MapReduce and Parallel DBMSs: Friends or Foes?

Teng Long

Class Presentation for CMSC818K

March. 8, 2011

Department of Computer Science

University of Maryland

Questions to Answer

- Are MapReduce systems complement or substitution of Parallel DBMSs?
  - What are their similarities and differences?
  - What kind of applications are they good at?
  - What can they learn from each other?

Features

- Essential Features of Parallel Database Systems:
  - Horizontal partitioning of relational table;
  - Partitioned execution of SQL operations.
- Features of MapReduce Model
  - Simplicity: Only Map() and Reduce() to be written by users;
  - Input data stored in a collection of partitions in a distributed file system;
  - Programs are injected into a distributed-processing framework.

Similarities

- DBMS can do whatever MR can
  - User-defined functions provides the equivalent functionality of a Map operation;
  - SQL aggregates augmented with UDF: the same MR-style ; reduce functionality;
  - GROUP BY operation in SQL equals to RESHUFFLE in MapReduce.

Differences

- Possible application classes that MR beats DBMS:
  - extract-transform-load(ETL) task and “read once” data set;
  - Complex analytics;
  - Semi-structured data, usually like key-value pairs;
  - Quick-and-dirty analyses;
  - Limited-budget operations.

Differences(cont.)

- MR beaten by DBMSs because:
  - Repetitive record parsing;
  - Less Compression;
  - Less Pipeline;
  - Weak Scheduling;
  - No Column-oriented storage.

Lessons

- DBMS to MR:
  - Efficient query parallel execution(Unfair!!!)
  - Higher, more productive-level abstraction, rather than MR algorithm code(What about flexibility?)
- MR to DBMS:
  - Quick setup, automatic tuning, better website with example code, better query generations, better documentation;
  - Try to deal with data on File system, skip the LOAD phase.

Conclusion

- MapReduce is complementary to DBMSs, NOT a competing technology.
  - Parallel DBMSs are for efficient querying of large data sets;
  - MR-style systems are for complex analytics and ETL tasks.

