HYPERTABLE INC

WHITE PAPER - MARCH 2012

Hypertable Architecture Overview

Hypertable is an open source, scalable NoSQL database modeled after Bigtable, Google's proprietary scalable database. It is written in C++ for optimum performance and is based on a design that has been proven in the real world for a wide variety of workloads. Hypertable is powerful scalable database infrastructure that can serve as the database underpinnings for next-generation big data applications. In this paper we provide an overview of the Hypertable architecture.

Contents

SYSTEM ARCHITECTURE 1

COMPARISON TO A RELATIONAL DATABASE 1

HYPERTABLE DATA REPRESENTATION 2

NAMESPACES 3

ACCESS GROUPS 3

CLIENT LIBRARY 3

RANGESERVER INSERT HANDLING 3

QUERY ROUTING 4

RANGESERVER QUERY HANDLING 4

FILESYSTEM CELLSTORE FORMAT 5

ADAPTIVE MEMORY ALLOCATION 5

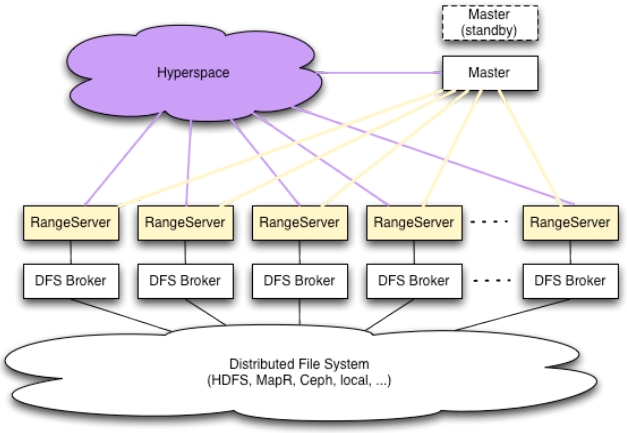
SCALING AND LOAD BALANCING 5

CONCLUSION 6

SYSTEM ARCHITECTURE

Hypertable is a high performance, open source, massively scalable database modeled after Bigtable, Google's proprietary scalable database managment system. Hypertable is designed to utilize a scalable and highly available file system such as Hadoop (HDFS), where high availability is achieved by replicating file data across multiple machines. (HDFS is an open source version of the proprietary Google file system, GFS.)

The diagram below provides an overview of the Hypertable system followed by a brief description of each system component.



Master - The master handles all Meta operations such as creating and deleting tables. Client data does not move through the Master, so the Master can be down for short periods of time without clients being aware. The master is also responsible for detecting range server failures and re-assigning ranges if necessary. The master is also responsible for range server load balancing. Current design provides a single Master process, but high availability is achieved through hot standbys.

Range Server - Range servers are responsible for managing ranges of table data, handling all reading and writing of data. They can manage up to potentially thousands of ranges and are agnostic to the set of ranges that they manage or the tables of which they're a part. Ranges can move freely from one range server to another, an operation that is mostly orchestrated by the Master.

DFS Broker - Hypertable is capable of running on top of any filesystem. To achieve this, Hypertable has abstracted the interface to the filesystem by sending all filesystem requests through a Distributed File System (DFS) broker process. The DFS broker provides a normalized filesystem interface and translates normalized filesystem requests into native filesystem requests and vice-versa. Currently, DFS brokers are available for HDFS, MapR, Ceph, KFS, and local (for single-machine installations running on top of a local filesystem).

Hyperspace - This is Hypertable's equivalent to Google's Chubby service. Hyperspace is a highly available lock manager and provides a filesystem for storing small amounts of metadata. Exclusive or shared locks may be obtained on any created file or directory. High availability is achieved by running in a distributed configuration with replicas running on different physical machines. Consistency is achieved through a distributed consensus protocol. Google refers to Chubby as, "the root of all distributed data structures" which is a good way to think of Hyperspace.

