Parallel Techniques for Big Data

Patrick Valduriez

Zenith

IBC
Institut de biologie computationnelle

Inria informatics mathematics

UNIVERSITÉ MONTPELLIER 2
SCIENCES ET TECHNIQUES

cnrs advancing the frontiers

Laboratoire d'Informatique de Robotique et de Microélectronique de Montpellier
LIRMM

Outline of the Talk

Big data: problem and issues

Parallel data processing

Parallel architectures

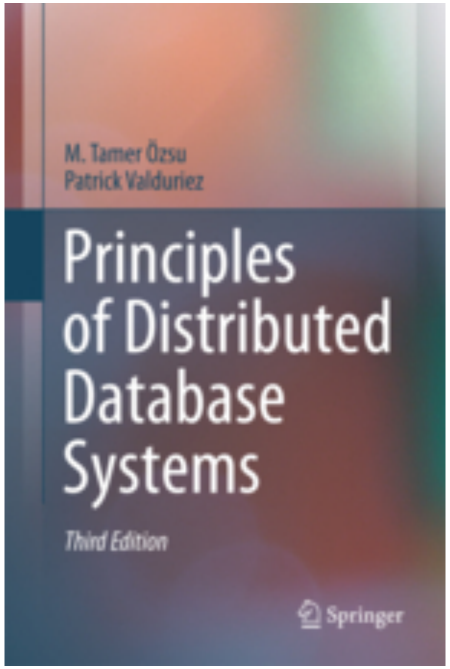
Parallel techniques

Cloud data mgt

NoSQL DBMS

MapReduce

Conclusion



Big Data: problem and issues

Big Data: what is it?

A buzz word!

- With different meanings depending on your perspective
  - E.g. 10 terabytes is big for a transaction processing system, but small for a world-wide web search engine

A definition (Wikipedia)

- Consists of data sets that grow so *large* that they become awkward to work with using on-hand database management tools
  - Difficulties: capture, storage, search, sharing, analytics, visualizing
- *But size is only one dimension of the problem*

How big is big?

- Moving target: terabyte (10^{12} bytes), petabyte (10^{15} bytes), exabyte (10^{18}), zetabyte (10^{21})
- Landmarks in DBMS products
  - 1980: Teradata database machine
  - 2010: Oracle Exadata database machine

Why Big Data Today?

Overwhelming amounts of data generated by all kinds of devices, networks and programs

- E.g. sensors, mobile devices, internet, social networks, computer simulations, satellites, radiotelescopes, LHC, etc.

Increasing storage capacity

- Storage capacity has doubled every 3 years since 1980 with prices steadily going down
  - 1 Gigabyte for: 1M$ in 1982, 1K$ in 1995, 0.12$ in 2011
- 1,8 zetabytes: an estimation for the data stored by humankind in 2011 (Digital Universe study of International Data Corporation)

Very useful in a digital world!

- Massive data can produce high-value information and knowledge
- Critical for data analysis, decision support, forecasting, business intelligence, research, (data-intensive) science, etc.

Big Data Dimensions: the three V's

Volume

- Refers to massive amounts of data
- Makes it hard to store and manage, but also to analyze (big analytics)

Velocity

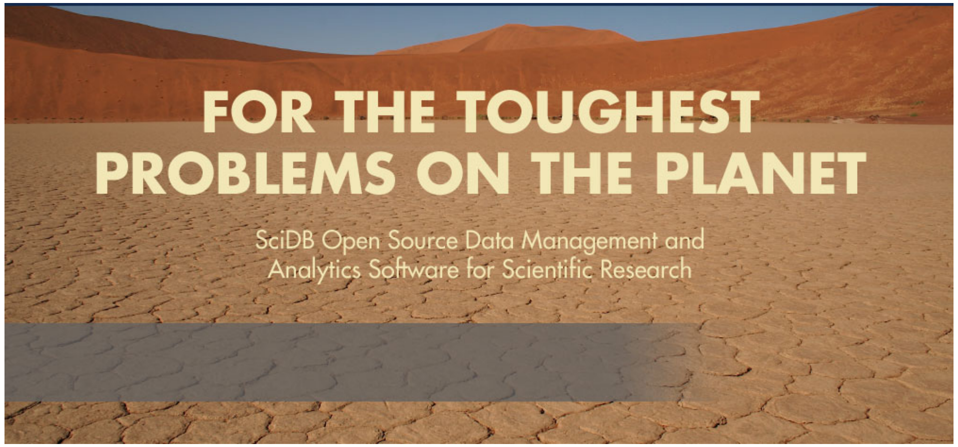
- Continuous data streams are being captured (e.g. from sensors or mobile devices) and produced
- Makes it hard to perform online processing

Variety

- Different data formats (sequences, graphs, arrays, …), different semantics, uncertain data (because of data capture), multiscale data (with lots of dimensions)
- Makes it hard to integrate and analyze

Scientific Data – *common features*

- **Big data**



- Manipulated through complex, distributed *workflows*
- Important *metadata* about experiments and their provenance
- Mostly append-only (with rare updates)

Parallel Data Processing

The solution to big data processing!

Exploit a massively parallel computer

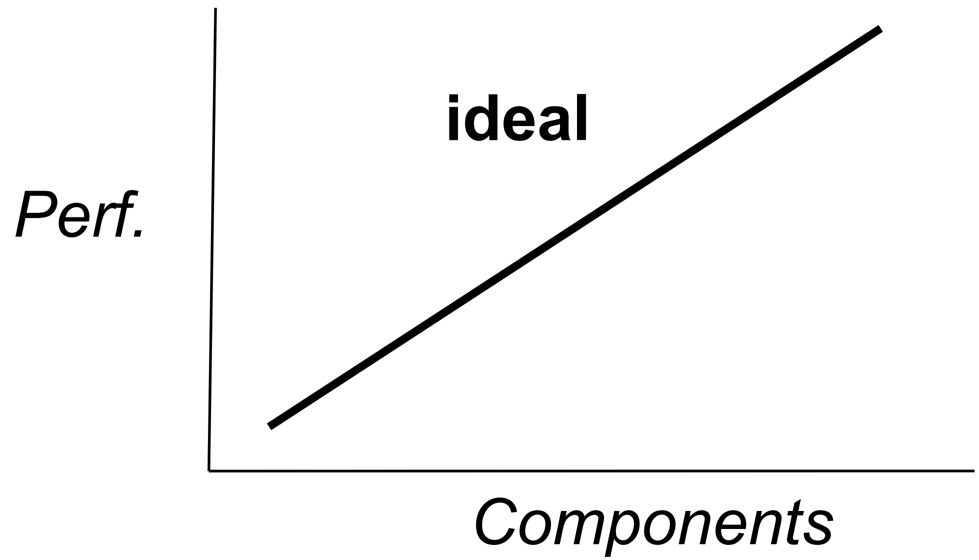
- A computer that interconnects lots of CPUs, RAM and disk units

