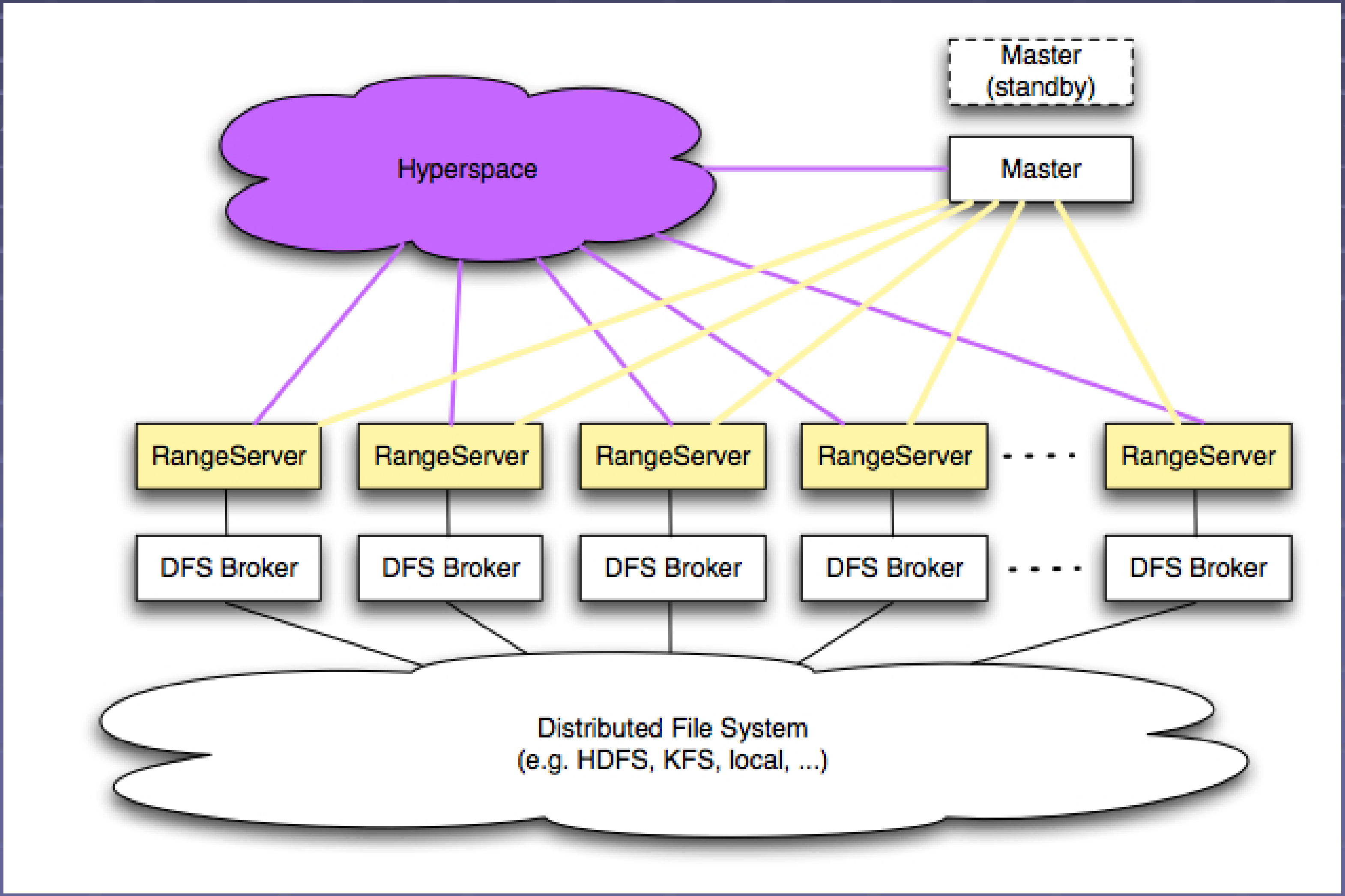