COMPARISON TO A RELATIONAL DATABASE

Hypertable is similar to a relational database in that it represents data as tables of information, with rows and columns, but that's about as far as the analogy goes. The following list provides some important differences. In Hypertable,

- Row keys are UTF-8 strings
- There is no support for data types (data values are treated as opaque byte sequences)
- There is no support for joins
- There is no support for transactions
- New cell data values result in a new row entry (older entries may remain in the table)

Tables in Hypertable can be thought of as massive lists of data records, sorted by a single primary key, the row key.

TABLE LAYOUT CONCEPT

A relational database assumes that each column defined in the table schema can have a value in each row that is present in the table, so every row contains data space for every specified column. NULL values are often represented with a special marker (e.g. \N). The primary key and column identifier are implicitly associated with each cell based on its physical position within the layout. The following diagram illustrates how a relational database table might be stored:

Item	Date	Qty	Supplier
Apples	2011-20-29	60	Figoni
Asparagus	2011-10-30	34	Giusti Farms
Bananas	\N	\N	\N
Cantelope	\N	\N	\N
Grapes	\N	\N	\N
Onions	2011-10-27	66	Pastorino
Oranges	\N	\N	\N
Peaches	\N	\N	\N
Pears	\N	\N	\N
Pineapples	\N	\N	\N
Plums	\N	\N	\N
Strawberries	\N	\N	\N
Yams	2011-11-03	52	Iacopi Farms

Hypertable (and Bigtable) takes its design from the Log Structured Merge Tree. It flattens out the table structure into a sorted list of key/value pairs, each one representing a cell in the table. The key includes the full row and column identifier, which means each cell is provided complete addressing information. Cells that are NULL are simply not included in the list which makes this design particularly well-suited for sparse data. The following diagram illustrates how Hypertable would store the above table data on-disk:

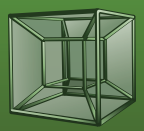
key		value
Apples	Date	2011-20-29
Apples	Qty	60
Apples	Supplier	Figoni
Asparagus	Date	2011-10-30
Asparagus	Qty	34
Asparagus	Supplier	Giusti Farms
Onions	Date	2011-10-27
Onions	Qty	66
Onions	Supplier	Pastorino
Yams	Date	2011-11-03
Yams	Qty	52
Yams	Supplier	Iacopi Farms

Though there can be a fair amount of redundancy in the row keys and column identifiers, Hypertable employs key-prefix and block data compression which helps to reduce the effects.

HYPERTABLE DATA REPRESENTATION

As described above, Hypertable stores data as tables of information. Each row in a table has cells containing related information, and each cell is identified, in part, by a row key and column name. Support for up to 255 column names is provided when the table is created. Hypertable provides two additional features:

- **column qualifier** - The column names defined in the table schema actually represent column families. Applications may supply an optional column qualifier formatted as family:qualifier, with each distinct qualifier representing a column instance belonging to the column family. An unlimited number of uniquely named instances of a column family can be defined by the application. Column data is stored in a sparse format such that wwhile one row may have millions of qualified column instances within a column family, another row might have none or just a few.
- **timestamp** - This is a 64-bit field associated with each cell that allows for different cell versions. The value represents nanoseconds since the **Unix** epoch and can be supplied by the application or auto-assigned by Hypertable. The number of versions stored is configured in the table schema and the number of versions returned can be specified in the query predicate. Versions are stored in reverse-chronological order, so that the newest version of the cell is returned first when queried.



To illustrate how data is represented in Hypertable, consider an example taken from a web crawler that stores information for each page that it crawls. The first table represents how the data might look stored in the style of a relational database table.

crawldb Table

row key	title	content	anchor	
			anchor:com.apple.www/	anchor:com.redherring.www/
com.facebook.www	Facebook | Home	<!DOCTYPE html ...	Facebook	Facebook
com.yahoo.www	Yahoo!	<html><head>...		
			anchor:org.slashdot.www/	
com.zvents.www	Discover Things To Do - Zvents	<html xmlns="http...	Zvents	
org.hypertable.www	Hypertable: An Open Source, High Performance	<!DOCTYPE html ...		

