Data Management in the Cloud - current issues and research directions

Patrick Valduriez
Esther Pacitti

DNAC Congress, Paris, nov. 2010
http://www.med-hoc-net-2010.org

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

UNIVERSITÉ MONTPELLIER 2
SCIENCES ET TECHNIQUES

INRIA
SOPHIA ANTIPOLIS - MÉDITERRANÉE

Is Research Needed in the Cloud?

Grand Challenge

- Cost-effective support of the very large scale of the infrastructure to manage lots of users and resources with high QoS

Current solutions are ad-hoc and proprietary

- Developed by Web industry giants such as Amazon, Google, Microsoft, Yahoo
  - E.g. Google File System (GFS)
- Specific, simple applications with low consistency needs

But the research community is catching up

- Many new conferences and journals on Cloud Computing
  - Distributed systems, OS, data management communities
- Open Source alternatives, e.g. Hadoop HDFS
- As the complexity of applications increases, the implication of the research community is needed

Outline

OLTP vs OLAP apps in the cloud

Grid vs cloud architecture

Cloud data management solutions

- Distributed file management with GFS
- Distributed database management with Bigtable and Pnuts
- Parallel data processing with MapReduce

Issues

Research directions

Cloud Benefits

Reduced cost

- Customer side: the IT infrastructure needs not be owned and managed, and billed only based on resource consumption
- Cloud provider side: by sharing costs for multiple customers, reduces its cost of ownership and operation to the minimum

Ease of access and use

- Customers can have access to IT services anytime, from anywhere with an Internet connection

Quality of Service (QoS)

- The operation of the IT infrastructure by a specialized, experienced provider (including with its own infrastructure) increases QoS

Elasticity

- Easy for customers to deal with sudden increases in loads by simply creating more virtual machines (VMs)

OLTP vs OLAP in the Cloud

OLTP

- Operational databases of average sizes (TB), write-intensive
- ACID transactional properties, strong data protection, response time guarantees

Not very suitable for cloud

- Requires shared-disk multiprocessors
- Corporate data gets stored at untrusted host

OLAP

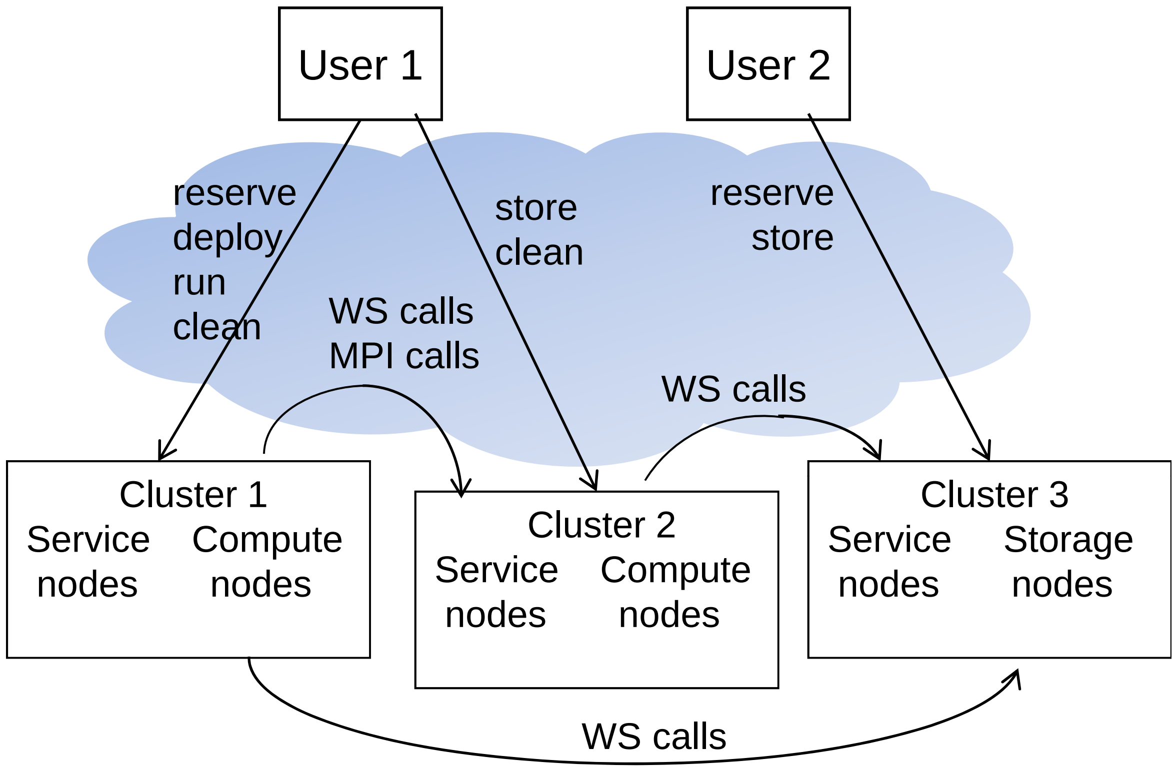
- Historical databases of very large sizes (PB), read-intensive, can accept relaxed ACID properties

