MAP REDUCE AND OTHER MATTERS REVISITED

GENOVEVA VARGAS SOLAR

FRENCH COUNCIL OF SCIENTIFIC RESEARCH, LIG-LAFMIA, FRANCE

Genoveva.Vargas@imag.fr

http://mapreducefest.wordpress.com/

MAPREDUCE: SIMPLIFIED DATA PROCESSING ON LARGE CLUSTERS

BY: JEFFREY DEAN AND SANJAY GHEMAWAT

PRESENTED BY: HIND ALHAKAMI

PROBLEM TO SOLVE

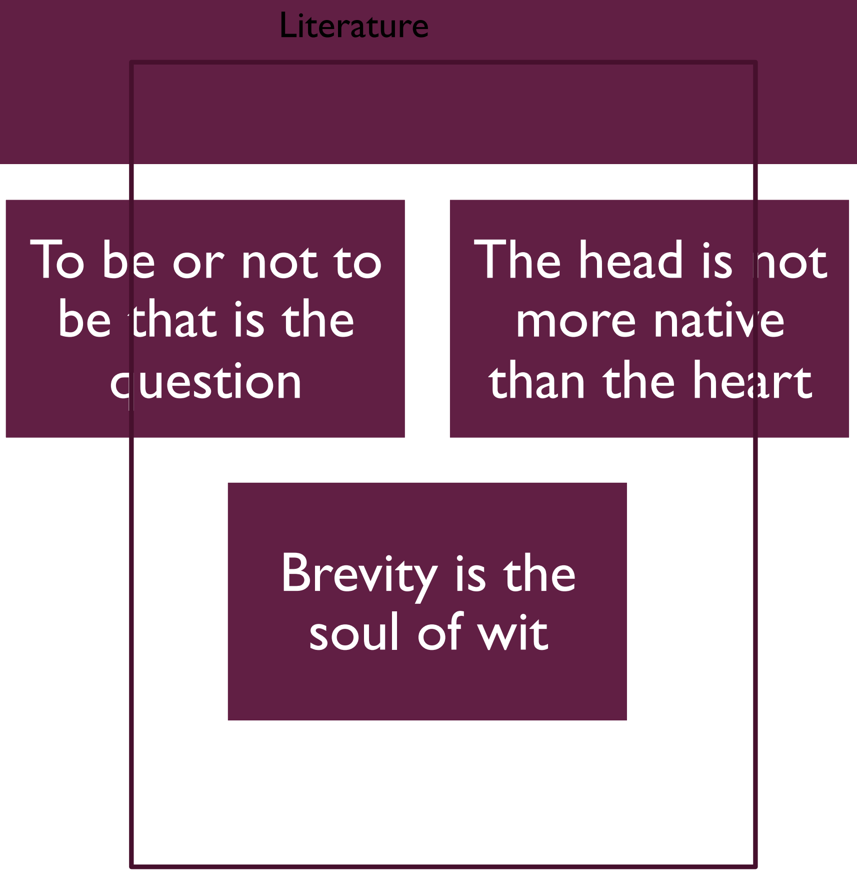
- Simple queries on huge amount of data.
- Distribution needed, which Complicate the problem.

SOLUTION

Map-Reduce System

EXAMPLE

- Count number of occurrences of each word in the given literature


Literature
To be or not to be that is the question
The head is not more native than the heart
Brevity is the soul of wit

SIMPLIFIED EXAMPLE

- For Simplicity, Consider counting words in one document.

To be or not to be that is the question

FIRST STEP - MAP

```
reduce(String key, Iterator values):
  // key: a word
  // values: a list of counts
  int result = 0;
  for each v in values:
    result += ParseInt(v);
  Emit(AsString(result));
```

{(" To",1), ("be",1), ("or", 1), ("not",1), ("to",1), ("be",1), ("that",1), ("is", 1), ("the",1), ("question",1)}

SECOND STEP - REDUCE

```
reduce(String key, Iterator values):
  // key: a word
  // values: a list of counts
  int result = 0;
  for each v in values:
    result += ParseInt(v);
  Emit(AsString(result));
```

{("to",2), ("be",2), ("or",1), ("not",1), ("that",1), ("is",1), ("the",1), ("question",1)}