To obtain

- *High performance* through data-based parallelism
  - High throughput for transaction-oriented (OLTP) loads
  - Low response time for decision-support (OLAP) queries
- *High availability* and reliability through data replication
- *Extensibility* with the ideal goals
  - Linear speed-up
  - Linear scale-up

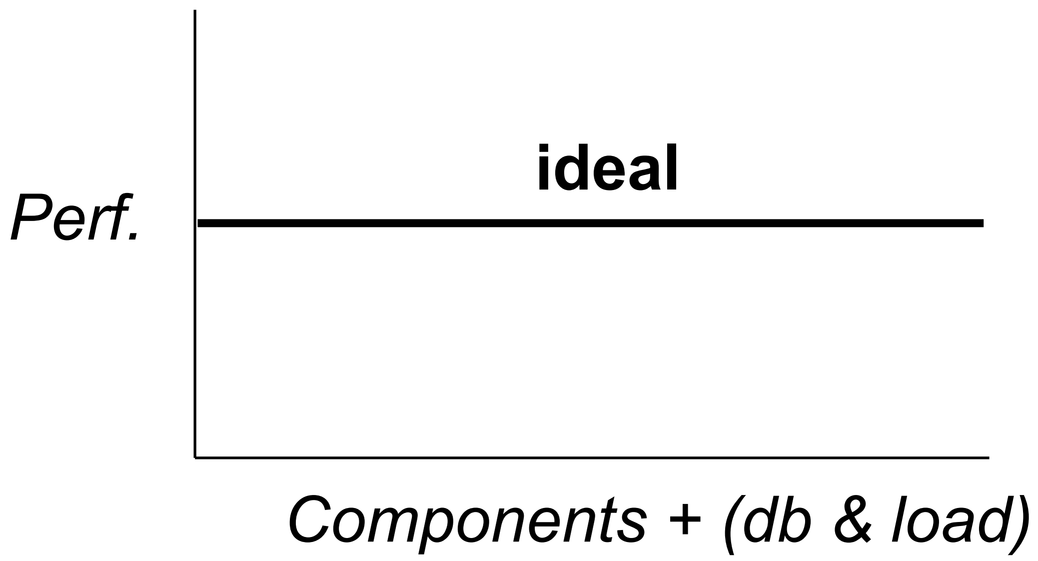
Linear Speed-up

Linear increase in performance for a constant database size and load, and proportional increase of the system components (CPU, memory, disk)



Linear Scale-up

Sustained performance for a linear increase of database size and load, and proportional increase of components



Data-based Parallelism

Inter-query

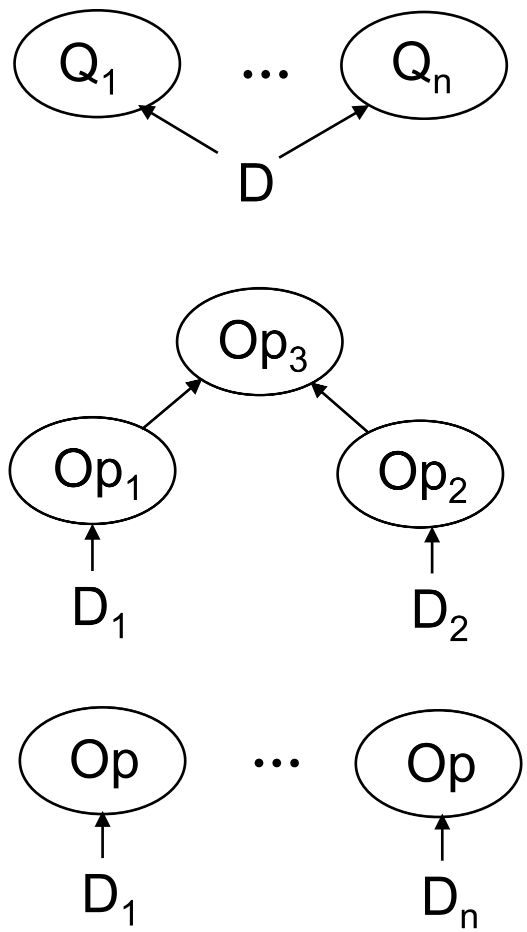
- Different queries on the same data
- For concurrent queries

Inter-operation

- Different operations of the same query on different data
- For complex queries

Intra-operation

- The same operation on different data
- For large queries



Parallel Architectures

Parallel Architectures for Data Management

Three main alternatives, depending on how processors, memory and disk are interconnected

- Shared-memory computer
- Shared-disk cluster
- Shared-nothing cluster

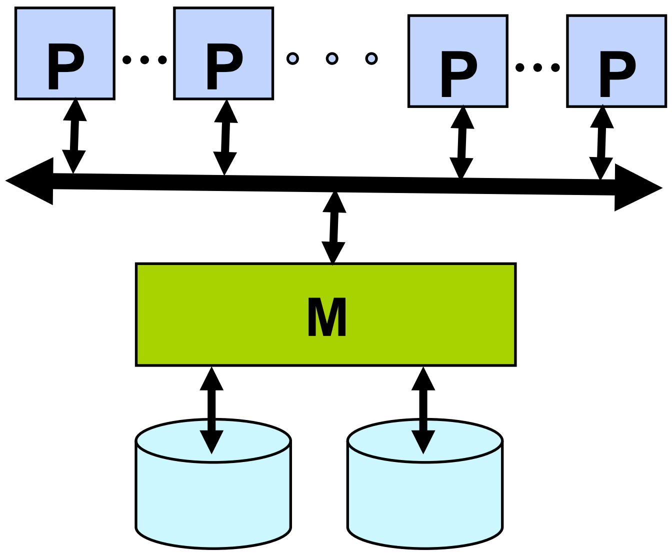
Shared-memory Computer

All memory and disk are shared

- Symmetric Multiprocessor (SMP)
- Non Uniform Memory Architecture (NUMA)
- Examples
  - IBM Numascale, HP Proliant, Data General NUMALiiNE, Bull Novascale

