Big Data Management

Alberto Abelló & Petar Jovanovic

Motivation

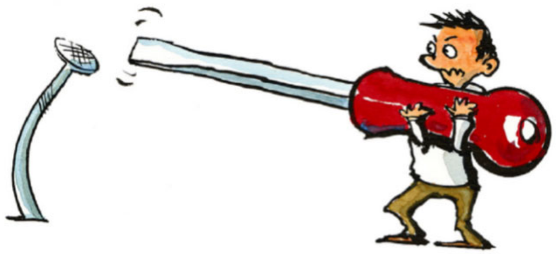
From SQL to NOSQL

Law of the instrument

"Over-reliance on a familiar tool."

Wikipedia

- *Golden hammer* anti-pattern: "A *familiar technology or concept applied obsessively to many software problems.*"



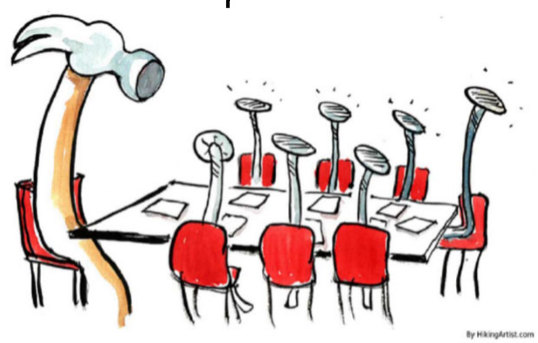
If the only tool you have is a hammer,
everything looks like a nail.

Law of the Relational Database

Object-relational impedance mismatch is "… *one in which a program written using an object-oriented language uses a relational database for storage.*"

Ireland et al.

- Since we only know relational databases, every time we want to model a new domain we'll automatically think on how to represent it as columns and rows



If the only tool you have is a relational database, everything looks like a table.

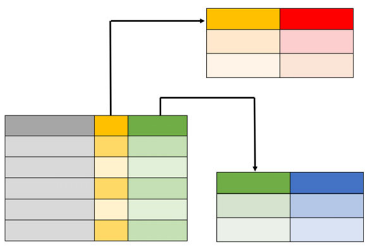
One size does not fit all (Michael Stonebraker)

Not Only SQL (different problems entail different solutions)

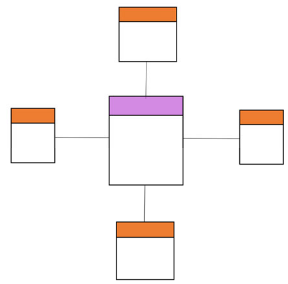
- OLTP
  - Object-Relational
- Data warehousing and OLAP
  - MOLAP
  - Column stores
- Scientific databases and other massive Big Data repositories
  - Key-value stores
  - Column-Family
- Semantic Web and Open Data
  - Graph databases
- Text/documents
  - Document stores (XML, JSON)
- Real-time processing
  - Stream processors

Different data models

Relational (OLTP)



Multidimensional (OLAP)