BACK TO THE ORIGINAL EXAMPLE

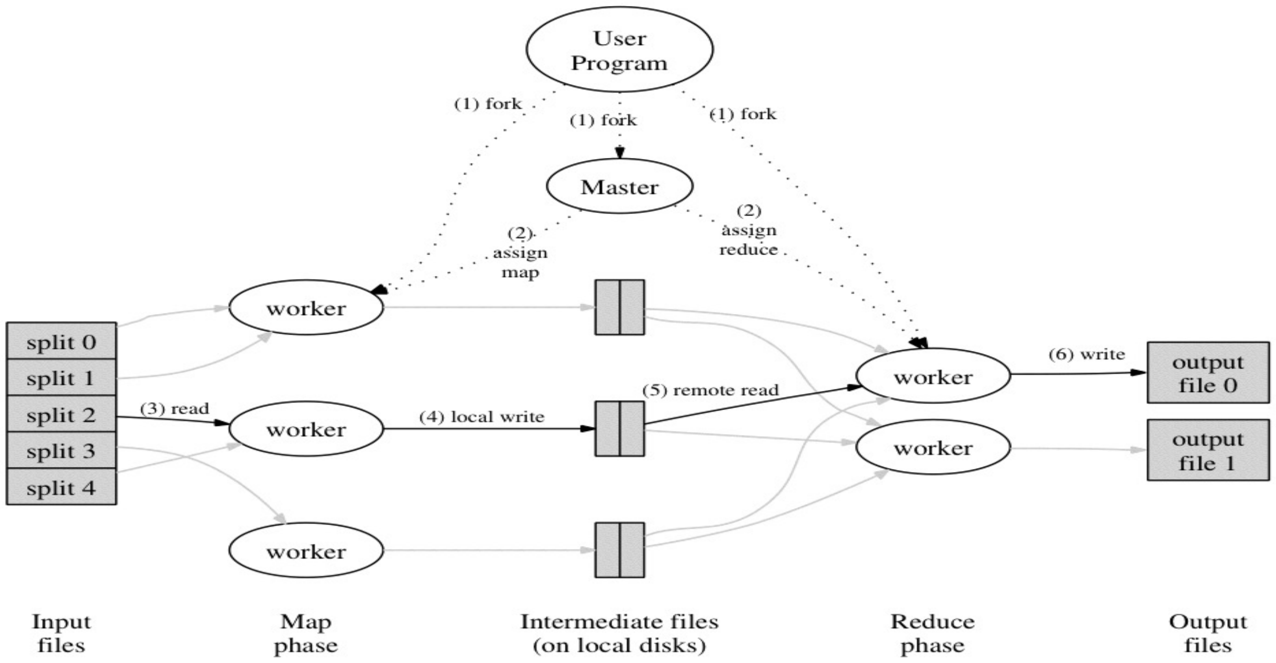
To be or not to be that is the question

The head is not more native than the heart

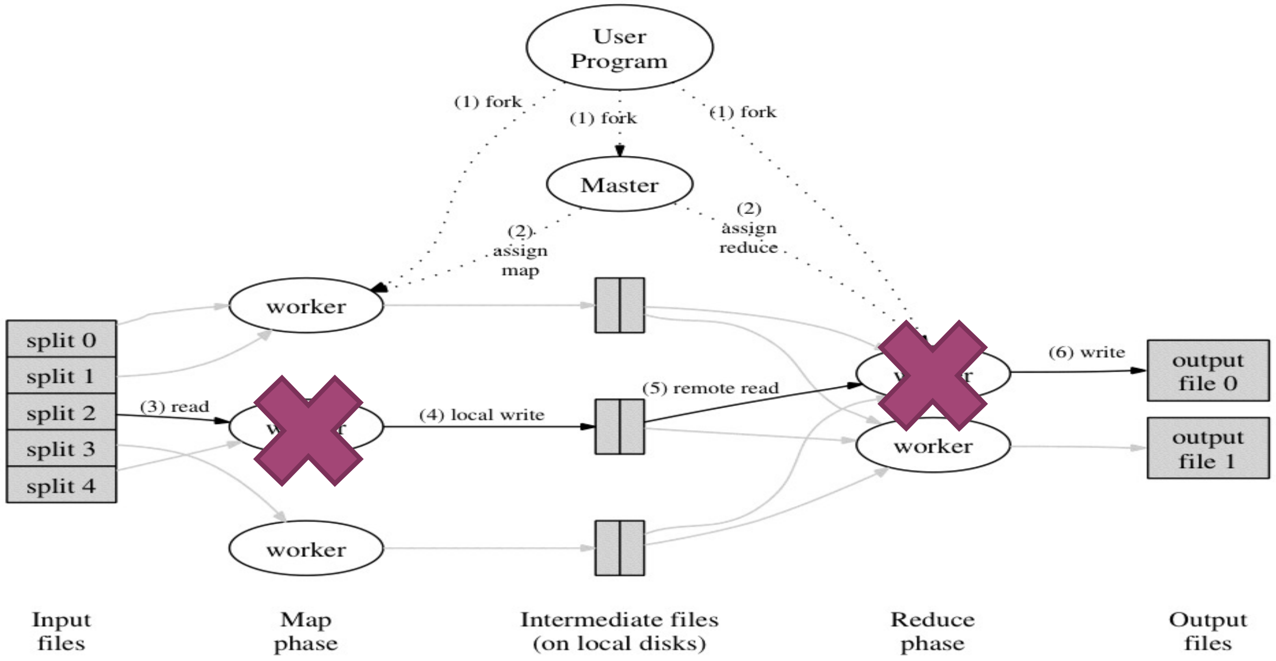
- {("the",3), …}

Brevity is the soul of wit

