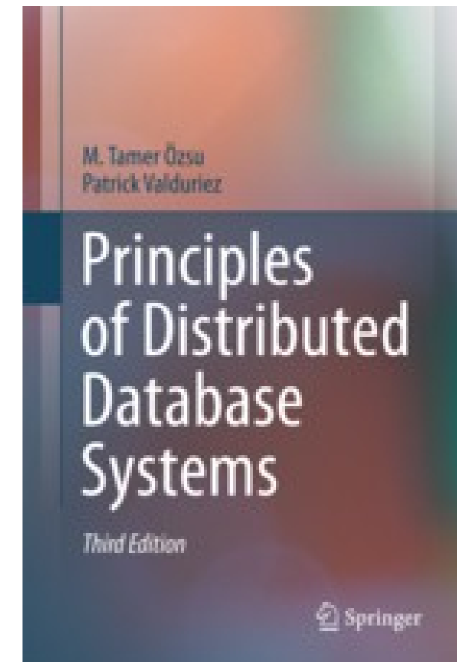
Parallel Techniques for Big Data

Patrick Valduriez
INRIA,
Montpellier

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 TP system, but small for a web search engine
- A definition (Wikipedia)
  - Consists of data sets that grow so *large* that they become awkward to work with using on-hand data 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 (1012 bytes), petabyte (1015 bytes), exabyte (1018), zetabyte (1021)

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, 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
- Very useful in a digital world!
  - Massive data can produce high-value information and knowledge

Some estimates

- 1,8 zetaoctets: an estimation for the data stored by humankind in 2011
- 40 zetaoctets in 2020
- Less than 1% of big data is analyzed
- Less than 20% of big data is protected

  - Source: Digital Universe study of International Data Corporation, december 2012

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)

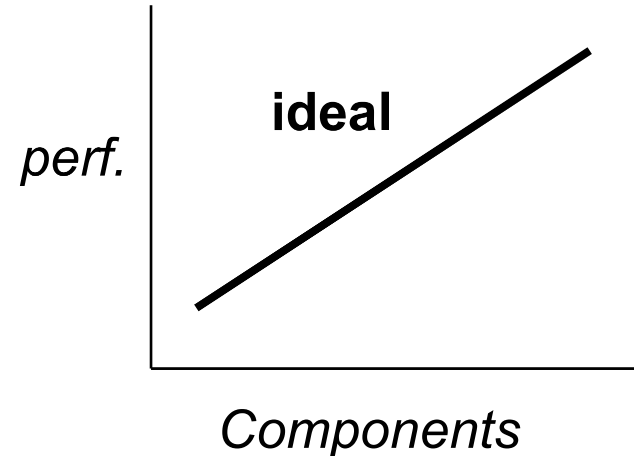
Parallel Data Processing

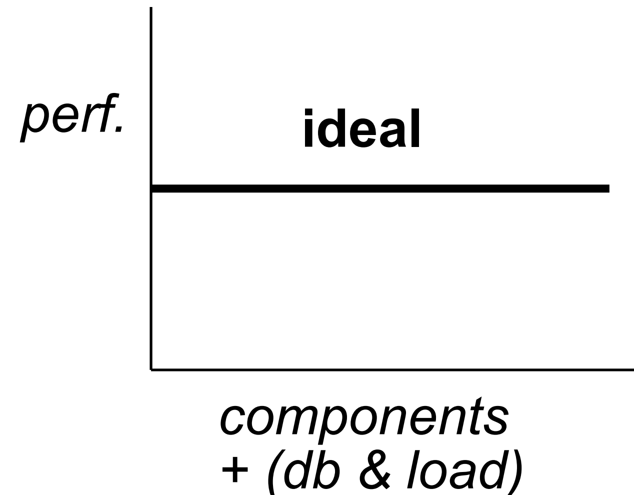
The solution to big data processing!

- 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 OLTP loads
    - Low response time for OLAP queries
  - *High availability* and reliability through data replication
  - *Extensibility* of the architecture

Extensibility : ideal goals

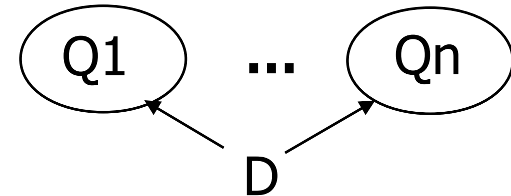
- *Linear increase* in performance for a constant database size and load, and proportional increase of the system components (CPU, memory, disk)
- *Sustained performance* for a linear increase of database size and load, and proportional increase of components