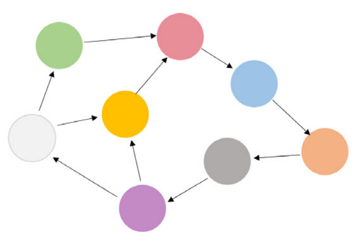
Key-Value

KEY	VALUE
KEY	VALUE
KEY	VALUE
KEY	VALUE
KEY	VALUE

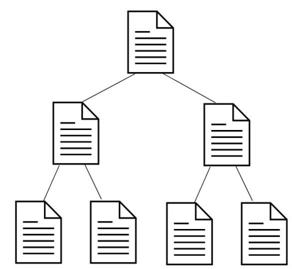
Column-Family

	Family1	Family2	Family3	Family4
Key				
Key				
Key				
Key				

Graph

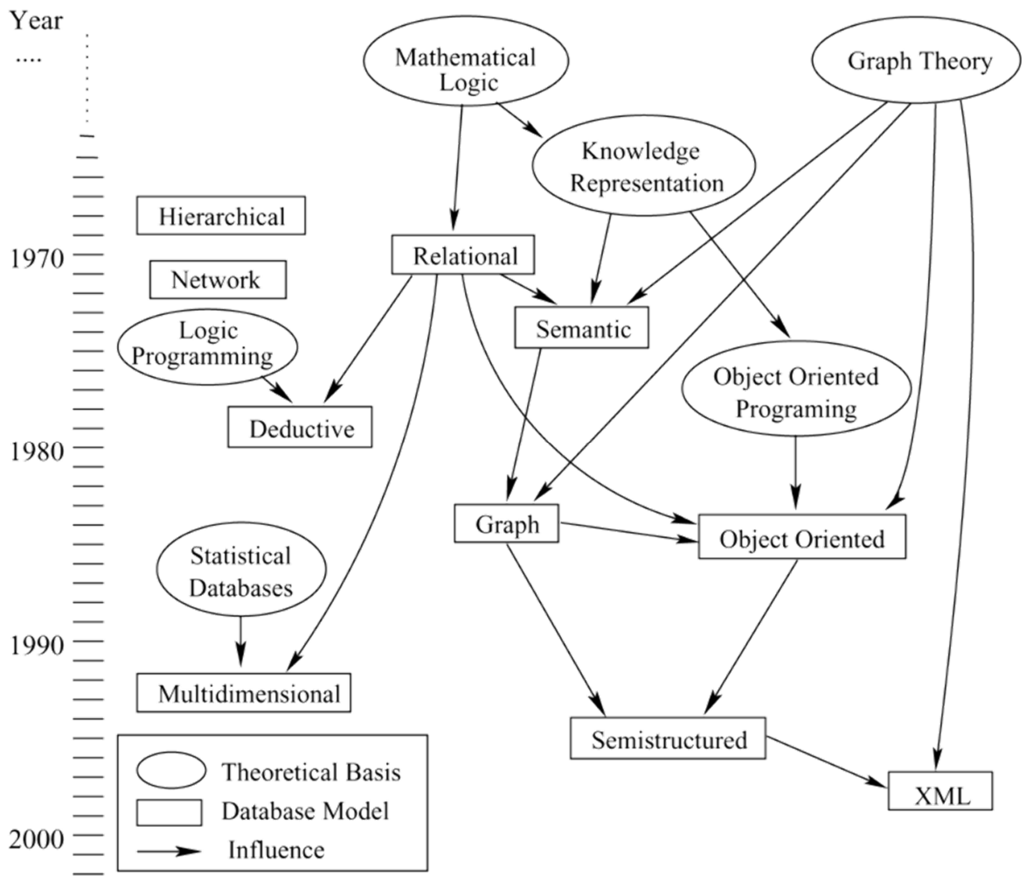


Document



By Aina Montalban, inspired by Daniel G. McCreary and Ann M. Kelly

Evolution of different data models



R. Angles and C. Gutierrez

Alternative storage structures

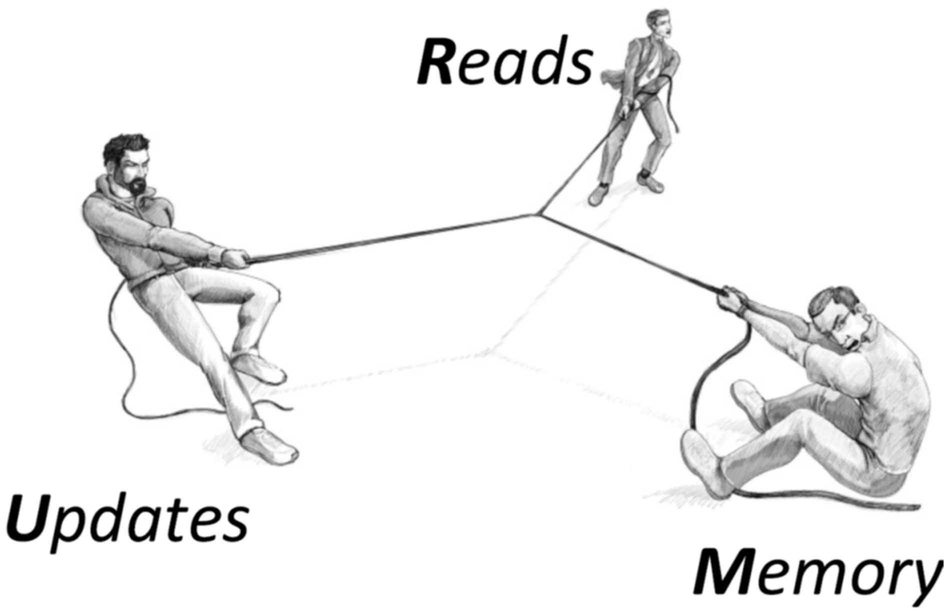
The problem is not SQL

- Relational systems are too generic…
  - OLTP: stored procedures and simple queries
  - OLAP: ad-hoc complex queries
  - Documents: large objects
  - Streams: time windows with volatile data
  - Scientific: uncertainty and heterogeneity
- …but the overhead of RDBMS has nothing to do with SQL
  - Low-level, record-at-a-time interface is not the solution

Michael Stonebraker
SQL Databases vs. NoSQL Databases
Communications of the ACM, 53(4), 2010

The RUM conjecture

“Designing access methods that set an upper bound for two of the RUM overheads, leads to a hard lower bound for the third overhead which cannot be further reduced.”



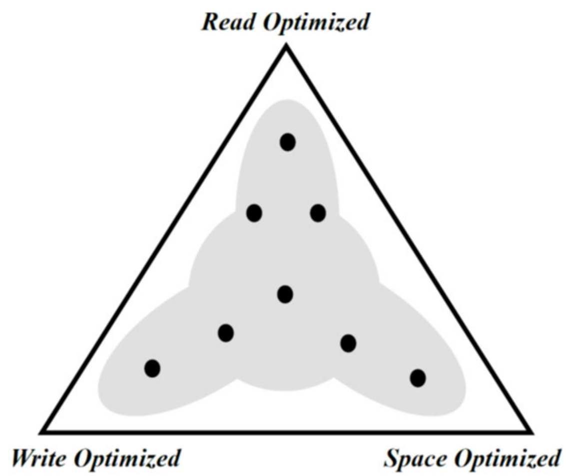
M. Athanassoulis et al.

Example of RUM conjecture

Find(x)

	x	

Space overhead, must update the index



Space overhead (appends), hard to find

Update(x)