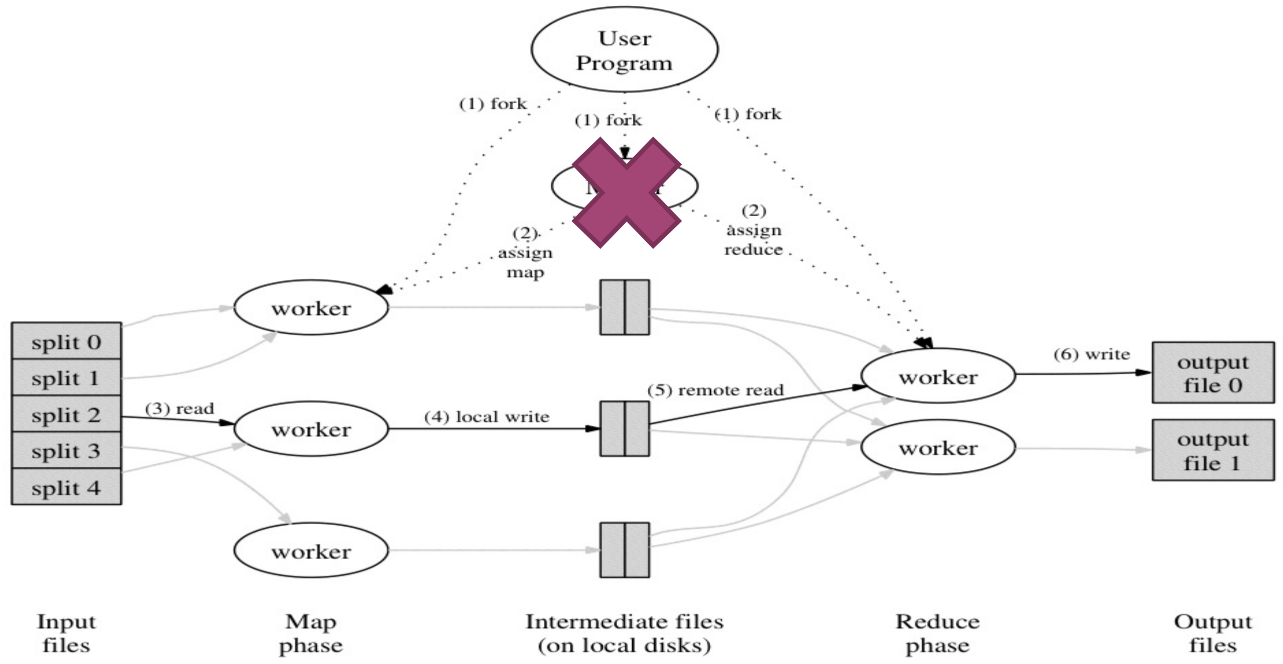
EXECUTION OVERVIEW



WORKER FAILURE



MASTER FAILURE

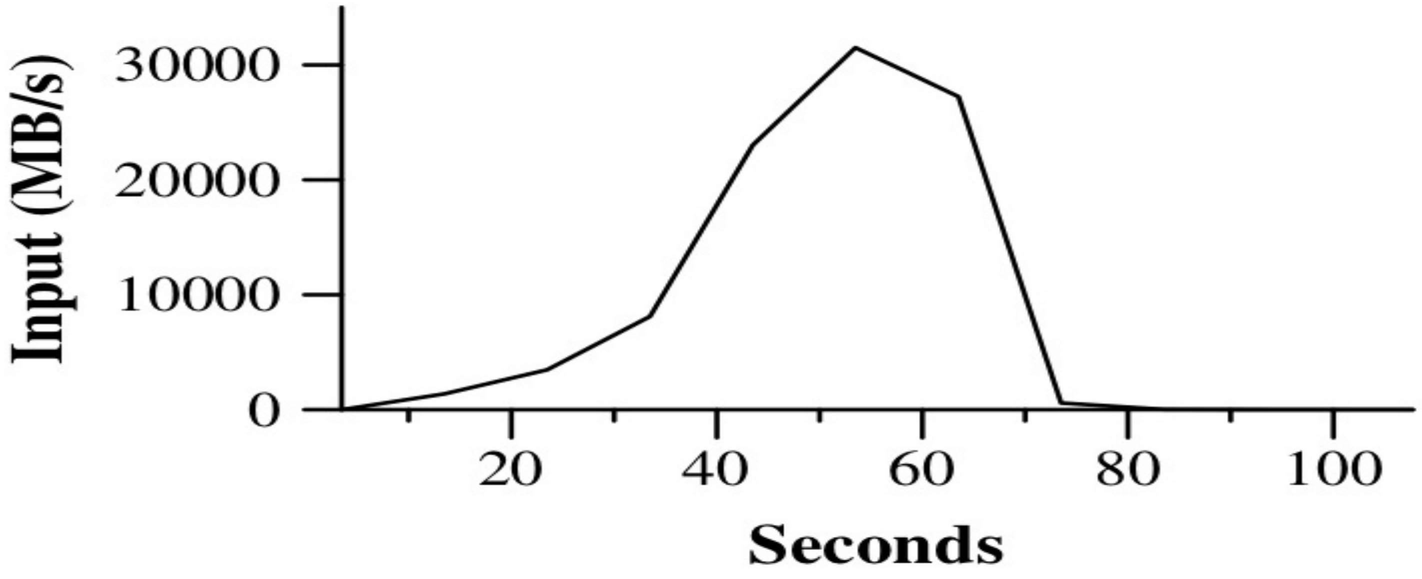


EXPERIMENT

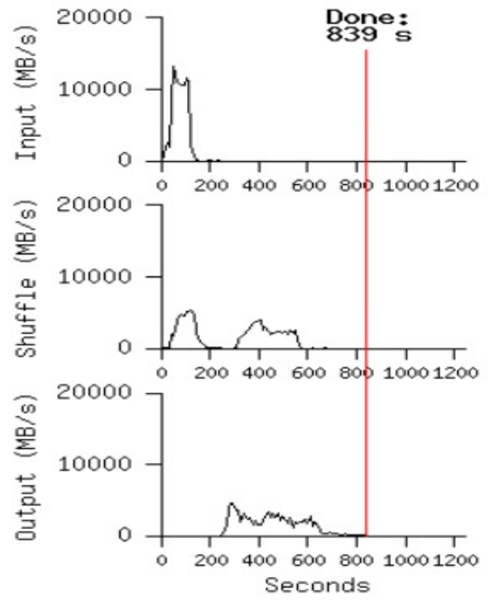
- Cluster of approximately 1800 machine.

