Distributed Data Management in 2020?

Patrick Valduriez

INRIA & LIRMM

Montpellier, France

DEXA, Toulouse, August 30, 2011

Basis for this Talk

T. Özsu and P. Valduriez. *Principles of Distributed Database Systems – Third Edition*. Springer, 2011

T. Özsu, P. Valduriez, S. Abiteboul, B. Kemme, R. Jiménez-Peris, and B. Chin Ooi. Distributed data management in 2020? ICDE Panel, April 2011

And some cool time at UCSB to get ready, summer 2011, with A. El Abbadi and D. Agrawal's group

Distributed Data Management: brief history

1980’s: client server and distributed relational database technology

- all commercial DBMSs today are distributed.

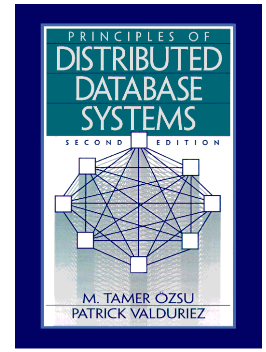
1990’s: maturation of client-server technology and parallel DBMS, introduction of object-orientation

2000’s: data integration, database clusters, Web and XML data management, P2P systems, stream data management, and cloud data management

Principles of Distributed Database Systems

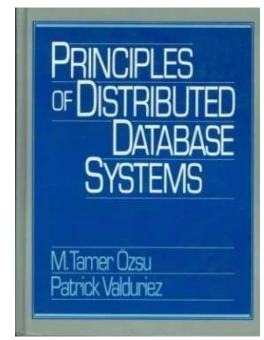
First edition, Prentice Hall, 1991, 560 pages

- Relational data distribution: principles and techniques



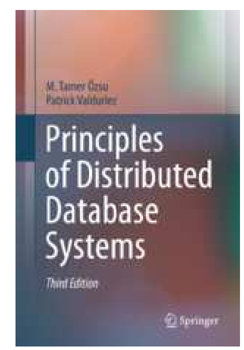
Second edition, Pearson, 1999, 660 pages

- Client-server, parallel DB, object systems



Third edition, Springer, 2011, 850 pages

- Replication, data integration, MDB QP, DB clusters, P2P, Web and XML, data streams, cloud



Main Question

Now, the question is:

What is likely to happen in the next decade?

Or to put it differently, if there were to be a fourth edition of our book in 2020, *what would it be? what would be new?*

Optional: how many pages for the fourth edition?

Observations wrt. the Last 20 Years

1. The fundamental principles of distributed data management have hold, and distributed data management can be still characterized on the three dimensions of the earlier editions
   - Distribution, heterogeneity, autonomy
2. What has changed much is the scale of the dimensions: very large scale distribution (cluster, P2P, web and cloud); very high heterogeneity (web); very high autonomy (web and P2P)
3. New techniques and algorithms could be presented as extensions of earlier material, using relational concepts

Acceleration of Changes

New data-intensive applications

- E.g. social networks, web data analytics, scientific apps, data streams

With different kinds of data

- Very large, complex, unstructured, semi-structured, heterogeneous, etc. and highly distributed

New data management technologies

- New file systems: GFS, HDFS, …
- NOSQL DBMS and key-value stores: Amazon SimpleDB, Amazon Dynamo, Google Base, Google Bigtable, Yahoo Pnuts, UCSB ElasTraS, etc.
- New parallel programming frameworks: MapReduce, Pregel
- And new architectures, e.g. MapReduce/GFS

Key Questions

1. What are the fundamental principles behind the emerging solutions?
2. Is there any generic architectural model to explain those principles?
3. Do we need new foundations to look at data distribution?