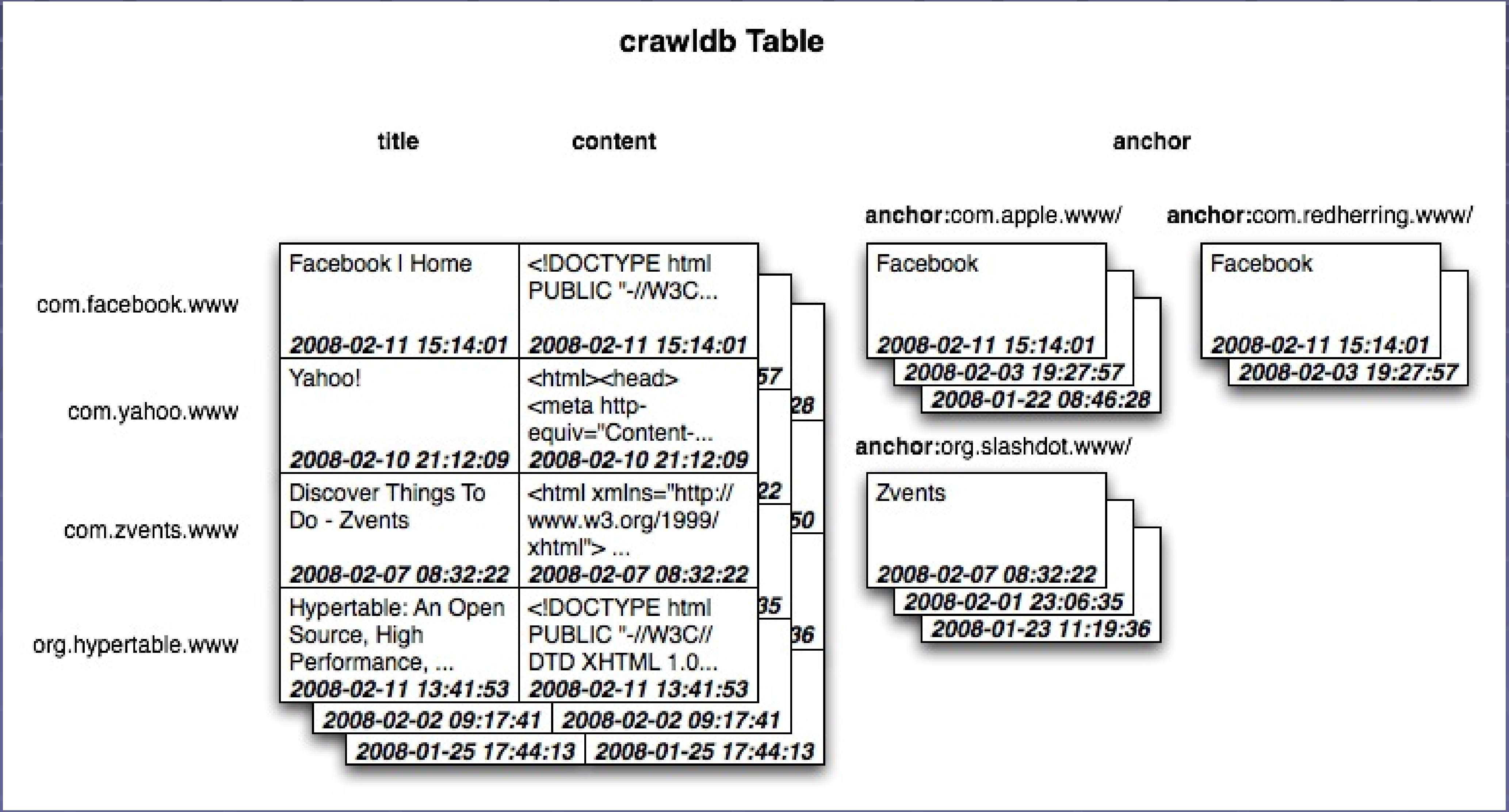