Suitable for cloud

- Shared-nothing clusters of commodity servers are cost-effective
- Sensitive data can be hidden (anonymized) in the cloud

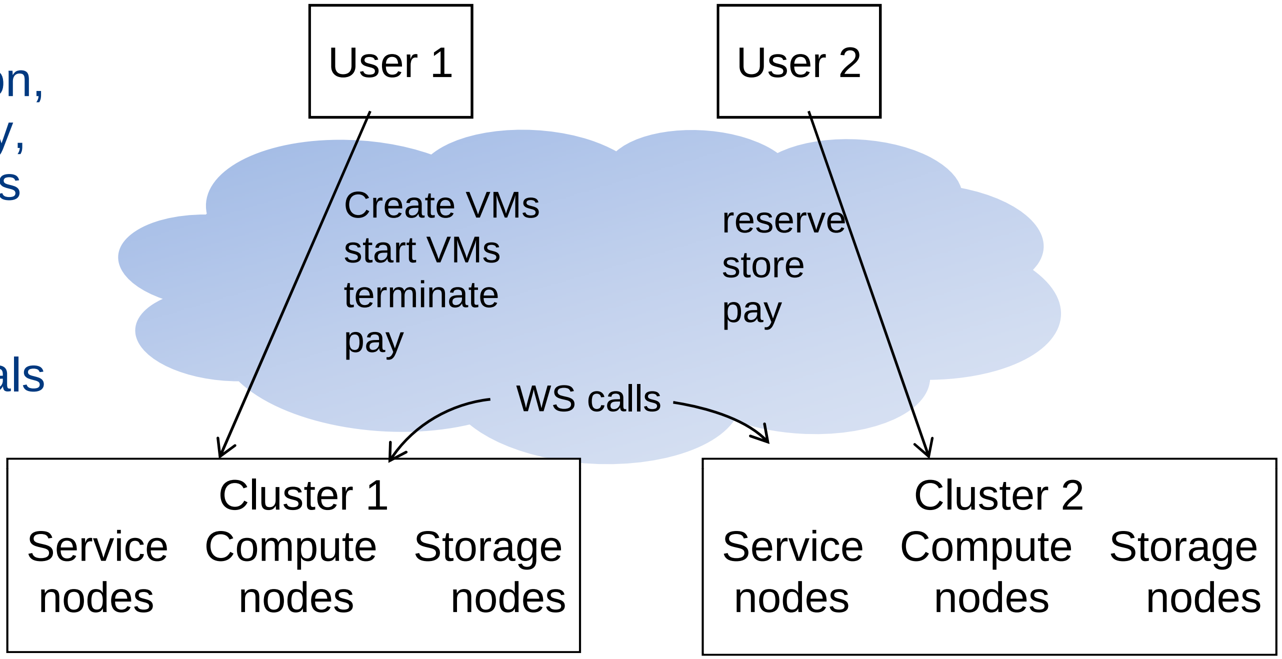
Grid Architecture

- Access through Web services to distributed, heterogeneous resources
  - supercomputers, clusters, databases, etc.
- For **Virtual Organizations**
  - which share the same resources, with common rules and access rights
- Grid middleware
  - security, database, provisioning, job scheduling, workflow management, etc.