	x	x	x

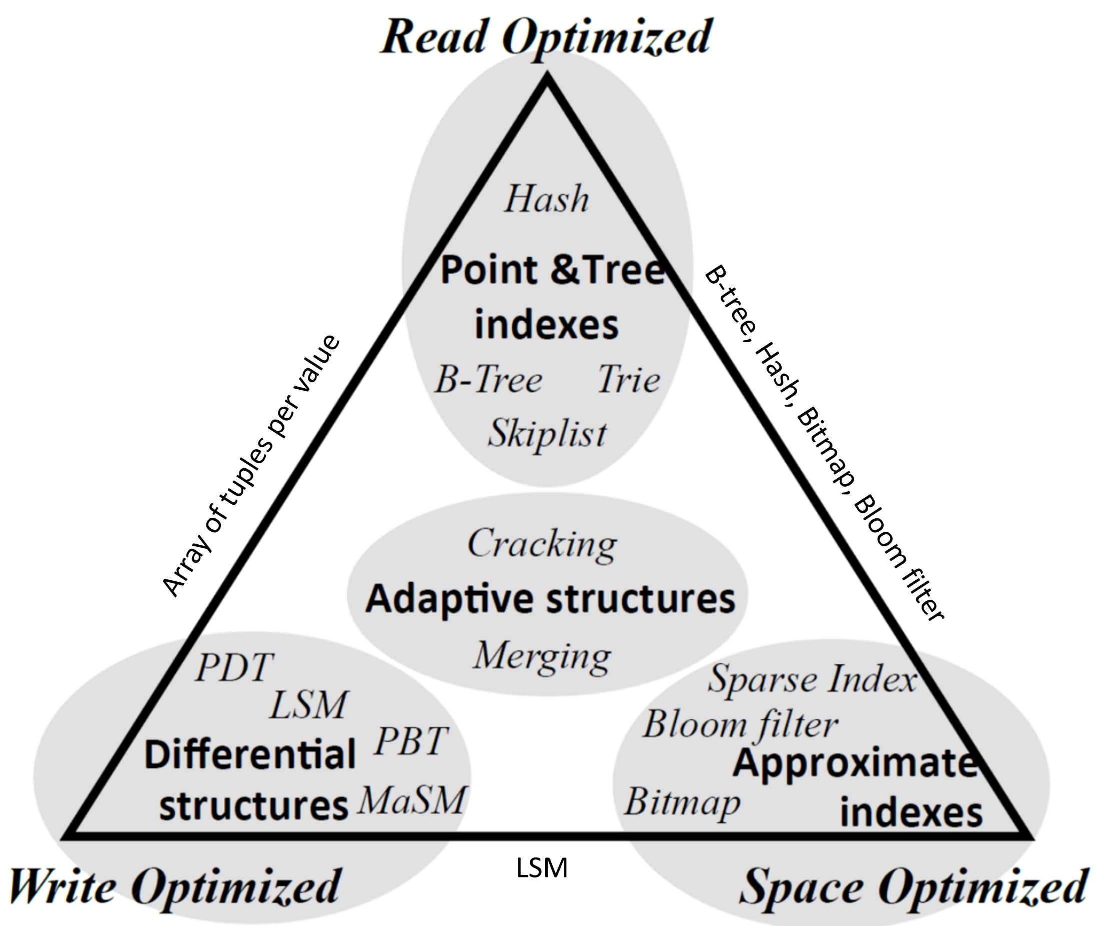
Updating means reordering

Find(x)
Binary search

sorted

M. Athanassoulis et al.

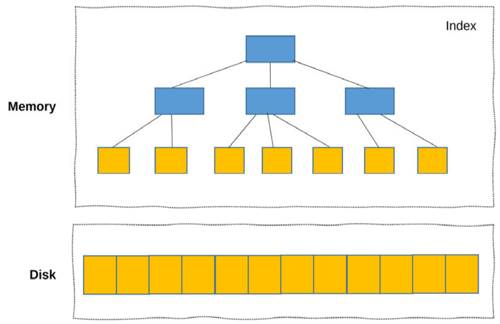
RUM classification space



M. Athanassoulis et al.

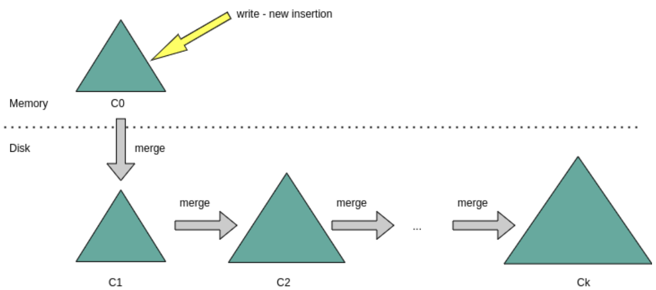
Different internal structures

B-tree



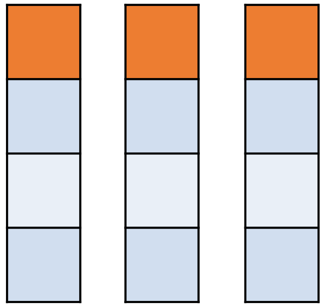
MongoDB, Riak

LSM



HBase, Cassandra, RocksDB

Vertical Partioning



Aina Montalban

Key-Value

BigTable

BigTable Data Model

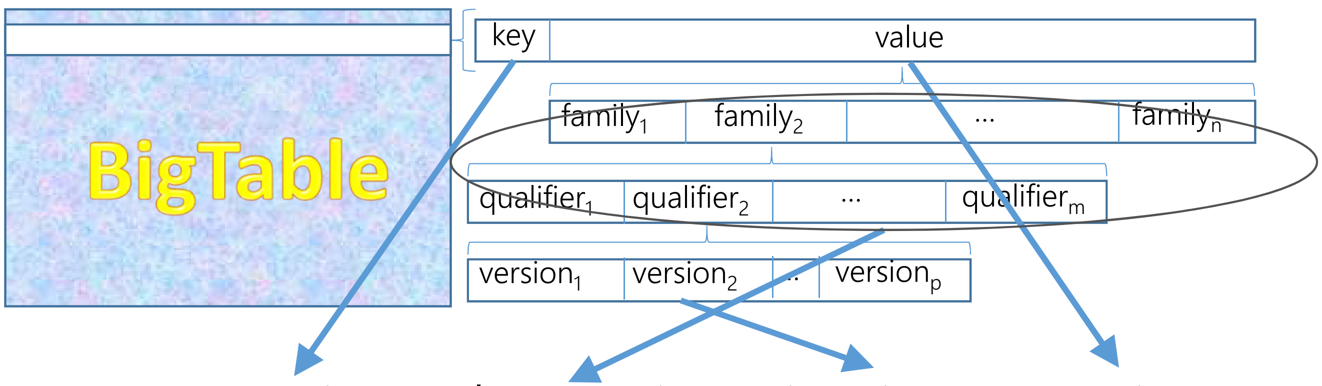
“A Bigtable is a sparse, distributed, persistent, multi-dimensional, sorted map.”

F. Chang et al.

- Sparse: most elements are unknown
- Distributed: cluster parallelism
- Persistent: disk storage (HDFS)
- Multi-dimensional: values with columns
- Sorted: sorting lexicographically by primary key
- Map: lookup by primary key

BigTable schema elements

- Stores tables (collections) and rows (instances)
  - Data is indexed using row and column names (arbitrary strings)
- Treats data as uninterpreted strings (without data types)
- Each cell of a BigTable can contain multiple versions of the same data
  - Stores different versions of the same values in the rows
  - Each version is identified by a timestamp
    - Timestamps can be explicitly or automatically assigned

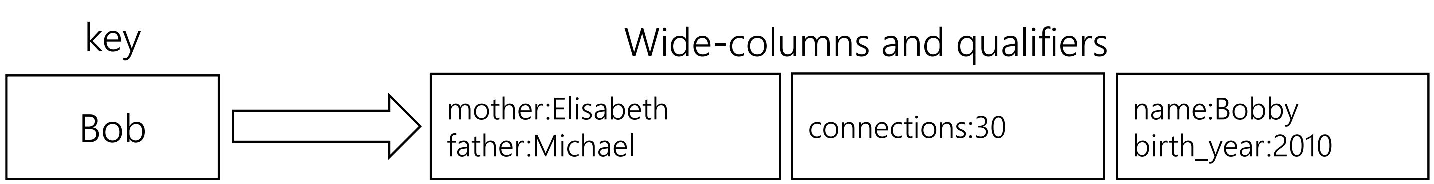


(row:string, column:string[, time:int64])→string

Key-Value

- Key-value stores
  - Entries in form of key-values
    - One key maps only to one value
  - Query on key only
  - Schemaless

- Bigtable (Wide-column key-value stores)
  - Entries in form of key-values
    - But now values are split in wide-columns
  - Typically query on key
    - May have some support for values
  - Schemaless within a wide-column



HBase

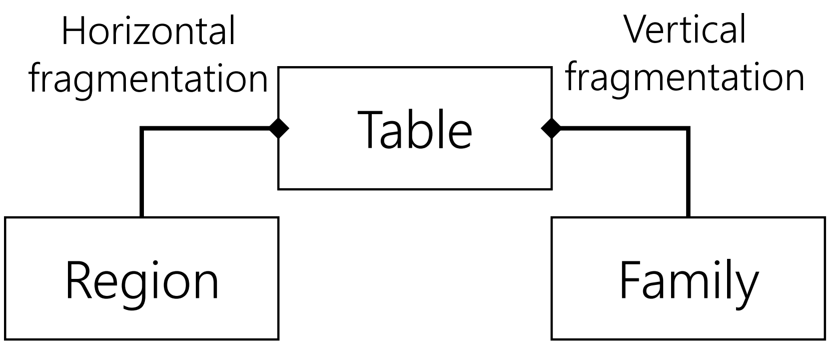
- Apache project
  - Based on Google's Bigtable
- Designed to meet the following requirements
  - Access specific data out of petabytes of data
  - It must support
    - Key search
    - Range search
    - High throughput file scans
  - It must support single row transactions

HBase shell

- ALTER <tablename>, <columnfamilyparam>
- COUNT <tablename>
- CREATE TABLE <tablename>
- DESCRIBE <tablename>
- DELETE <tablename>, <rowkey>[, <columns>]
- DISABLE <tablename>
- DROP < tablename>
- ENABLE <tablename>
- EXIT
- EXISTS <tablename>
- GET <tablename>, <rowkey>[, <columns>]
- LIST
- PUT <tablename>, <rowkey>, <columnid>, <value>[, <timestamp>]
- SCAN <tablename>[, <columns>]
- STATUS [{summary|simple|detailed}]
- SHUTDOWN