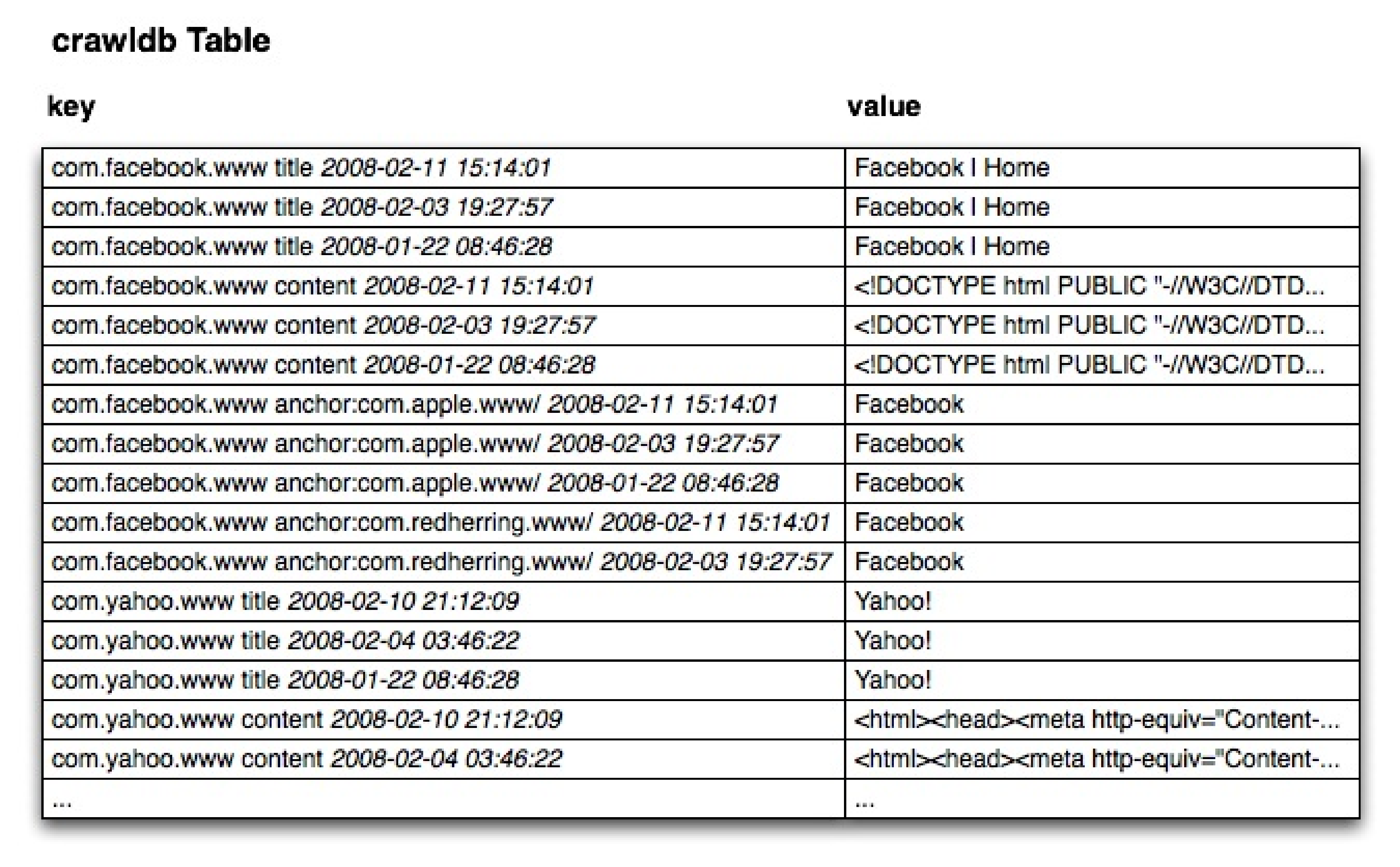
Table: Visual Representation



crawldb Table

	title	content	anchor		
			anchor:com.apple.www/	anchor:com.redherring.www/	anchor:org.slashdot.www/
com.facebook.www	Facebook I Home 2008-02-11 15:14:01	<!DOCTYPE html PUBLIC "-//W3C... 2008-02-11 15:14:01	Facebook 2008-02-11 15:14:01 2008-02-03 19:27:57 2008-01-22 08:46:28	Facebook 2008-02-11 15:14:01 2008-02-03 19:27:57	
com.yahoo.www	Yahoo! 2008-02-10 21:12:09	<html><head><meta http-equiv="Content-... 2008-02-10 21:12:09			
com.zvents.www	Discover Things To Do - Zvents 2008-02-07 08:32:22	<html xmlns="http://www.w3.org/1999/xhtml"> ... 2008-02-07 08:32:22			Zvents 2008-02-07 08:32:22 2008-02-01 23:06:35 2008-01-23 11:19:36
org.hypertable.www	Hypertable: An Open Source, High Performance, ... 2008-02-11 13:41:53 2008-02-02 09:17:41 2008-01-25 17:44:13	<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0... 2008-02-11 13:41:53 2008-02-02 09:17:41 2008-01-25 17:44:13			