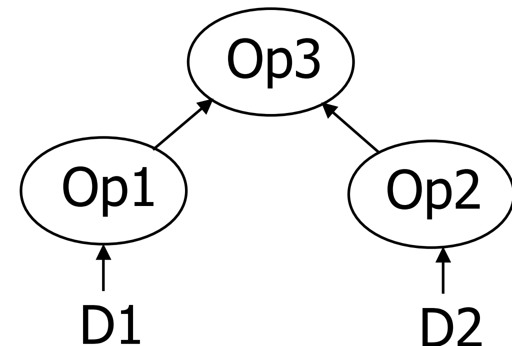


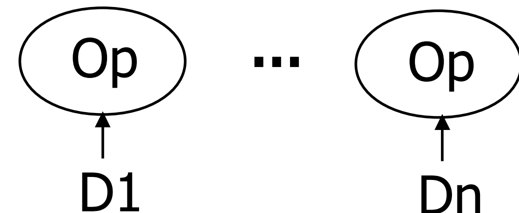


Data-based Parallelism

- 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







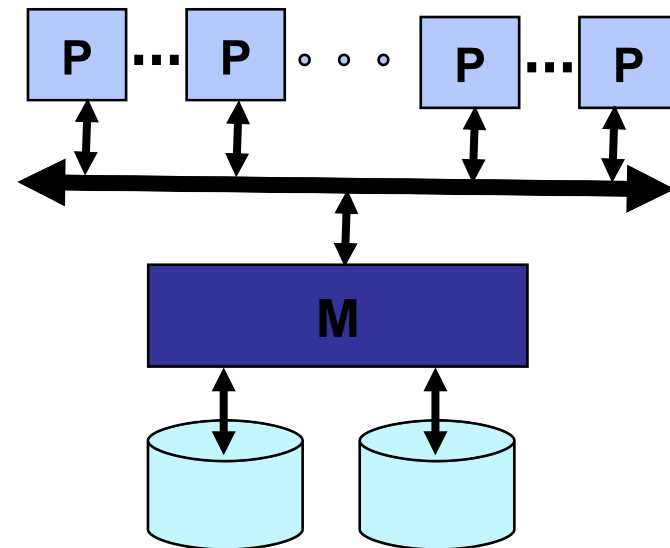
Parallel Architectures

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

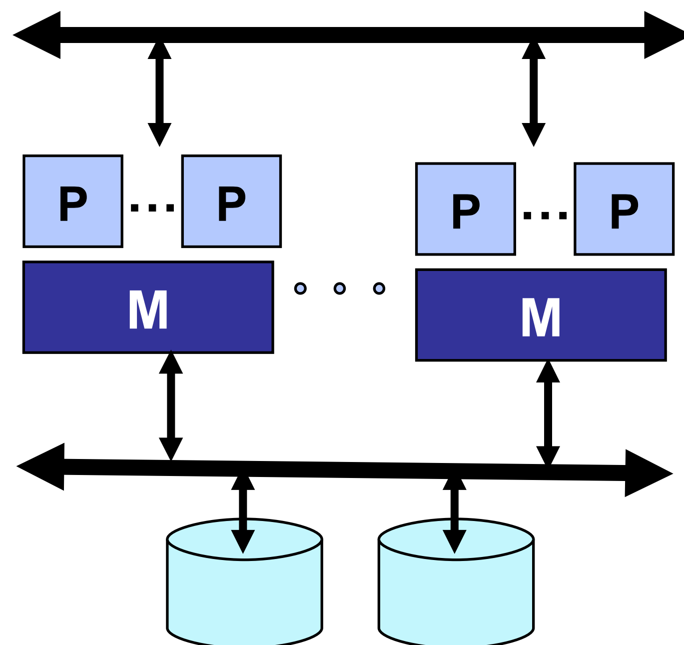
\- Complex interconnect limits extensibility, cost

For write-intensive workloads, expensive for big data

Shared-disk (SD) 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

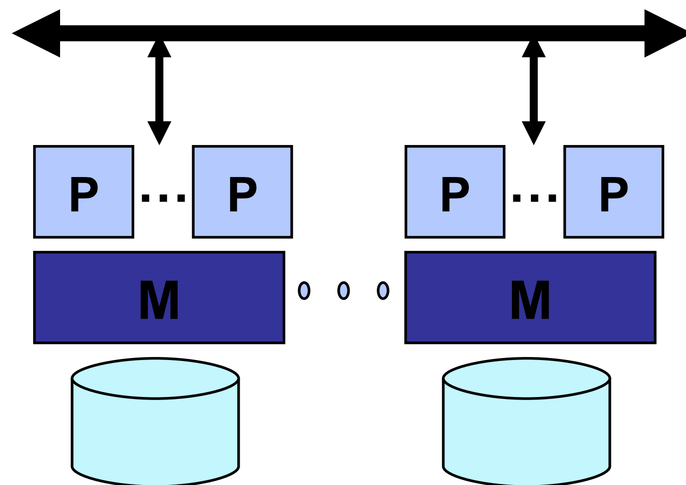
\- Complex DLM, cost

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 match for big data (read intensive)