Outline of the Talk

Principles of distributed data management

New challenges for distributed data management

- Cloud computing
- e-Science

Emerging solutions

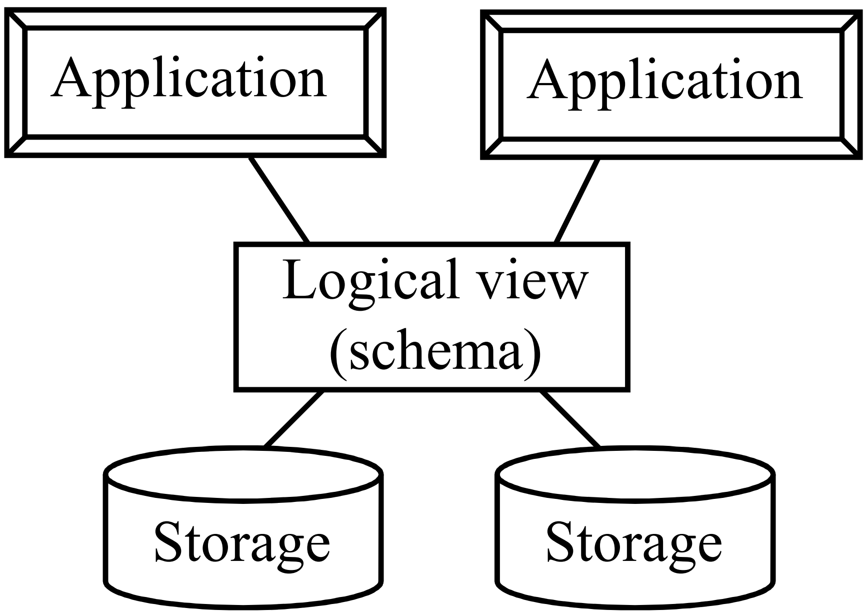
Conclusion

Note: some topics are subject to much POLEMICS

Principles of Distributed Data Management

Fundamental Principle: Data Independence

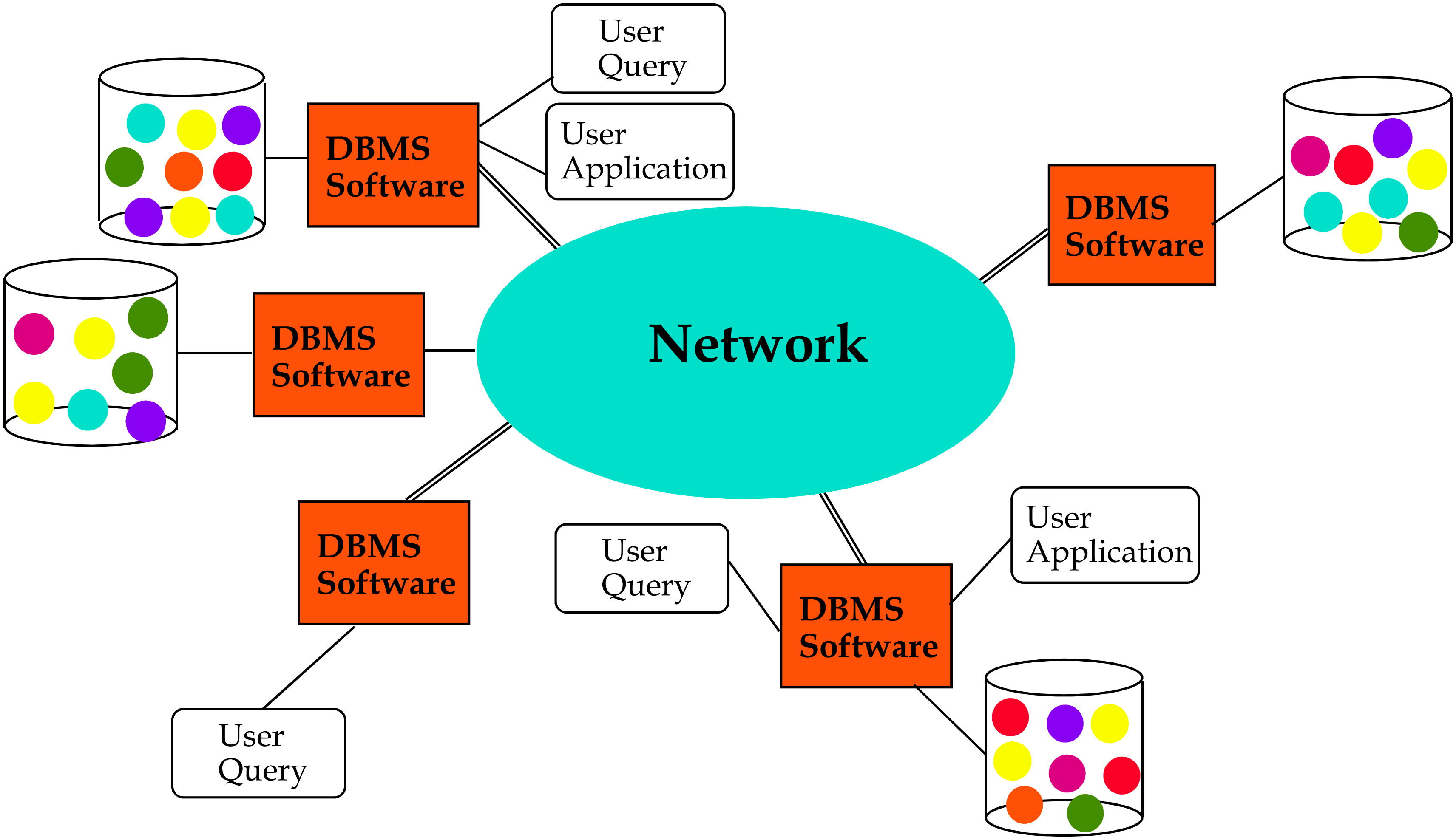
Enables hiding implementation details



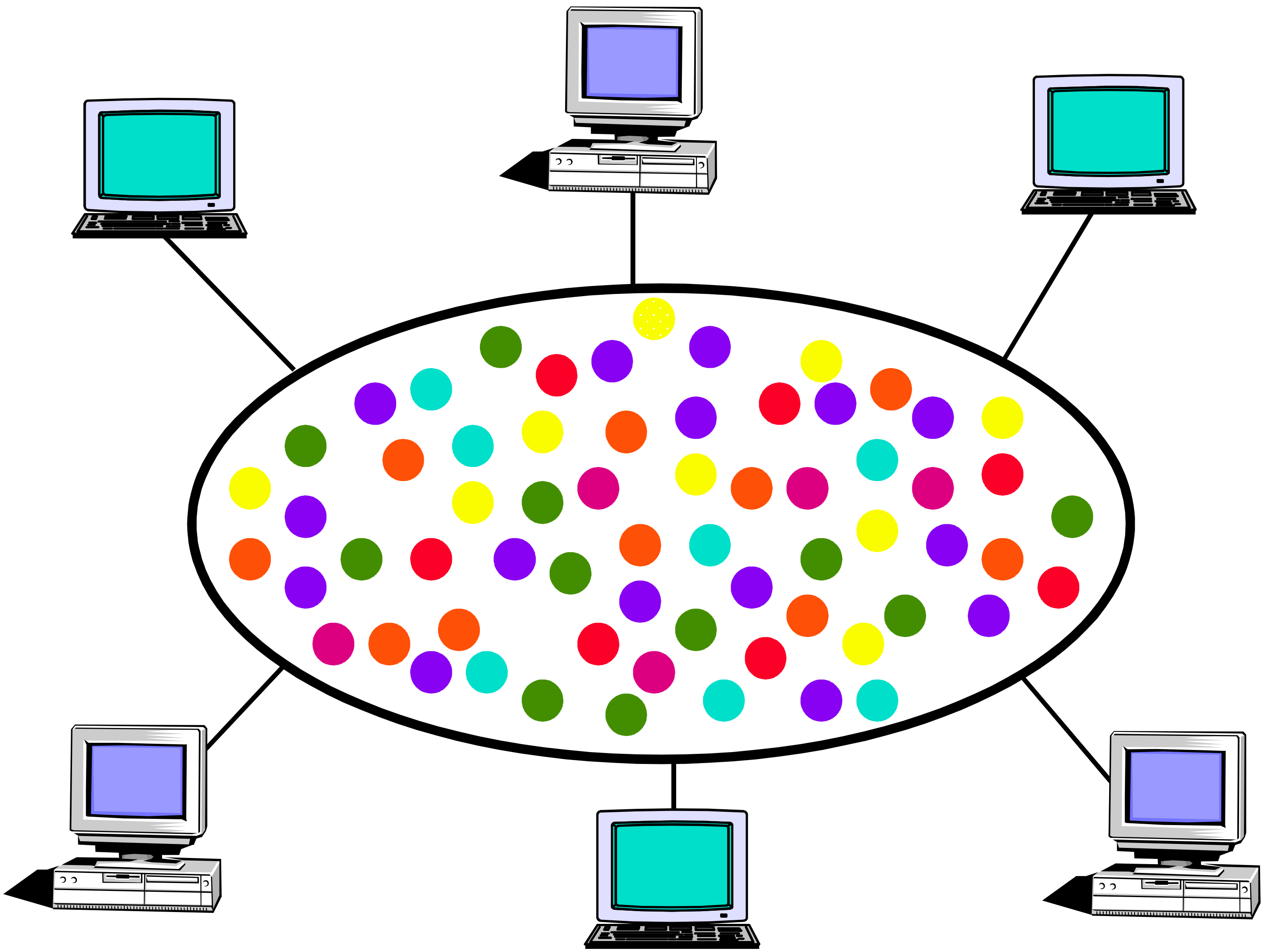
Provision for high-level services

- Schema
- Queries (SQL, XQuery)
- Automatic optimization
- Transactions
- Consistency
- Access control
- …