\+ Simple for apps, fast com., load balancing

\- Complex interconnect limits extensibility, cost



For write-intensive workloads, expensive for big data

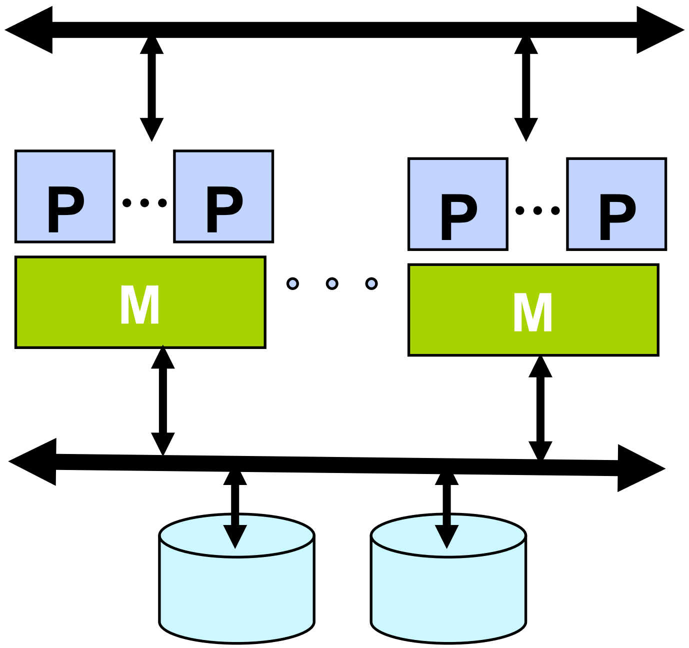
Shared-disk Cluster

Disk is shared, memory is private

- Storage Area Network (SAN) to interconnect memory and disk (block level)
- Needs distributed lock manager (DLM) for cache coherence
- Examples
  - Oracle RAC and Exadata
  - IBM PowerHA

+ Simple for apps, extensibility

- Complex DLM, cost



For write-intensive workloads or big data

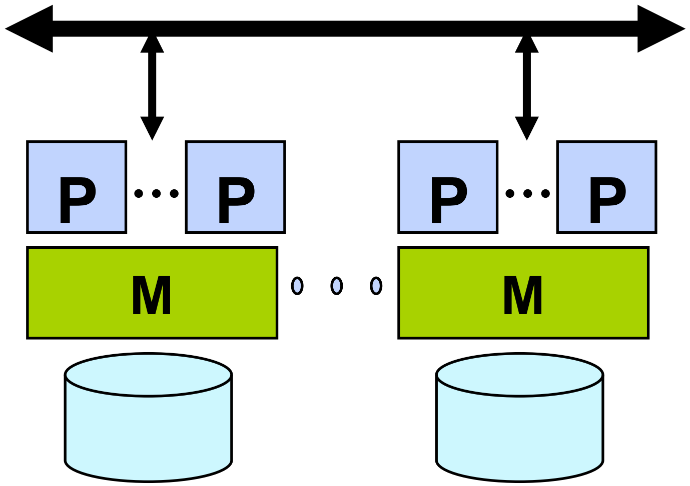
Shared-nothing (SN) Cluster

No sharing of either memory or disk across nodes

- No need for DLM
- But needs data partitioning
- Examples
  - DB2 DPF, SQL Server Parallel DW, Teradata, MySQLcluster
  - Google search engine, NoSQL key-value stores (Bigtable, …)

+ highest extensibility, cost

- updates, distributed trans.



\- *Perfect for big data (read intensive)*

Parallel Data Management Techniques

A Simple Model for Parallel Data

Shared-nothing architecture

- The most general and most scalable

Set-oriented

- Each dataset *D* is represented by a *table* of rows

Key-value

- Each row is represented by a <*key, value*> pair, where
  - Key uniquely identifies the value in *D*
  - Value is a list of (attribute name : attribute value) pairs

Can represent structured (relational) data or NoSQL data

- But graph is another story (see Pregel or DEX)

Examples

- <row-id_5, (part-id:5, part-name:iphone5, supplier:Apple)>
- <doc-id_{10}, (content:<html> html text … </html>)>
- <akeyword, (doc-id:id_1, doc-id:id_2, doc-id:id_{10})>

Design Considerations

Big datasets

- Data partitioning and indexing
  - Problem with skewed data distributions
- Parallel algorithms for algebraic operators
  - Select is easy, Join is difficult
- Disk is very slow (10K times slower than RAM)
  - Exploit RAM data structures and compression
  - Exploit fash memory (read 10 times faster than disk)

Query parallelization and optimization

- Automatic if the query language is declarative (e.g. SQL)
- Programmer-assisted otherwise (e.g. MapReduce)

Transaction support

- Hard: need for distributed transactions (distributed locks and 2PC)
  - NoSQL systems don't provide transactions

Fault-tolerance and availability

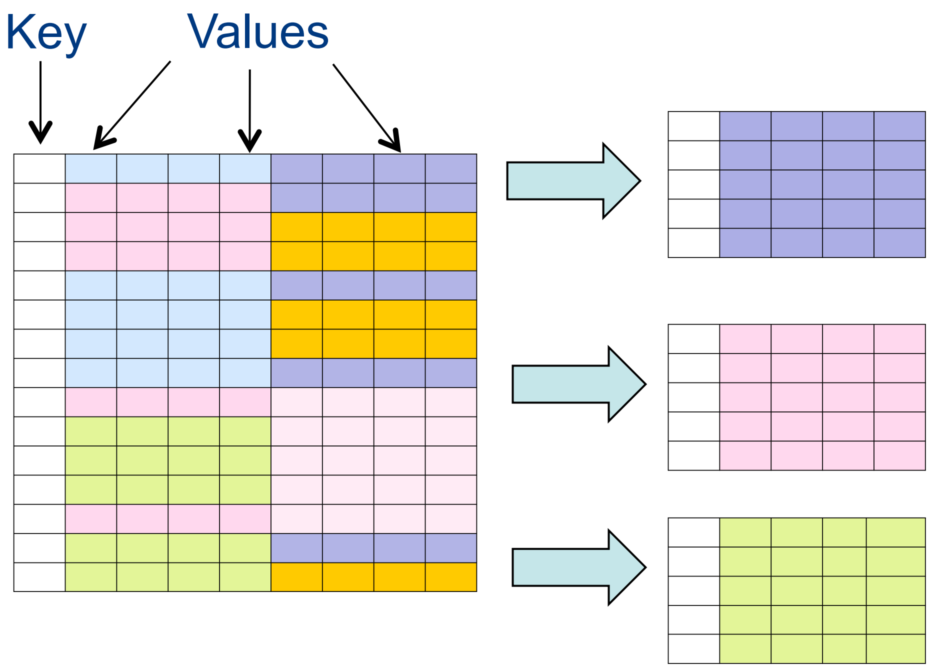
- With many nodes (e.g. several thousand), node failure is the norm, not the exception
  - Exploit replication and failover techniques

Data Partitioning

Vertical partitioning

- Basis for column stores (e.g. MonetDB, Vertica): efficient for OLAP queries
- Easy to compress, e.g. using Bloom filters