Parallel Data Management Techniques

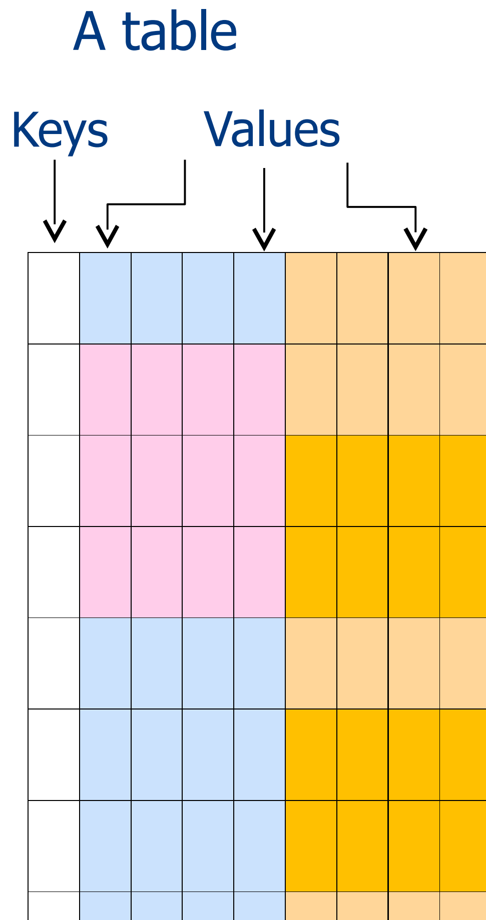
A Simple Model for Parallel Data

- 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

Design Considerations

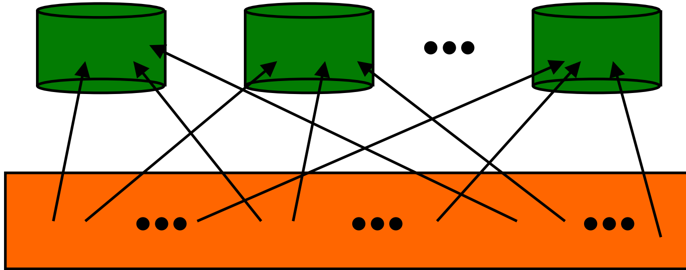
- Big datasets
  - Data partitioning and indexing
    - Problem with skewed data distributions
  - Disk is very slow (10K times slower than RAM)
    - Exploit RAM data structures and compression
    - Exploit flash memory (read 10 times faster than disk)
- Query parallelization and optimization
  - Automatic if the query language is declarative (e.g. SQL)
  - Parallel algorithms for algebraic operators
    - Select is easy, Join is difficult
  - Programmer-assisted otherwise (e.g. MapReduce)

Data Partitioning



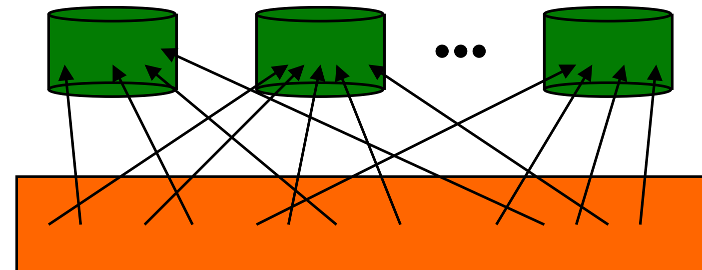
- Vertical
  - Base Basis for column stores (e.g. MonetDB, Vertica): efficient for OLAP queries
  - Easy to compress, e.g. using Bloom filters
- Horizontal (sharding)
  - Shards can be stored (and replicated) at different nodes

Sharding Schemes



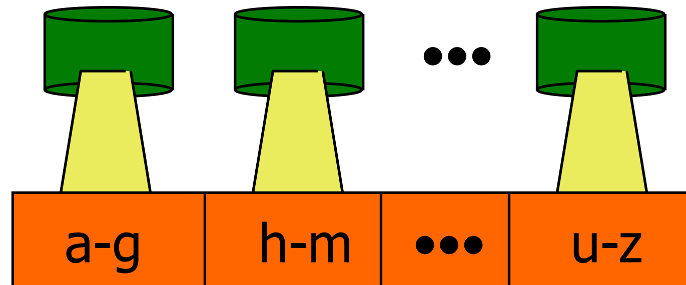
Round-Robin

- *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

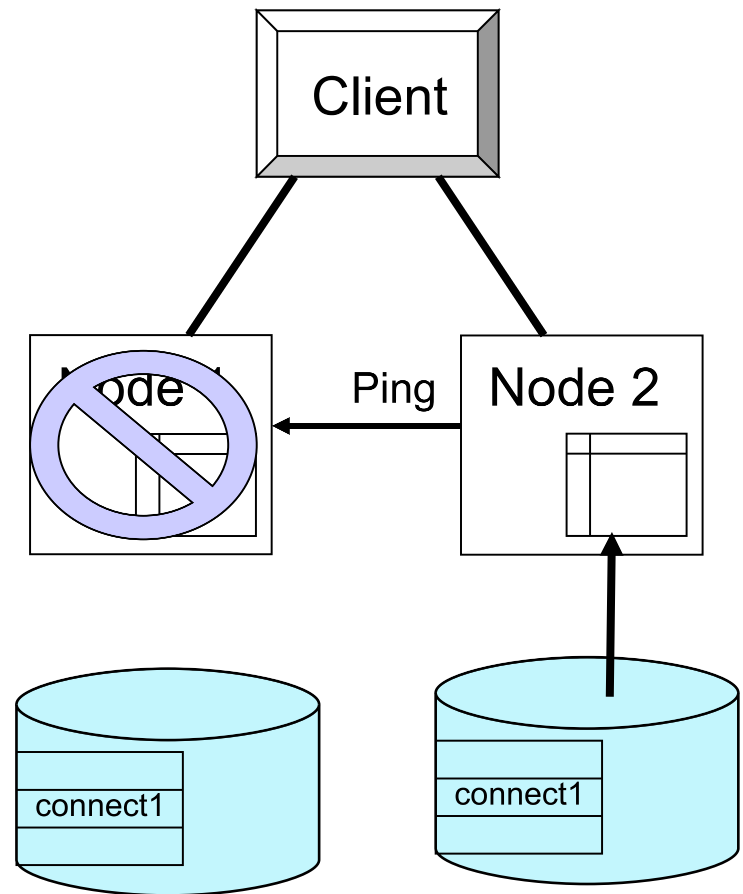
- (k,v) to node that holds k's interval
  - exact-match and range queries
  - deals with skew