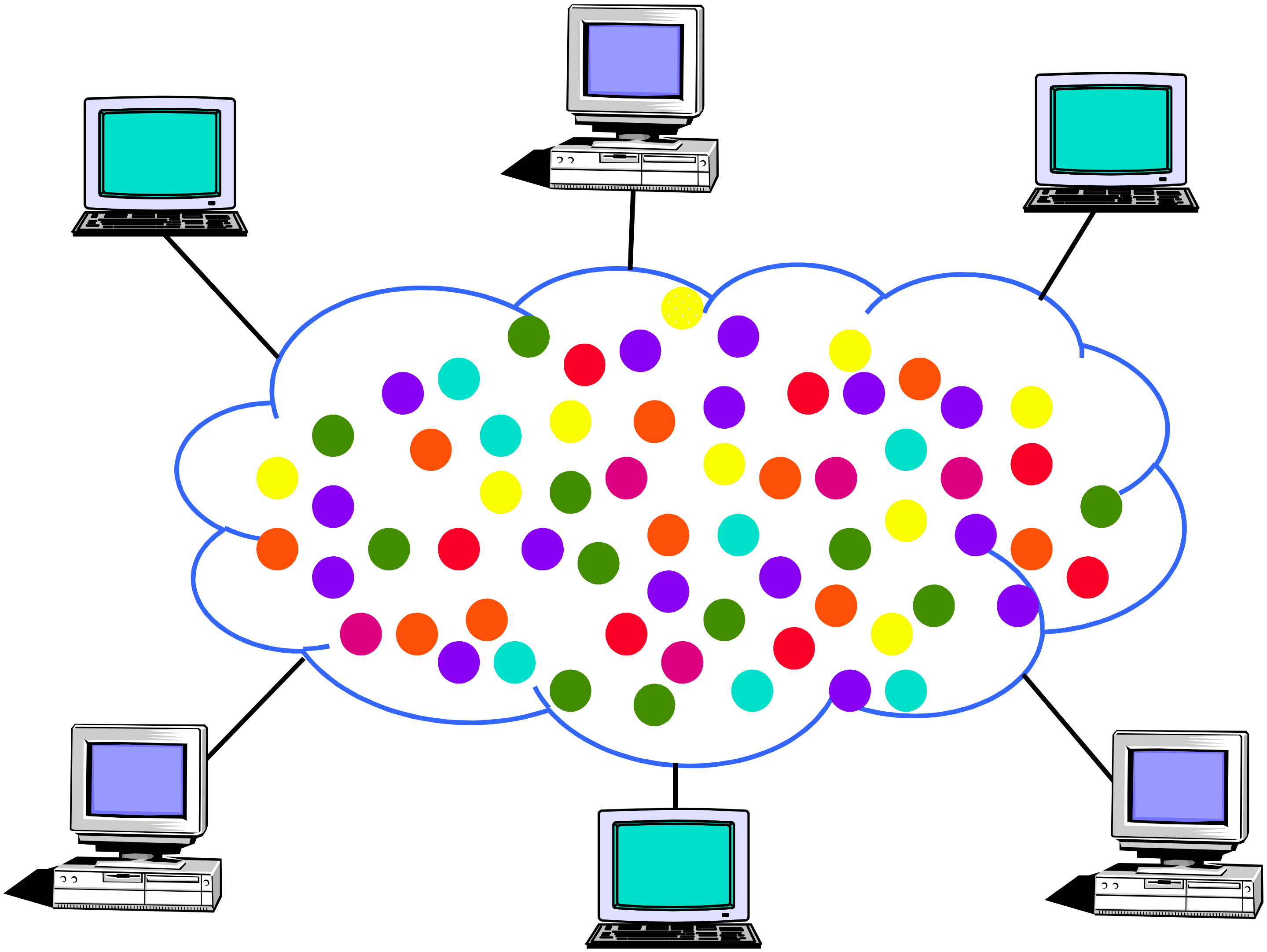
Distributed Database – System View



Distributed Database – User View (1991)



Distributed Database – User View (2011)



Principles of Distributed Data Management

Set-oriented data (relational tables)

- Fragmentation: the basis for distributed and parallel processing

High-level languages (calculus, algebra)

- The basis for *data independence*
- Programmer productivity, automatic optimization and tuning

Data consistency

- ACID transactions: atomicity, integrity control, concurrency control, reliability

Data semantics (schemas, integrity constraints, taxonomies, folksonomies, ontologies, …)

- To improve information retrieval and automate data integration

Horizontal Fragmentation

PROJ$_1$: projects with budgets less than $200,000

PROJ$_2$: projects with budgets greater than or equal to $200,000

PROJ

PNO	PNAME	BUDGET	LOC
P1	Instrumentation	150000	Montreal
P2	Database Develop.	135000	New York
P3	CAD/CAM	250000	New York
P4	Maintenance	310000	Paris
P5	CAD/CAM	500000	Boston

PROJ$_1$

PNO	PNAME	BUDGET	LOC
P1	Instrumentation	150000	Montreal
P2	Database Develop.	135000	New York

PROJ$_2$

PNO	PNAME	BUDGET	LOC
P3	CAD/CAM	250000	New York
P4	Maintenance	310000	Paris
P5	CAD/CAM	500000	Boston

Basis for distributed database design, data integration (LAV/GAV), data partitioning in parallel DBMS and key-value stores, etc.

Vertical Fragmentation

$PROJ_1$: information about project budgets

$PROJ_2$: information about project names and locations

PROJ

PNO	PNAME	BUDGET	LOC
P1	Instrumentation	150000	Montreal
P2	Database Develop.	135000	New York
P3	CAD/CAM	250000	New York
P4	Maintenance	310000	Paris
P5	CAD/CAM	500000	Boston

$PROJ_1$

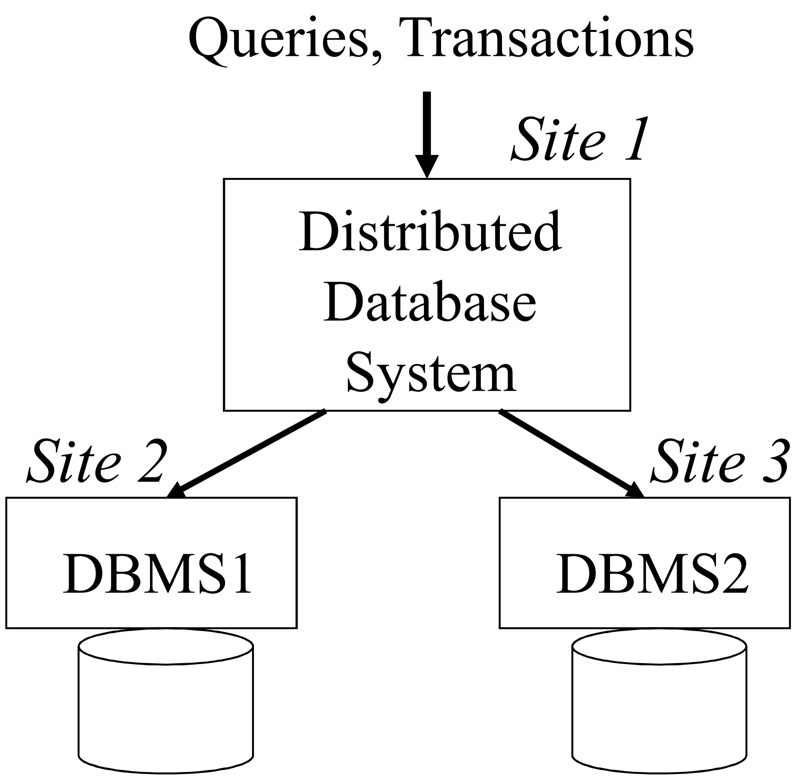
PNO	BUDGET
P1	150000
P2	135000
P3	250000
P4	310000
P5	500000

$PROJ_2$

PNO	PNAME	LOC
P1	Instrumentation	Montreal
P2	Database Develop.	New York
P3	CAD/CAM	New York
P4	Maintenance	Paris
P5	CAD/CAM	Boston

Basis for column-store DBMS

Distributed Database System



Provides distribution transparency

- Global schema
  - Common data descriptions
  - Data placement information
- Centralized admin. through global catalog
- Distributed functions
  - Schema mapping
  - Query processing
  - Transaction management
  - Access control
  - Etc.