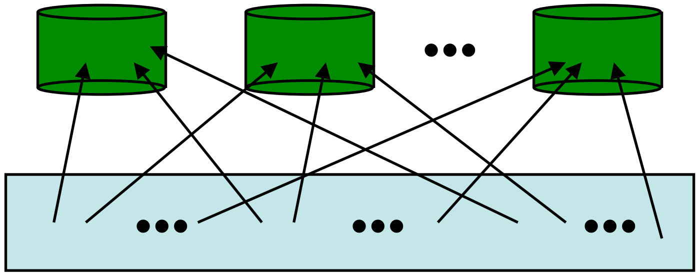
A table



Horizontal partitioning (sharding)

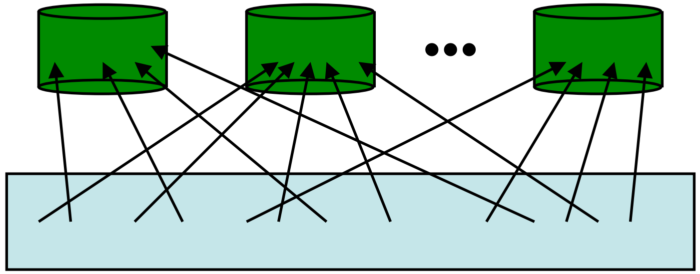
- Shards can be stored (and replicated) at different nodes

Sharding Schemes



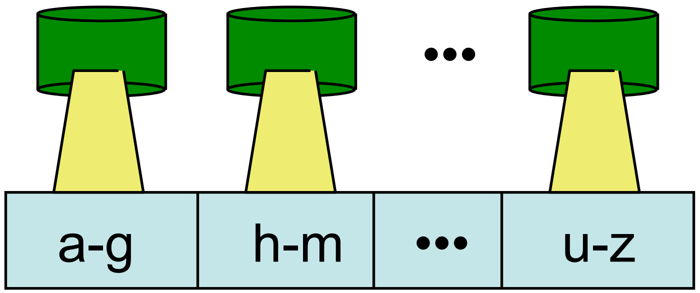
Round-Robin

- *i*th row to node (*i mod n*)
- perfect balancing
- but full scan only



Hashing

- (k,v) to node h(k)
- exact-match queries
- but problem with skew



Range

- (k,v) to node that holds k's interval
- exact-match and range queries
- deals with skew

Indexing

Can be supported by special tables with rows of the form: <*attribute, list of keys*> pairs

- Ex. <att-value, (doc-id:id_1, doc-id:id_2, doc-id:id_{10})>
- Given an attribute value, returns all corresponding keys
- These keys can in turn be used to access the corresponding rows in shards

Complements sharding with secondary indices or inverted files to speed up attribute-based queries

Can be partitioned

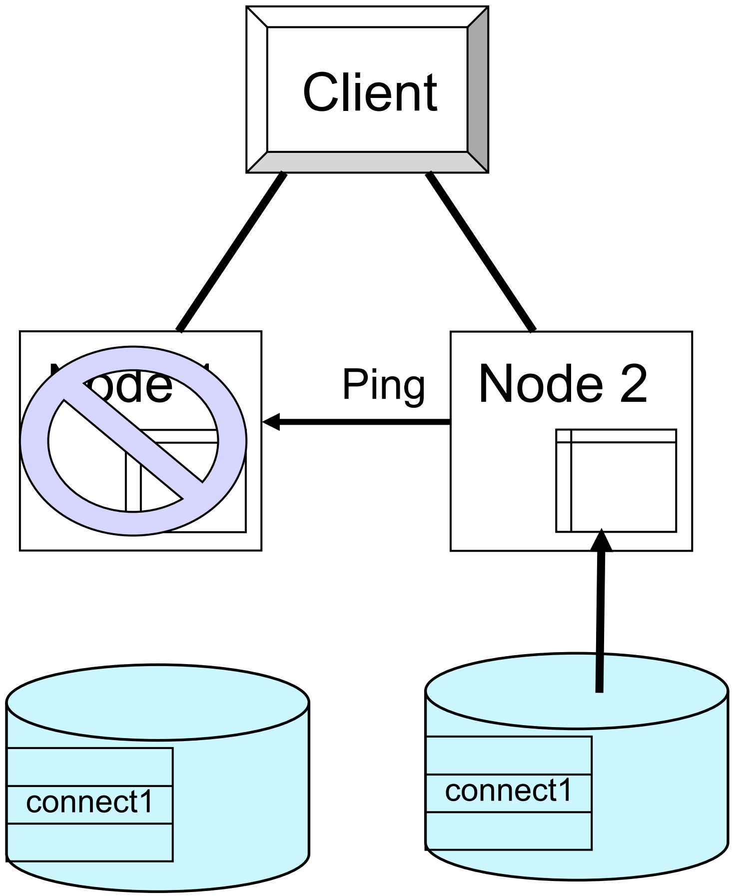
Replication and Failover

Replication

- The basis for fault-tolerance and availability
- Have several copies of each shard

Failover

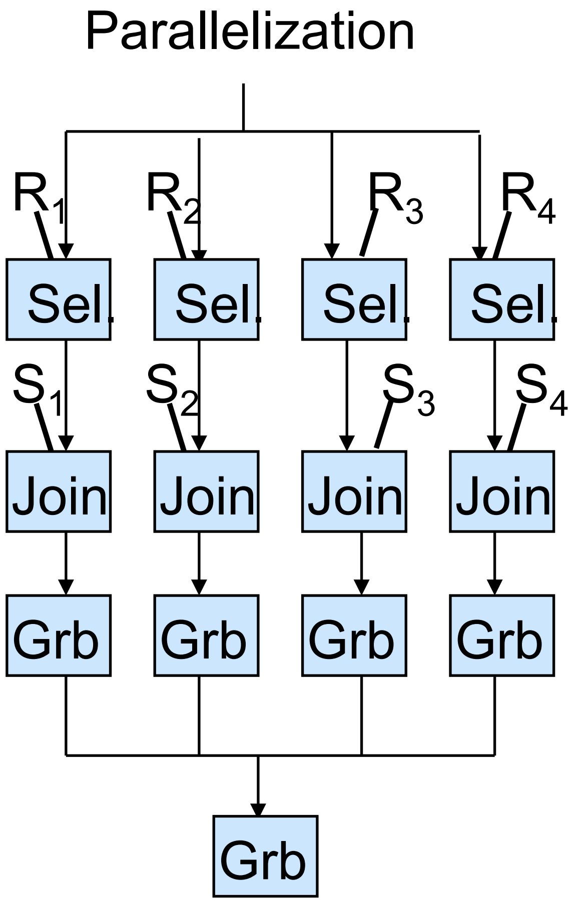
- On a node failure, another node detects and recovers the node's tasks