https://learnhbase.wordpress.com/2013/03/02/hbase-shell-commands

Logical structure



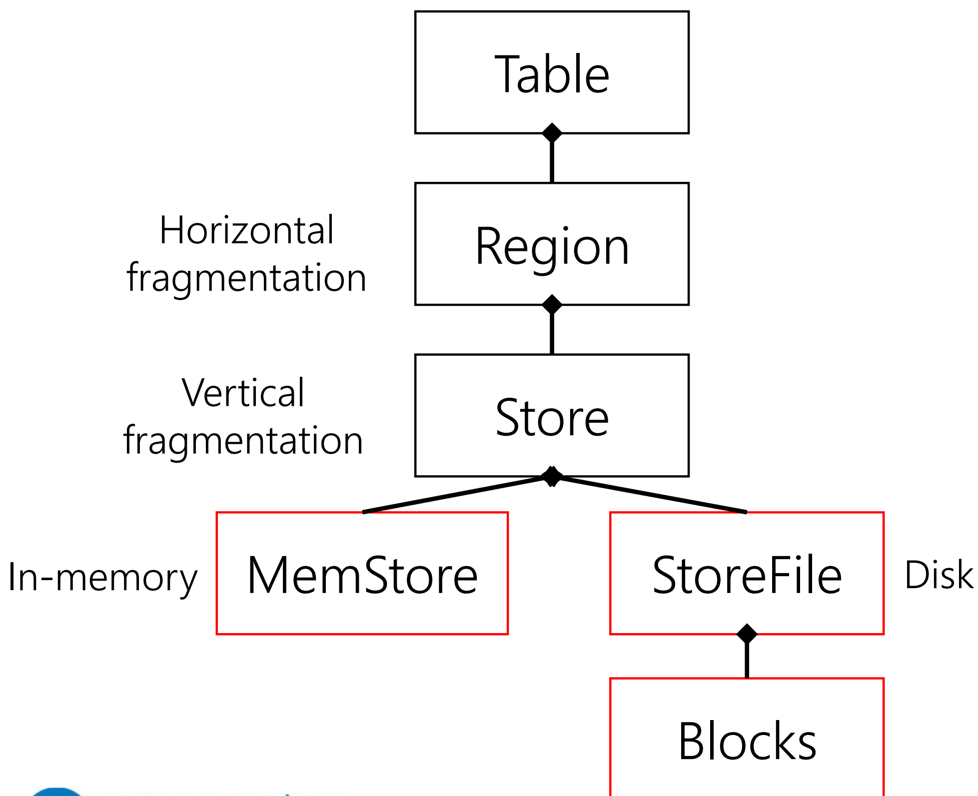
Region 1

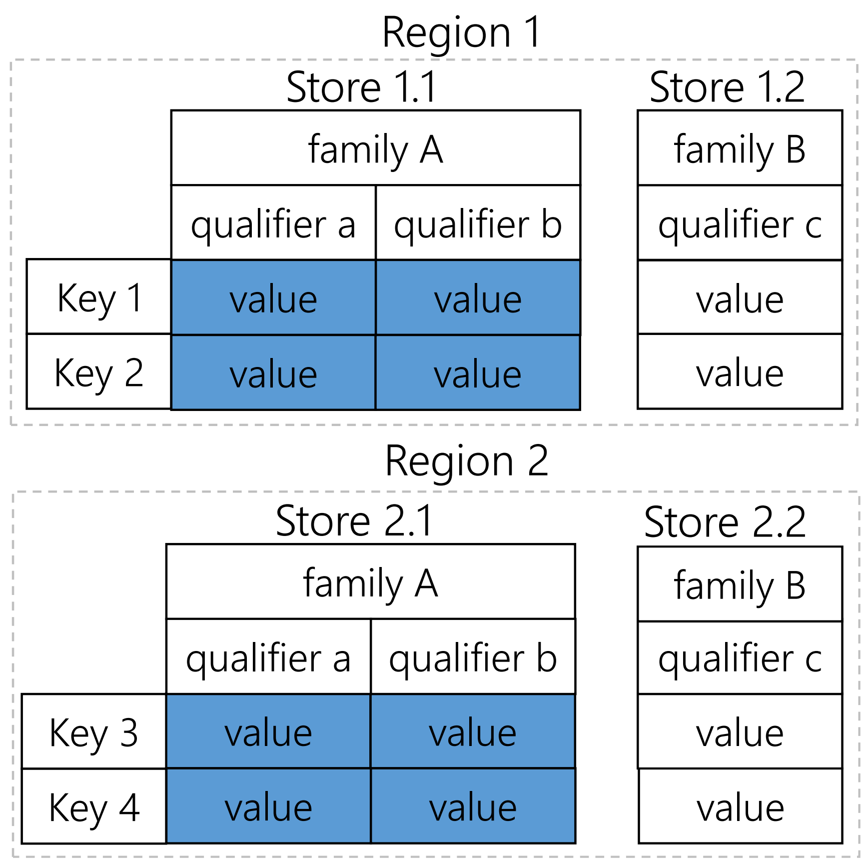
	family A		family B
	qualifier a	qualifier b	qualifier c
Key 1	value	value	value
Key 2	value	value	value

Region 2

	family A		family B
	qualifier a	qualifier b	qualifier c
Key 3	value	value	value
Key 4	value	value	value

Physical structure





HBase Component Roles

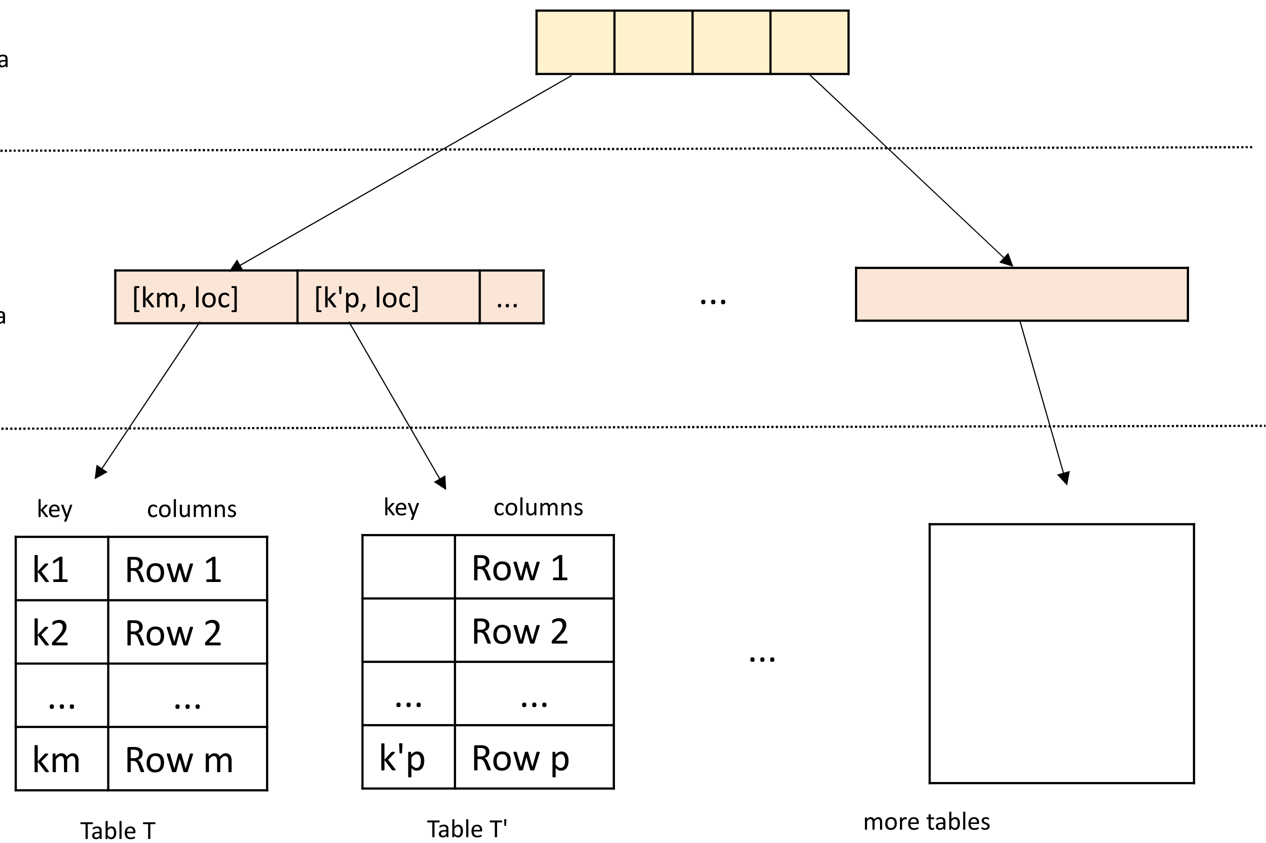
- One coordinator server
  - Maintenance of the table schemas
    - Root region
  - Monitoring of services (heartbeating)
  - Assignment of regions to servers
- Many region servers
  - Each handling around 100-1.000 regions
  - Apply concurrency and recovery techniques
  - Managing split of regions
    - Regions can be sent to another server (load balancer)
  - Managing merge of regions
- Client nodes
  - Cache the metadata sent by the region servers