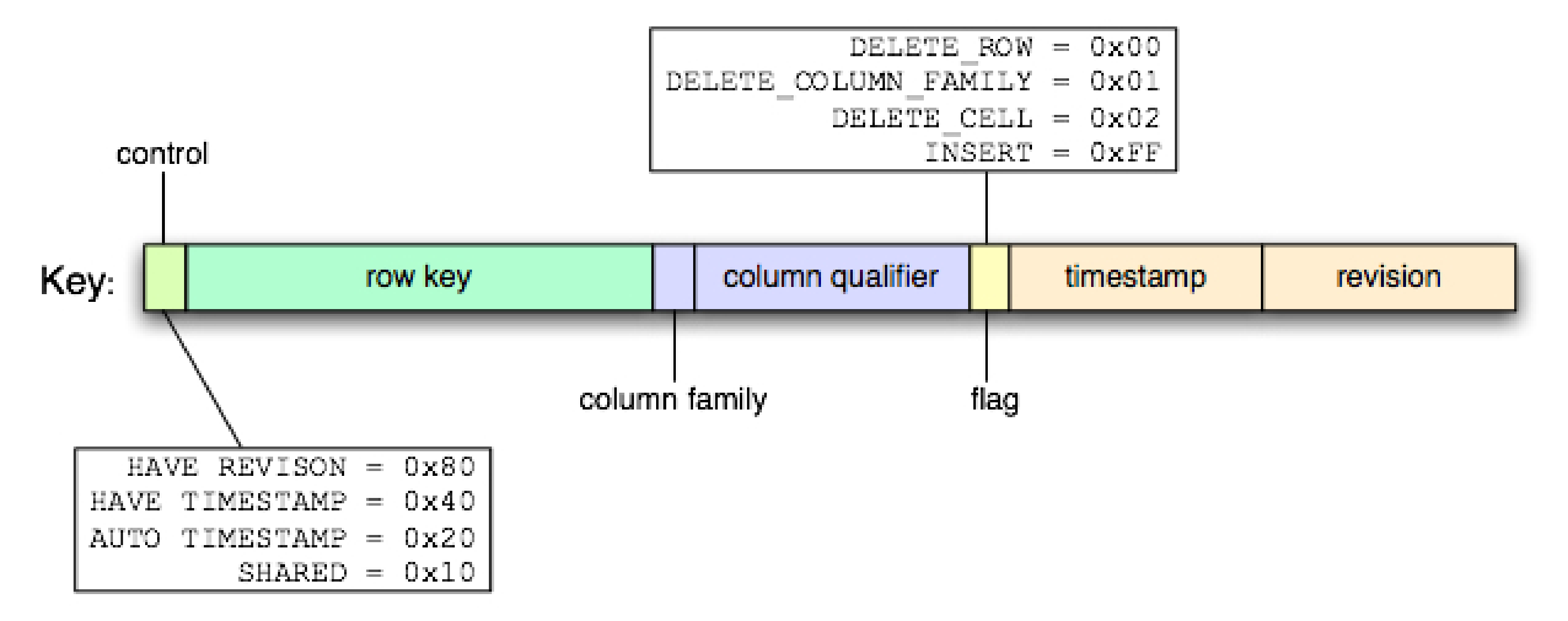
Table: Actual Representation



crawldb Table

key	value
com.facebook.www title *2008-02-11 15:14:01*	Facebook I Home
com.facebook.www title *2008-02-03 19:27:57*	Facebook I Home
com.facebook.www title *2008-01-22 08:46:28*	Facebook I Home
com.facebook.www content *2008-02-11 15:14:01*	<!DOCTYPE html PUBLIC "-//W3C//DTD...
com.facebook.www content *2008-02-03 19:27:57*	<!DOCTYPE html PUBLIC "-//W3C//DTD...
com.facebook.www content *2008-01-22 08:46:28*	<!DOCTYPE html PUBLIC "-//W3C//DTD...
com.facebook.www anchor:com.apple.www/ *2008-02-11 15:14:01*	Facebook
com.facebook.www anchor:com.apple.www/ *2008-02-03 19:27:57*	Facebook
