Principles of Distributed Database Systems

TS. Phan Thị Hà

© 2020, M.T. Özsu & P. Valduriez TS. Phan Thị Hà

Outline

- Introduction
- Distributed and Parallel Database Design
- Distributed Data Control
- Distributed Query Processing
- Distributed Transaction Processing
- Data Replication
- Database Integration – Multidatabase Systems
- Parallel Database Systems
- Peer-to-Peer Data Management
- Big Data Processing
- NoSQL, NewSQL and Polystores
- Web Data Management

© 2020, M.T. Özsu & P. Valduriez

Outline

- NoSQL, NewSQL and Polystores
  - Motivations
  - NoSQL systems
  - NewSQL systems
  - Polystores

© 2020, M.T. Özsu & P. Valduriez

Motivations

- Trends
  - Big data
    - Unstructured data
  - Data interconnexion
    - Hyperlinks, tags, blogs, etc.
  - Very high scalability
    - Data size, numbers of users
- Limits of relational DBMSs (SQL)
  - Need for skilled DBA and well-defined schemas
  - SQL and complex tuning
  - Hard to make updates scalable
    - Parallel RDBMS use a shared-disk for OLTP
- The CAP theorem

© 2020, M.T. Özsu & P. Valduriez

The CAP Theorem

- Polemical topic
  - “A database can't provide consistency AND availability during a network partition”
  - Argument used by NoSQL to justify their lack of ACID properties
  - But has nothing to do with scalability
- Two different points of view
  - Relational databases
    - Consistency is essential
      - ACID transactions
  - Distributed systems
    - Service availability is essential
      - Inconsistency tolerated by the user, e.g. web cache

© 2020, M.T. Özsu & P. Valduriez

What is the CAP Theorem?

- The desirable properties of a distributed system
  - ***C****onsistency*: all nodes see the same data values at the same time
  - ***A****vailability*: all requests get an answer
  - ***P****artition tolerance*: the system keeps functioning in case of network failure
- History
  - At the PODC 2000 conference, Brewer (UC Berkeley) conjectures that one can have only two properties at the same time
  - In 2002, Gilbert and Lynch (MIT) prove the conjecture, which becomes a theorem

© 2020, M.T. Özsu & P. Valduriez

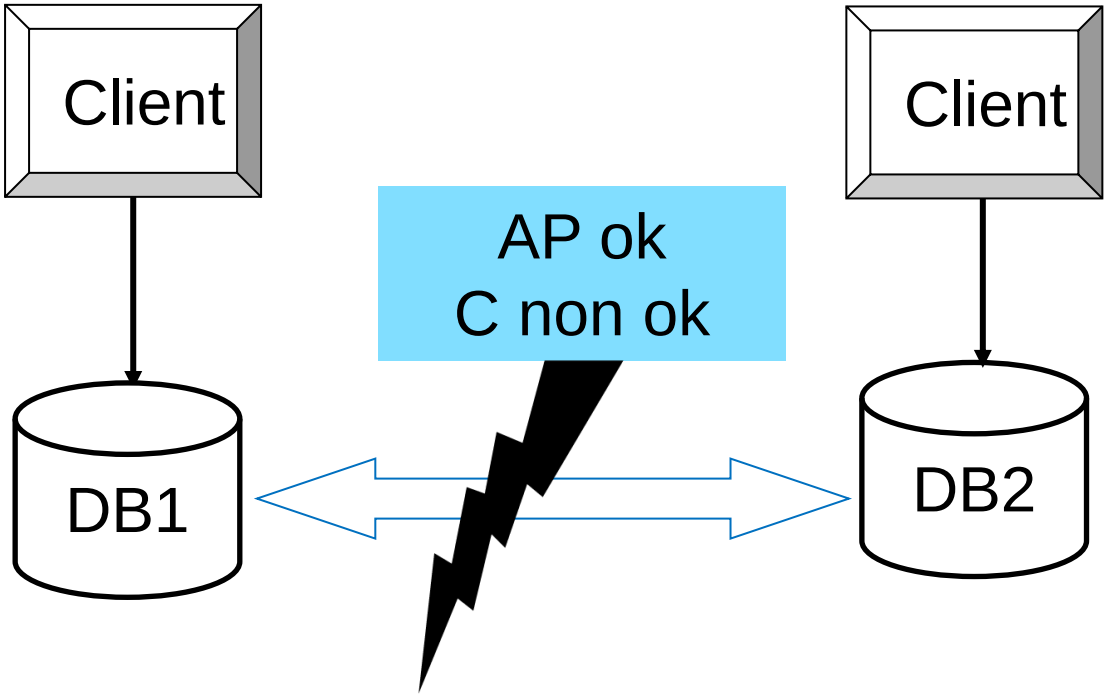
Strong vs Eventual Consistency

- Strong consistency (ACID)
  - All nodes see the same data values at the same time
- Eventual consistency
  - Some nodes may see different data values at the same time
  - But if we stop injecting updates, the system reaches strong consistency

- Illustration with symmetric, asynchronous replication in databases

© 2020, M.T. Özsu & P. Valduriez

Symmetric, Asynchronous Replication



But we have eventual consistency

- After reconnection (and resolution of update conflicts), consistency can be obtained

Outline

- NoSQL, NewSQL and Polystores
  - Motivations
  - NoSQL systems
  - NewSQL systems
  - Polystores

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

© 2020, M.T. Özsu & P. Valduriez

NoSQL Approaches

- Characterized by the data model, in increasing order of complexity:
  1. Key-value: DynamoDB
  2. Tabular: Bigtable
  3. Document: MongoDB
  4. Graph: Neo4J
  5. Multimodel: OrientDB
- What about object DBMS or XML DBMS?
  - Were there much before NoSQL
  - Sometimes presented as NoSQL
  - But not really scalable

© 2020, M.T. Özsu & P. Valduriez

Key-value Stores

- Simple (key, value) data model
  - Key = unique id
  - Value = a text, a binary data, structured data, etc.
- Simple queries
  - Put (key, value)
    - Inserts a (key, value) pair
  - Value = get (key)
    - Returns the value associated with key
  - {(key, value)} = get_range (key1, key2)
    - Returns the data whose key is in interval [key1, key2]