Cloud Architecture

- Like grid, access to resources using Web services
  - But less distribution, more homogeneity, and bigger clusters
- For different customers
  - Including individuals
- Replication across sites for high availability
- Scalability, SLA, accounting and pricing essential



Cloud Data Management: why not RDBMS?

RDBMS all have a distributed and parallel version

- With SQL support for all kinds of data (structured, XML, multimedia, streams, etc.)

But the “one size fits all” approach has reached the limits

- Loss of performance, simplicity and flexibility for applications with specific, tight requirements
- New specialized DBMS engines better: column-oriented DBMS for OLAP, DSMS for stream processing, etc.

For the cloud, RDBMS provide both

- Too much: ACID transactions, complex query language, lots of tuning knobs
- Too little: specific optimizations for OLAP, flexible programming model, flexible schema, scalability

Cloud Data Management Solutions

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, …
- New DBMS: 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

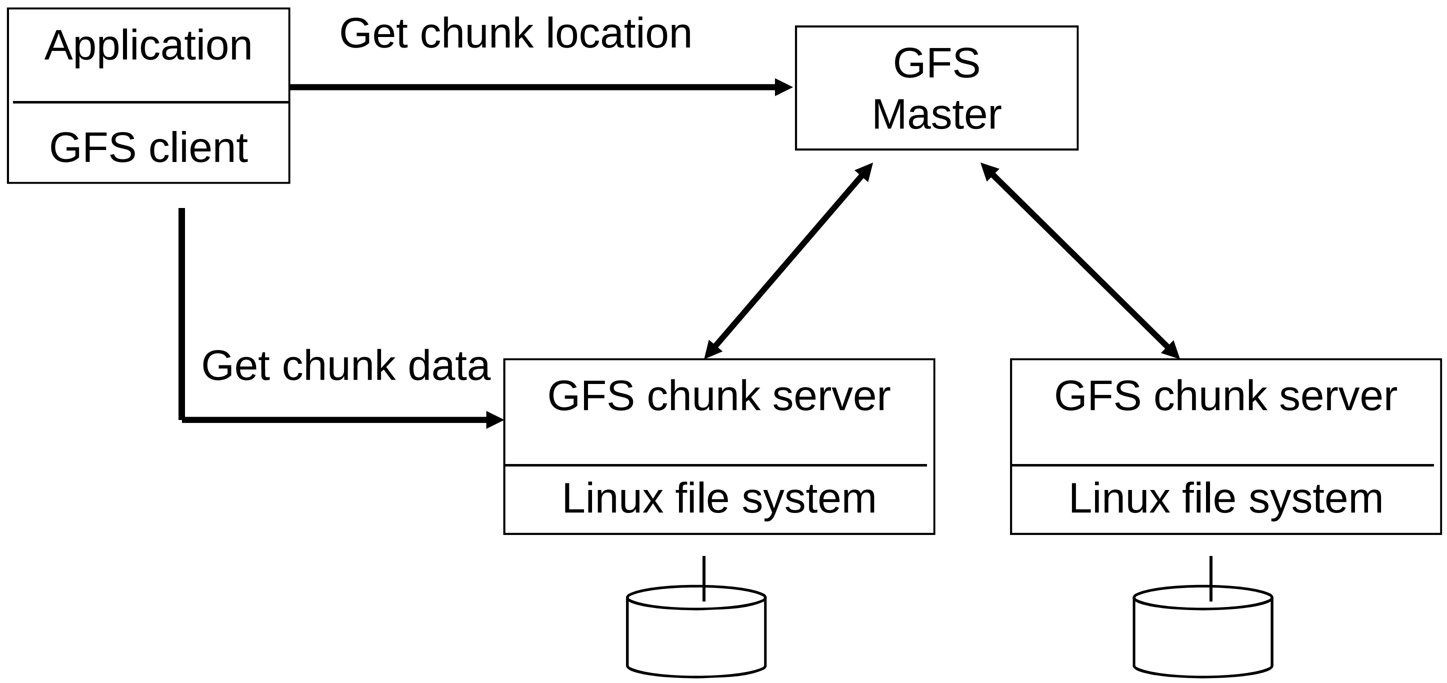
Single GFS master

- 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



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, Orkut, etc.