Metadata hierarchical structure

Root table
Store locations of metadata tables

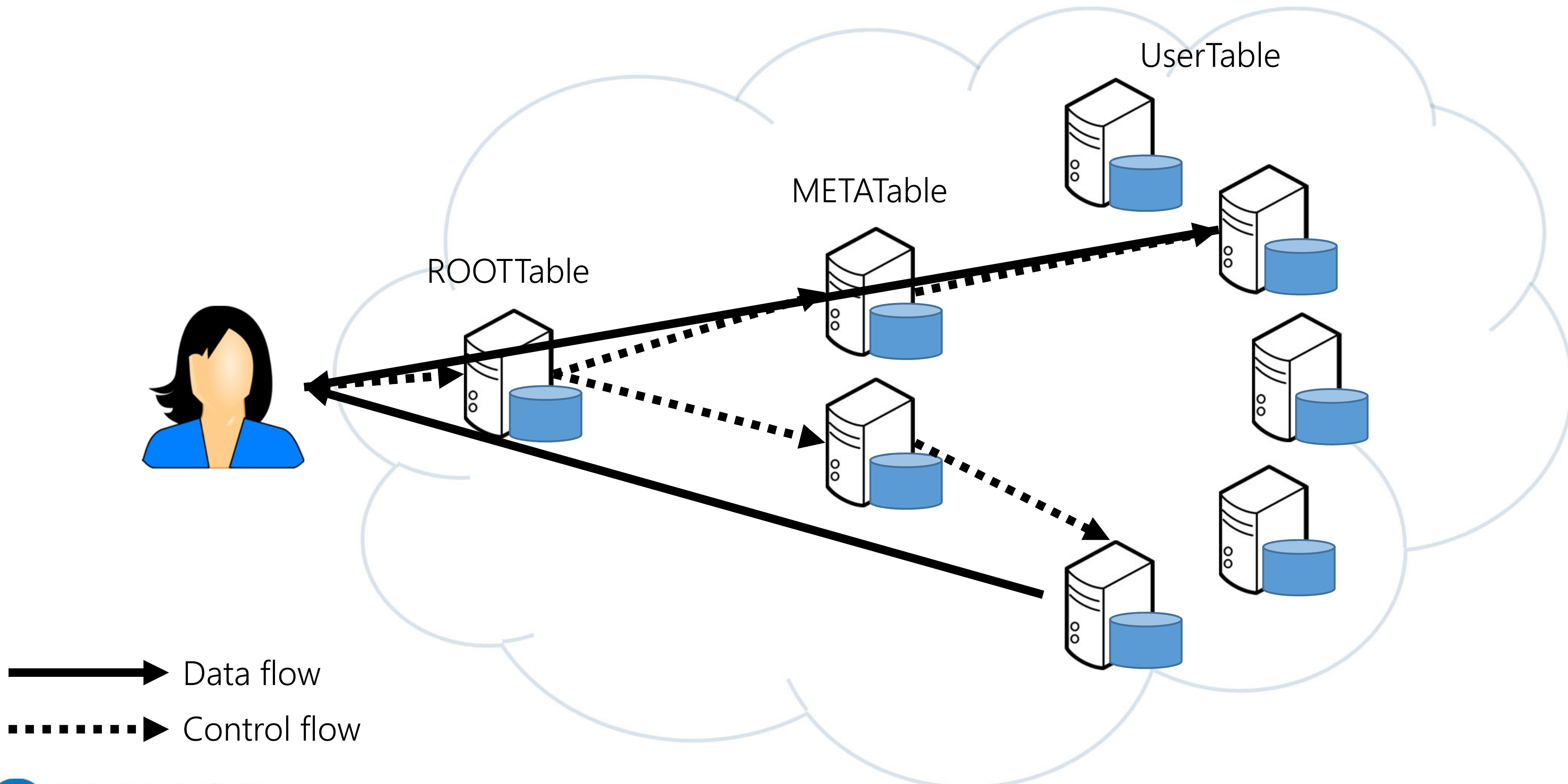
Metadata tables
Store locations of user data tables

User data tables
Store data

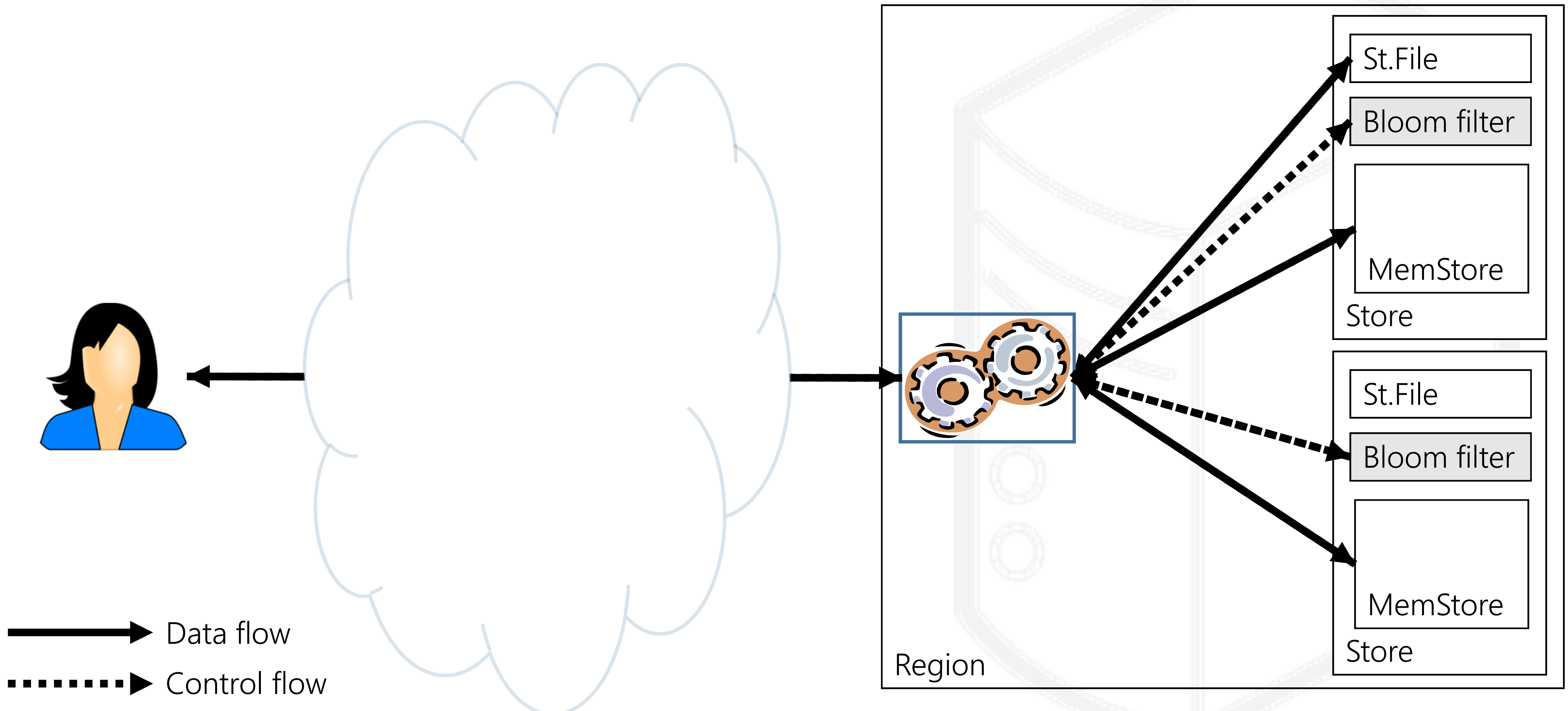


Aina Montalban, based on S. Abiteboul et al.