© 2020, M.T. Özsu & P. Valduriez

Amazon DynamoDB

- Major service of AWS for data storage
  - E.g. product lists, shopping carts, user preferences
- Data model (key, structured value)
  - Partitioning on the key and secondary indices on attributes
  - Simple queries on key and attributes
  - Flexible: no schema to be defined (but automatically inferred)
- Consistency
  - Eventual consistent reads (default)
  - Strong consistent reads
  - Atomic updates with atomic counters
- High availability and fault-tolerance
  - Synchronous replication between data centers
- Integration with other AWS services
  - Identity control and access
  - MapReduce
  - Redshift data warehouse

© 2020, M.T. Özsu & P. Valduriez

DynamoDB – data model

- Table (items)
- Item (key, attributes)
  - 2 types of primary (unique) keys
    - Hash (1 attribute)
    - Hash & range (2 attributes)
  - Attributes of the form "name":"value"
    - Type of value: scalar, set, or JSON
- Java API with methods
  - Add, update, delete item
  - GetItem: returns an item by primary key in a table
  - BatchGetItem: returns the items of same primary key in multiple tables
  - Scan : returns all items
  - Query
    - Range on hash & range key
    - Access on indexed attribute

Table: Forum_Thread

Forum	Subject	Date of last post	Tags
"S3"	"abc"	"2017 . . ."	"a" "b"
"S3"	"acd"	"2017 . . ."	"c"
"S3"	"cbd"	"2017 . . ."	"d" "e"
"RDS"	"xyz"	"2017 . . ."	"f"
"EC2"	"abc"	"2017 . . ."	"a" "e"
"EC2"	"xyz"	"2017 . . ."	"f"

Hash key — Range key

GetItem (Forum="EC2", Subject="xyz")

Query (Forum="S3", Subject > "ac")

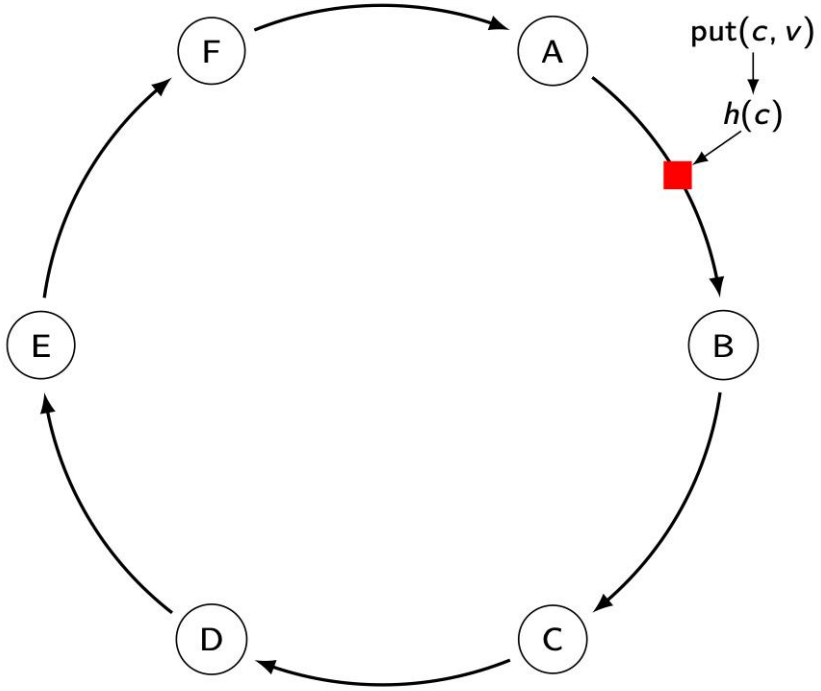
© 2020, M.T. Özsu & P. Valduriez

DynamoDB - data partitioning

DynamoDB

amazon web services

- Consistent hashing: the interval of hash values is treated as a ring
- Advantage: if a node fails, its successor takes over its data
  - No impact on other nodes
- Data is replicated on next nodes



Node B is responsible for the hash value interval (A,B]. Thus, item (c,v) is assigned to node B

Document Stores

- Main applications
  - Document systems
  - Content Management Systems
  - Catalogs
  - Personalization
  - Analysis of messages (tweets, etc.) in real time
  - Etc.

© 2020, M.T. Özsu & P. Valduriez

Data Models for Documents

- Documents
  - Hierarchical structure, with nesting of elements
  - Weak structuring, with "similar" elements
  - Base types: text, but also integer, real, date, etc.
- Two main data models
  - XML (eXtensible Markup Language): W3C standard (1998) for exchanging data on the Web
    - Complex and heavy
  - JSON (JavaScript Object Notation) by Douglas Crockford (2005) for exchanging data JavaScript
    - Simple and light

© 2020, M.T. Özsu & P. Valduriez

MongoDB

- Objective: performance and scalability
  - A document is a collection of (key, typed value) with a unique key (generated by MongoDB)
- Data model and query language based on JSON
  - Binary JSON (BSON): more compact
- No schema, no join, no complex transaction
- Shared-nothing cluster architecture
- Secondary indices
- Integration with MapReduce & Spark

A MongoDB Collection (posts)

_id: ObjectId("abc")	author: "alex", title: "No Free Lunch", text: "This is ...", tags: ["business","ramblings"], comments: [{who:"jane", what:"I agree."},{who:"joe", what:"No ..."}]
_id: ObjectId("abd")	A post by X
_id: ObjectId("acd")	A post by Y