Note that the diagram also illustrates the use of the column qualifier.

When the above data is stored by Hypertable, a timestamp is added and it is stored as sorted lists of key/value pairs as illustrated in the diagram below. (The diagram shows up to three timestamped versions of each cell, the older versions having been stored from previous crawls. When queried, the most recent cell version is returned first.)

crawldb Table

key	value
com.facebook.www title *2008-02-11 15:14:01*	Facebook | Home
com.facebook.www title *2008-02-03 19:27:57*	Facebook | Home
com.facebook.www title *2008-01-22 08:46:28*	Facebook | 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-...
...	...

NAMESPACES

Namespaces logically group tables together and are analogous to the directory hierarchy in a modern filesystem. They allow the user to organize tables into related groups, keeping table names simple, as table names need only be unique within the namespace in which they are created.

ACCESS GROUPS

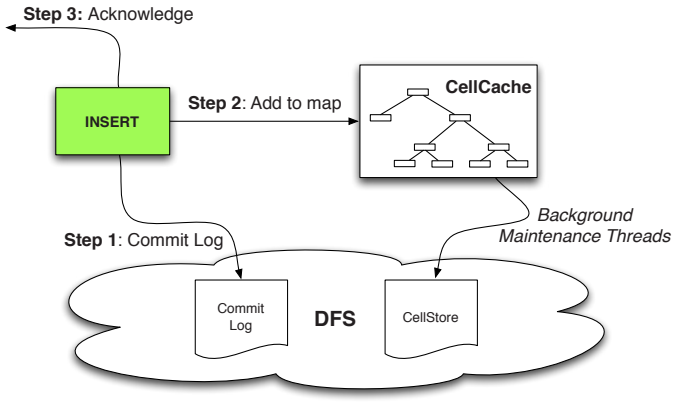
Access Groups provide a way for the user to control the physical storage of column data to optimize disk I/O. Access Groups are defined in the table schema and instruct Hypertable to physically store all data for columns within the same access group together on disk. This feature allows the user to optimize queries for columns that are accessed with high frequency by reducing the amount of data transferred from disk during query execution. Disk I/O then is limited to just the data from the access groups of the columns specified in the query

CLIENT LIBRARY

The Client Library provides the application programming interface (API) that allows an application to communicate with Hypertable. This library is linked into each Hypertable application and handles query routing.

RANGE SERVER INSERT HANDLING

As noted earlier, Range servers are responsible for managing ranges of table data (contiguous table rows sorted by key), handling all reading and writing of data. The following diagram illustrates how inserts are handled inside the RangeServer.



- **Step 1:** Commit Log - Inserts are appended to the Commit log which resides in the distributed filesystem (DFS) and followed by a sync operations that tells the filesystem to persist any buffered writes to disk. If multiple insert requests are pending, or a GROUP_COMMIT_INTERVAL is configured for the table, then the sync operation is performed after multiple Commit log appends to improve throughput.

- **Step 2:** Add to map - The inserts are added to the in-memory CellCache (equivalent to the Memtable in Bigtable).

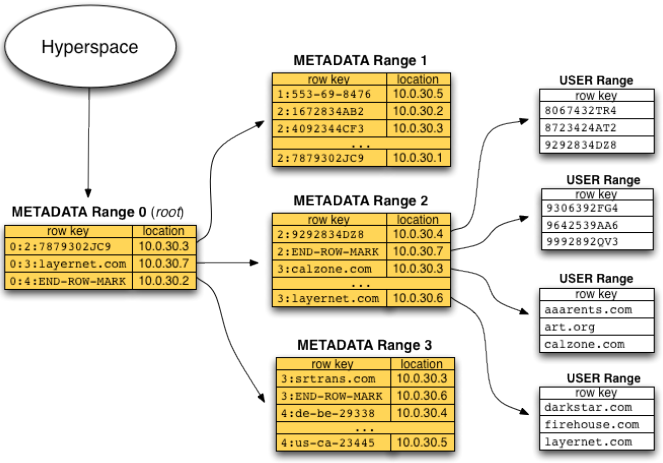
- **Step 3:** Acknowledge - Acknowledgement is sent back to the application.