Indexing

- Can be supported by special tables with rows of the form: *<attribute, list of keys>* pairs
  - Ex. <att-value, (doc-id:id1, doc-id:id2, doc-id:id10)>
  - 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

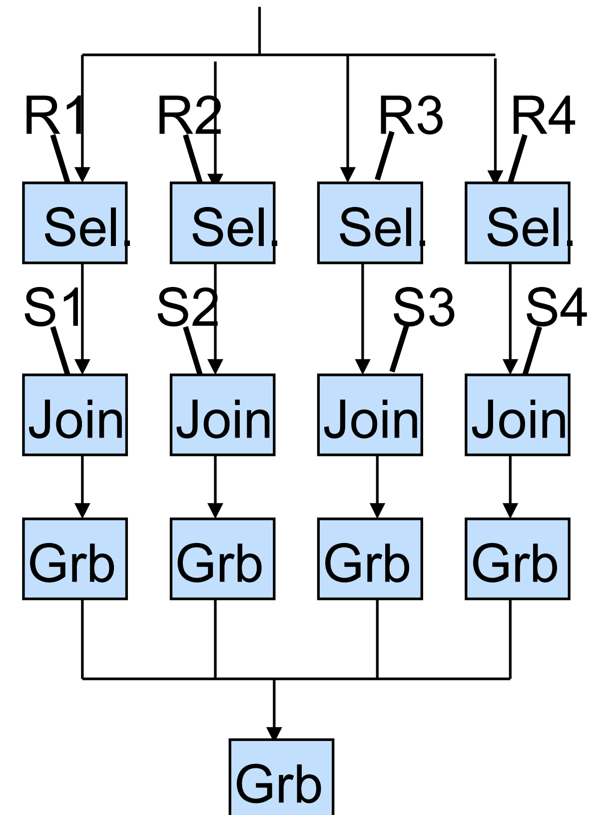
- Replication
  - The basis for fault-tolerance and availability
  - Have several copies of each shard
- Failover
  - On a node failure, another node detects and recovers the node’s tasks



Parallel Query Processing

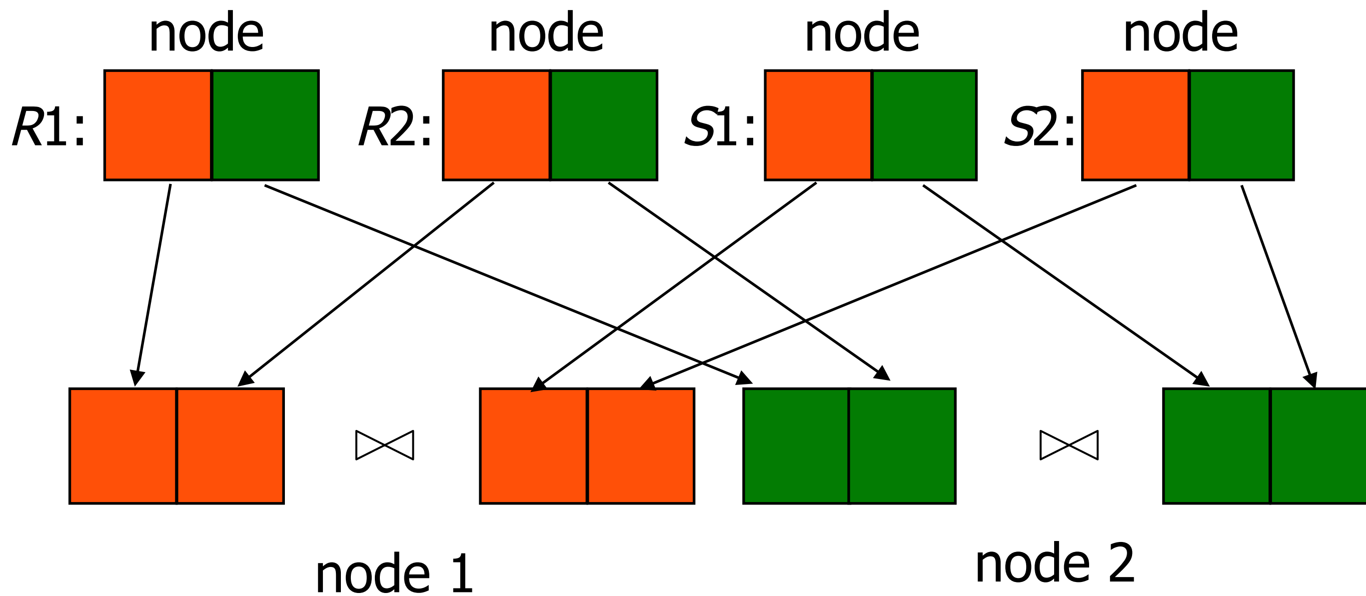
1. Query parallelization
   - Produces an optimized parallel execution plan, with operators
   - Based on partitioning, replication, indexing