DDBS Architectures

Distributed DBMS (DDBMS)

- Homogeneity: same DBMS, same middleware
- P2P components: each node has same functionality
  - Issue query, execute transaction, etc.
- Full DBMS functionality
- Used by Parallel DBMS and (modern) P2P DBMS
- C/S DBMS as a simpler alternative

Multidatabase System (MDBMS)

- Strong heterogeneity and autonomy of data sources (files, databases, XML documents, ..)
- Limited DBMS functionality (queries)
- Used by data integration systems (Mediator/Wrapper)

Scaling up DDBS

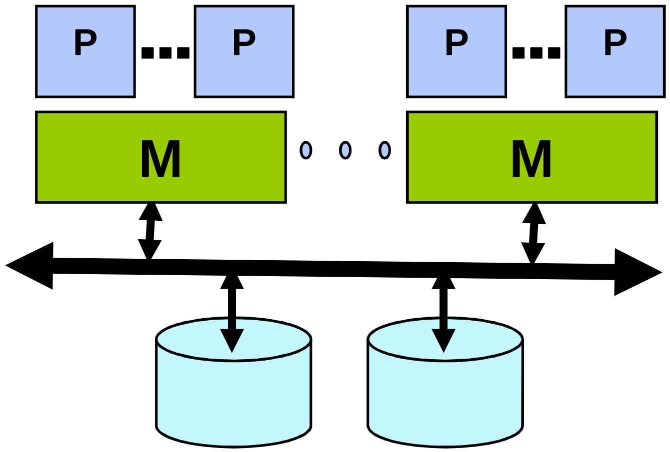
Homogeneity of system components makes it easier to scale up in numbers of nodes

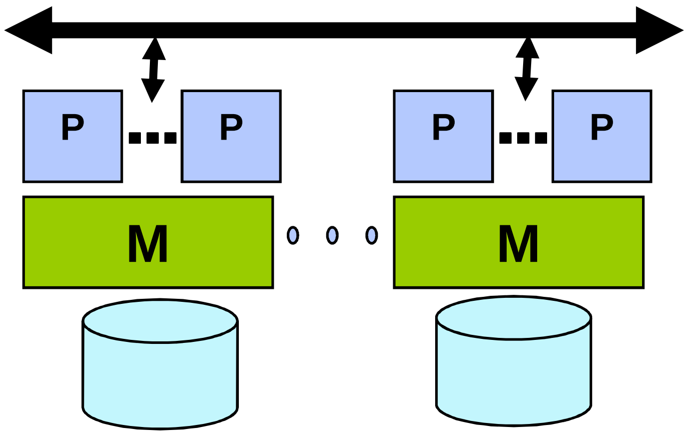
- Thousands in PDBMS, Millions in P2P
- High-performance in local networks

Data source heterogeneity and autonomy makes it hard

- But critical for Web data integration with thousands of data sources
- Solution: restrict functionality (simple read-only queries)

Shared-disk vs Shared-nothing in PDBMS





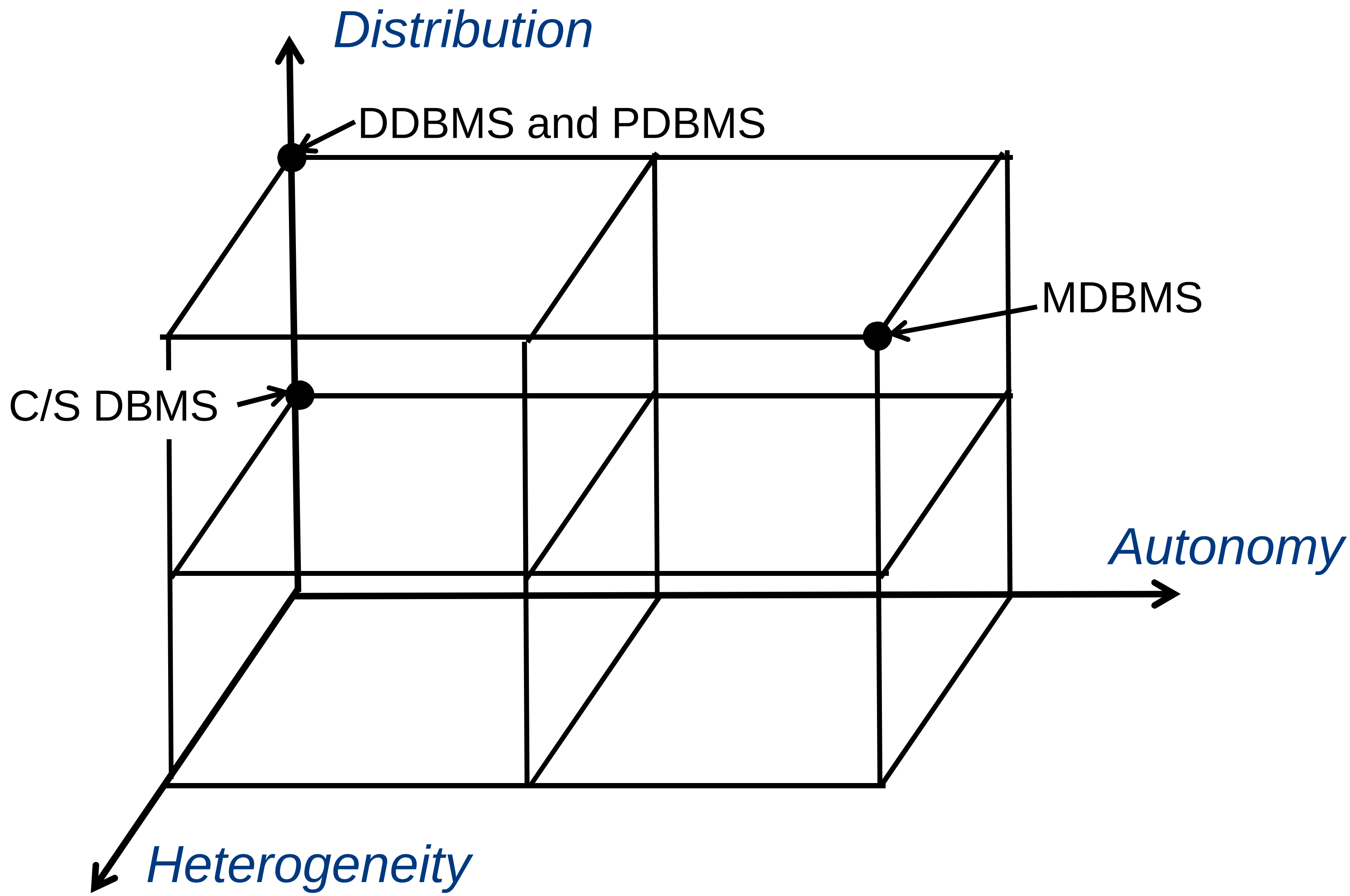
- Requires distributed cache coherency
- Simple for admins
- Can scale well
- Well adapted for OLTP

- Requires complex data partitioning
- Needs 2PC
- Can scale to VLDB
- Well adapted for OLAP

Typical approach separates OLTP and OLAP

- Notable exception: Oracle, using multiversions

Dimensions of the Problem (1991)



New Challenges for Distributed Data Management

New Distributed Data-intensive Applications

Spurred by the pervasiveness of the web, as well as recent advents in high-speed networks, fast commodity hardware

- Sensors, i-appliances, smartcards, multicores, flash memories, etc.

Data is more and more distributed

- **Cloud computing**
- **Scientific applications**
- Personal dataspaces (social networks, webmail, blogs, etc.)
- Computer games, data streams, etc.
- Not to forget corporate apps
  - Need to scale out
  - Need data integration, search engines, etc.

Cloud Computing: a new paradigm?

The vision

- On demand, reliable services provided over the Internet (the "cloud") with easy access to virtually infinite computing, storage and networking resources

Simple and effective!