The basis for popular Open Source implementations

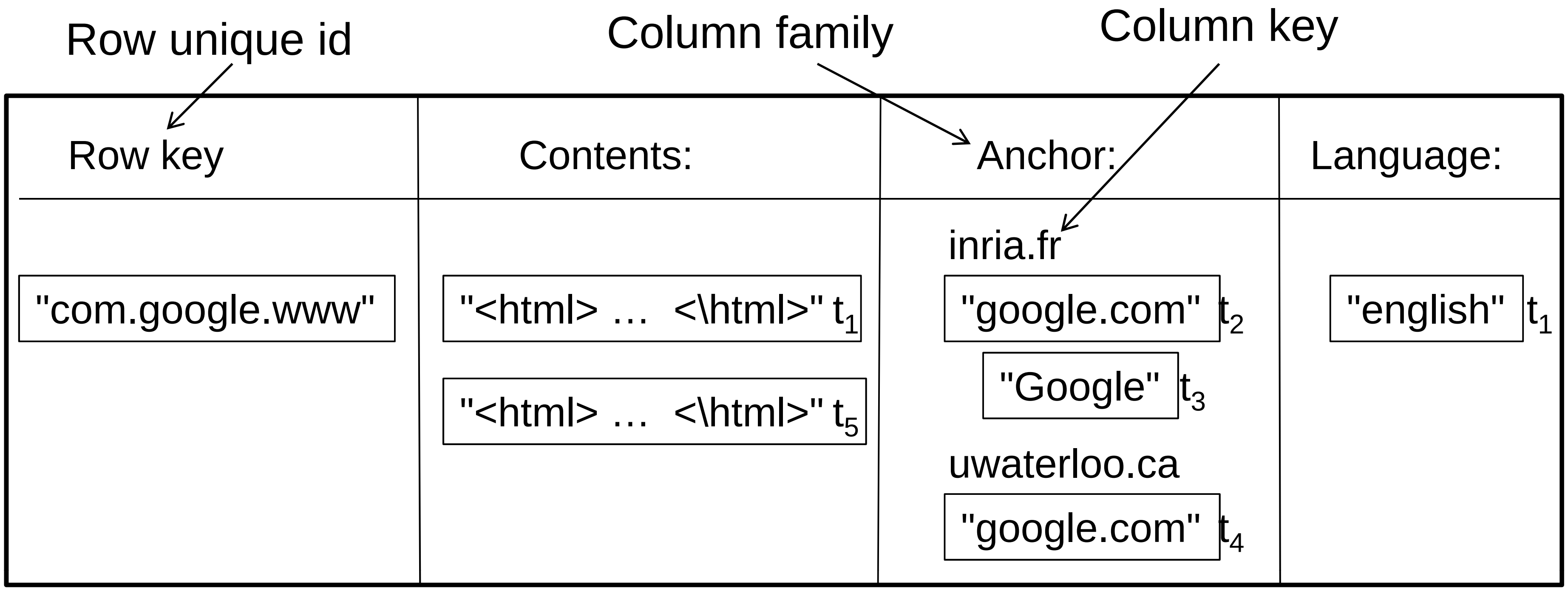
- Hadoop Hbase on top of HDFS (Apache & Yahoo)

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 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

MapReduce

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(k1,v1) -> list(k2,v2)
  - Reduce(k2, list (v2)) –> list(v3)