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



Parallel Hash Join Algorithm

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



$$R \text{ join } S = \bigcup_{i=1}^{p} Ri \text{ join } Si$$

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, Teradata

Main products

Vendor	Product	Architecture	Remarks
EMC	GreenPlum	SN	Hybrid SQL/MapReduce, based on PostgreSQL
HP	Vertica	SN	Column-based
IBM	DB2 Pure Scale DB2 Database Partitioning Feature (DPF)	SD SN	AIX on SP Linux on cluster
Microsoft	SQL Server SQL Server Parallel Data Warehouse	SD SN	Windows only Acquisition of Netezza
Oracle	Real Application Cluster Exadata Database machine MySQL	SD SD SN	Portability OSS on linux cluster
ParAccel	ParAccel Analytic Database	SN	Column-based
Teradata	Teradata Database Aster	SN with Bynet SN	Unix and Windows Hybrid SQL/MapReduce and row/column

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 systems: Amazon SimpleDB, Google Base,

Google File System (GFS)

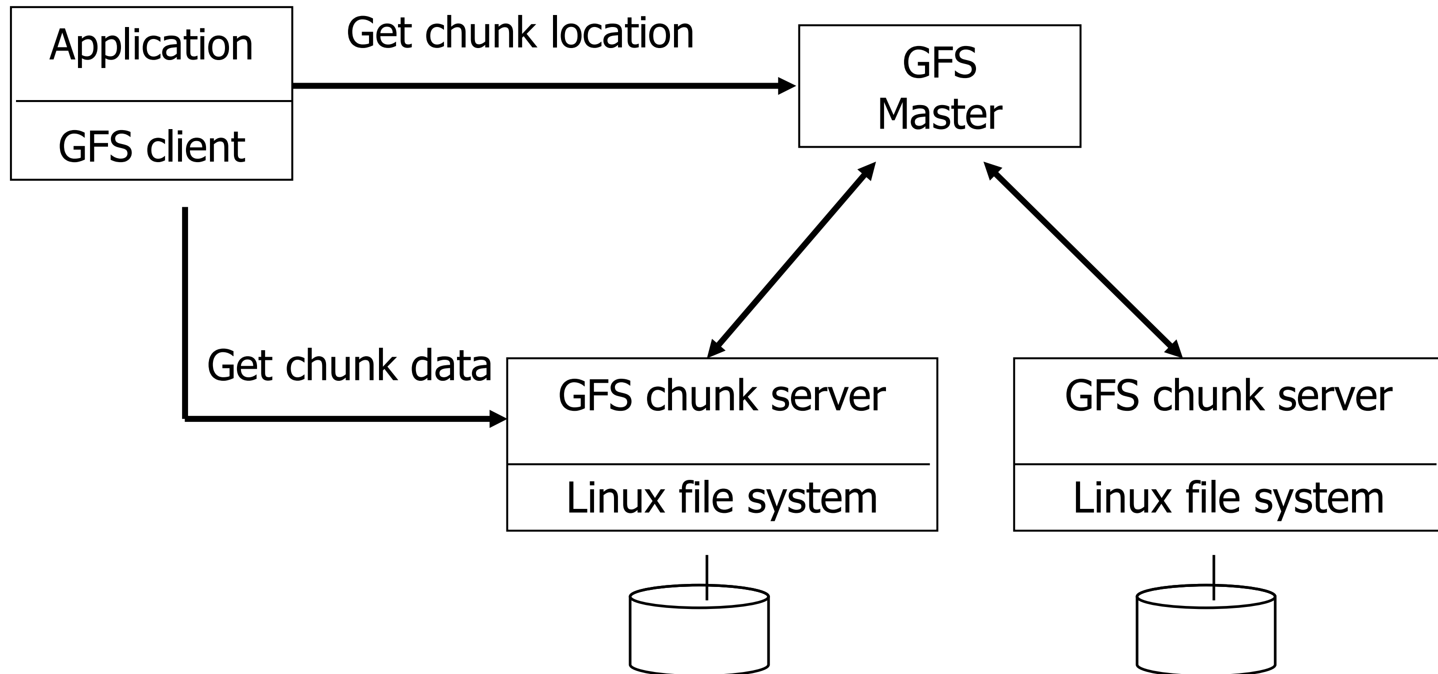
- Used by many Google applications
  - Search engine, Bigtable, Mapreduce, etc.
- The basis for Hadoop HDFS (Apache & Yahoo)
- Optimized for specific needs
  - Shared-nothing cluster of thousand nodes, built from inexpensive harware => node failure is the norm!
  - Very large files, of typically several 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