Bigtable: A Distributed Storage System for Structured Data

Teng Long

Class Presentation for CMSC818K

March. 8, 2011

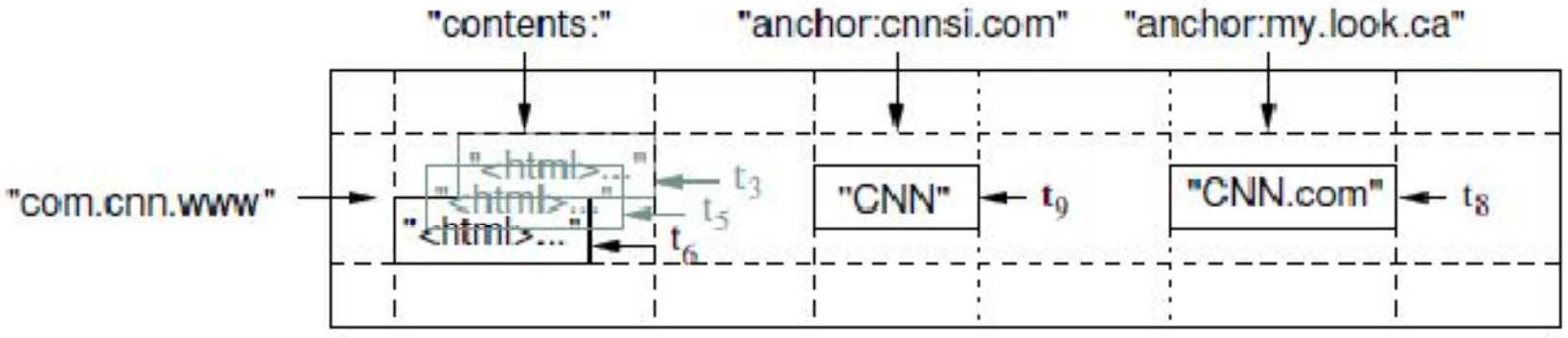
Department of Computer Science

University of Maryland

Some slides are from Shahram Ghandeharizadeh's talk on Bigtable in USC

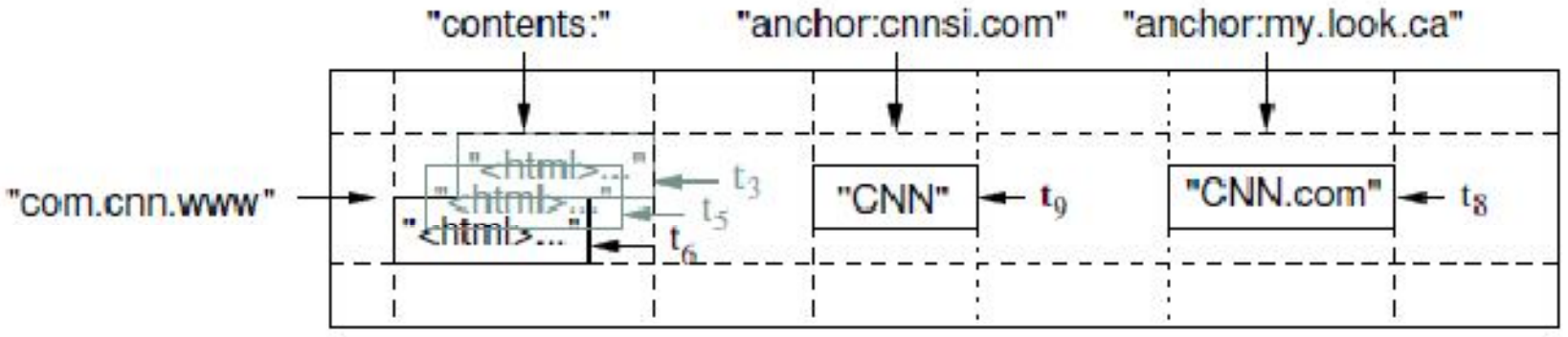
Introduction

- What is Bigtable?
  - A data model (a schema).
  - A sparse, distributed persistent multi-dimensional sorted map.
  - Data is partitioned across the nodes seamlessly.
  - The map is indexed by a row key, column key, and a timestamp.
  - Output value in the map is an un-interpreted array of bytes.
    - (row: byte[ ], column: byte[ ], time: int64) → byte[ ]



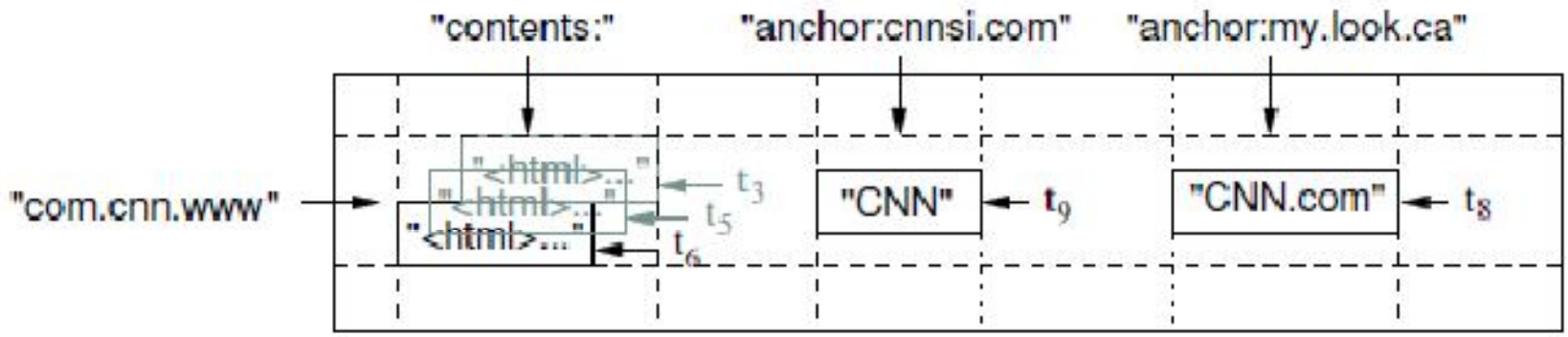
Data Model – Rows

- Every read or write of data under a single row is atomic.
- Data is maintained in lexicographic order by row key.
- The row range for a table is dynamically partitioned.
  - Objective: make read operations single-sited!
- Each partition (row range) is named a *tablet*.
  - Unit of distribution and load-balancing.



Data Model – Column Families

- Column keys are grouped into sets called column families;
  - It is the basic unit of access control;
- All data in the same family are often of the same type;
- A column family must be created before data can be stored in a column key(like schema);
- Hundreds of static column families;
- Use family: qualifier to refer to a column key.