- Two Benchmarks:
  - Grep
  - Sort

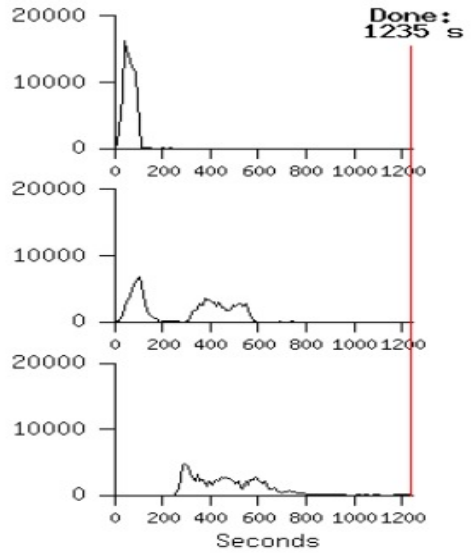
PERFORMANCE



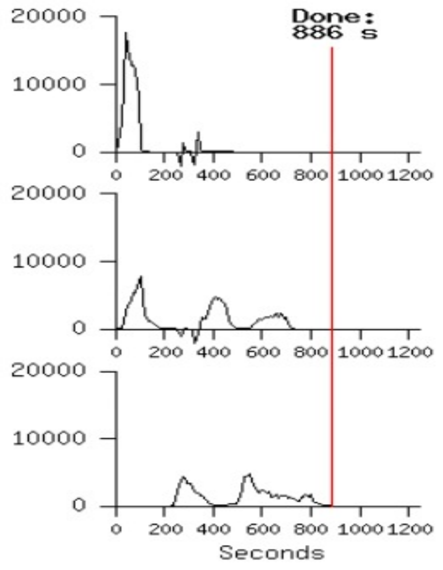
PERFORMANCE



Normal execution



No backup tasks



200 tasks killed

OPTIMIZATIONS

- Combiner function.
- Skipping bad records.
- Local execution.
- Status Information.

CONCLUSION

- MapReduce provides a convenience level of abstraction.
- Supports scalability and fault-tolerance.
- Preserves network bandwidth.

BIGTABLE: A DISTRIBUTED STORAGE SYSTEM FOR STRUCTURED DATA

FAY CHANG, JEFFREY DEAN, SANJAY GHEMAWAT, WILSON C. HSIEH, DEBORAH A. WALLACH, MIKE BURROWS, TUSHAR CHANDRA, ANDREW FIKES, ROBERT E. GRUBER @ GOOGLE

PRESENTED BY RICHARD VENUTOLO

INTRODUCTION

- BigTable is a distributed storage system for managing structured data.
- Designed to scale to a very large size
  - Petabytes of data across thousands of servers
- Used for many Google projects
  - Web indexing, Personalized Search, Google Earth, Google Analytics, Google Finance, …
- Flexible, high-performance solution for all of Google's products

MOTIVATION

- Lots of (semi-)structured data at Google
  - URLs:
    - Contents, crawl metadata, links, anchors, pagerank, …
  - Per-user data:
    - User preference settings, recent queries/search results, …
  - Geographic locations:
    - Physical entities (shops, restaurants, etc.), roads, satellite image data, user annotations, …
- Scale is large
  - Billions of URLs, many versions/page (~20K/version)
  - Hundreds of millions of users, thousands or q/sec
  - 100TB+ of satellite image data

WHY NOT JUST USE COMMERCIAL DB?

- Scale is too large for most commercial databases
- Even if it weren't, cost would be very high
  - Building internally means system can be applied across many projects for low incremental cost
- Low-level storage optimizations help performance significantly
  - Much harder to do when running on top of a database layer

GOALS

- Want asynchronous processes to be continuously updating different pieces of data
  - Want access to most current data at any time
- Need to support:
  - Very high read/write rates (millions of ops per second)
  - Efficient scans over all or interesting subsets of data
  - Efficient joins of large one-to-one and one-to-many datasets
- Often want to examine data changes over time
  - E.g. Contents of a web page over multiple crawls

BIGTABLE

- Distributed multi-level map
- Fault-tolerant, persistent
- Scalable
  - Thousands of servers
  - Terabytes of in-memory data
  - Petabyte of disk-based data
  - Millions of reads/writes per second, efficient scans
- Self-managing
  - Servers can be added/removed dynamically
  - Servers adjust to load imbalance

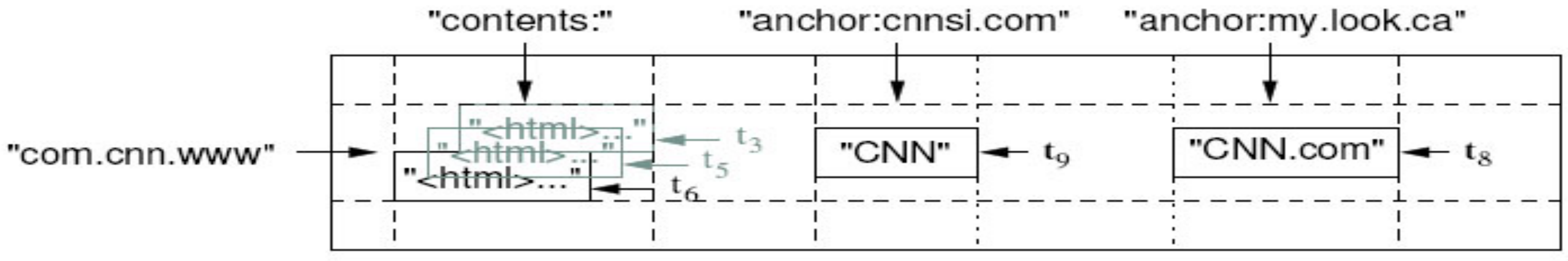
BUILDING BLOCKS

- Building blocks:
  - Google File System (GFS): Raw storage
  - Scheduler: schedules jobs onto machines
  - Lock service: distributed lock manager
  - MapReduce: simplified large-scale data processing
- BigTable uses of building blocks:
  - GFS: stores persistent data (SSTable file format for storage of data)
  - Scheduler: schedules jobs involved in BigTable serving
  - Lock service: master election, location bootstrapping
  - Map Reduce: often used to read/write BigTable data