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 Systems

NOSQL (Not Only SQL): definition

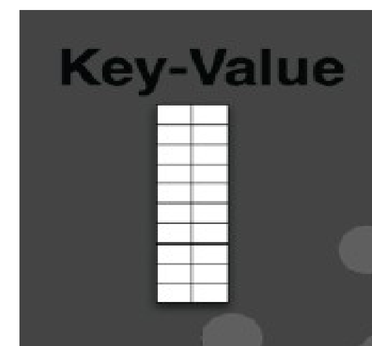
- Specific DBMS: for web-based data
  - Specialized data model
    - Key-value, table, document, graph
  - Trade relational DBMS properties
    - Full SQL, ACID transactions, data independence
  - For
    - Simplicity (schema, basic API)
    - Scalability and performance
    - Flexibility for the programmer (integration with programming language)
- NB: SQL is just a language and has nothing to do with the story

NoSQL Approaches

- Characterized by the data model, in increasing order of complexity:
  1. key-value: Amazon DynamoDB and SimpleDB
  2. big table: Google Bigtable
  3. document: 10gen MongoDB
  4. graph: Neo4J

- What about object DBMS or XML DBMS?
  - Were there much before NoSQL
  - Sometimes presented as NoSQL
  - But not really scalable

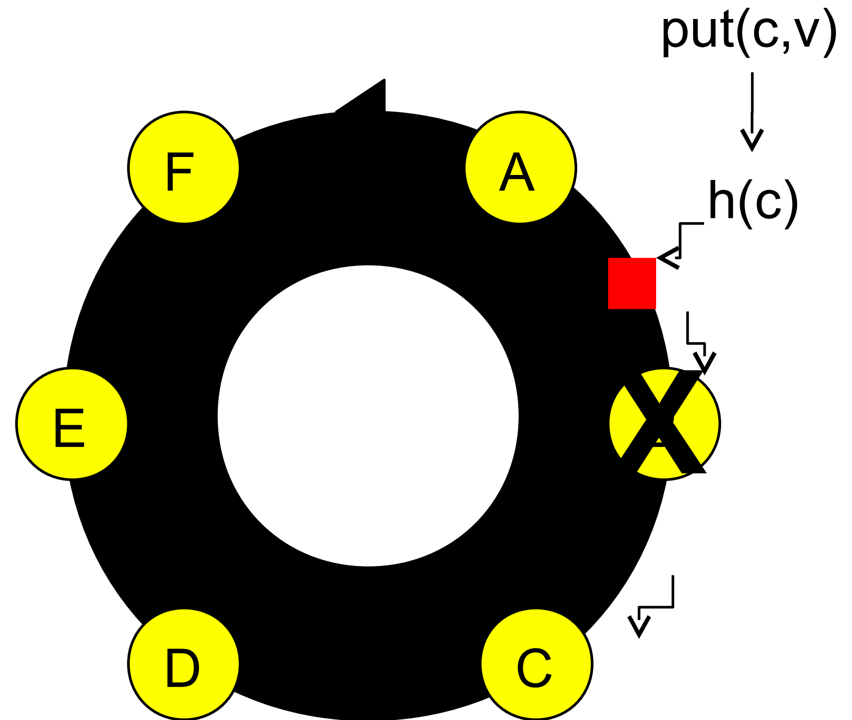
Key-value store: DynamoDB



- The basis for many systems
  - Cassandra, Voldemort
- Simple (key, value) data model
  - Key = unique id
  - Value = a small object (< 1 Mbyte)
- Simple queries
  - Put (key, value)
  - Get (key)
- Replication and eventual consistency
  - If no more updates, the replicas get mutually consistent
- No security
  - Assumes the environment is secured (cloud)
- High availability, performance and scalability using

DynamoDB – consistent hashing

- Hash-based partitioning
- The interval of hash values is treated as a ring
  - Ex. Node B is resp. for interval [A,B]
- Advantage: if a node fails, its successor takes over its data
  - No impact on other nodes
- Data is replicated on



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,

A Bigtable Row

Row unique id   Column family   Column key

Row key	Contents:	Anchor:	Language:
"com.google.www"	"<html>...<\html>" t1 "<html>...<\html>" t5	inria.fr "google.com" t2 "Google" t3 uwaterloo.ca "google.com" t4	"english" t1

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

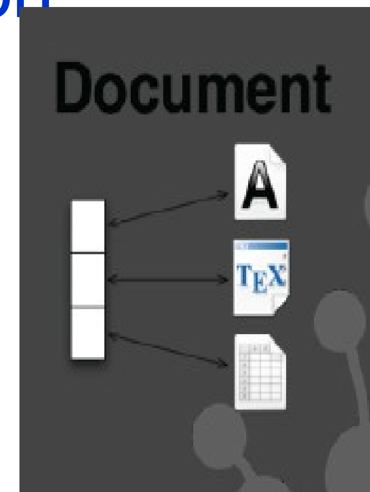
- No such thing as SQL
- Basic API for defining and manipulating tables, within a programming language such as C++
  - No impedance mismatch
  - Various operators to write and update values, and to iterate over subsets of data, produced by a scan operator
  - Various ways to restrict the rows, columns 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