Parallel Query Processing

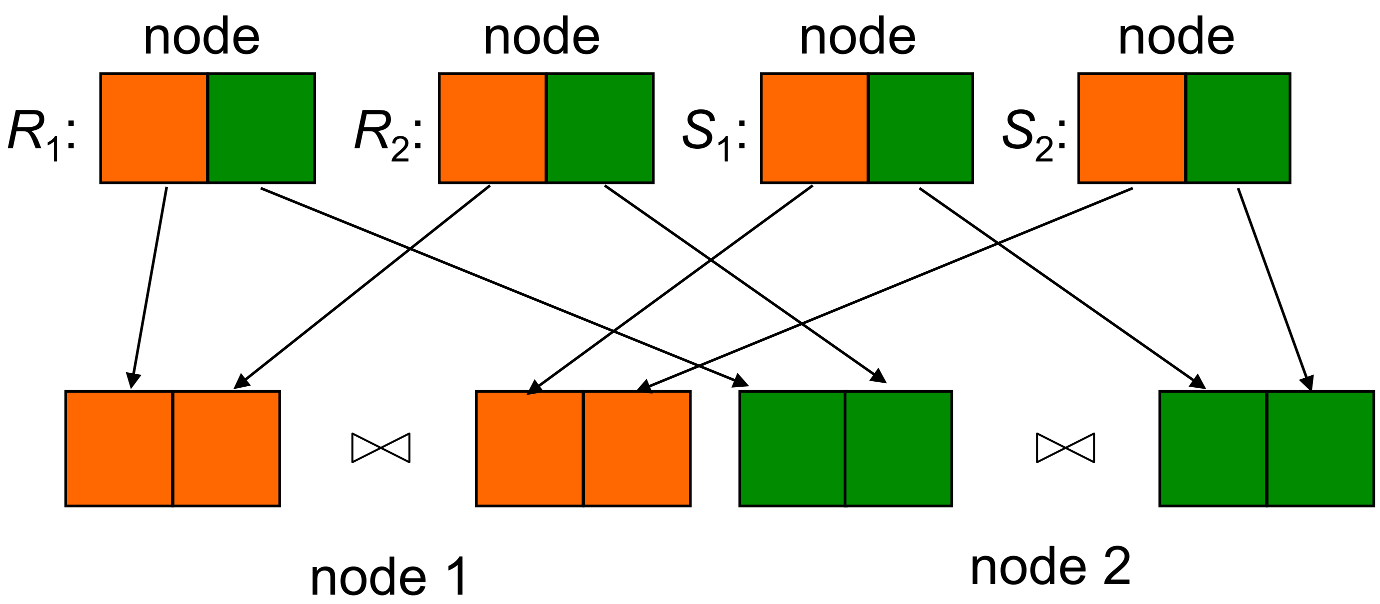
1. Query parallelization
   - Produces an optimized parallel execution plan, with operators
   - Based on partitioning, replication, indexing
2. Parallel execution
   - Relies on parallel main memory algorithms for operators
   - Use of hashed-based join algorithms
   - Adaptive degree of partitioning to deal with skew

Select … from R,S
where …group by…



Parallel Hash Join Algorithm

Both tables R and S are partitioned by hashing on the join attribute



$$R \textit{ join } S = \bigcup_{i=1}^{p} R_i \textit{ join } S_i$$

Parallel DBMS

Generic: with full support of SQL, with user defined functions

- Structured data, XML, multimedia, etc.
- Automatic optimization and parallelization

Transactional guarantees

- Atomicity, Consistency, Isolation, Durability
- Transactions make it easy to program complex updates

Performance through

- Data partitioning, indexing, caching
- Sophisticated parallel algorithms, load balancing

Two kinds

- Row-based: Oracle, MySQL, MS SQLserver, IBM DB2
- Column-based: MonetDB, HP Vertica

Cloud Data Management

Cloud Data: problem and solution

Cloud data

- Can be very large (e.g. text-based or scientific applications), unstructured or semi-structured, and typically append-only (with rare updates)

Cloud users and application developers

- In very high numbers, with very diverse expertise but very little DBMS expertise

Therefore, current cloud data management solutions trade consistency for scalability, simplicity and flexibility

- New file systems: GFS, HDFS, …
- NOSQL: Amazon SimpleDB, Google Base, Google Bigtable, Yahoo Pnuts, etc.
- New parallel programming: Google MapReduce (and its many variations)

Google File System (GFS)

Used by many Google applications

- Search engine, Bigtable, Mapreduce, etc.

The basis for popular Open Source implementations: Hadoop HDFS (Apache & Yahoo)

Optimized for specific needs

- Shared-nothing cluster of thousand nodes, built from inexpensive harware => node failure is the norm!
- Very large files, of typically several GB, containing many objects such as web documents
- Mostly read and append (random updates are rare)
  - Large reads of bulk data (e.g. 1 MB) and small random reads (e.g. 1 KB)
  - Append operations are also large and there may be many concurrent clients that append the same file
  - High throughput (for bulk data) more important than low latency

Design Choices

Traditional file system interface (create, open, read, write, close, and delete file)

- Two additional operations: snapshot and record append.

Relaxed consistency, with atomic record append

- No need for distributed lock management
- Up to the application to use techniques such as checkpointing and writing self-validating records