Data Model – Timestamps

- 64 bit integers
- Generated by:
  - Bigtable: real-time in microseconds,
  - Client application: when unique timestamps are a necessity.
- Items in a cell are stored in decreasing timestamp order.
- Garbage-collect setting on column-family levels: keep either last n versions, or new-enough versions.

Bigtable APIs

- Implements interfaces to
  - Database-like APIs
    - create and delete tables and column families,
    - Write or delete values in Bigtable,
    - Look up values from individual rows,
  - Bigtable system APIs
    - modify cluster, table, and column family metadata such as access control rights,

Bigtable APIs(Cont.)

- Complex Data manipulation
  - Single-row atomic read-modify-write transaction;
  - Allows cells to be used as integer counters;
  - Supports the execution of client-supplied scripts in the address space of the server.
    - Filtering based on arbitrary expressions, and summarization via a variety of operators.
- Can be used as I/0 in the MapReduce framework.

Building Blocks

- Uses GFS to store logs and data files.
- Bigtable processes share the same machines with processes from other applications.
  - A shared cluster of commodity PCs.
- A cluster management system:
  - Schedules jobs,
  - Manages resources on shared machines,
  - Monitors PC status and handles failures.

SSTable

- A database similar to a BDB database:
  - Stores and retrieves key/data pairs.
    - Key and data are arbitrary byte arrays.
  - Cursors to iterate key/value pairs given a selection predicate (exact and range).
  - Configurable to use either persistent store (disk) or main-memory based.
- An SSTable is stored in GFS.

Chubby

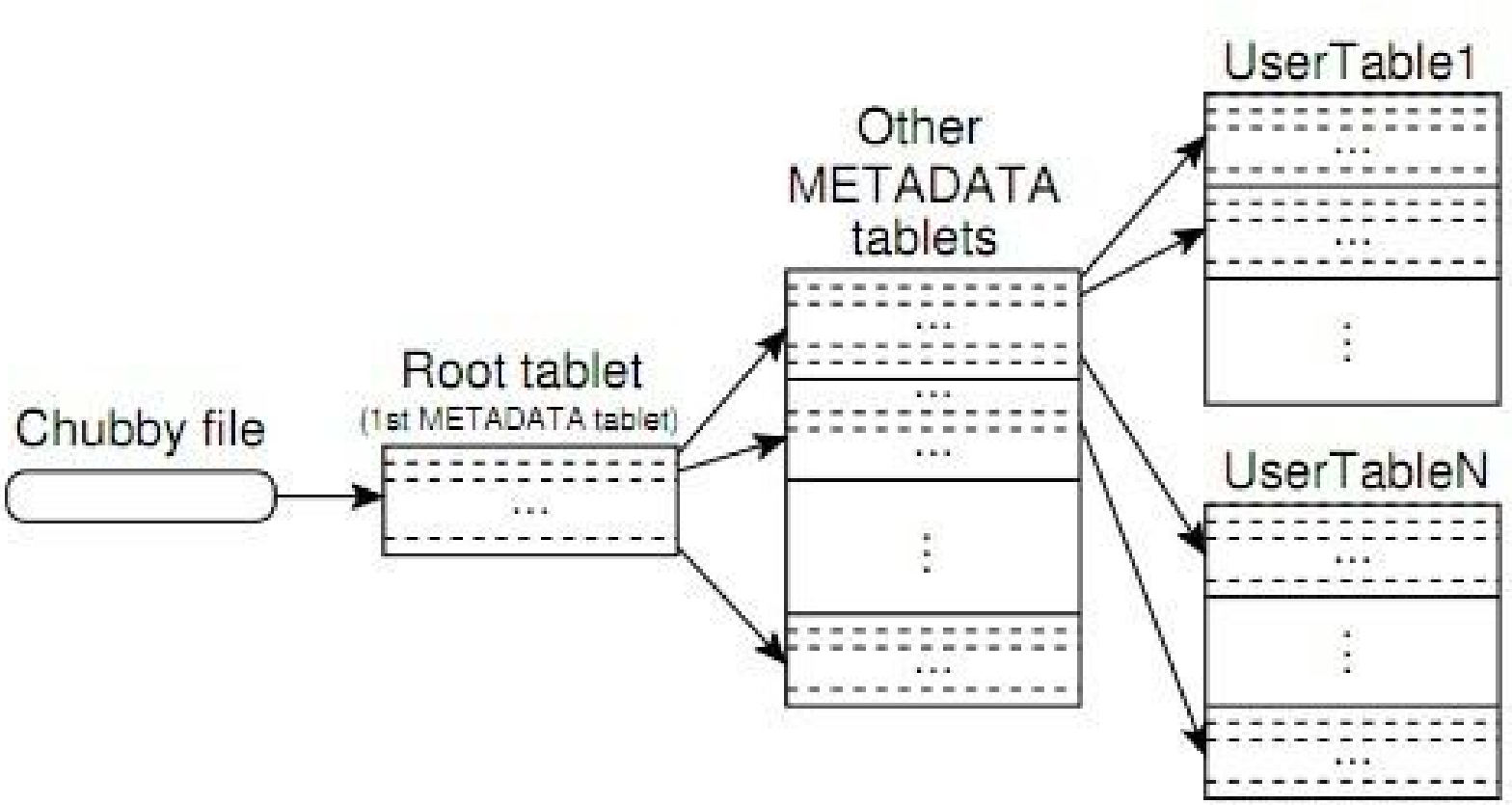
- A persistent and distributed lock service.
- Consists of 5 active replicas, one replica is the master and serves requests.
  - Service is functional when majority of the replicas are running and in communication with one another
- Implements a nameservice that consists of directories and files.