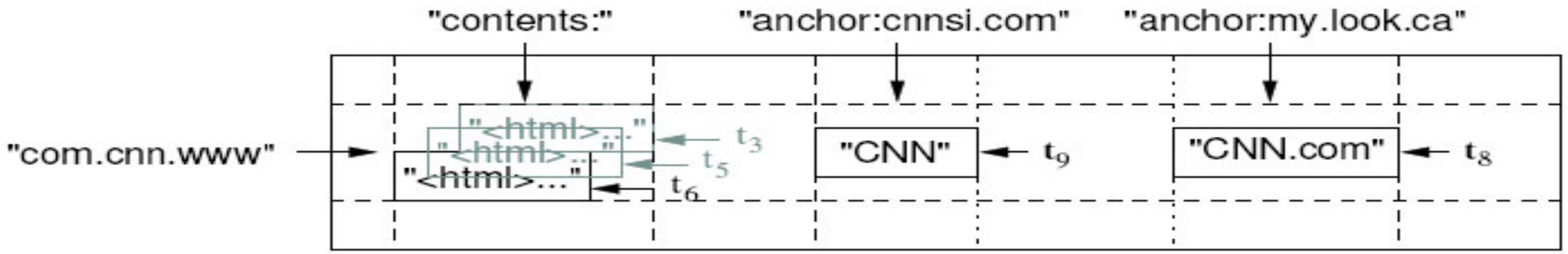
BASIC DATA MODEL

- A BigTable is a sparse, distributed persistent multi-dimensional sorted map

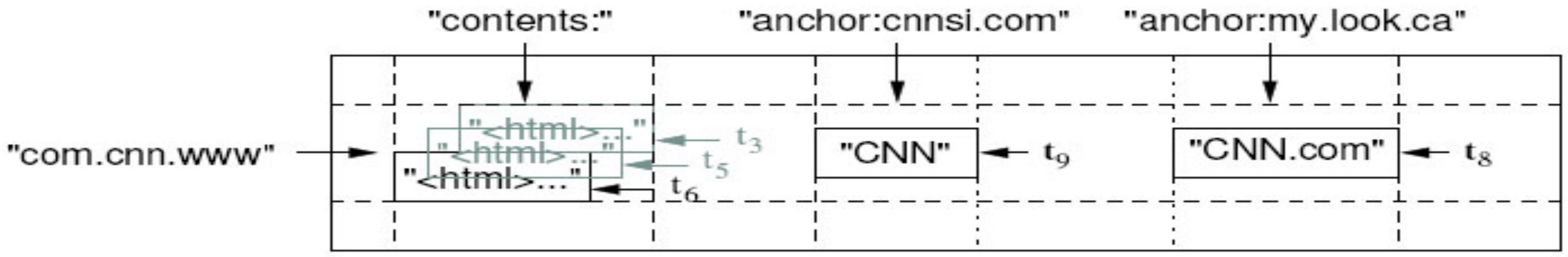
  (row, column, timestamp) -> cell contents



- Good match for most Google applications



- Want to keep copy of a large collection of web pages and related information
- Use URLs as row keys
- Various aspects of web page as column names
- Store contents of web pages in the `contents:` column under the timestamps when they were fetched.



- Name is an arbitrary string
  - Access to data in a row is atomic
  - Row creation is implicit upon storing data
- Rows ordered lexicographically
  - Rows close together lexicographically usually on one or a small number of machines

ROWS (CONT.)

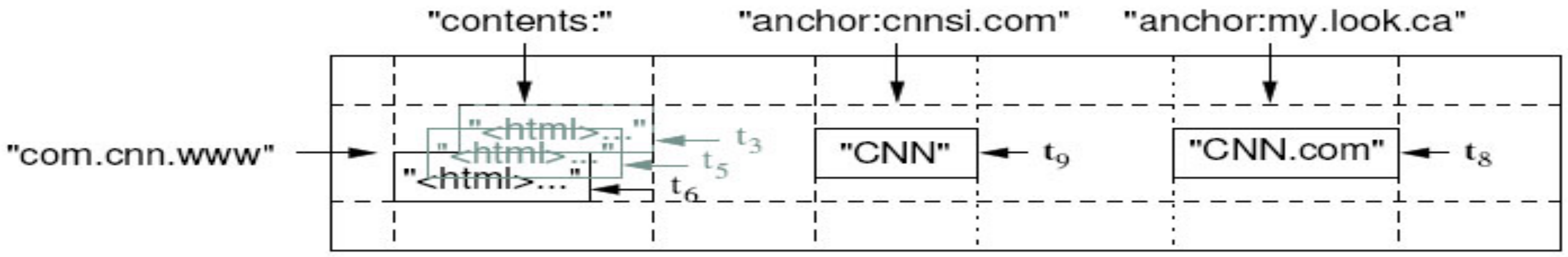
Reads of short row ranges are efficient and typically require communication with a small number of machines.