- Through simple Web interfaces, users can outsource complex tasks
  - Data mgt, system administration, application deployment
- The complexity of managing the infrastructure gets shifted from the users' organization to the cloud provider

Capitalizes on previous computing models

- Web services, utility computing, cluster computing, virtualization, grid computing

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)

Barrier to Entry: Security and Privacy

Current solutions

- **Internal (or private) cloud as opposed to public cloud** : the use of cloud technologies but in a private network behind a firewall
  - Much tighter security
  - But reduced cost advantage because the infrastructure is not shared with other customers (as in public cloud)
  - Compromise: **hybrid cloud** (internal cloud for OLTP + public cloud for OLAP)
- **Virtual private cloud**: Virtual Private Network (VPN) within a public cloud with security services
  - Promise of a similar level of security as an internal cloud and tighter integration with internal cloud security
  - *But such security integration is complex*

Much room for innovation

OLAP vs OLTP in the Cloud

OLAP

- Historical databases of very large sizes (PB), read-intensive
- Relaxed ACID properties
- Shared-nothing clusters of commodity servers cost-effective
- Sensitive data can be hidden (anonymized) in the cloud

OLTP

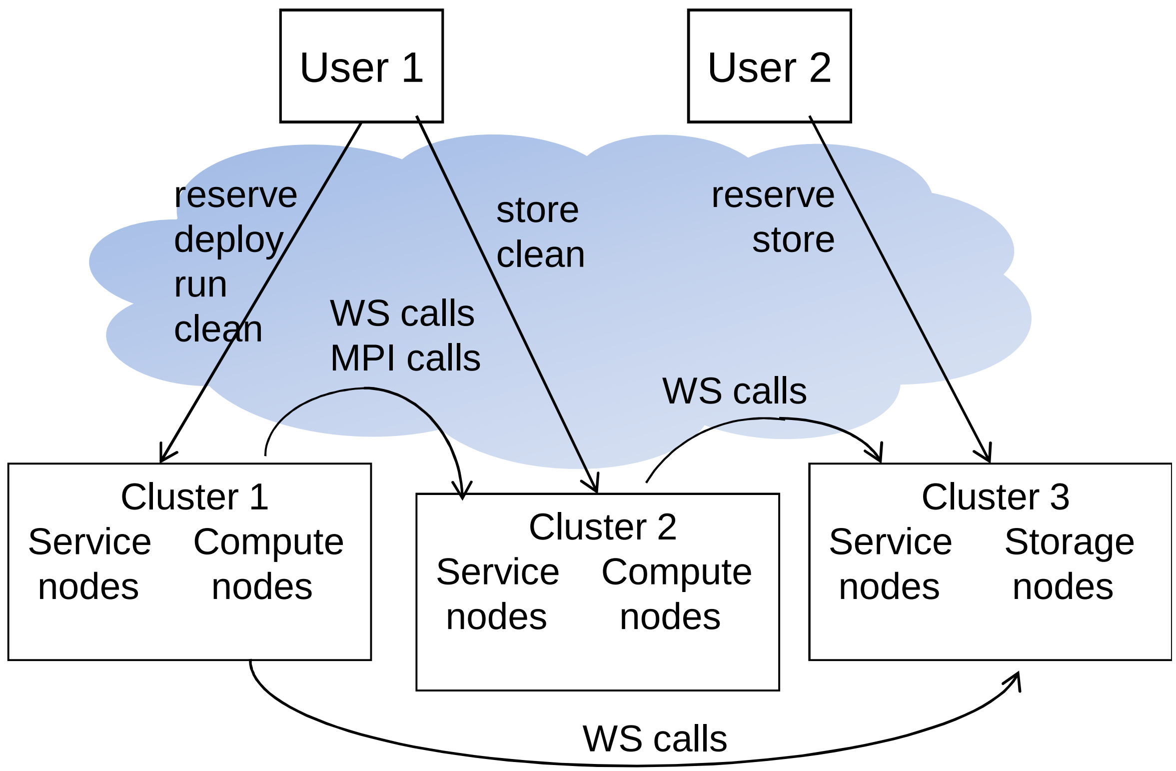
- Operational databases of average sizes (TB), write-intensive
- ACID transactions, strong data protection, response time guarantees
- Shared-disk multiprocessors preferred
  - Notable exception: Tandem NonStopSQL in the 1980s
- Corporate data gets stored at untrusted host

OLAP easier, but OLTP doable

- e.g. UCSB ElasTraS, MS SQL Azure, MIT Relational Cloud

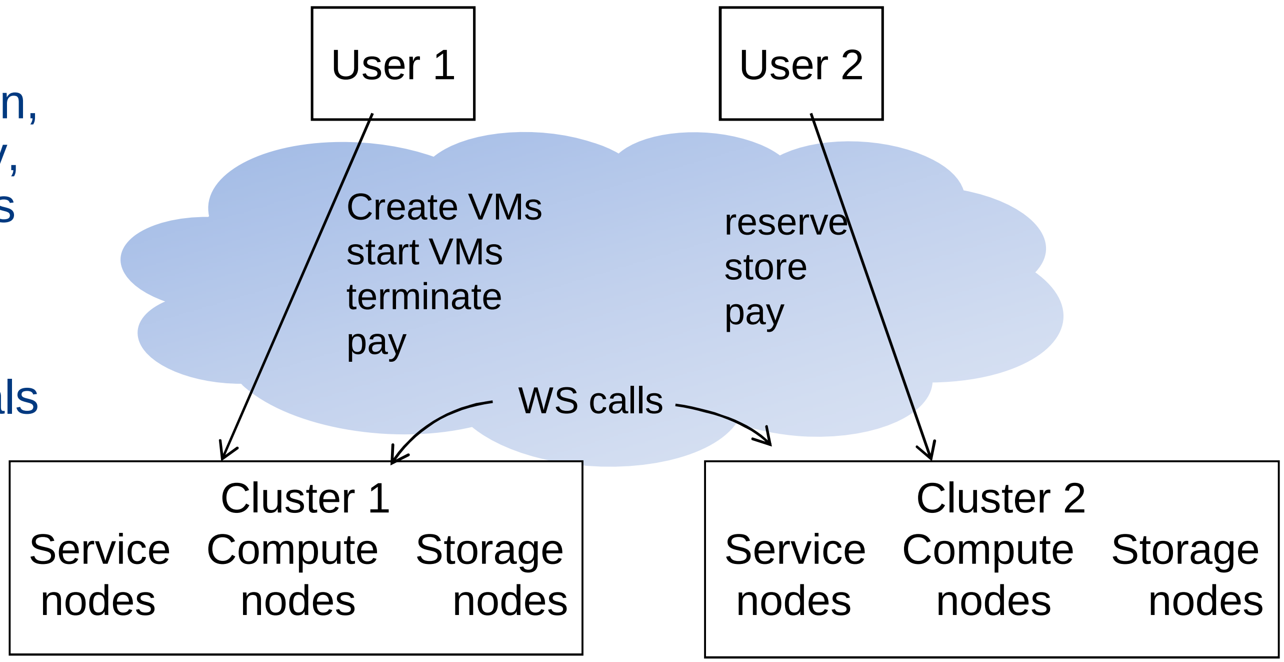
Grid Architecture

- 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 Problem

Cloud data

- Very large (lots of dataspaces, very large collections, multimedia, etc.)
- Complex, unstructured or semi-structured
- Heterogeneous
- Often schemaless but metadata (tags, …)
- Typically append-only (with rare updates)

Cloud users and application developers

- In very high numbers
- With very diverse expertise but very little DBMS expertise