Implementation – Main Components

- A library that is linked into every client;
- A master server;
  - Assign tablets to tablet servers;
  - Detect addition and expiration of tablet servers;
  - Balance tablet-server load;
  - Garbage collection of files in GFS
  - Handle schema changes
- Many tablet servers.
  - Handle read/write requests;
  - Split tablets;
  - Data does not move through the master.

Location of Tablets

- A 3-level hierarchy:



Location of Tablets(Cont.)

- A 3-level hierarchy:
  - First level: A file stored in chubby contains location of the root tablet, i.e., a directory of ranges (tablets) and associated meta-data.
  - Second level: Each meta-data tablet contains the locat ion of a set of user tablets.
  - Third level: A set of SSTable identifiers for each tablet.

Tablet Management:

- A tablet is assigned to one tablet server at a time.
- Master Tasks:
  - Maintain the set of live tablet servers,
  - Maintain current assignment of tablets to tablet servers (including the unassigned ones)
  - Master monitors *server directory* to discover tablet servers,
- Chubby maintains tablet servers:
  - A tablet server creates and acquires an eXclusive lock on a uniquely named file in a specific chubby directory (named *server directory*),
  - A tablet server stops processing requests if it loses its X
    - Tablet server will try to obtain an X lock on its uniqely named file as long as it exists.
    - If the uniquely named file of a tablet server no longer exists then the tablet server kills itself. Goes back to a free pool to be assigned tablets by the master.

Tablet Management(Cont.)