- **Background Maintenance Threads** - Over time, as the CellCaches fill memory, background maintenance threads will "spill" the in-memory CellCache data to on-disk CellStore files which frees up memory inside the RangeServer, allowing it to accept more inserts.

This design makes Hypertable writes durable and consistent because inserts are not acknowledged until insert has been successfully written to the Commit log.

QUERY ROUTING

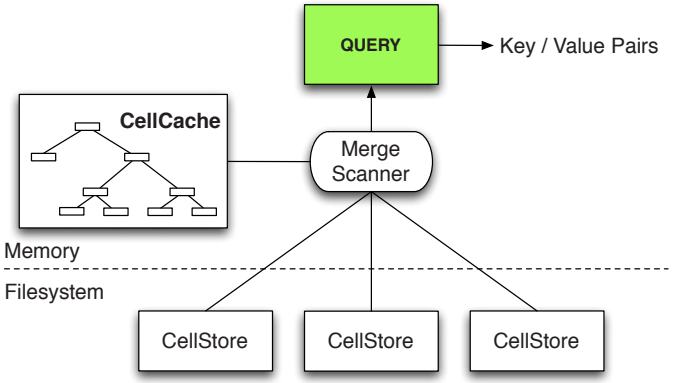
The following illustrates and describes the data structures that support the query routing algorithm that sends queries to the relevant RangeServers.



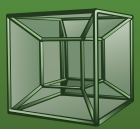
The query routing algorithm makes use of a special table called sys/METADATA that contains a row for each range in the system. There is a column Location that indicates which RangeServer is currently serving the range. Though the diagram shows IP addresses in the Location column, the system stores a proxy name for the RangeServer in that column so that the system can be run on public clouds such as Amazon's EC2 and operate correctly in the face of server restarts and IP address changes. A two-level hierarchy is overlaid on top of the METADATA table. The first range is the ROOT range which contains pointers to the second-level ranges which, in turn, contain pointers to the USER ranges, which are the ranges that make up regular user or application defined tables.

RANGESERVER QUERY HANDLING

The following diagram illustrates how queries are handled inside the RangeServer.

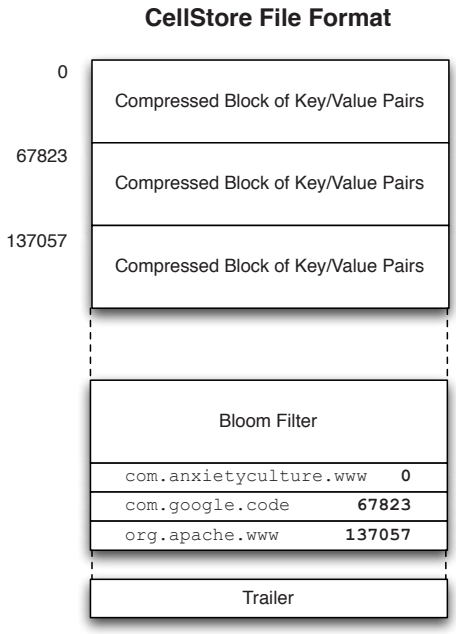


Data for a range can reside in the in-memory CellCache as well as in some number of on-disk CellStores (see following section). To evaluate a query over a table range, the RangeServer must create a unified view of the data, which it does through the use of a MergeScanner object, which merges together the sorted key/value pairs coming from the CellCache and from the CellStores. This unified stream of key/value pairs is then filtered to produce the query results.