Scientific Applications

Modern science such as agronomy, bio-informatics, physics and environmental science must deal with overwhelming amounts of experimental data produced through empirical observation and simulation

Such data must be processed (cleaned, transformed, analyzed) in all kinds of ways in order to draw new conclusions, prove scientific theories and produce knowledge

Scientific Data – *hard problems*

Massive scale

- Constant progress in scientific observational instruments (e.g. satellites, sensors, large hadron collider) and simulation tools creates a huge data overload.
- For example, climate modeling data are growing so fast that they will lead to collections of hundreds of exabytes expected by 2020

Complexity

- Because of heterogeneous methods used for producing data and the inherently multi-scale nature of many sciences, resulting in data with hundreds (or thousands) of attributes or dimensions, making data analysis very hard

Heterogeneity

- Modern science research is a highly collaborative process, involving scientists from different disciplines (e.g. biologists, soil scientists, and geologists working on an environmental project) and organizations
- Each discipline or organization tends to produce and manage its own data, in specific formats, with its own processes, making data integration very hard

Scientific Data – *common features*

- **Massive scale, complexity and heterogeneity**
- Manipulated through complex, distributed *workflows*
- Important *metadata* about experiments and their provenance
- Heterogeneous schemas and ontologies
- Mostly append-only (with rare updates)

Scientific Data Management Problem

Current solutions

- Typically file-based, application-specific (ad hoc) and low-level
- Deployed in large-scale HPC environments
  - Cluster, grid, cloud

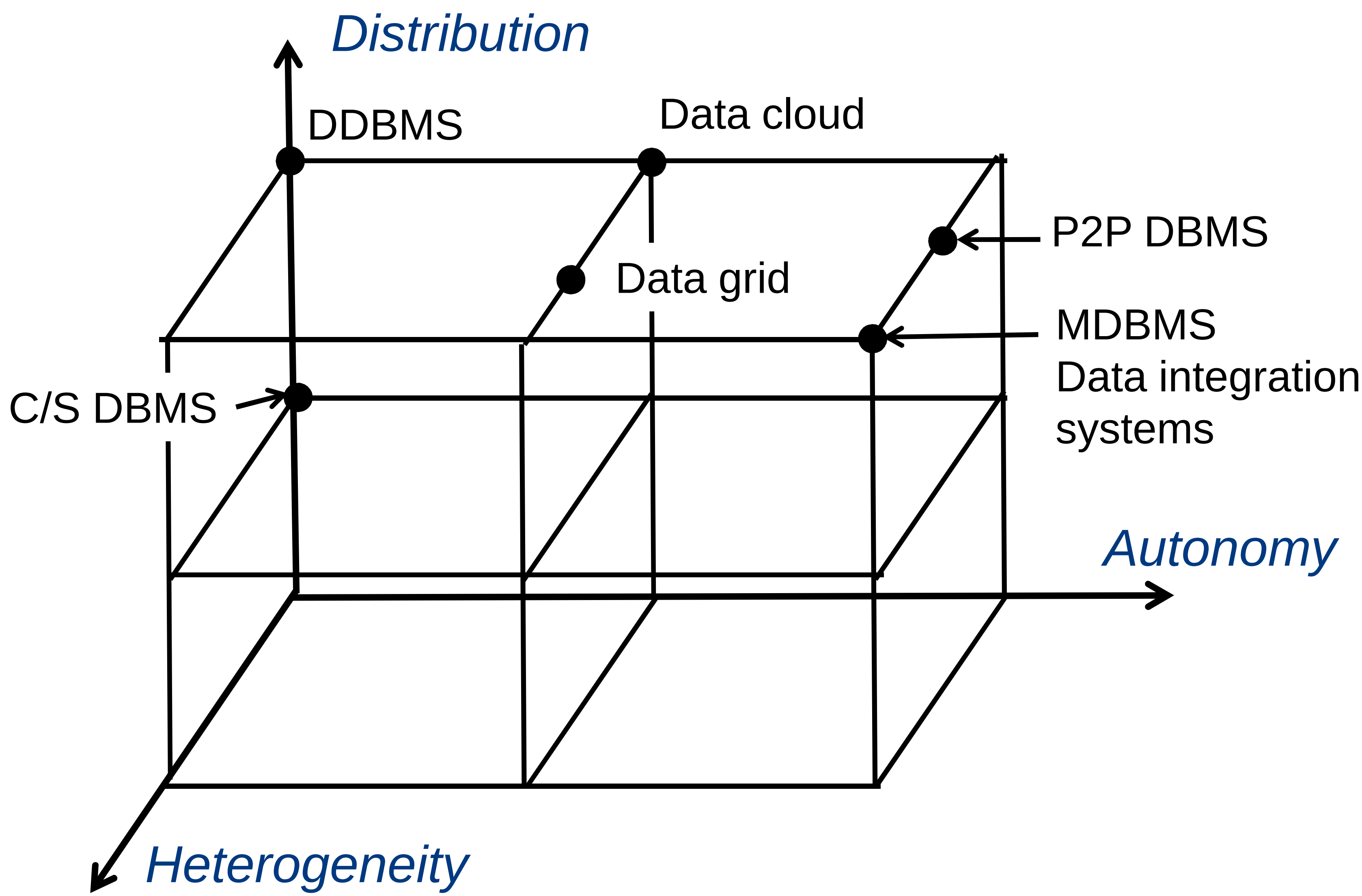
Problem

- Labor-intensive (development, maintenance)
- Cannot scale (hard to optimize disk access)
- Cannot keep pace (the data overload will just make it worse)

"*Scientists are spending most of their time manipulating, organizing, finding and moving data, instead of researching. And it's going to get worse*" (DoE Office of Science Data Management Challenge)

Emerging Solutions

Dimensions of the problem (2011)



Why not RDBMS?

RDBMS all have a distributed and parallel version

- With SQL support for all kinds of data (structured, XML, multimedia, streams, etc.)
- Standard SQL a major argument for adoption by tool vendors (e.g. analytics, business intelligence)

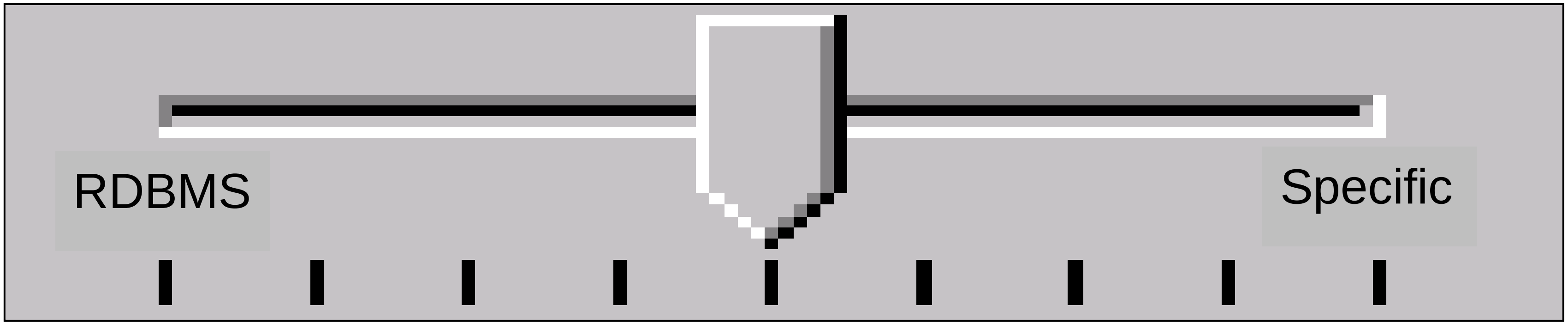
But the “one size fits all” approach has reached the limits

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

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

Generic vs Specific



Emerging solutions trade data independence and consistency for scalability, flexibility and performance

Generic vs Specific

How to provide application-specific optimizations in a generic fashion?