com.facebook.www anchor:com.apple.www/ *2008-01-22 08:46:28*	Facebook
com.facebook.www anchor:com.redherring.www/ *2008-02-11 15:14:01*	Facebook
com.facebook.www anchor:com.redherring.www/ *2008-02-03 19:27:57*	Facebook
com.yahoo.www title *2008-02-10 21:12:09*	Yahoo!
com.yahoo.www title *2008-02-04 03:46:22*	Yahoo!
com.yahoo.www title *2008-01-22 08:46:28*	Yahoo!
com.yahoo.www content *2008-02-10 21:12:09*	<html><head><meta http-equiv="Content-...
com.yahoo.www content *2008-02-04 03:46:22*	<html><head><meta http-equiv="Content-...
...	...

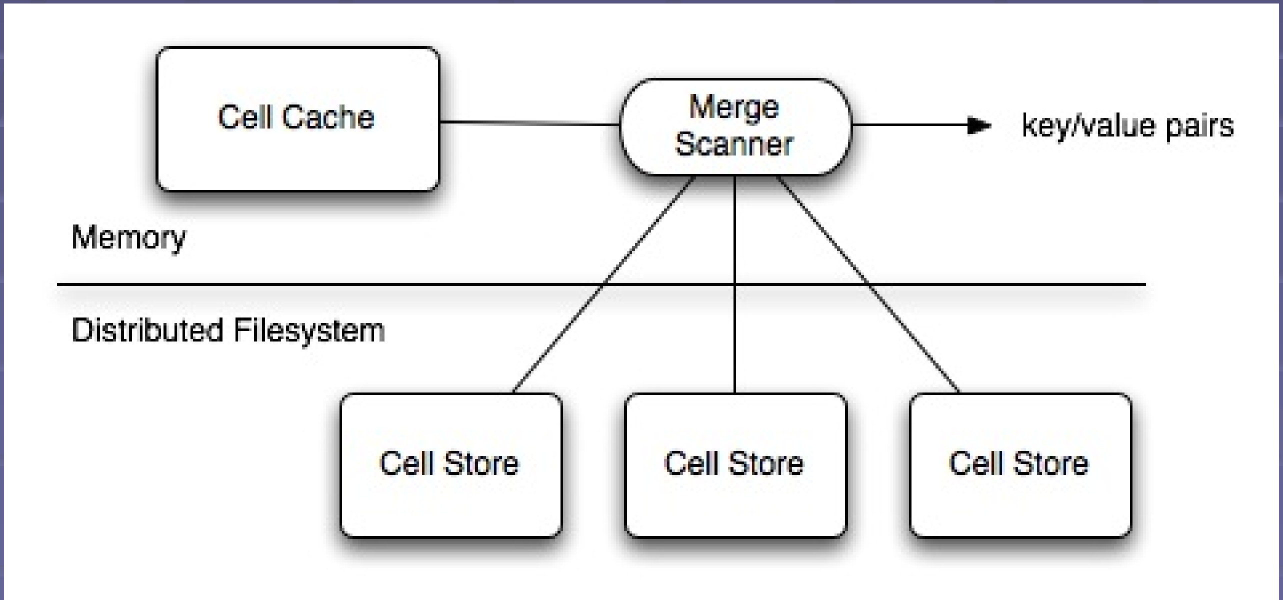
Anatomy of a Key



- MVCC - snapshot isolation
- Bigtable uses copy-on-write
- Timestamp and revision shared by default
- Simple byte-wise comparison