Unique key generated by MongoDB

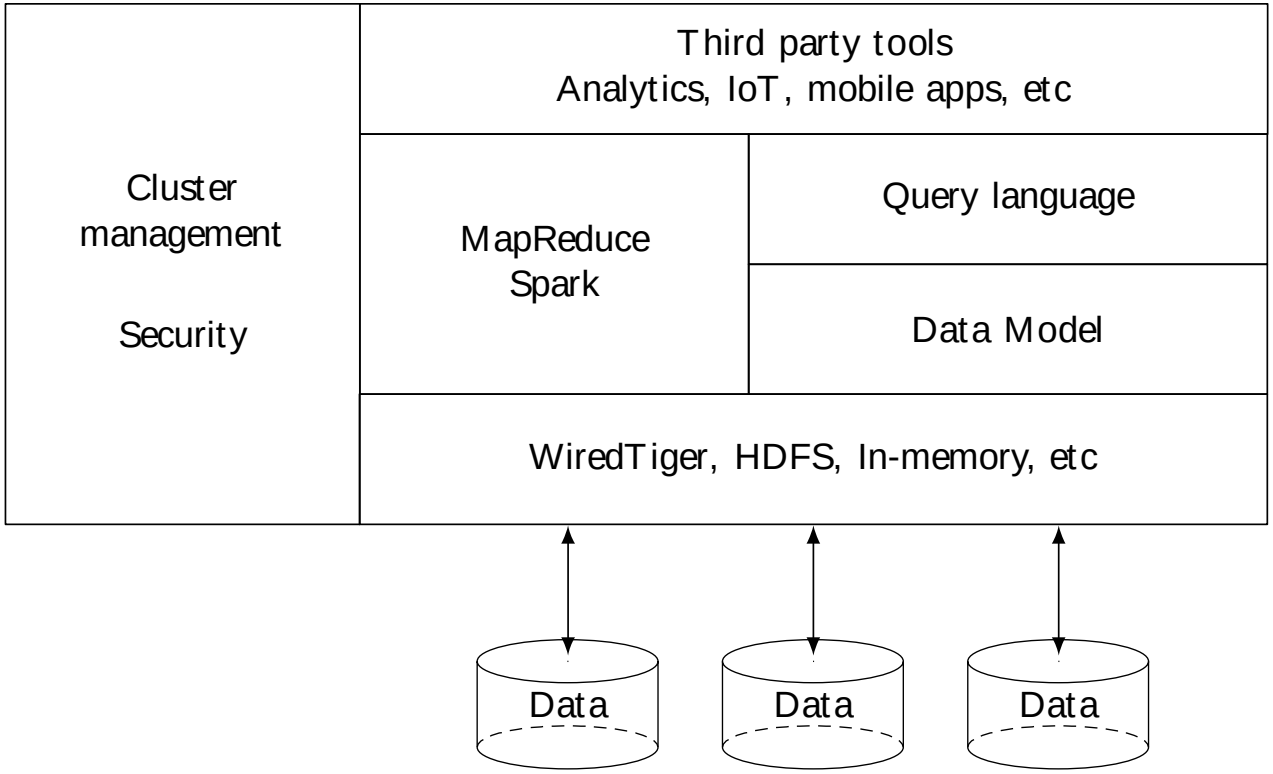
Value = JSON object with nested arrays

MongoDB – query language

- Expression of the form
  - db.nomBD.function (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

© 2020, M.T. Özsu & P. Valduriez

MongoDB - architecture



© 2020, M.T. Özsu & P. Valduriez

Main NoSQL JSON DBMSs

Vendor	Product	Langages	Comments
Apache	CouchDB	JavaScript	Open source (Apache)
Couchbase Inc.	Couchbase	N1QL	Open source (Apache)
djondb.com	djondb	JSON	
ejdb.org	EJDB	Mongo-DB like	Open source (LGPL)
linkedin	Espresso	JSON	
MarkLogic	Marklogic server	JSON	Integration with Hadoop
Mongodb.com	MongoDB	Ext. JSON	
zorba.io	Zorba	JSONiq	Open source (Apache)

© 2020, M.T. Özsu & P. Valduriez

Tabular Stores: 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
- Dynamic partitioning of tables for scalability

© 2020, M.T. Özsu & P. Valduriez

A BigTable Row

Google BigTable

Row key	Name	Email	Web page
100	"Prefix": "Dr." "Last": "Dobb"	"email: gmail.com": "dobb@gmail.com"	<!DOCTYPE html PUBLIC . . . >
101	"First": "Alice" "Last": "Martin"	"email: gmail.com": "amartin@gmail.com" "email: free.fr": "amartin@free.fr"	<!DOCTYPE html PUBLIC . . . >

- Column family = a kind of multi-valued attribute
  - Set of columns (of the same type), each identified by a key
    - Column key = attribute value, but used as a name e.g. gmail.com, free.fr
- Unit of access control and compression

BigTable API

- 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

© 2020, M.T. Özsu & P. Valduriez

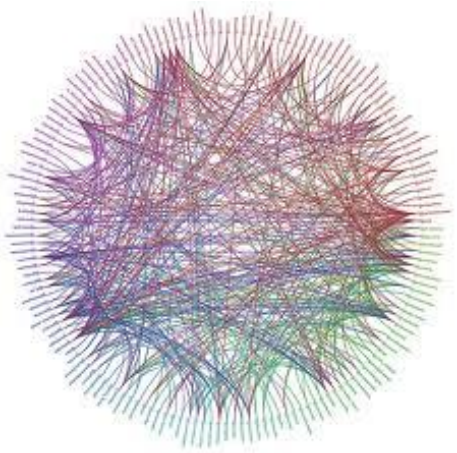
Dynamic Range Partitioning

Google BigTable

- 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

© 2020, M.T. Özsu & P. Valduriez

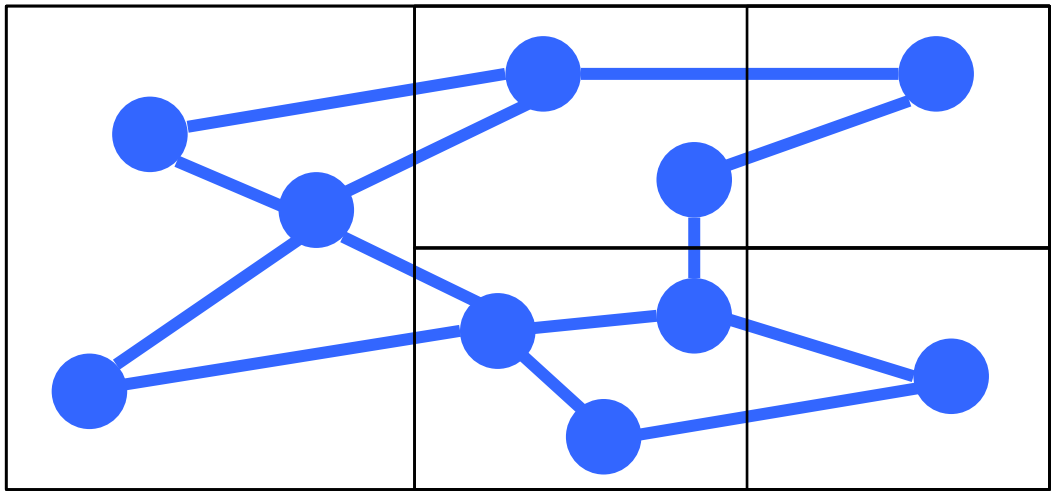
Graph DBMS



- Database graphs
  - Very big: billions of nodes and links
  - Many: millions of graphs
- Main applications
  - Social networks
    - Recommendation, sharing, sentiment analysis
  - Master data management
    - Reference business objects, data governance
  - Fraud detection in real time
    - E-commerce, insurance, etc.
  - Enterprise networks
    - Impact analysis, QoS
  - Identity management
    - Group management, provenance