- For instance, to perform scientific data analysis efficiently, scientists typically resort to dedicated indexes, compression techniques and specific in-memory algorithms
- Generic DB-like techniques should be able to cope with these specific techniques

One way to do this is through user-defined functions

- MapReduce allows user-defined functions (map and reduce)
- Pig latin raises the level of abstraction with an algebraic query language

Examples of Emerging Solutions

Bigtable

MapReduce

Algebraic approach for scientific workflows

Bigtable

Key-value storage system from Google for a shared-nothing cluster

- Uses a distributed file system (GFS) for structured data, with fault-tolerance and availability

Used by popular Google applications

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

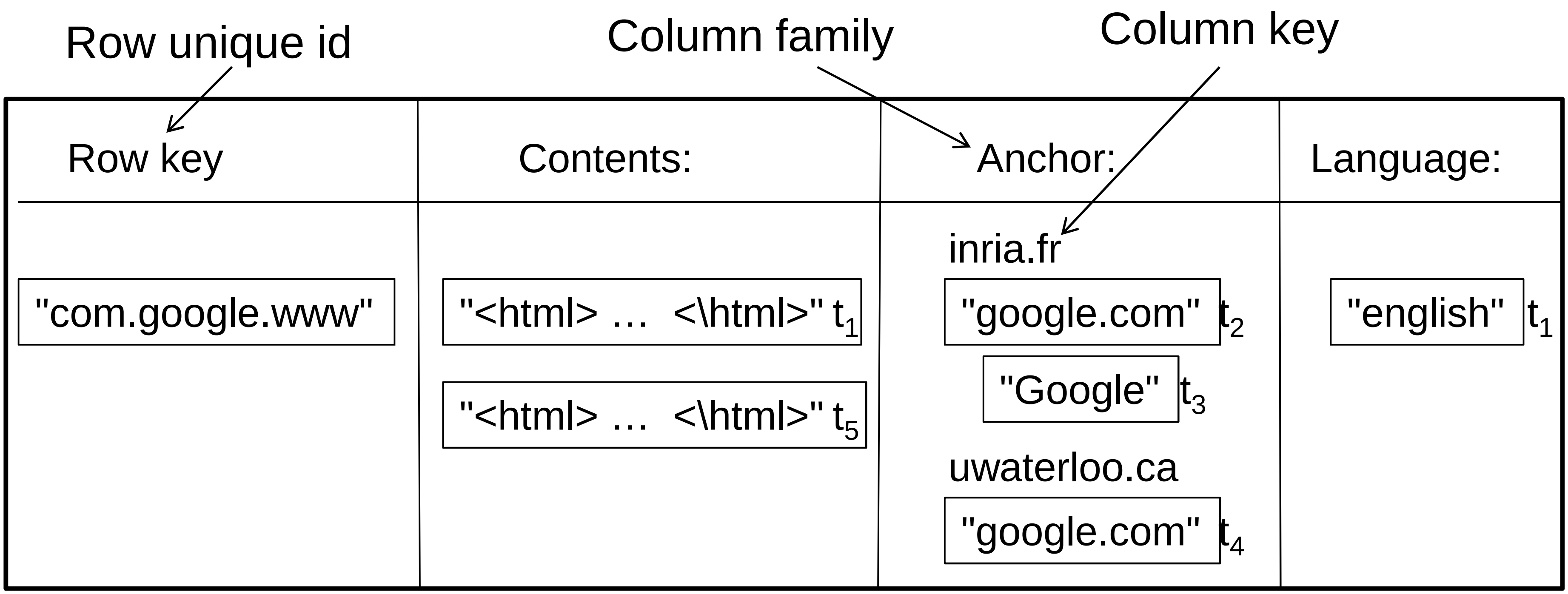
The basis for popular Open Source implementations

- 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)

A Bigtable Row



Can be represented as a special kind of nested tuple

Bigtable’s Principles

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

- Simple algebraic operators
- And no “impedance mismatch” (like with OODB)
  - A major incentive for developers

Transactional atomicity for single row updates only

Key-value storage by range partitioning of a table (as tablets) on the row key

- Partitioning is dynamic, starting with one tablet (the entire table range) which is subsequently split into multiple tablets as the table grows
- Efficient implementation of tablets:
  - Compression of column families, grouping of column families with high locality of access, aggressive caching of metadata information by clients

MapReduce

Parallel programming framework from Google for data analysis of very large data sets

- Highly dynamic, irregular, schemaless, etc.
- SQL or Xquery too heavy
- Typical usage: computing an inverted index for a set of documents, counting URL access frequencies in Web logs, computing a reverse Web-link graph, etc.

Simple functional programming model

- Data structured as (key, value) pairs
  - E.g. (doc-id, content), (word, count), etc.
- Only two functions to be given:
  - Map(k1,v1) -> list(k2,v2)
  - Reduce(k2, list (v2)) –> list(v3)

Implemented on GFS on very large clusters

- Provides high fault-tolerance

MapReduce Example

EMP (ENAME, TITLE, CITY)

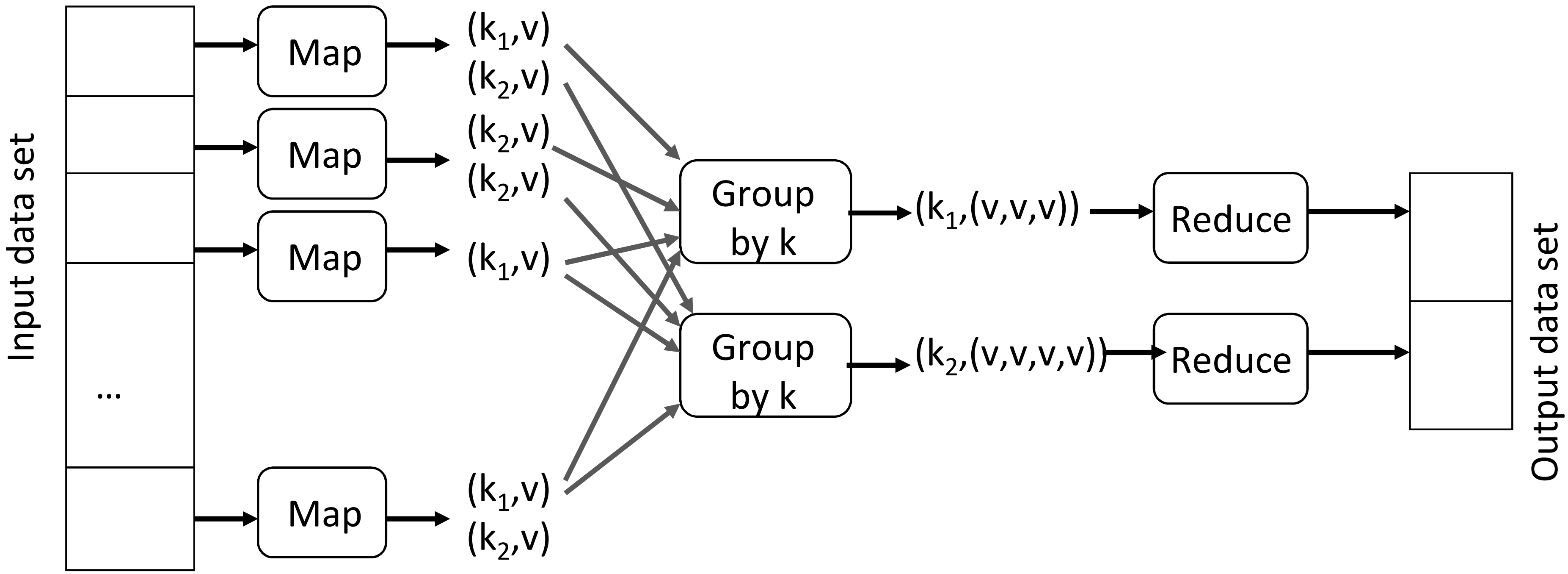
Query: for each city, return the number of employees whose name is "Smith"

```
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*))
```