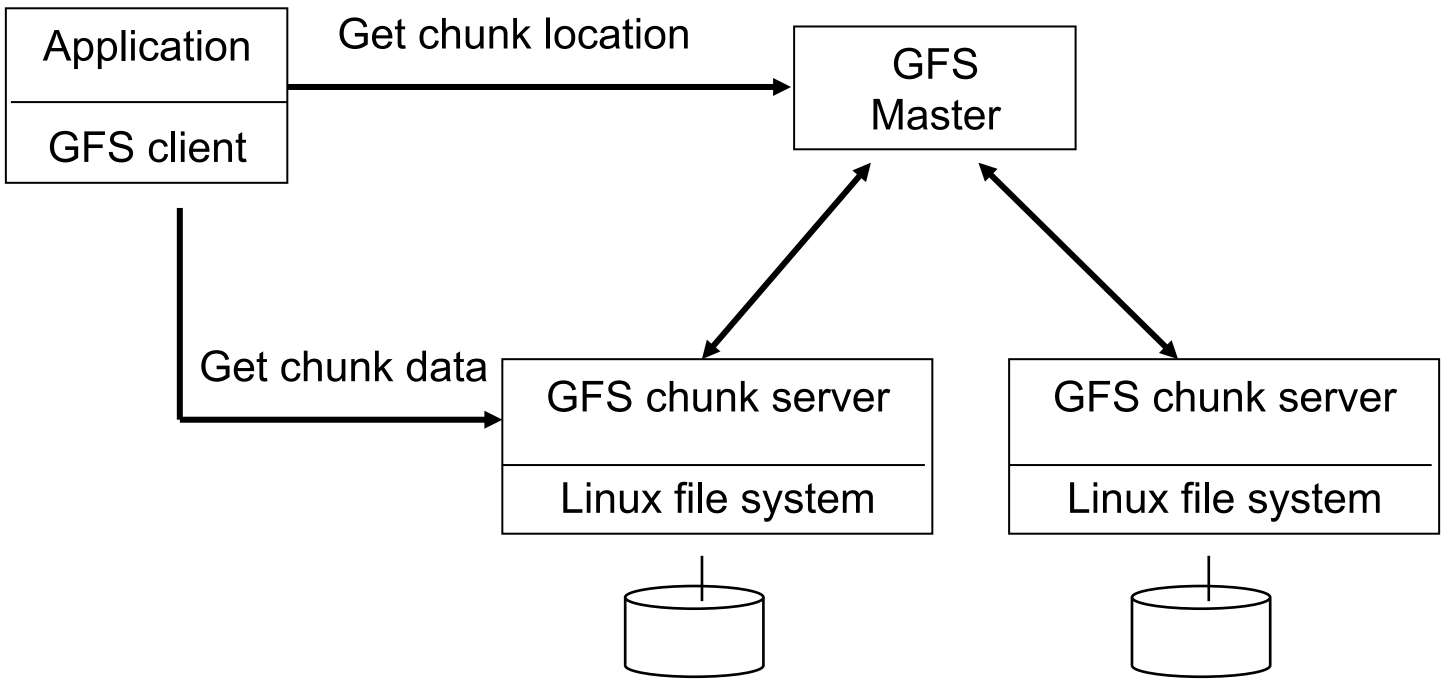
Single GFS master

- Maintains file metadata such as namespace, access control information, and data placement information
- Simple, lightly loaded, fault-tolerant

Fast recovery and replication strategies

GFS Distributed Architecture

Files are divided in fixed-size partitions, called *chunks*, of large size, i.e. 64 MB, each replicated at several nodes



NoSQL DBMS

NOSQL (Not Only SQL) Systems

Specific DBMS: for web-based data

- Trade relational DBMS properties
  - Full SQL, transactions, data independence
- For
  - Simplicity (flexible schema, basic API)
  - Scalability

Different kinds

- Key-value, ex. Google Bigtable, Amazon SimpleDB
- Structure-specific: document, graph, array, etc.

NB: SQL is just a language and has nothing to do with the story

Google Bigtable

Database storage system for a shared-nothing cluster

- Uses GFS to store structured data, with fault-tolerance and availability

Used by popular Google applications

- Google Earth, Google Analytics, Google+, etc.

The basis for popular Open Source implementations

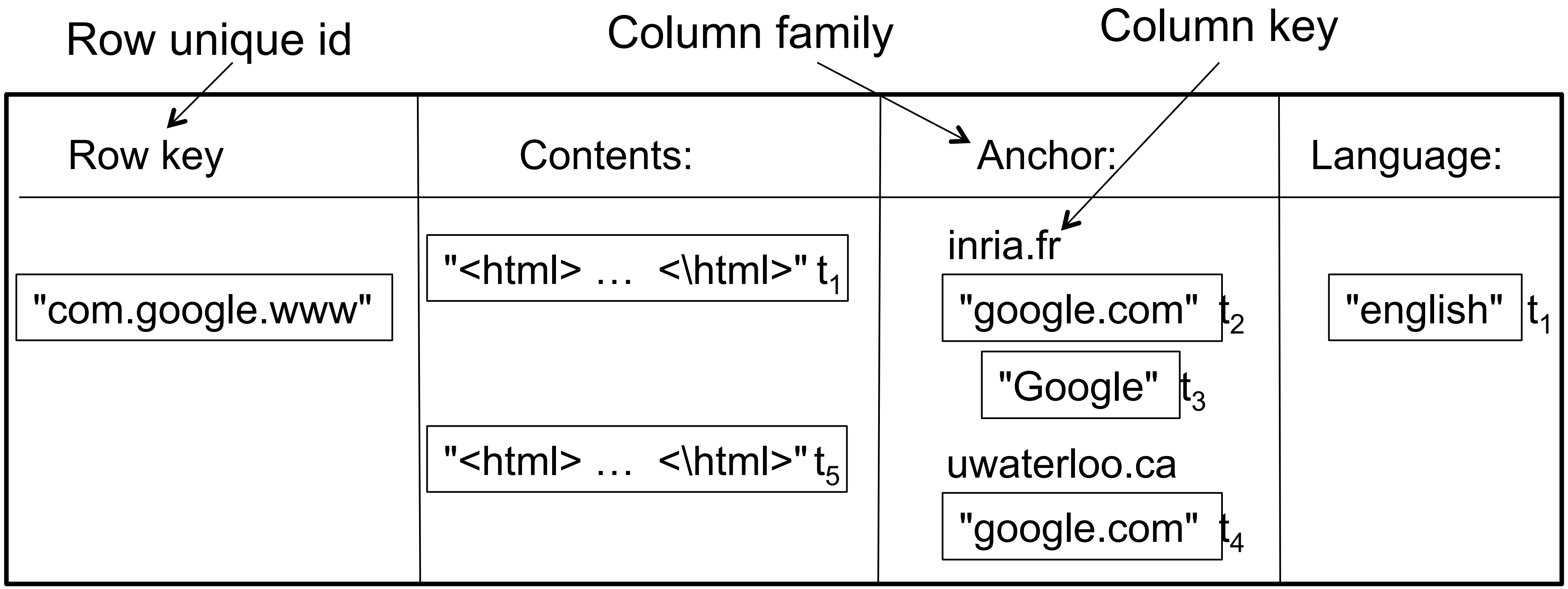
- Hadoop Hbase on top of HDFS (Apache & Yahoo)

Specific data model that combines aspects of row-store and column-store DBMS

- Rows with multi-valued, timestamped attributes
  - A Bigtable is defined as a multidimensional map, indexed by a row key, a column key and a timestamp, each cell of the map being a single value (a string)

Dynamic partitioning of tables for scalability

A Bigtable Row



Column family = a kind of multi-valued attribute

- Set of columns (of the same type), each identified by a key
  - Colum key = attribute value, but used as a name
- Unit of access control and compression

Bigtable DDL and DML

Basic API for defining and manipulating tables, within a programming language such as C++

- Various operators to write and update values, and to iterate over subsets of data, produced by a scan operator
- Various ways to restrict the rows, columns and timestamps produced by a scan, as in relational select, but no complex operator such as join or union
- Transactional atomicity for single row updates only

Dynamic Range Partitioning

Range partitioning of a table on the row key

- Tablet = a partition (shard) corresponding to a row range
- Partitioning is dynamic, starting with one tablet (the entire table range) which is subsequently split into multiple tablets as the table grows
- Metadata table itself partitioned in metadata tablets, with a single root tablet stored at a master server, similar to GFS's master

Implementation techniques

- Compression of column families
- Grouping of column families with high locality of access
- Aggressive caching of metadata information by clients

Yahoo! PNUTS

Parallel and distributed database system

Designed for serving Web applications

- No need for complex queries
- Need for good response time, scalability and high availability
- Relaxed consistency guarantees for replicated data

Used internally at Yahoo!