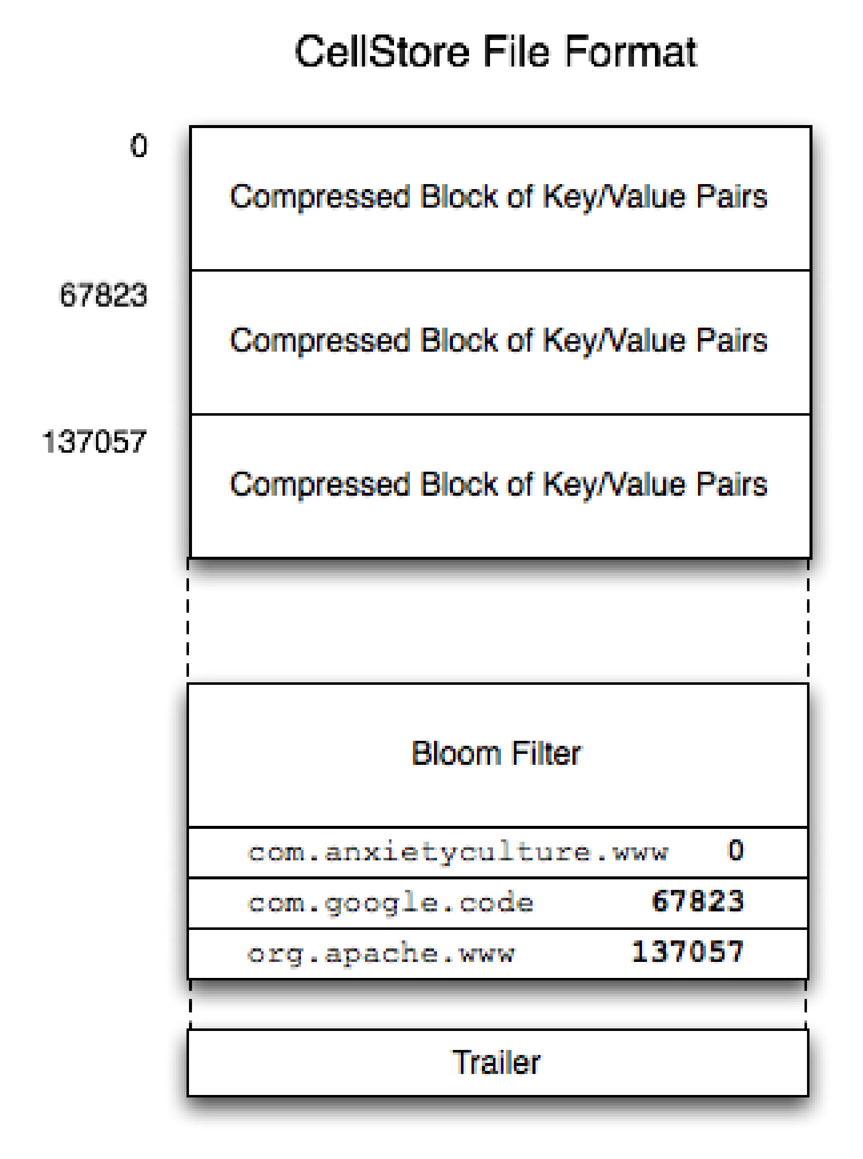
Range Server

- Manages ranges of table data
- Caches updates in memory (CellCache)
- Periodically spills (compacts) cached updates to disk (CellStore)



Range Server: CellStore

- Sequence of 65K blocks of compressed key/value pairs


CellStore File Format
0
67823
137057
Compressed Block of Key/Value Pairs
Compressed Block of Key/Value Pairs
Compressed Block of Key/Value Pairs
Bloom Filter
Trailer

com.anxietyculture.www	0
com.google.code	67823
org.apache.www	137057

Compression

- CellStore and CommitLog Blocks
- Supported Compression Schemes
  - zlib --best
  - zlib --fast
  - lzo
  - quicklz
  - bmz
  - none

Performance Optimizations

- Block Cache
  - Caches CellStore blocks
  - Blocks are cached uncompressed
- Bloom Filter
  - Avoids unnecessary disk access
  - Filter by rows or rows+columns
  - Configurable false positive rate
- Access Groups
  - Physically store co-accessed columns together
  - Improves performance by minimizing I/O

Commit Log

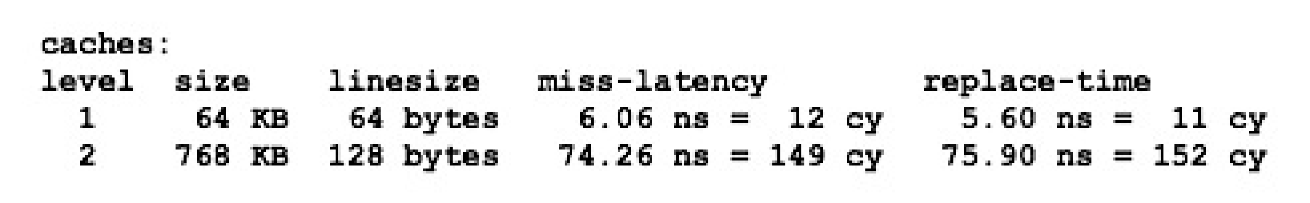
- One per RangeServer
- Updates destined for many Ranges
  - One commit log write
  - One commit log sync
- Log is directory
  - 100MB fragment files
  - Append by creating a new fragment file
- NO_LOG_SYNC option
- Group commit (TBD)