Global execution



Local execution

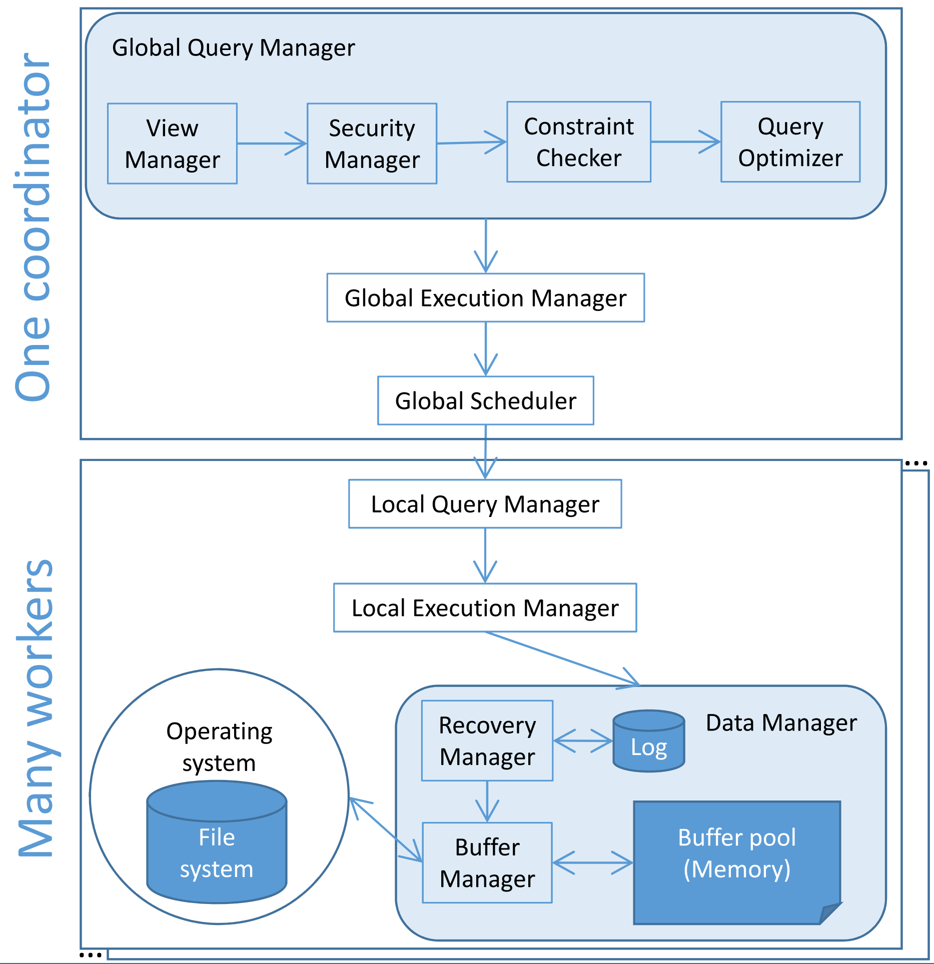


Distributed processing framework

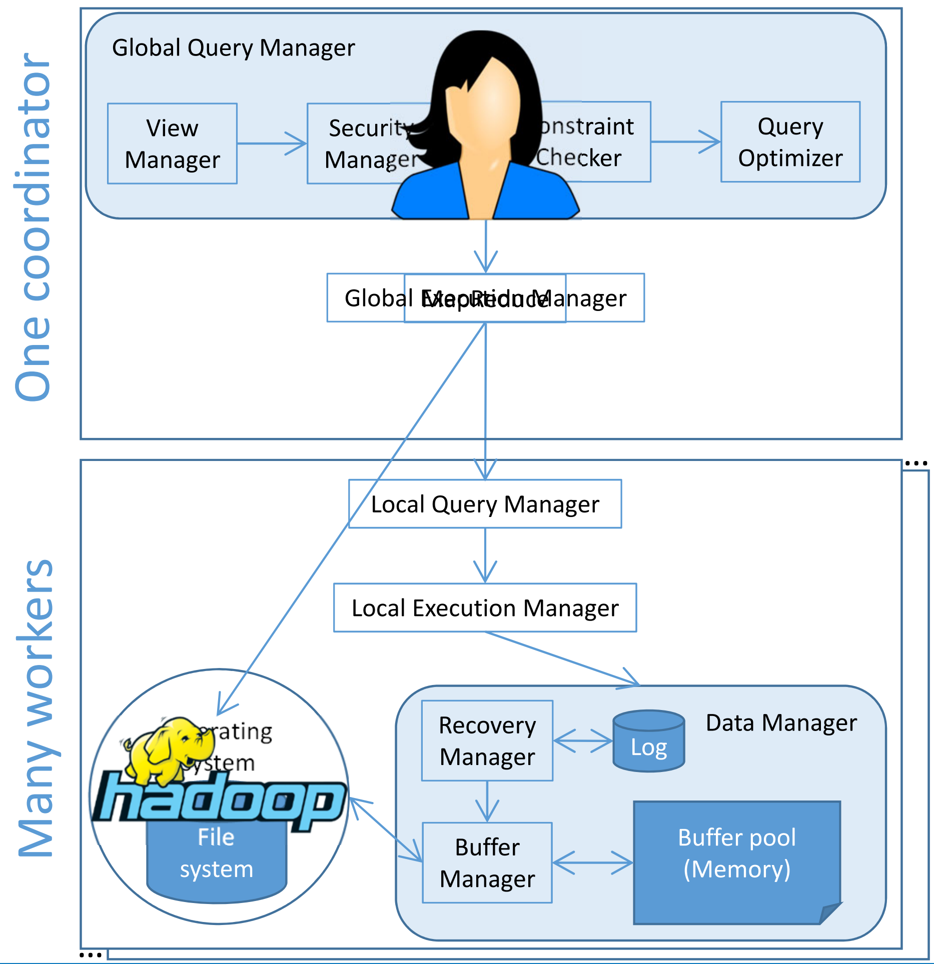
Origins

- Based on Google development
  - Conceived to compute the page Rank
- Data processing framework
  - Facilitate scalability
  - Hidden parallelism
  - Transparent distribution
    - Exploit data locality
    - Balance workload
  - Resilience to failure
    - Fine grained fault tolerance
- Useful in any domain