© 2020, M.T. Özsu & P. Valduriez

Graph Partitioning

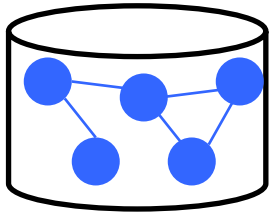


- Objective: get balanced partitions
  - NP-hard problem: no optimal algorithm
  - Solutions: approximate, heuristics, based on the graph topology
  - See Chapter 10 for details on graph partitioning

© 2020, M.T. Özsu & P. Valduriez

Neo4J

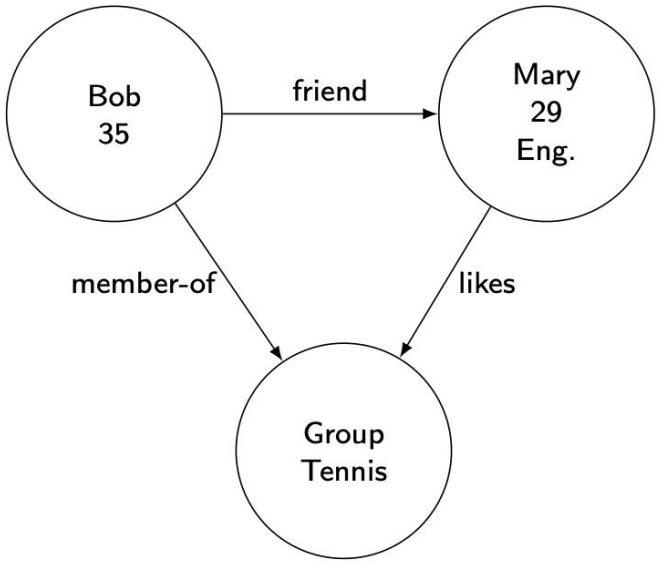
- Direct support of graphs
  - Data model, API, query language
  - Implemented by linked lists on disk
  - Optimized for graph processing
  - Transactions
- Implemented on SN cluster
  - Asymmetric replication
  - But no graph partitioning
    - Planned for a future version



© 2020, M.T. Özsu & P. Valduriez

Neo4J – data model

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



A 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");
n1.setProperty("age", 29);
n1.setProperty("job", "engineer");
n1.createRelationshipTo(n2,
    RelTypes.friend);
tx.Commit();
```

© 2020, M.T. Özsu & P. Valduriez

Neo4J - languages

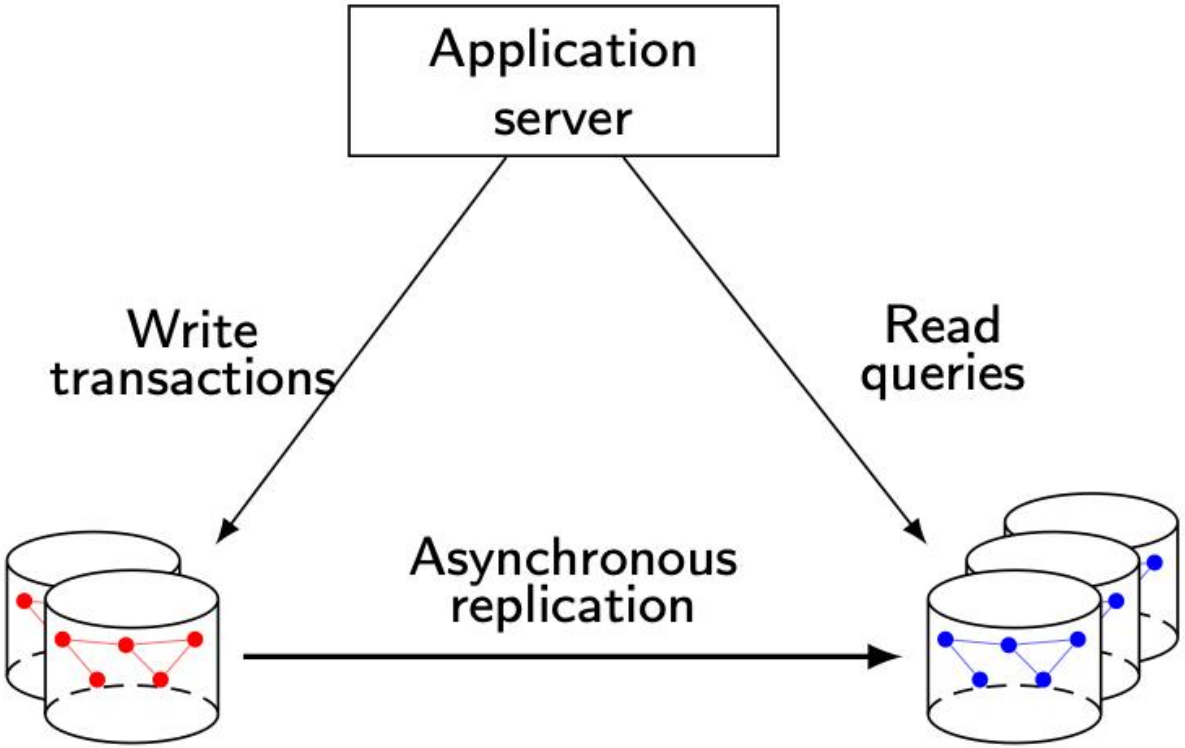
- Java API (navigational)
- Cypher query language
  - Queries and updates with graph traversals
- Support of SparQL for RDF data

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