Request Throttling

- RangeServer tracks memory usage
- Config properties
  - `Hypertable.RangeServer.MemoryLimit`
  - `Hypertable.RangeServer.MemoryLimit.Percentage (70%)`
- Request queue is paused when memory usage hits threshold
- Heap fragmentation
  - tcmalloc - good
  - glibc - not so good

C++ vs. Java

- Hypertable is CPU intensive
  - Manages large in-memory key/value map
  - Lots of key manipulation and comparisons
  - Alternate compression codecs (e.g. BMZ)
- Hypertable is memory intensive
  - GC less efficient than explicitly managed memory
  - Less memory means more merging compactions
  - Inefficient memory usage = poor cache performance



```
caches:
level  size    linesize   miss-latency        replace-time
  1     64 KB   64 bytes    6.06 ns =  12 cy    5.60 ns =  11 cy
  2    768 KB  128 bytes   74.26 ns = 149 cy   75.90 ns = 152 cy
```

Language Bindings

- Primary API is C++
- Thrift Broker provides bindings for:
  - Java
  - Python
  - PHP
  - Ruby
  - And more (Perl, Erlang, Haskell, C#, Cocoa, Smalltalk, and Ocaml)

Client API

```
class Client {

  void create_table(const String &name,
                    const String &schema);

  Table *open_table(const String &name);

  void alter_table(const String &name,
                   const String &schema);

  String get_schema(const String &name);

  void get_tables(vector<String> &tables);

  void drop_table(const String &name,
                  bool if_exists);
};
```

Client API (cont.)

```
class Table {
  TableMutator *create_mutator();
  TableScanner *create_scanner(ScanSpec &scan_spec);
};
class TableMutator {
  void set(KeySpec &key, const void *value, int value_len);
  void set_delete(KeySpec &key);
  void flush();
};
class TableScanner {
 bool next(CellT &cell);
};
```

Client API (cont.)

```
class ScanSpecBuilder {
  void set_row_limit(int n);
  void set_max_versions(int n);
  void add_column(const String &name);
  void add_row(const String &row_key);
  void add_row_interval(const String &start, bool sinc,
                        const String &end, bool einc);
  void add_cell(const String &row, const String &column);
  void add_cell_interval(…)
  void set_time_interval(int64_t start, int64_t end);
  void clear();
  ScanSpec &get();
}
```

Testing: Failure Inducer

- Command line argument

--induce-failure=<label>:<type>:<iteration>

- Class definition

```
class FailureInducer {
  public:
    void parse_option(String option);
    void maybe_fail(const String &label);
};
```

- In the code

```
 if (failure_inducer)
    failure_inducer->maybe_fail("split-1");
```

1TB Load Test

- 1TB data
- 8 node cluster
  - 1 1.8 GHz dual-core Opteron
  - 4 GB RAM
  - 3 x 7200 RPM 250MB SATA drives
- Key size = 10 bytes
- Value size = 20KB (compressible text)
- Replication factor: 3
- 4 simultaneous insert clients
- ~ 50 MB/s load (sustained)
- ~ 30 MB/s scan

Performance Test (random read/write)

- Single machine
  - 1 x 1.8 GHz dual-core Opteron
  - 4 GB RAM
- Local Filesystem
- 250MB / 1KB values
- Normal Table / lzo compression

Batched writes	31K inserts/s (31MB/s)
Non-batched writes (serial)	500 inserts/s (500KB/s)
Random reads (serial)	5800 queries/s (5.8MB/s)

Project Status

- Current release is 0.9.2.4 “alpha”
- Waiting for Hadoop 0.21 (fsync)
- TODO for “beta”
  - Namespaces
  - Master directed RangeServer recovery
  - Range balancing

Questions?

- www.hypertable.org