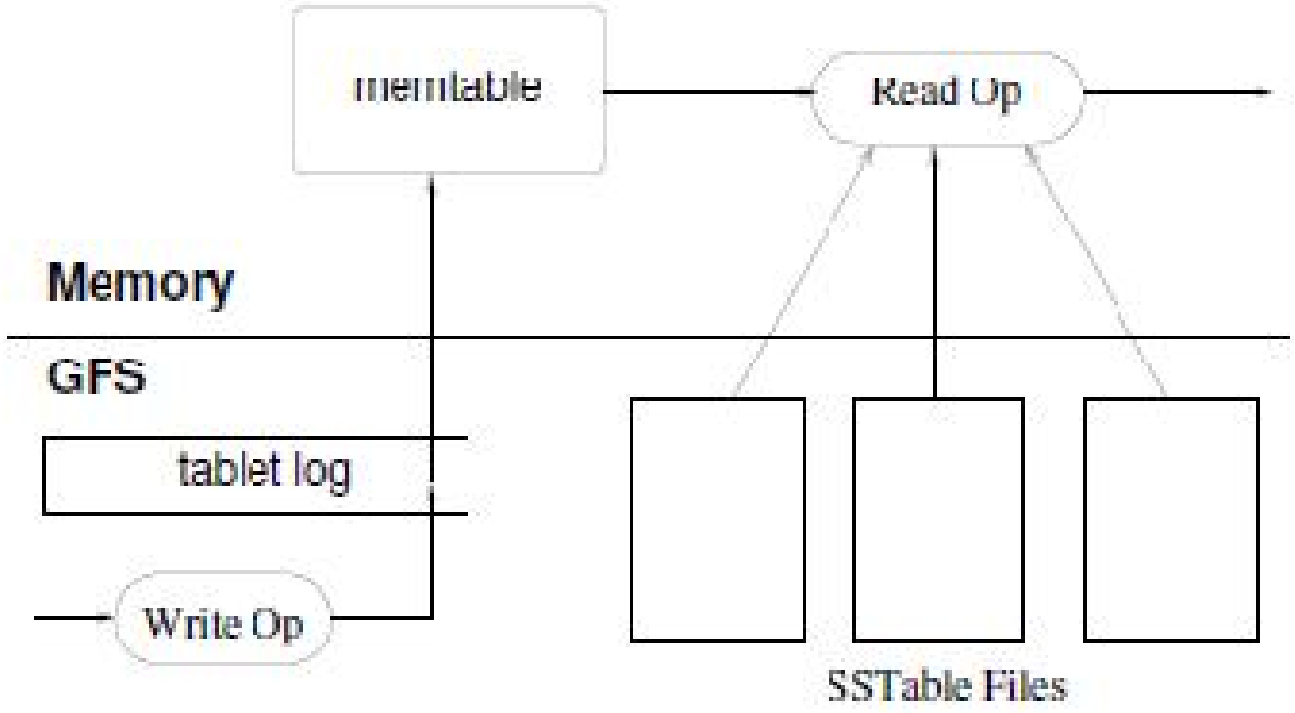
- The master detects when a tablet server stops, and reassigns its tablets ASAP.
  - The master periodically asks tablet servers for their status of locks;
  - If someone lost the lock, the master tries to reach to Chubby before deleting that tablet server, in order to make sure it is the tablet server not working, not Chubby;
  - The master kills itself if its Chubby session expires

Bigtable and Chubby

- Bigtable uses Chubby to:
  - Ensure there is at most one active master at a time,
  - Store the bootstrap location of Bigtable data (Root tablet),
  - Discover tablet servers and finalize tablet server deaths,
  - Store Bigtable schema information (column family information),
  - Store access control list.
- If Chubby becomes unavailable for an extended period of time, Bigtable becomes unavailable.

Memtable

- Recently committed updates are stored in memtable (a sorted buffer in memory)
- Memtable can be converted into an SSTable when getting to big, and new memtable is created.



Client Read and Write Operation

- Write operation arrives at a tablet server:
  - Server ensures the client has sufficient privileges for the write operation (Chubby),
  - A log record is generated to the commit log file,
  - Once the write commits, its contents are inserted into the memtable.
- Read operation arrives at a tablet server:
  - Server ensures client has sufficient privileges for the read operation (Chubby),
  - Read is performed on a merged view of (a) the SSTables that constitute the tablet, and (b) the memtable.

Compactions

- As writes execute, size of memtable increases.
- Once memtable reaches a threshold:
  - Memtable is frozen,
  - A new memtable is created,
  - Frozen metable is converted to an SSTable and written to GFS.
- This *minor compaction* minimizes memory usage of tablet server, and reduces recovery time in the presence of crashes ;
- *Merging compaction* (in the background) reads a few SSTables and memtable to produce one SSTable.
  (Input SSTables and memtable are discareded.)
- *Major compaction* rewrites all SSTables into exactly one SSTable (containing no deletion entries).

Refinements

- Objective: Higher Performance
- Methods:
  - Locality groups: Try to read locally;
  - Compression on locality groups: Possible IO boost;
  - Caching for read performance
    - Higher-level cache for key-value pairs;
    - Lower-level cache for SSTables;
  - Bloom Filters for SSTables: to avoid disk accesses;
  - Single commit-log per tablet server to avoid inefficiency on GFS

Conclusion

- Bigtable has high performance and high availability, and scales easily;
- The Bigtable framework brings great flexibility to Google applications(including MapReduce practices).

Bigtable: A Distributed Storage System for Structured Data

Teng Long

Class Presentation for CMSC818K

March. 8, 2011

Department of Computer Science

University of Maryland