- User database, social networks, content metadata management and shopping listings management apps

Design Choices

Basic relational data model

- Tables of flat records, Blob attributes
- Flexible schemas
  - New attributes can be added at any time even though the table is being queried or updated
  - Records need not have values for all attributes

Simple query language

- Selection and projection on a single relation
- Updates and deletes must specify the primary key

Range partitioning or hashing of tables into tablets

- Placement in a cluster (at a site)
- Sites in different geographical regions maintain a complete copy of the system and of each table

Publish/subscribe mechanism with guaranteed delivery, for both reliability and replication

- Used to replay lost updates, thus avoiding a traditional database log

Relaxed Consistency Model

Between strong consistency and eventual consistency

- Motivated by the fact that Web applications typically manipulate only one record at a time, but different records may be used under different geographic locations

Per-record timeline consistency: guarantees that all replicas of a given record apply all updates to the record in the same order

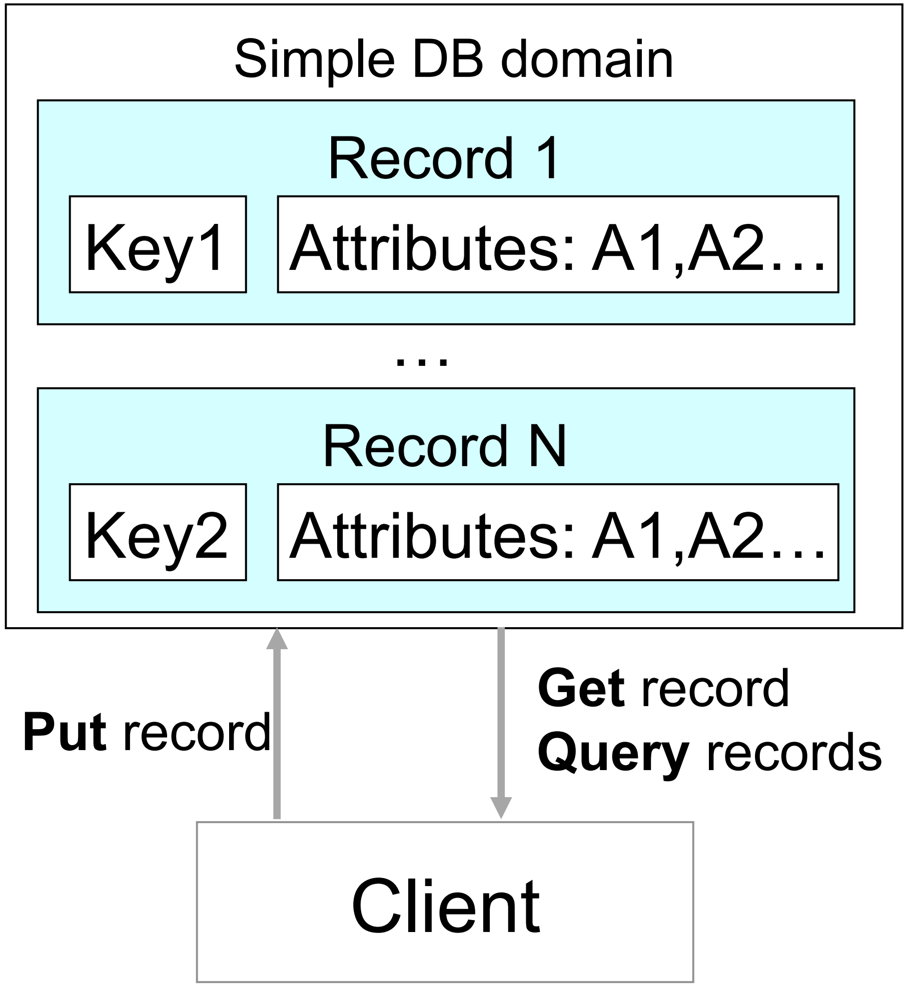
Several API operations with different guarantees

- Read-any: returns a possibly stale version of the record
- Read-latest: returns the latest copy of the record
- Write: performs a single atomic write operation

Amazon SimpleDB

- A basic key-value store DBMS, without imposed schema
  - Flat files
  - Basic operators (scan, filter, join, aggregate)
  - Cache, replication
  - Transactions
  - SQL frontend
- But no
  - Query optimizer
  - Complex relational operators (union, etc)
  - Fault tolerance
  - Index definition (all fields automatically indexed)

SimpleDB Data Model



- Flexible data model
  - Each attribute is indexed
  - Zero administration

SimpleDB Example

item	description	color	material
123	Sweater	Blue, Red	
456	Dress shirt	White, Blue	
789	Shoes	Black	Leather

Inserts

Put (item, 123), (description, Sweater), (color, Blue), (color, Red)

Put (item, 456), (description, Dress shirt), (color, White), (color, Blue)

Put (item, 789), (description, Shoes), (color, Black), (material, Leather)

A simple query

Domain = MyStore [‘description’ = ‘Sweater’]

Other NOSQL Systems

Company	Product	Category	Comment
Amazon	Dynamo	KV store	
Apache	Cassandra Accumulo	KV store KV store	Orig. Facebook Orig. NSA
Google	Pregel	Graphs	
Hadoop	Hbase	KV store	Orig. Yahoo
LinkedIn	Voldemort	KV store	
10gen	MongoDB	Documents	
Neo4J.org	Neo4J	Graphs	
Sparcity	DEX	Graphs	Orig. UPC, Barcelone
Ubuntu	CouchDB	Documents	

MapReduce

MapReduce

Parallel programming framework from Google
- Proprietary (and protected by software patents)
- But popular Open Source version by Hadoop (Apache & Yahoo)

For data analysis of very large data sets
- Highly dynamic, irregular, schemaless, etc.
- SQL or Xquery too heavy

New, simple parallel programming model
- Data structured as (key, value) pairs
  - E.g. (doc-id, content), (word, count), etc.
- Functional programming style with two functions to be given:
  - Map(key, value) -> ikey, ivalue
  - Reduce(ikey, list (ivalue)) –> list(fvalue)

Implemented on GFS on very large clusters

The basis for popular implementations
- Hadoop, Hadoop++, Amazon MapReduce, etc.