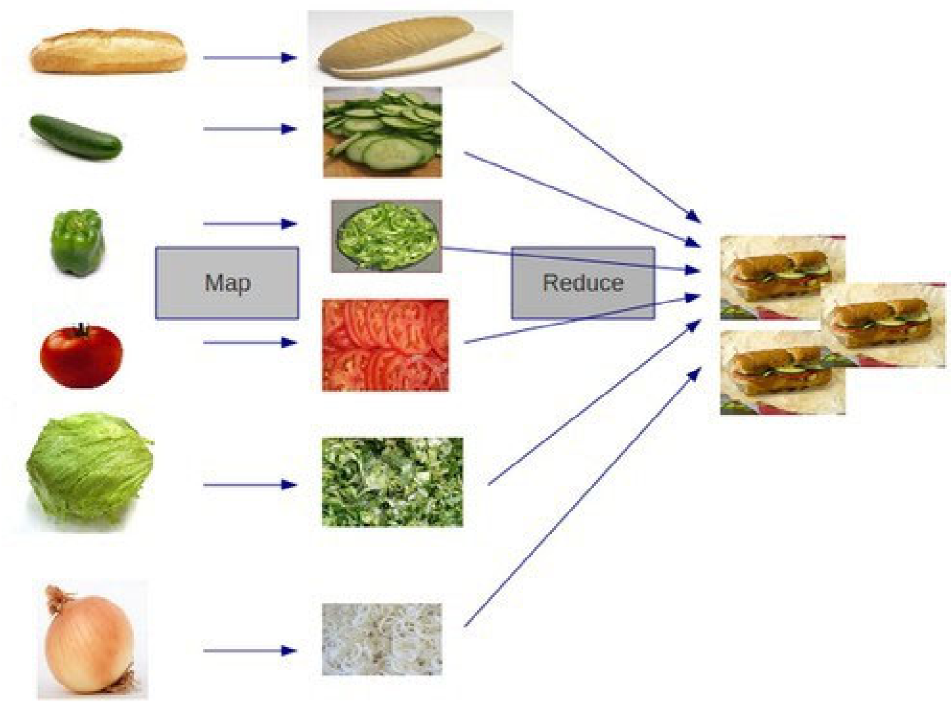
MapReduce as a DDBMS component



MapReduce as a DDBMS component



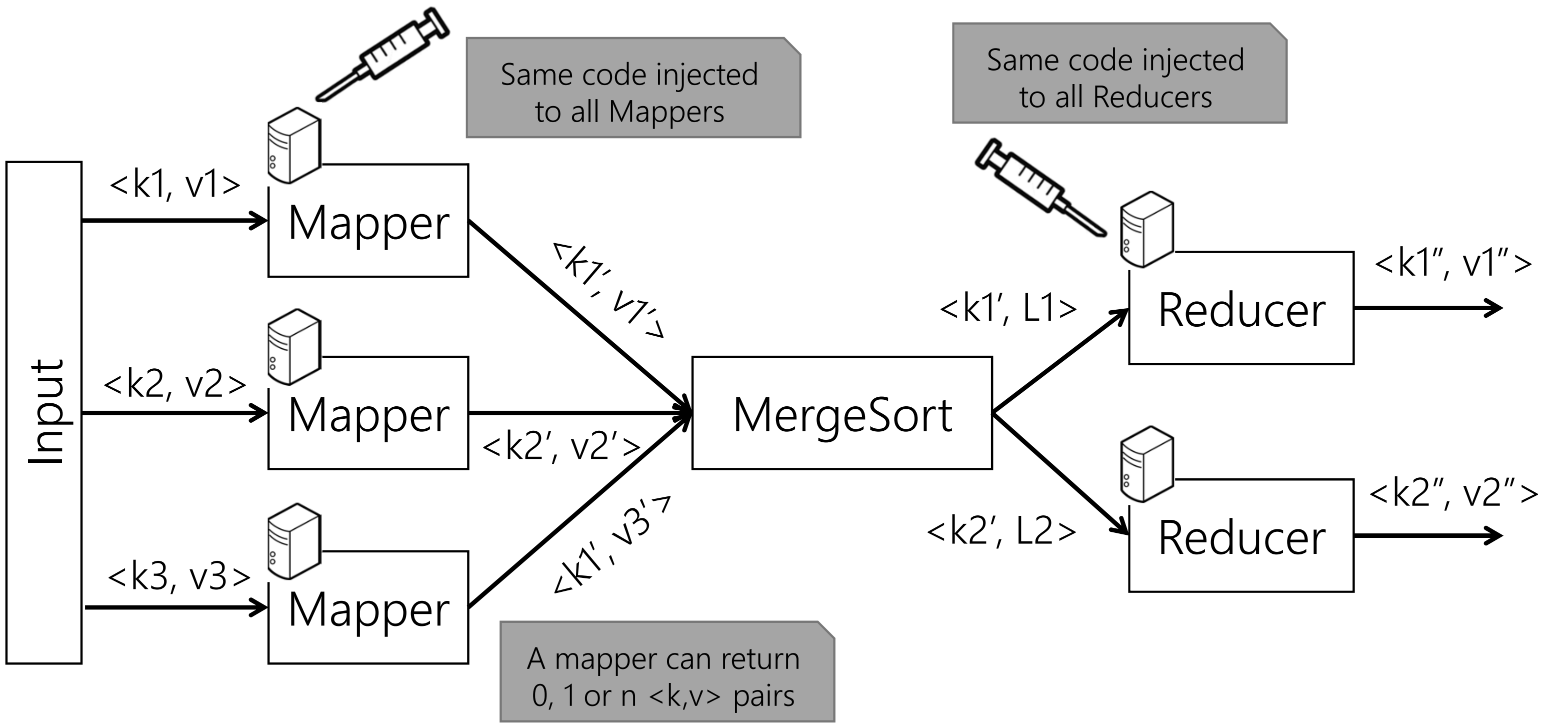
Chain production



By Mohamed Nabeel

Components and use

The MapReduce framework



The MapReduce framework in detail

1. Input: read input from a DFS
2. Map: for each input <key_{in}, $value_{in}$>
   - generate zero-to-many <key_{map}, $value_{map}$>
3. Partition: assign sets of <key_{map}, $value_{map}$> to reducer machines
4. Shuffle: data are shipped to reducer machines using a DFS
5. Sort&Merge: reducers sort their input data by key
6. Reduce: for each key_{map}
   - the set $value_{map}$ is processed to produce zero-to-many <key_{red}, $value_{red}$>
7. Output: writes the result of reducers to the DFS

Formal definition

- Single input
  - Data are represented as <key, value> pairs
    - Value can be anything (structured or not)
- Functional programming
  - Map phase, for each input <key, value> a function *f* is applied that returns a multiset of new <key, value> pairs:

$$f(\langle k, v\rangle) \mapsto \{\langle k_1, v_1\rangle, \ldots, \langle k_n, v_n\rangle\}$$