```
MATCH bob-[:friend]->()-[:friend]->follower
WHERE bob.name="Bob" AND follower.name =~ "M.*"
RETURN bob, follower.name
```

© 2020, M.T. Özsu & P. Valduriez

Neo4J – architecture



© 2020, M.T. Özsu & P. Valduriez

Multimodel Stores: OrientDB

- Integration of key-value, document and graph
  - Extension of an objet model, with direct connections between objects/nodes
  - SQL extended with graph traversals
- Distributed architecture
  - Graph partitioning in a cluster
  - Symmetric replication between data centers
  - ACID transactions
  - Web technologies: JSON, REST

© 2020, M.T. Özsu & P. Valduriez

Main NoSQL Systems

Vendor	Product	Category	Comments
Amazon	DynamoDB	KV	Proprietary
Apache	Cassandra Accumulo	KV Tabular	Open source, Orig. Facebook Open source, Orig. NSA
Couchbase	Couchbase	KV, document	Origin: MemBase
Google	Bigtable	Tabular	Proprietary, patents
FaceBook	RocksDB	KV	Open source
Hadoop	Hbase	Tabular	Open source, Orig. Yahoo
LinkedIn	Voldemort Expresso	KV Document	Open source ACID transactions
10gen	MongoDB	Document	Open source
Oracle	NoSQL	KV	Based on BerkeleyDB
OrientDB	OrientDB	Graph, KV, document	Open source, ACID transactions
Neo4J.org	Neo4J	Graph	Open source, ACID transactions
Ubuntu	CouchDB	Document	Open source

© 2020, M.T. Özsu & P. Valduriez

Outline

- NoSQL, NewSQL and Polystores
  - Motivations
  - NoSQL systems
  - NewSQL systems
  - Polystores

NewSQL

- Pros NoSQL
  - Scalability
    - Often by relaxing strong consistency
  - Performance
  - Practical APIs for programming
- Pros Relational
  - Strong consistency
  - Transactions
  - Standard SQL
    - Makes it easy for tool vendors (BI, analytics, …)
- NewSQL = NoSQL/relational hybrid

© 2020, M.T. Özsu & P. Valduriez

Google F1

- F1 inspired by the term *hybrid filial 1* in genetics
  - For the AdWords killer app
    - More than 100 Terabytes, 100K requests per sec
    - Scalability problem with the MySQL / cluster solution
- Objective
  - "F1 combines high availability, the scalability of NoSQL systems like Bigtable, and the consistency and usability of traditional SQL databases."
- Geographic distribution of the data centers