MapReduce Typical Usages

Counting the numbers of some words in a set of docs

Distributed grep: text pattern matching

Counting URL access frequencies in Web logs

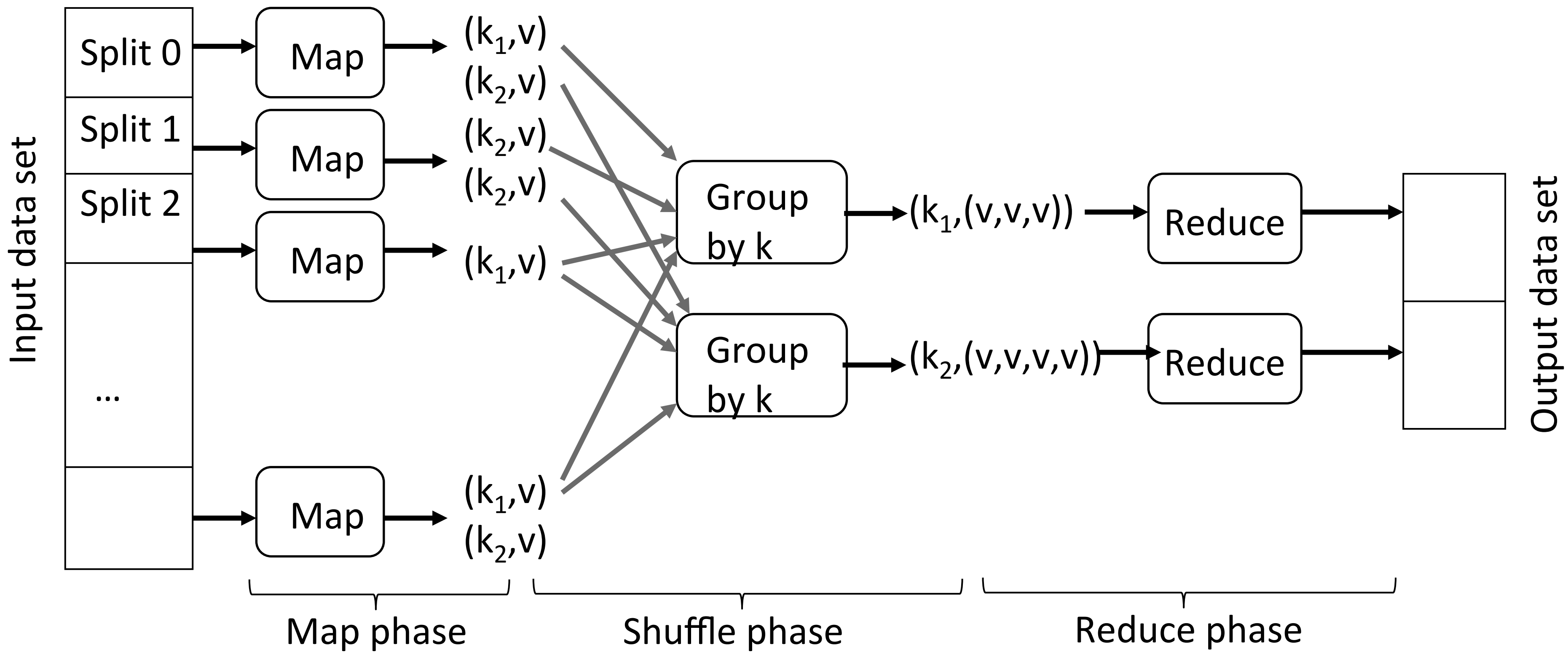
Computing a reverse Web-link graph

Computing the term-vectors (summarizing the most important words) in a set of documents

Computing an inverted index for a set of documents

Distributed sorting

MapReduce Processing



Simple programming model

- Key-value data storage
- Hash-based data partitioning

MapReduce Example

EMP (ENAME, TITLE, CITY)

Query: for each city, return the number of employees whose name is "Smith"

```
SELECT CITY, COUNT(*)
FROM EMP
WHERE  ENAME LIKE "\%Smith"
GROUP BY CITY
```

With MapReduce

```
Map (Input (TID,emp), Output: (CITY,1))
    if emp.ENAME like "%Smith" return (CITY,1)
Reduce (Input (CITY,list(1)), Output: (CITY,SUM(list(1)))
    return (CITY,SUM(1*))
```

Fault-tolerance

Fault-tolerance is fine-grain and well suited for large jobs

Input and output data are stored in GFS

- Already provides high fault-tolerance

All intermediate data is written to disk

- Helps checkpointing Map operations, and thus provides tolerance from soft failures

If one Map node or Reduce node fails during execution (hard failure)

- The tasks are made eligible by the master for scheduling onto other nodes
- It may also be necessary to re-execute completed Map tasks, since the input data on the failed node disk is inaccessible

MapReduce vs Parallel DBMS

[Pavlo et al. SIGMOD09]: Hadoop MapReduce vs two parallel DBMS, one row-store DBMS and one column-store DBMS

- Benchmark queries: a grep query, an aggregation query with a group by clause on a Web log, and a complex join of two tables with aggregation and filtering
- Once the data has been loaded, the DBMS are significantly faster, but loading is much time consuming for the DBMS
- Suggest that MapReduce is less efficient than DBMS because it performs repetitive format parsing and does not exploit pipelining and indices

[Dean and Ghemawat, CACM10]

- Make the difference between the MapReduce model and its implementation which could be well improved, e.g. by exploiting indices

[Stonebraker et al. CACM10]

- Argues that MapReduce and parallel DBMS are complementary as MapReduce could be used to extract-transform-load data in a DBMS for more complex OLAP.

MapReduce Performance

Much room for improvement (see MapReduce yearly workshop)

- Map phase
  - Minimize I/0 cost using indices (Hadoop++)
- Shuffle phase
  - Minimize data transfers by partitioning data on the same IK
  - Current work in Zenith
- Reduce phase
  - Exploit fine-grain parallelism of Reduce tasks
  - Current work in Zenith

Conclusion

Basic techniques are not new

- Parallel database machines, shared-nothing cluster
- Data partitioning, replication, indexing, parallel hash join, etc.
- But need to scale up

NoSQL key-value stores

- Trade consistency and transactional guarantees for scalability
- Simple API with data-oriented operators available to the programmer
- Less structure, but more parsing

Towards hybrid NoSQL/RDBMS?

- Google F1: "combines the scalability, fault tolerance, transparent sharding, and cost benefits so far available only in NoSQL systems with the usability, familiarity, and transactional guarantees expected from an RDBMS"

Much room for research and innovation

- MapReduce extensions, dynamic workload-based partitioning, data-oriented scientific workflows, uncertain data mining, content-based IR, etc.