Document DBMS: MongoDB

- Objectives: performance and scalability as in (key, value) stores, but with typed
  - A document is a collection of (key, typed value) with a unique key (generated by MongoDB)
- Data model and query language based on the BSON (Binary JSON) format
- No schema, no join, no complex transaction
- Sharding, replication and failover
- Secondary indices
- Integration with MapReduce



MongoDB – document (post) example

{ _id : ObjectId("4e77bb3b8a3e000000004f7a"),
when : Date("2012-09-19T02:10:11.3Z"),
author : "alex",
title : "No Free Lunch",
text : "This is the text of the post. It could be very long.",
tags : ["business", "ramblings"],
votes : 5, voters : ["jane", "joe", "spencer", "phyllis", "li"],
comments :
[{ **who** : "jane",
when : Date("2012-09-19T04:00:10.112Z"),
comment : "I agree." },
{ **who** : "meghan",
when : Date("2012-09-20T14:36:06.958Z"),
comment : "You must be joking. etc etc ..." }
] }

Generated by MongoDB

Arrays

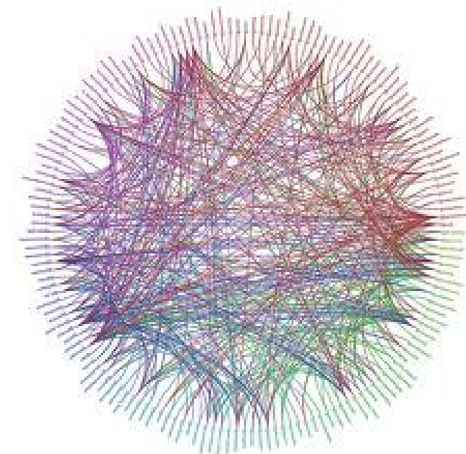
Arrays of Documents

MongoDB – query language

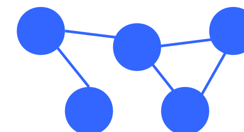
- Expression of the form
  - db.nomBD.fonction (JSON expression)
- Update examples
  - db.posts.insert({author:'alex', title:'No Free Lunch'})
  - db.posts.update({author:'alex', {$set:{age:30}}})
  - db.posts.update({author:'alex', {$push:{tags:'music'}}})
- Select examples
  - db.posts.find({author:"alex"})
    - All posts from Alex
  - db.posts.find({comments.who:"jane"})
    - All posts commented by Jane

Graph DBMS: Neo4J

- Applications with very big graphs
  - Billions of nodes and links
  - Social networks, hypertext documents, linked open data, etc.



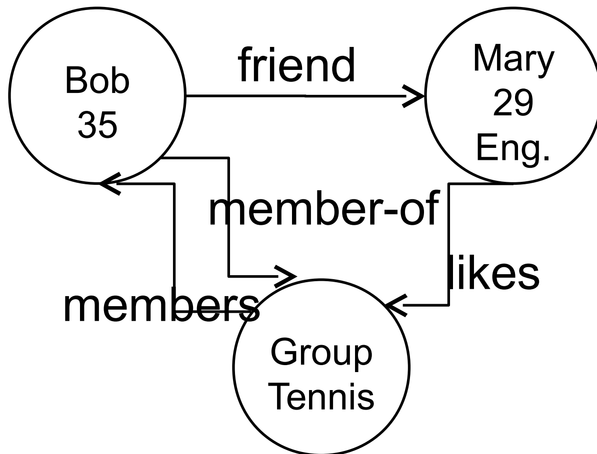
- Direct support of graphs
  - Data model, API, query language
  - Implemented by linked lists on disk
  - Optimized for graph processing
  - Transactions



- Implemented on SN cluster
  - Asymmetric replication
  - Graph partitioning

Neo4J – data model

- Nodes
- Links between nodes
- Properties on nodes and links



Ex. of Neo transaction

```
NeoService neo = ... // factory
Transaction tx = neo.beginTx();
Node n1 = neo.CreateNode();
n1.setProperty("name", "Bob");
n1.setProperty("age", 35);
Node n2 = neo.createNode();
n2.setProperty("name", "Mary");
n2.setProperty("age", 29);
```

Neo4J - languages

- Java API (navigational)
- Cypher query language
  - Queries and updates with graph traversals
- Support of SparQL for RDF data

Ex. of Cypher query that returns the (indirect) friends of Bob whose name starts with "M"

```
START
bob=node:node_auto_index
(name = 'Bob')

MATCH bob-[:friend]->()-
[:friend]->follower

WHERE follower.name =~ 'M.*'

RETURN bob, follower.name
```