  - Synchronous replication between data centers for high availability
  - Sharding and parallel processing within data center

© 2020, M.T. Özsu & P. Valduriez

Google F1

- Based on Spanner, a large scale distributed system
  Synchronous replication between data centers with Paxos
  - Load balancing between F1 servers
    - Favor the geographical zone of the client
- Different levels of consistency
  - ACID transactions
  - *Snapshot* (read only) transactions
    - Based on data versioning
  - Optimistic transactions (read without locking, then write)
    - Validation phase to detect conflicts and abort conflicting transactions
- Two interfaces
  - SQL
  - NoSQL key-value interface
- Hierarchical relational storage
  - Precomputed joins

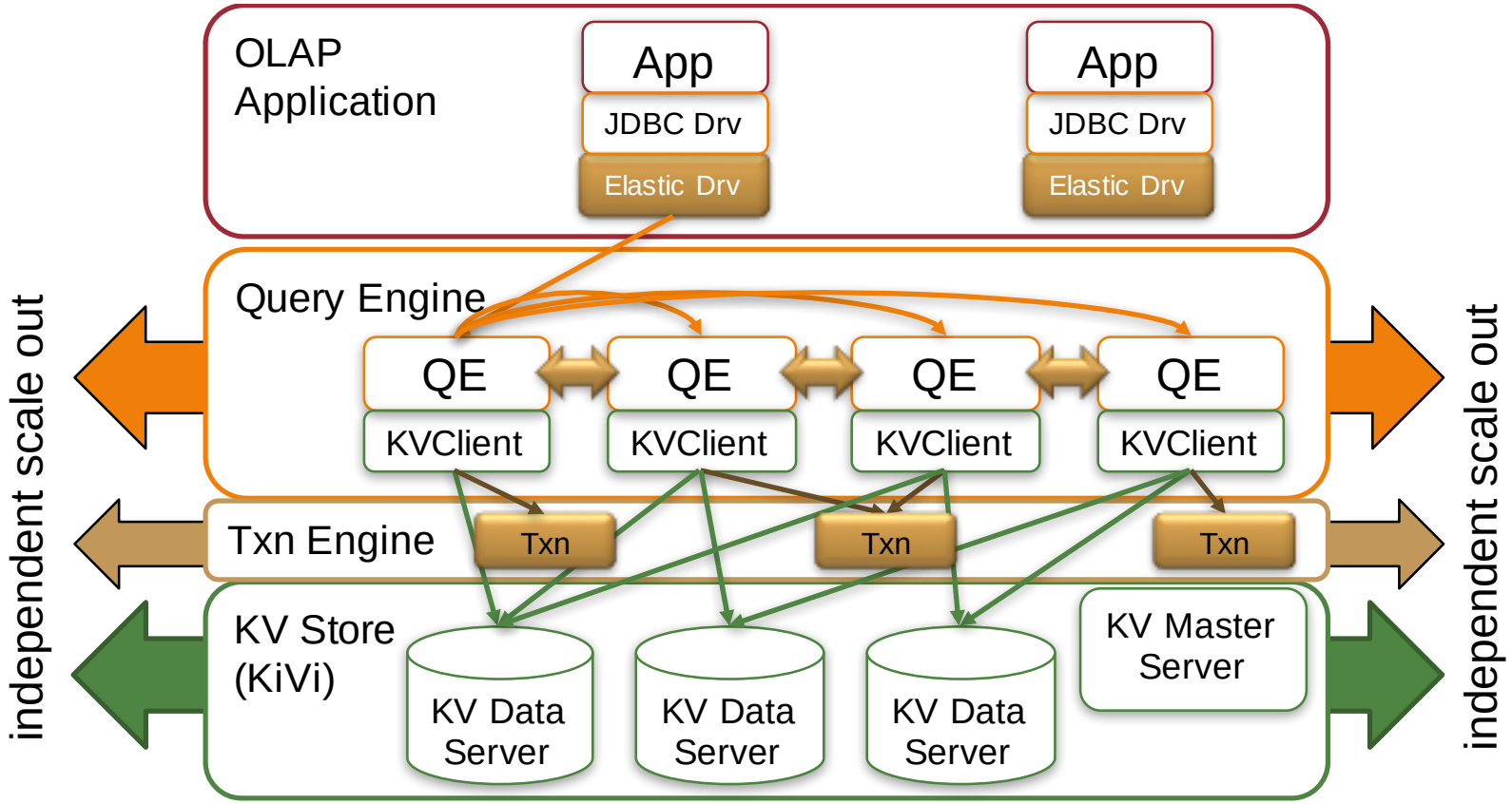
© 2020, M.T. Özsu & P. Valduriez

LeanXcale

- SQL/JSON DBMS
  - Access from a JDBC driver
- Key-value store (KiVi)
  - Dual SQL/KV interface over relational data with efficiency, elasticity, high availability, indexing, …
  - Fast, parallel data ingestion
  - Polystore access: HDFS, NoSQL, …
- OLAP parallel processing
  - Based on the Apache Calcite optimizer
  - Extensive push down of operators to KiVi
- Ultra-scalable transaction processing

© 2020, M.T. Özsu & P. Valduriez

Distributed Architecture

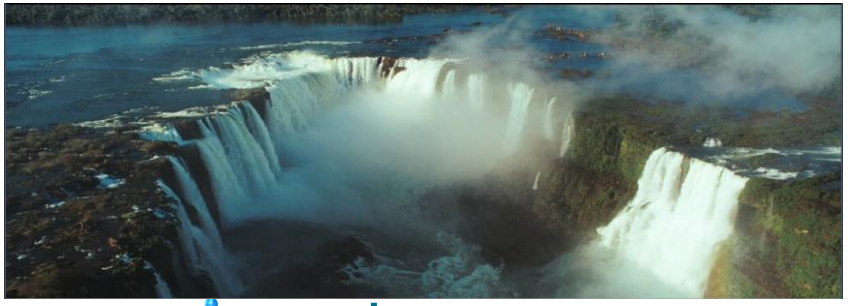


© 2020, M.T. Özsu & P. Valduriez

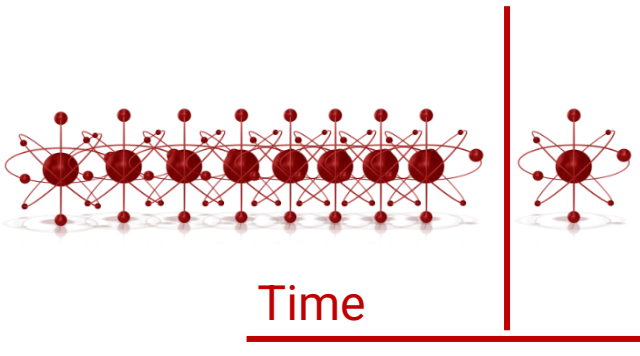
Transaction Processing

LEANXCALE

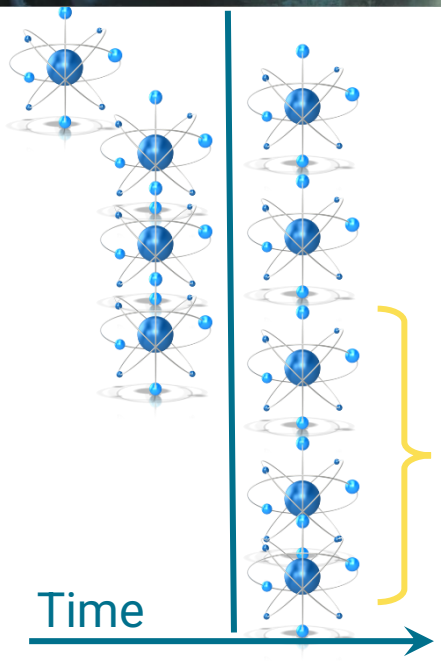
Traditional approach



Single-node bottleneck



vs



Processes & commits transactions **in parallel**

Provides a **consistent view**

© 2020, M.T. Özsu & P. Valduriez

Main NewSQL systems

Vendor	Product	Objective	Comment
Clustrix Inc., San Francisco	Clustrix	Analytics and transactional	First version out in 2006
CockroachDB Labs, NY	CockroachDB	Transactional	By ex-googlers. Open source inspired by F1, based on RocksDB
Google	F1/Spanner	Transactional	Proprietary
SAP	HANA	Analytics	In-memory, column-oriented
MemSQL Inc.	MemSQL	Analytics	In-memory, column/row-oriented, compatible with MySQL
LeanXcale, Madrid	LeanXcale	Analytics and transactional	Based on Apache Derby and Hbase Multistore access (KV, Hadoop, CEP, etc.)