- Can exploit this property by selecting row keys so they get good locality for data access.
- Example:

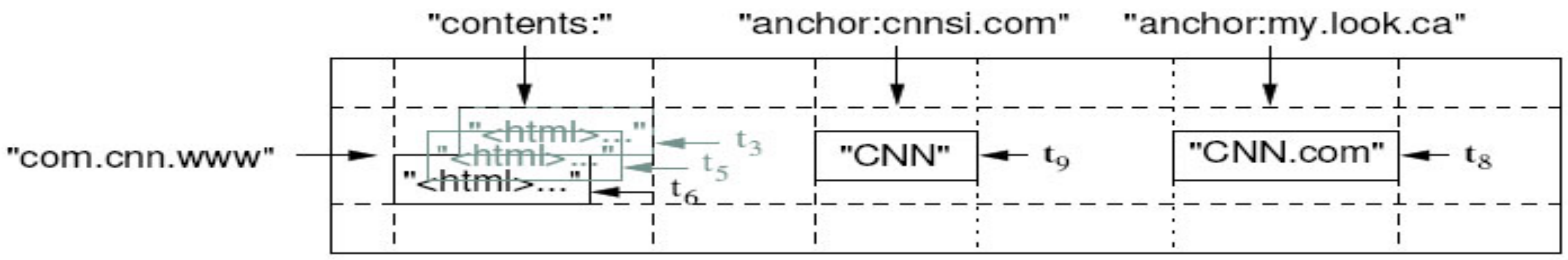
```
math.gatech.edu, math.uga.edu, phys.gatech.edu, phys.uga.edu

VS

edu.gatech.math, edu.gatech.phys, edu.uga.math, edu.uga.phys
```



- Columns have two-level name structure:
  - family:optional_qualifier
- Column family
  - Unit of access control
  - Has associated type information
- Qualifier gives unbounded columns
  - Additional levels of indexing, if desired



- Used to store different versions of data in a cell
  - New writes default to current time, but timestamps for writes can also be set explicitly by clients
- Lookup options:
  - *"Return most recent K values"*
  - *"Return all values in timestamp range (or all values)"*
- Column families can be marked w/ attributes:
  - *"Only retain most recent K values in a cell"*
  - *"Keep values until they are older than K seconds"*

IMPLEMENTATION – THREE MAJOR COMPONENTS

- Library linked into every client
- One master server
  - Responsible for:
    - Assigning tablets to tablet servers
    - Detecting addition and expiration of tablet servers
    - Balancing tablet-server load
    - Garbage collection
- Many tablet servers
  - Tablet servers handle read and write requests to its table
  - Splits tablets that have grown too large

IMPLEMENTATION (CONT.)

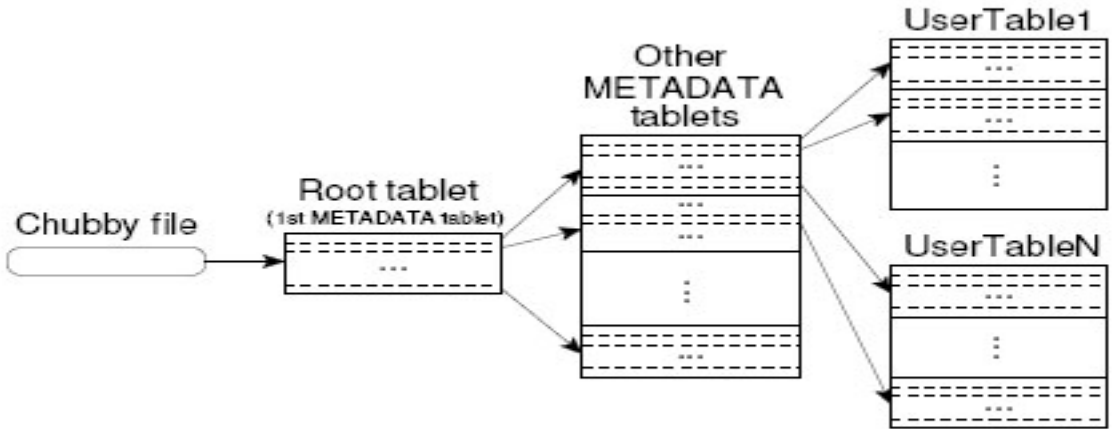
- Client data doesn't move through master server. Clients communicate directly with tablet servers for reads and writes.
- Most clients never communicate with the master server, leaving it lightly loaded in practice.

TABLETS

- Large tables broken into tablets at row boundaries
  - Tablet holds contiguous range of rows
    - Clients can often choose row keys to achieve locality
  - Aim for ~100MB to 200MB of data per tablet
- Serving machine responsible for ~100 tablets
  - Fast recovery:
    - 100 machines each pick up 1 tablet for failed machine
  - Fine-grained load balancing:
    - Migrate tablets away from overloaded machine
    - Master makes load-balancing decisions

TABLET LOCATION

- Since tablets move around from server to server, given a row, how do clients find the right machine?
  - Need to find tablet whose row range covers the target row



TABLET ASSIGNMENT

- Each tablet is assigned to one tablet server at a time.
- Master server keeps track of the set of live tablet servers and current assignments of tablets to servers. Also keeps track of unassigned tablets.
- When a tablet is unassigned, master assigns the tablet to an tablet server with sufficient room.