Main NoSQL Systems

Vendor	Product	Category	Comments
Amazon	Dynamo SimpleDB	KV	Proprietary
Apache	Cassandra Accumulo	KV Big table	Open source, Origin Facebook Open source, Origin NSA
Armadillo S.A.	Armadillo	Document	Proprietary, security
Google	Bigtable Pregel	Big table Graph	Proprietary, patents
Hadoop	Hbase	Big table	Open source, Orig. Yahoo
LinkedIn	Voldemort	KV	Open source
10gen	MongoDB	Document	Open source
Neo4J.org	Neo4J	Graph	Open source
Sparcity	DEX	Graph	Proprietary, Orig. UPC, Barcelone
Ubuntu	CouchDB	Document	Open source

NoSQL versus Relational

- The techniques are not new
  - Database machines, SN cluster
  - But very large scale
- Pros NoSQL
  - Scalability, performance
  - APIs suitable for programming
- Pros Relational
  - Strong consistency, transactions
  - Standard SQL (many tools)
- Towards NoSQL/Relational hybrids?
  - Google F1: "combines the scalability, fault tolerance, transparent sharding, and cost benefits so far available only in NoSQL systems with the usability, familiarity,

MapReduce

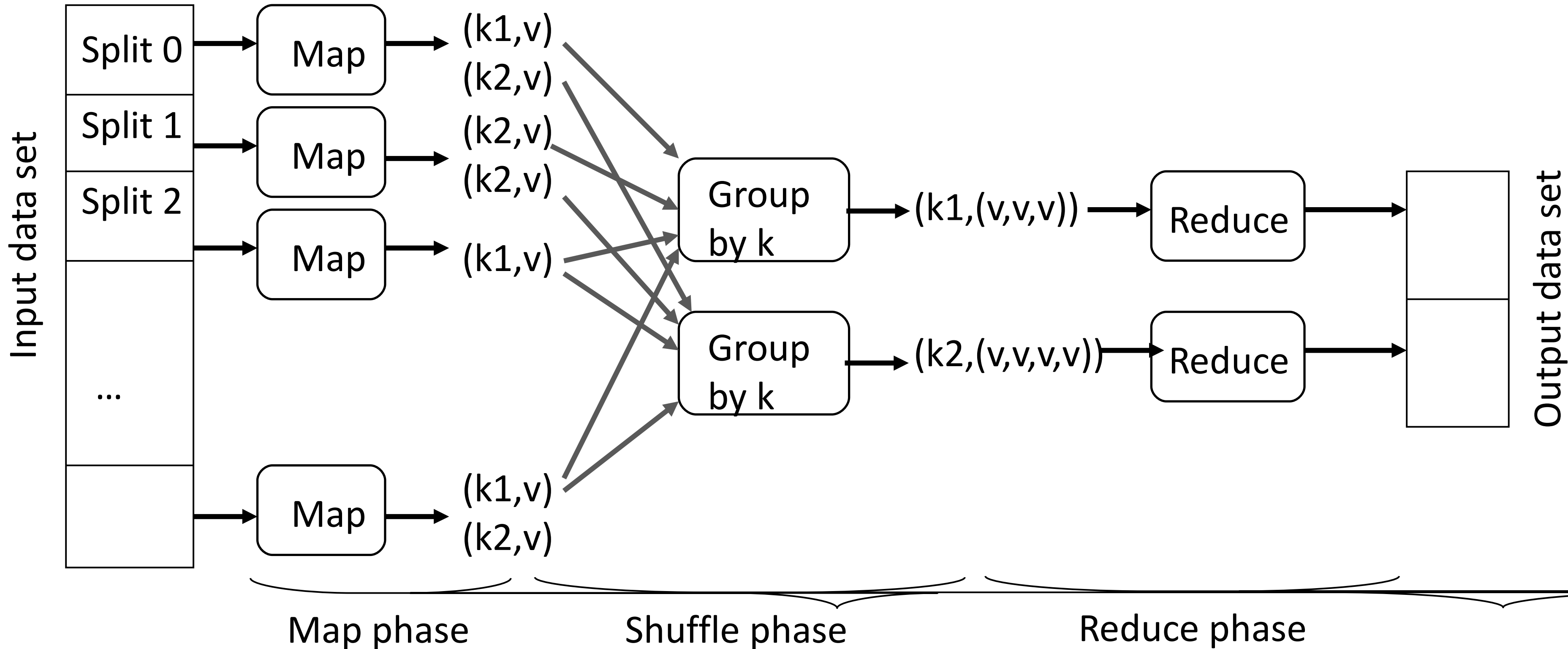
MapReduce Framework

- Invented by 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

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)
```

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

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

MapReduce Performance

- Much room for improvement (see MapReduce workshops)
  - Map phase
    - Minimize I/0 cost using indices (Hadoop++)
  - Shuffle phase
    - Minimize data transfers by partitioning data on the same intermediate key
    - Current work in Zenith
  - Reduce phase
    - Exploit fine-grain parallelism of Reduce tasks
    - Current work in Zenith

Conclusion

Research Directions

- Basic techniques are not new
  - Parallel database machines, shared-nothing cluster
  - Data partitioning, replication, indexing, parallel hash join, etc.
  - But need to scale up
- Much room for research and innovation
  - MapReduce extensions
  - Dynamic workload-based partitioning
  - Data-oriented scientific workflows
  - Uncertain data mining
  - Content-based IR
  - Data privacy