NuoDB, Cambridge	NuoDB	Analytics and transactionall	Solution cloud (Amazon)
GitHub	TiDB	Transactional	Open source inspired by Google F1
VoltDB Inc.	VoltDB	Analytics and transactional	Open source and proprietary versions In-memory

© 2020, M.T. Özsu & P. Valduriez

Which Data Store for What?

Category	Systems	Requirements
Key-value	DynamoDB, SimpleDB, Cassandra	Access by key Flexibility (no schema) Very high scalability and performance
Document	MongoDB, CouchDB, Expresso	Web content management Flexibility (no schema) Limited transactions
Tabular	BigTable, Hbase, Accumulo	Very big collections Scalability and high availability
Graph	Neo4J, Sparcity, Titan	Efficient storage and management of large graphs
Multimodel	OrientDB, ArangoDB	Integrated key-value, document and graph management
NewSQL	Google F1, CockroachDB, VoltDB	ACID transactions , flexibility and scalability SQL and key-value access

© 2020, M.T. Özsu & P. Valduriez

Outline

- NoSQL, NewSQL and Polystores
  - Motivations
  - NoSQL systems
  - NewSQL systems
  - Polystores

Polystores

- Also called *Multistores*
- Provide integrated access to multiple cloud data stores such as NoSQL, HDFS and RDBMS
- Great for integrating structured (relational) data and big data
- Much more difficult than distributed databases
- A major area of research & development

Differences with Distributed Databases

- Multidatabase systems
  - A few databases (e.g. less than 10)
    - Corporate DBs
  - Powerful queries (with updates and transactions)
- Web data integration systems
  - Many data sources (e.g. 1000's)
    - DBs or files behind a web server
  - Simple queries (read-only)
- Mediator/wrapper architecture
- In the cloud, more opportunities for an efficient multistore architecture
  - No restriction to where mediator and wrapper components need be installed

© 2020, M.T. Özsu & P. Valduriez

Classification of Polystores

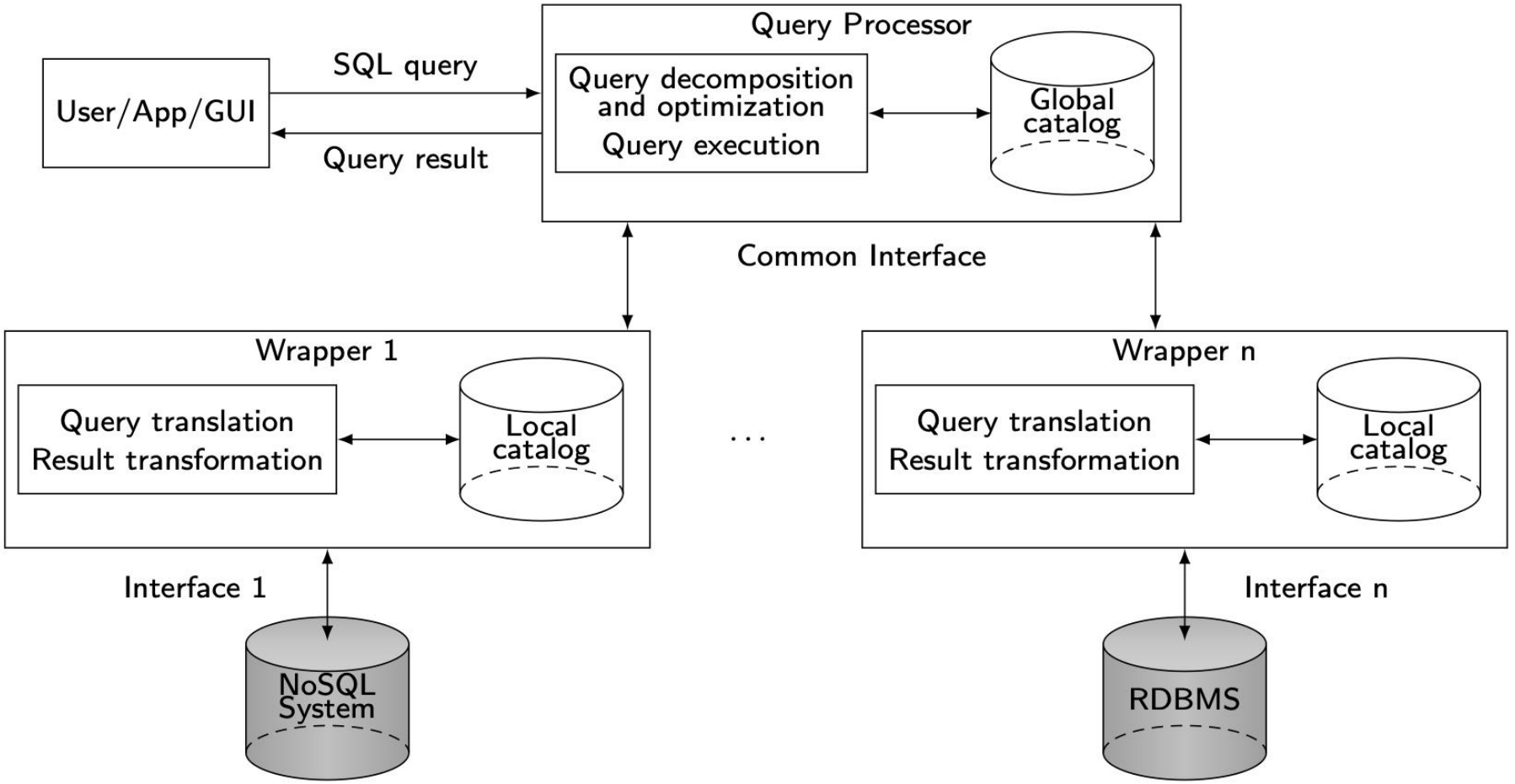
- We divide polystores based on the level of coupling with the underlying data stores
  - Loosely-coupled
  - Tightly-coupled
  - Hybrid

Loosely-coupled Polystores

- Reminiscent of multidatabase systems
  - Mediator-wrapper architecture