FILESYSTEM CELLSTORE FORMAT

Over time, the RangeServers will write in-memory CellCaches to on-disk files, called CellStores, whose format is as follows:



- **Compressed blocks of cells (key/value pairs)** - This section consists of a series of sorted blocks of compressed sorted key/value pairs. By default, the compressed blocks are approximately 64KB in size, but the size is configurable. These blocks are the minimum unit of data transfer from disk.

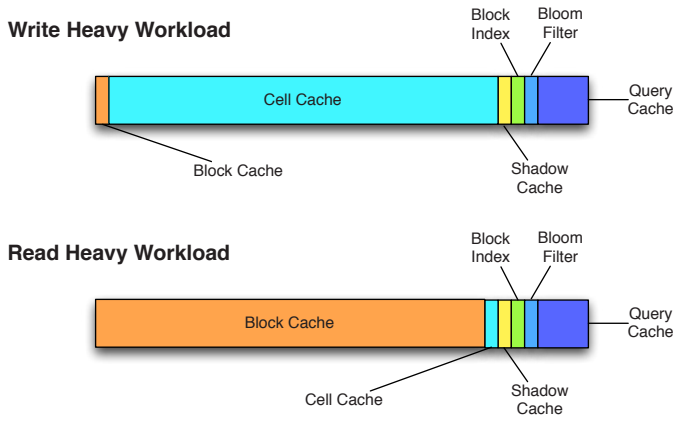
- **Bloom Filter** - The compressed blocks of key/value pairs are followed by a bloom filter. This is a probabilistic data structure that describes the keys that exist (with high likelihood) in the CellStore and which also signals if a key is definitively not present. This helps the RangeServer avoid unnecessary block transfer and decompression.

- **Block Index** - After the bloom filter comes the block index. This index lists, for each block, the last key in the block followed by the block offset.

- **Trailer** - At the end of the CellStore is the trailer. The trailer contains general statistics about the CellStore

ADAPTIVE MEMORY ALLOCATION

The following diagram illustrates how the RangeServer adapts its memory usage based on changes in workload.

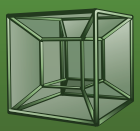


Under write-heavy workload, the RangeServer will give more memory to the CellCaches so that they can grow as large as possible, which minimizes the amount of spilling and merging work required. Under read-heavy workload, the system gives most of the memory to block cache, which significantly improves query throughput and latency.

SCALING AND LOAD BALANCING

Hypertable typically runs on a cluster of slave server machines, with a RangeServer process running on each slave. Initially, a table is assigned to be managed by a single RangeServer. As entries are added over time, Hypertable will break these tables into ranges of rows and distribute the ranges to other RangeServer processes.

Increasing cluster capacity can be accomplished by simply adding one or more new commodity servers and starting RangeServer processes on the new machines. Hypertable will detect that there are new servers available with plenty of spare capacity and will automatically reconfigure and migrate ranges from the relatively overloaded machines onto the new ones. This range migration process has the effect of balancing load across the entire cluster and utilizing the added capacity.



CONCLUSION

Hypertable is a scalable NoSQL database modeled after Bigtable, Google's proprietary scalable database. It is open source and freely available to anyone who wants to use it. Implemented in C++ for optimum performance, Hypertable can deliver scalable database capacity on a fraction of the hardware as compared to competing implementations. This means Hypertable can be run with less equipment, less power consumption, and less datacenter real estate, which translates to lower cost. Hypertable has been successfully deployed in a wide array of sectors including, Automotive Engineering, Biotechnology, Financial, and Internet. For more information, see **www.hypertable.com**.

HYPERTABLE INC

PHONE +1 650.401.6038
FAX +1 650.230.7176
info@hypertable.com

702 Marshall St.
Suite 615
Redwood City, CA 94063