API

- Metadata operations
  - Create/delete tables, column families, change metadata
- Writes (atomic)
  - Set(): write cells in a row
  - DeleteCells(): delete cells in a row
  - DeleteRow(): delete all cells in a row
- Reads
  - Scanner: read arbitrary cells in a bigtable
    - Each row read is atomic
    - Can restrict returned rows to a particular range
    - Can ask for just data from 1 row, all rows, etc.
    - Can ask for all columns, just certain column families, or specific columns

REFINEMENTS: LOCALITY GROUPS

- Can group multiple column families into a *locality group*
  - Separate SSTable is created for each locality group in each tablet.
- Segregating columns families that are not typically accessed together enables more efficient reads.
  - In WebTable, page metadata can be in one group and contents of the page in another group.

REFINEMENTS: COMPRESSION

- Many opportunities for compression
  - Similar values in the same row/column at different timestamps
  - Similar values in different columns
  - Similar values across adjacent rows
- Two-pass custom compressions scheme
  - First pass: compress long common strings across a large window
  - Second pass: look for repetitions in small window
- Speed emphasized, but good space reduction (10-to-1)

REFINEMENTS: BLOOM FILTERS

- Read operation has to read from disk when desired SSTable isn't in memory
- Reduce number of accesses by specifying a Bloom filter.
  - Allows us ask if an SSTable might contain data for a specified row/column pair.
  - Small amount of memory for Bloom filters drastically reduces the number of disk seeks for read operations
  - Use implies that most lookups for non-existent rows or columns do not need to touch disk

THE FUTURE OF HIGH PERFORMANCE DATABASE SYSTEMS

AUTHORS: DAVID DEWITT AND JIM GRAY

PRESENTER: JENNIFER TOM

BASED ON PRESENTATION BY AJITH KARIMPANA

OUTLINE

- Why Parallel Databases?
- Parallel Database Implementation
- Parallel Database Systems
- Future Directions and Research Problems

WHY PARALLEL DATABASES?

- Relational Data Model – Relational queries are ideal candidates for parallelization
- Multiprocessor systems using inexpensive microprocessors provide more power and scalability than expensive mainframe counterparts

Why do the relational queries allow for parallelization?

PROPERTIES OF IDEAL PARALLELISM

- Linear Speedup – An N-times large

$$Speedup = \frac{small\_system\_elapsed\_time}{big\_system\_elapsed\_time}$$

- Linear Scaleup – An N-times larger system performs an N-times larger job in the same elapsed time as the original system

Why are linear speedup and linear

$$Scaleup = \frac{small\_system\_elapsed\_time\_on\_small\_problem}{big\_system\_elapsed\_time\_on\_big\_problem}$$

TYPES OF SCALEUP

- **Transactional** – N-times as many clients submit N-times as many requests against an N-times larger database.
- **Batch** – N-times larger problem requires an N-times larger computer.

Should we look at both transactional scaleup and batch scaleup for all situations when assessing performance?

BARRIERS TO LINEAR SPEEDUP AND LINEAR SCALEUP

- **Startup:** The time needed to start a parallel operation dominates the total execution time
- **Interference:** A slowdown occurs as new processes are added when accessing shared resources
- **Skew:** A slowdown occurs when the variance exceeds the mean in job service times

Give example situations where you might come across interference and skew barriers?

SHARED-NOTHING MACHINES

- **Shared-memory** – All processors have equal access to a global memory and all disks
- **Shared-disk** – Each processor has its own private memory, but has equal access to all disks
- **Shared-nothing** – Each processor has its own private memory and disk(s)

Why do shared-nothing systems work well for database systems?

PARALLEL DATAFLOW APPROACH

- Relational data model operators take relations as input and produce relations as output
- Allows operators to be composed into dataflow graphs (operations can be executed in parallel)

How would you compose a set of operations for an SQL query to be done in parallel?

PIPELINED VS. PARTITIONED PARALLELISM

- **Pipelined:** The computation of one operator is done at the same time as another as data is streamed into the system.
- **Partitioned:** The same set of operations is performed on tuples partitioned across the disks. The set of operations are done in parallel on partitions of the data.

Can you give brief examples of each? Which parallelism is generally better for databases?

DATA PARTITIONING

- **Round-robin** – For n processors, the ith tuple is assigned to the "i mod n" disk.
- **Hash partitioning** – Tuples are assigned to the disks using a hashing function applied to select fields of the tuple.
- **Range partitioning** – Clusters tuples with similar attribute values on the same disk.

What operations/situations are ideal (or not ideal) for the above partitioning types?

What issues to database performance may arise due to partitioning?

SKEW PROBLEMS

- **Data skew** – The number of tuples vary widely across the partitions.
- **Execution skew** – Due to data skew, more time is spent to execute an operation on partitions that contain larger numbers of tuples.

How could these skew problems be minimized?