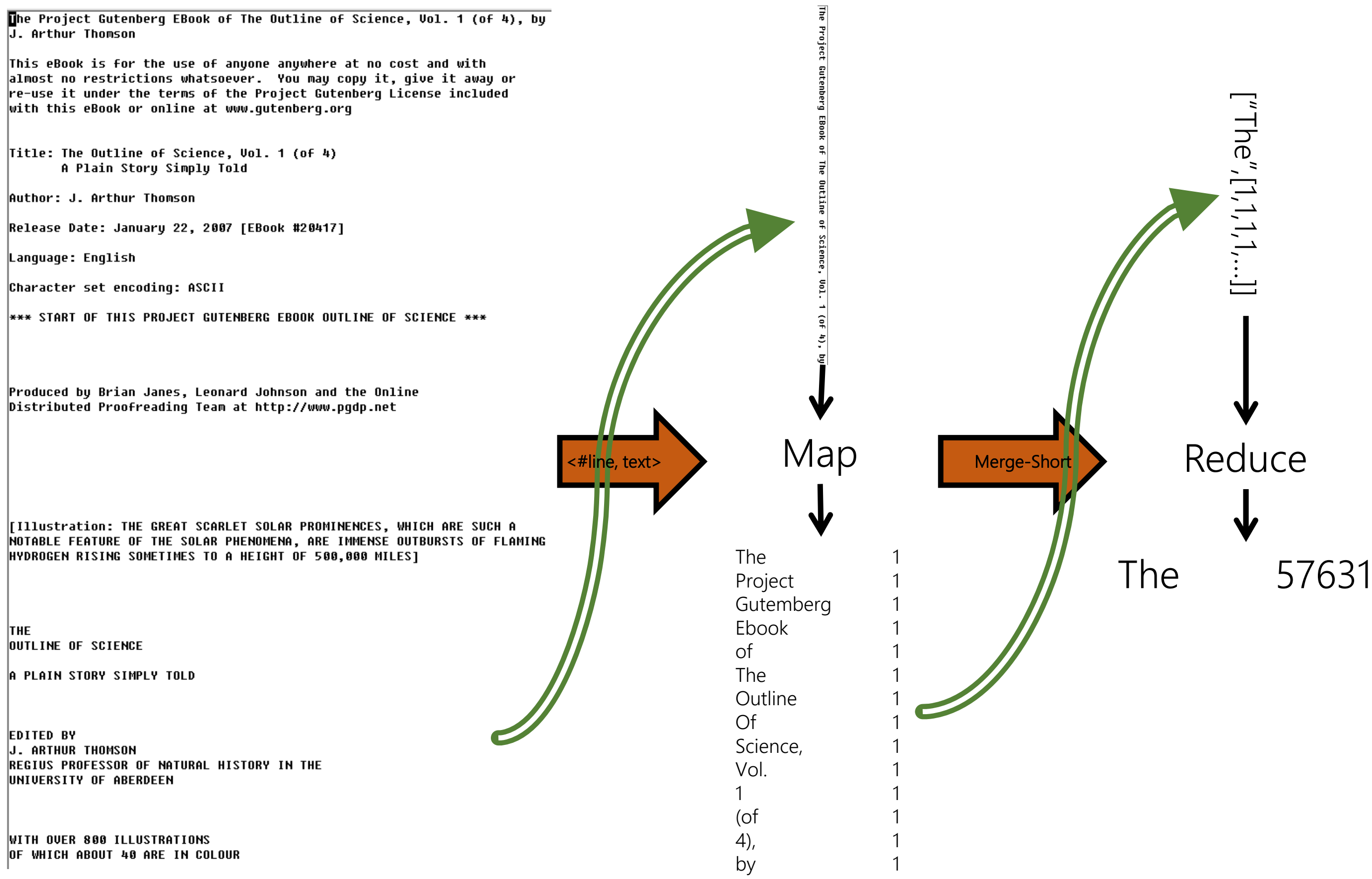
  - Reduce phase, all pairs with the same key are grouped and a function *g* is applied, which returns also a multiset of new <key, value> pairs:

$$g(\langle k, \{v_1, \ldots, v_n\}\rangle) \mapsto \{\langle k_1, v_1\rangle, \ldots, \langle k_m, v_m\rangle\}$$

MapReduce examples

Word count

Word count example



WordCount Code Example

```
public void map(LongWritable key, Text value) {
    String line = value.toString();
    StringTokenizer tokenizer = new StringTokenizer(line);
    while (tokenizer.hasMoreTokens()) {
        write(new Text(tokenizer.nextToken()), new IntWritable(1));
    }
}

public void reduce(Text key, Iterable<IntWritable> values) {
    int sum = 0;
    for (IntWritable val : values) {
        sum += val.get();
    }
    write(key, new IntWritable(sum));
}
```

WordCount Code Example

```
public void map( Key , Value ) {
                    Blackbox
        write( Key , Value );
    }
}

public void reduce( Key , Values ) {
                    Blackbox
    write( Key , Value );
}
```

Relational algebra in MapReduce

MapReduce Genericity

- Supported in many store systems
  - HBase, MongoDB, CouchDB, etc.
- Programming paradigm is computationally complete
  - Any data process can be adapted to it
    - Some tasks better adapt to it than others
    - Not necessarily efficient
      - Optimization is very limited because of lack of expressivity
- Signature is closed
  - Iterations can be chained
    - Fault tolerance is not guaranteed in between
    - Resources are released to be just requested again
- Criticized for being too low-level
  - APIs for Ruby, Python, Java, C++, etc.
  - Attempts to build declarative languages on top
    - SQL-like
      - HiveQL
      - Cassandra Query Language (CQL)

Relational operations: Projection

$$\pi_{a_{i_1},\dots,a_{i_n}}(T) \Rrightarrow \begin{cases} \texttt{map}(\text{key } k, \text{value } v) \mapsto [(\texttt{prj}_{a_{i_1},\dots,a_{i_n}}(k \oplus v), 1)] \\ \texttt{reduce}(\text{key } ik, \text{vset } ivs) \mapsto [(ik)] \end{cases}$$

Relational operations: Cross Product

$$
T \times S \quad \Longmapsto \quad \begin{cases}
\mathtt{map}(\text{key } k, \text{value } v) \mapsto \\
\begin{cases}
[(\mathrm{h}_T(k) \bmod D, k \oplus v)] & \text{if input}(k \oplus v) = T, \\
[(0, k \oplus v), .., (D-1, k \oplus v)] & \text{if input}(k \oplus v) = S.
\end{cases} \\
\mathtt{reduce}(\text{key } ik, \text{vset } ivs) \mapsto \\
\Big[\mathtt{crossproduct}(T_{ik}, S) \mid \\
\quad T_{ik} = \{iv \mid iv \in ivs \wedge \text{input}(iv) = T\}, \\
\quad S = \{iv \mid iv \in ivs \wedge \text{input}(iv) = S\}\Big]
\end{cases}
$$

References

- W. J. Brown et al. *AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis*. Wiley, 1998
- C. Ireland et al. *A classification of object-relational impedance mismatch* . DBKDA 2009
- M. Stonebraker et al. *The End of an Architectural Era (It's Time for a Complete Rewrite)*. VLDB, 2007
- L. Liu, M.T. Özsu (Eds.). *Encyclopedia of Database Systems*. Springer, 2009
- R. Cattell. *Scalable SQL and NoSQL Data Stores*. SIGMOD Record 39(4), 2010
- M. Stonebraker. *SQL Databases vs. NoSQL Databases.* Communications of the ACM, 53(4), 2010
- E. Meijer and G. Bierman. *A Co-Relational model of data for large shared data banks*. Communications of the ACM 54(4), 2011
- P. Sadagale and M. Fowler. *NoSQL distilled*. Addison-Wesley, 2013
- V. Herrero et al. *NOSQL Design for Analytical Workloads: Variability Matters*. ER, 2016
- M. Athanassoulis et al. *Designing Access Methods: The RUM Conjecture*. EDBT, 2016
- R. Tan et al. *Enabling query processing across heterogeneous data models: A survey*. BigData 2017

References

- P. O'Neil et al. *The log-structured merge-tree (LSM-tree).* Acta Informatica, 33(4). Springer, 1996
- F. Chang et al. *Bigtable: A Distributed Storage System for Structured Data*. OSDI'06
- S. Abiteboul et al. Web Data Management. Cambridge University Press, 2011. http://webdam.inria.fr/Jorge
- N. Neeraj. *Mastering Apache Cassandra*. Packt, 2015
- O. Romero et al. *Tuning small analytics on Big Data: Data partitioning and secondary indexes in the Hadoop ecosystem*. Information Systems, 54. Elsevier, 2016
- A. Petrov. *Algorithms Behind Modern Storage Systems*. Communications of the ACM 61(8), 2018

References

- J. Dean et al. *MapReduce: Simplified Data Processing on Large Clusters*. OSDI'04
- A. Pavlo et al. *A Comparison of Approaches to Large-Scale Data Analysis*. SIGMOD, 2009
- J. Dittrich et al. *Hadoop++: Making a Yellow Elephant Run Like a Cheetah (Without It Even Noticing)*. Proc. VLDB Endow. 3(1-2), 2010
- M. Stonebraker et al. *MapReduce and parallel DBMSs: friends or foes?* Communication of ACM 53(1), 2010
- S. Abiteboul et al. *Web data management*. Cambridge University Press, 2011
- A. Rajaraman et al. *Mining massive data sets*. Cambridge University Press, 2012
- P. Sadagale and M. Fowler. *NoSQL distilled*. Addison-Wesley, 2013