Implemented on GFS on very large clusters

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

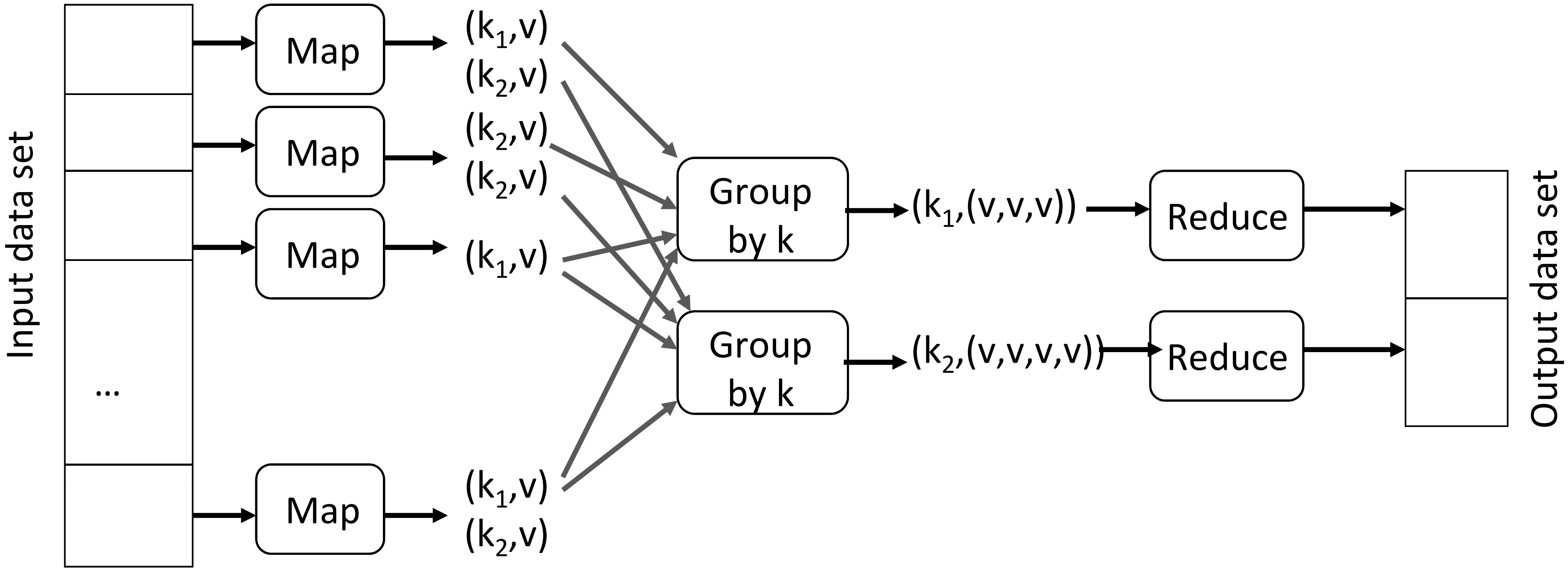
Computing a reverse Web-link graph

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



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.

Issues in Cloud Data Management

Main challenge: provide ease of programming, consistency, scalability and elasticity at the same time, over cloud data

Current solutions

- Quite successful for specific, relatively simple applications
- Have sacrificed consistency and ease of programming for the sake of scalability
- Force applications to access data partitions individually, with a loss of consistency guarantees across data partitions

For more complex apps. with tighter consistency requirements

- Developers are faced with a very difficult problem: providing isolation and atomicity across data partitions through careful engineering

Research Directions in Data Management

Declarative programming languages for the cloud

- E.g. BOOM project (UC Berkeley] using Overlog

Parallel OLAP query processing with consistency guarantees wrt concurrent updates

- E.g. using snapshot isolation

Scientific workflow management

- E.g. with P2P worker nodes

Data privacy preserving query processing

- E.g. queries on encrypted data

Autonomic data management

- E.g. automatic management of replication to deal with load changes

Green data management

- E.g. optimizing for energy efficiency

Cloud research @ Montpellier