MapReduce Processing



Principle: independent parallelism through hash partitioning

MapReduce vs PDBMS

[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

Algebraic Approach for Scientific Workflows

Data-centric scientific workflows

- Typically complex and manipulating many large datasets
- Computationally-intensive and data-intensive activities, thus requiring execution in large-scale parallel computers

However, parallelization of scientific workflows remains low-level, ad-hoc and labor-intensive, which makes it hard to exploit optimization opportunities

App. Example in deepwater oil exploitation (joint work with UFRJ and Petrobras)

- Pumping oil from ultra-deepwater from thousand meters up to the surface through *risers*
- Maintaining and repairing risers under deep water is difficult, costly and critical for the environment (e.g. to prevent oil spill)
- Problem: risers fatigue analysis (RFA) requires a complex workflow of data-intensive activities which may take a very long time to compute

Solution: an algebraic approach (inspired by relational algebra) and a parallel execution model that enable automatic parallelization of scientific workflows

- E. Ogasarawa, J. Dias, D. Oliveira, F. Porto, P. Valduriez, M. Mattoso. An Algebraic Approach for Data-Centric Scientific Workflows. VLDB 2011

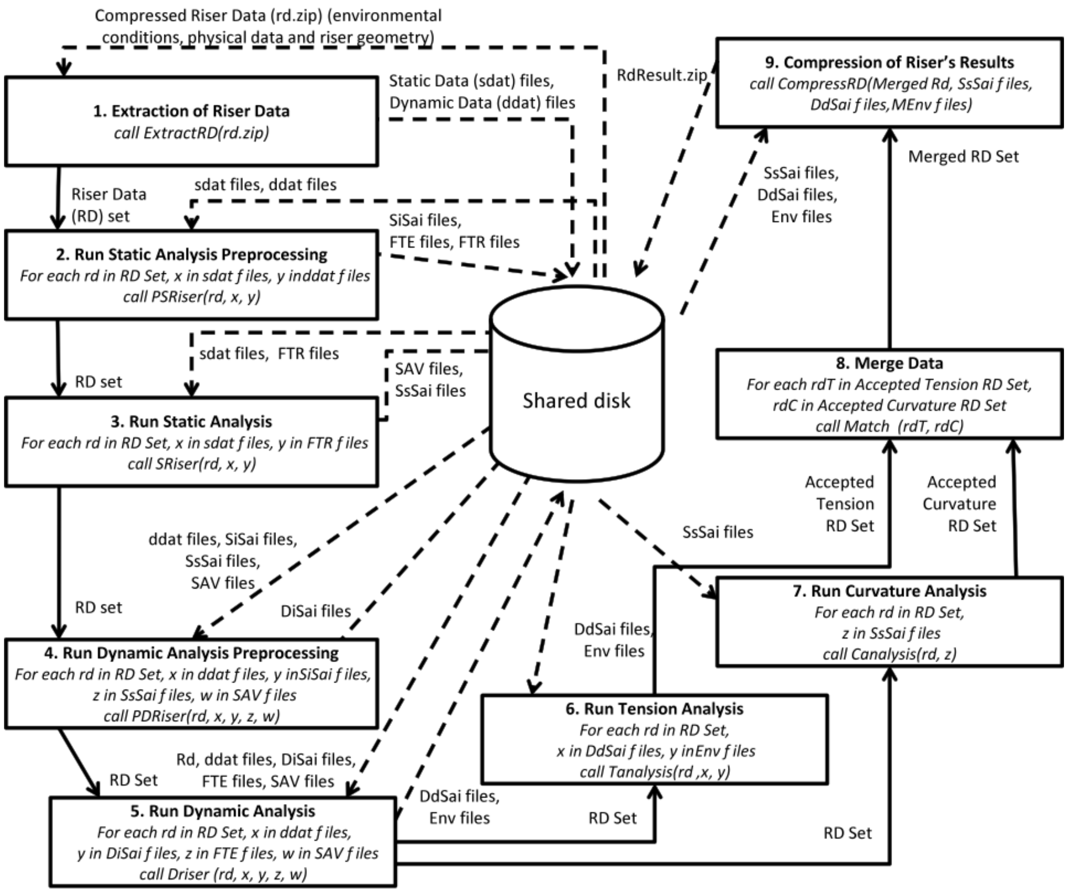
RFA Workflow Example

A typical RFA workflow

- Input: 1,000 files (about 600MB) containing riser information, such as finite element meshes, winds, waves and sea currents, and case studies
- Output: 6,000 files (about 37GB)

Some activities, e.g. dynamic analysis, are repeated for many different input files, and depending on the mesh refinements and other riser's information, each single execution may take hours to complete

The sequential execution of the workflow, on a SGI Altix ICE 8200 (64 CPUs Quad Core) cluster, may take as long as 37 hours



Algebraic Approach

Activities consume and produce relations

- E.g. dynamic analysis consumes tuples, with input parameters and references to input files and produces tuples, with analysis results and references to output files

Operators that provide semantics to activities

- Operators that invoke user programs (map, splitmap, reduce, filter)
- Relational expressions: SRQuery, Join Query

Algebraic transformation rules for optimization and parallelization

An execution model for this algebra based on self-contained units of activity activation

- Inspired by tuple activations for hierarchical parallel systems[Bouganim, Florescu & Valduriez, VLDB 1996]

Conclusion

What are the fundamental principles behind the emerging solutions?

Variations of the relational model

- Key-value stores

As well as specific data models

- Arrays, graphs, sequences, etc.

Simple API or algebraic language for manipulating data from a programming language

- No impedance mismatch
- Comeback of OODB or DBPL?

Relaxed consistency guarantees

- Stronger consistency doable, but at the expense of much more complex code
  - Isn’t ACID simpler for developers?

Hash and range partitioning for parallelism

Replication for fault-tolerance

Is there any generic architectural model to explain those principles?

The three main dimensions (distribution, autonomy, heterogeneity) remain

- Yet pushing the scales high up

But we need more dimensions to capture important architectural aspects

- Generic vs specific
- Dynamic distribution and partitioning for elasticity
- Others?

Do we need new foundations to look at data distribution?

The hardest question, when put in context

- Data -> information -> knowledge

To deal with very distributed data (e.g. personal dataspaces), data semantics and knowledge are important

- Uniform treatment of data, metadata, ontologies, access control, localization of information, trust, provenance, etc.

Is Datalog making a comeback?

- BOOM's Overlog at UCB (Hellerstein et al.)
- WebDamLog at INRIA (Abiteboul et al.)