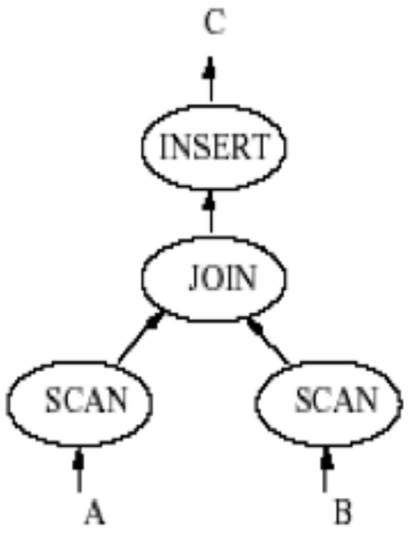
PARALLEL EXECUTION OF RELATIONAL OPERATORS

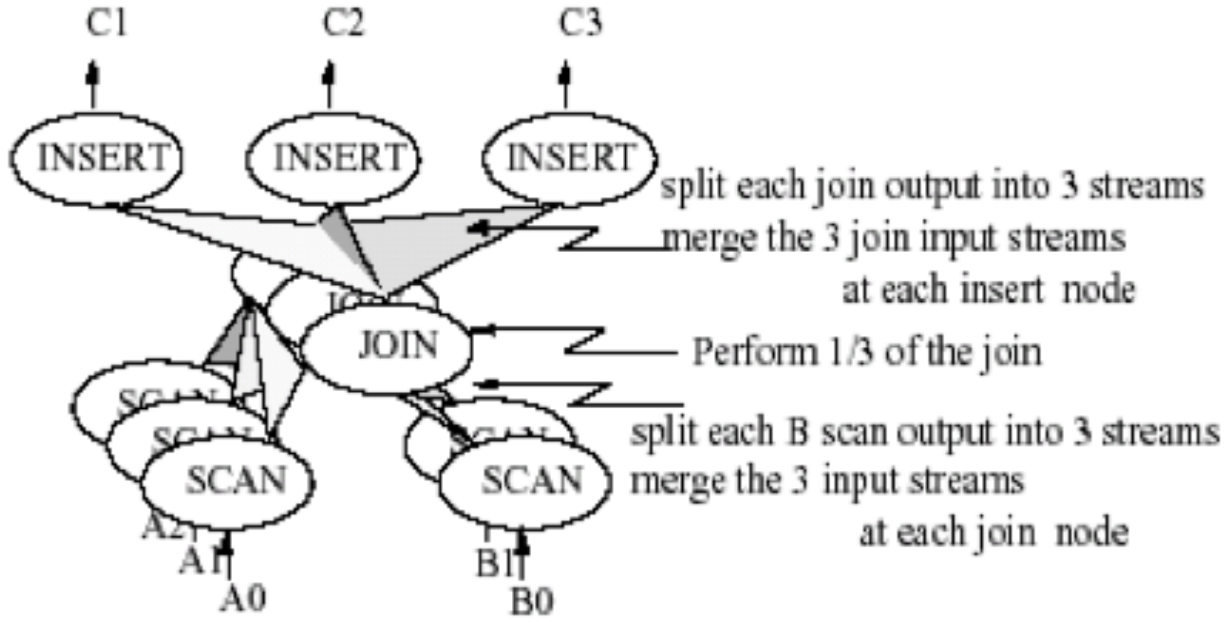
- Rather than write new parallel operators, use parallel streams of data to execute existing relational operators in parallel
- Relational operators have (1) a set of **input ports** and (2) an **output port**
- **Parallel dataflow** – partitioning and merging of data streams into the operator ports

Are there situations in which parallel dataflow does not work well?

SIMPLE RELATIONAL DATAFLOW GRAPH

```
insert into C
  select    *
  from      A, B
  where     A.x = B.y;
```




C1
C2
C3
INSERT
INSERT
INSERT
split each join output into 3 streams
merge the 3 join input streams
at each insert node
JOIN
Perform 1/3 of the join
SCAN
SCAN
split each B scan output into 3 streams
merge the 3 input streams
at each join node
A2
A1
A0
B1
B0

PARALLELIZATION OF HASH-JOIN

- Sort-merge join does not work well if there is data skew
- **Hash-join** is more resistant to data skew
- To join 2 relations, we divide the join into a set of smaller joins
- The union of these smaller joins should result in the join of the 2 relations

How might other relational algorithms/operations (like sorting) be parallelized to improve query performance?

PARALLEL DATABASE SYSTEMS

- **Teradata** – Shared-nothing systems that use commodity hardware. Near linear speedup and scaleup on relational queries.
- **Tandem NonStop SQL** – Shared-nothing architecture. Near linear speedup and scaleup for large relational queries.
- **Gamma** – Shared-nothing system. Provides hybrid-range partitioning. Near linear speed and scaleup measured.
- **The Super Database Computer** – Has special-purpose components. However, is shared-nothing architecture with dataflow.
- **Bubba** – Shared-nothing system, but uses shared-memory for message passing. Uses FAD rather than SQL. Uses single-level storage

Should we focus on the use of commodity hardware to build parallel database systems?

DATABASE MACHINES AND GROSCH’S LAW

- **Grosch’s Law** – The performance of computer systems increase as the square of their cost. The more expensive systems have better cost to performance ratios.
- Less expensive shared-nothing systems like Gamma, Tandem, and Teradata achieve near-linear performance (compared to mainframes)
- Suggests that Grosch’s Law no longer applies

FUTURE DIRECTIONS AND RESEARCH PROBLEMS

- **Concurrent execution of simple and complex queries**
  - Problem with locks and large queries
  - Priority scheduling (priority inversion scheduling)
- **Parallel Query Optimization** – Optimizers do not take into account parallel algorithms. Data skew creates poor query plan cost estimates.
- **Application Program Parallelization** – Database applications are not parallelized. No effective way to automate the parallelization of applications.

FUTURE DIRECTIONS AND RESEARCH PROBLEMS

- **Physical Database Design** – Tools are needed to determine indexing and partitioning schemes given a database and its workload.
- **On-line Data Reorganization** – Availability of data for concurrent reads and writes as database utilities create indices, reorganize data, etc. This process must be (1) online, (2) incremental, (3) parallel, and (4) recoverable.

Which of these issues may be extremely difficult to resolve?

Thanks

Merci

Gracias

Contact: Genoveva Vargas-Solar, CNRS, LIG-LAFMIA

Genoveva.Vargas@imag.fr

http://vargas-solar.imag.fr

http://mapreducefest.wordpress.com/