  - Deal with autonomous data stores
  - One common interface to all data stores
  - Common interface translated to local API
- Examples
  - BigIntegrator (Uppsala University)
  - Forward (UC San Diego)
  - QoX (HP Labs)

© 2020, M.T. Özsu & P. Valduriez

Architecture

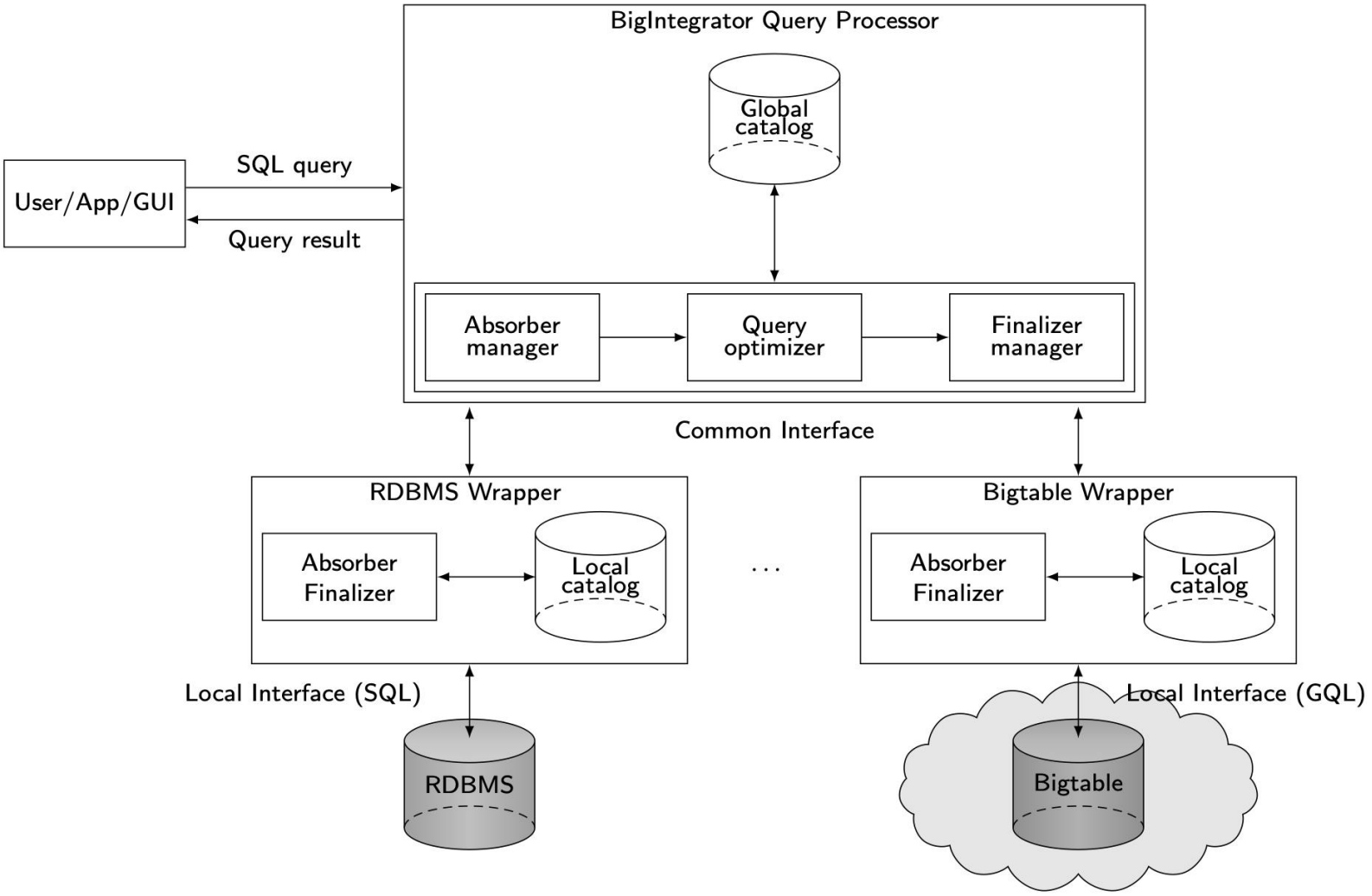


Big Integrator

- Combines Bigtable data stores in the cloud and relational data stores
- SQL-like queries
  - Google Query Language (GQL)
  - No join, only basic select predicates
- Query processing mechanism based on plugins
  - Absorber and finalizer
- Uses Local As View approach
  - To define the global schema of the Bigtable and relational data sources as flat relational tables

© 2020, M.T. Özsu & P. Valduriez

Big Integrator Architecture



Forward

- Supports SQL++
  - SQL-like language
  - Semi-structured data model
    - Extends JSON and relational data models
- Rich web development frameworks
  - Integrate visualization components (e.g., Google Maps)
- Global As View approach for the global schema
  - Each data source (SQL or NoSQL) appears to the user as an SQL++ virtual view, defined over SQL++ collections
- Architecture
  - Query processor
    - Performs SQL++ query decomposition
    - Cost based optimization
  - One wrapper per data store

© 2020, M.T. Özsu & P. Valduriez

QoX

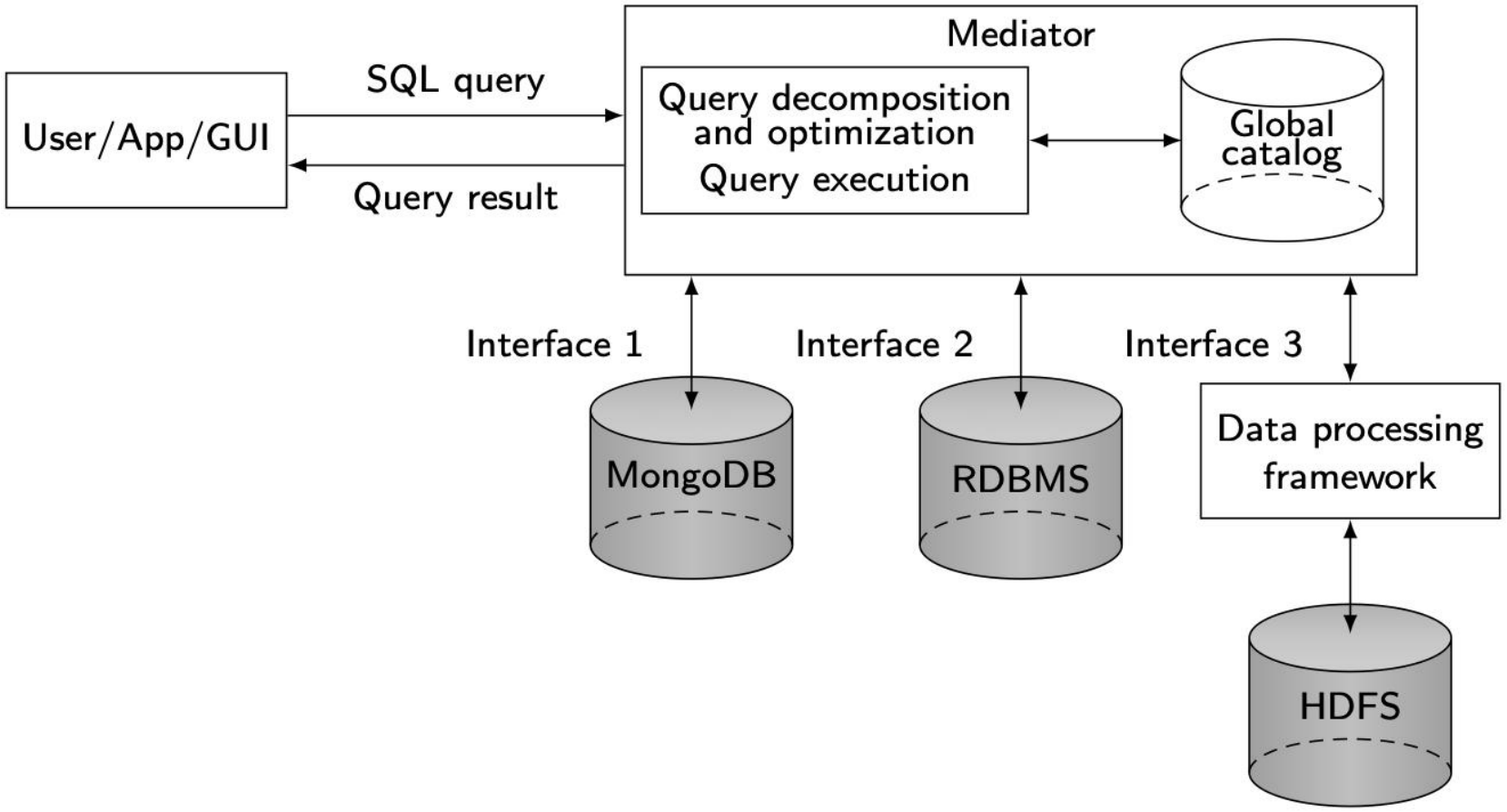
- Integrates data from relational databases and various execution engines (MapReduce or ETL)
  - Relational data model
  - SQL-like language
- Queries are analytical data-driven workflows (or dataflows)
- QoX optimizer
  - Integrates the back-end ETL pipeline and the front-end query operations into a single analytics pipeline
  - Evaluates alternative execution plans, estimates theirs costs and generates executable code

© 2020, M.T. Özsu & P. Valduriez

Tightly-coupled Polystores

- Use the local interfaces of the data stores
- Use a single query language for data integration in the query processor
- Allow data movement across data stores
- Optimize queries using materialized views or indexes
- Examples
  - Polybase (Microsoft Research, Madison)
  - HadoopDB (Yale Univ. & Brown Univ.)
  - Estocada (Inria)

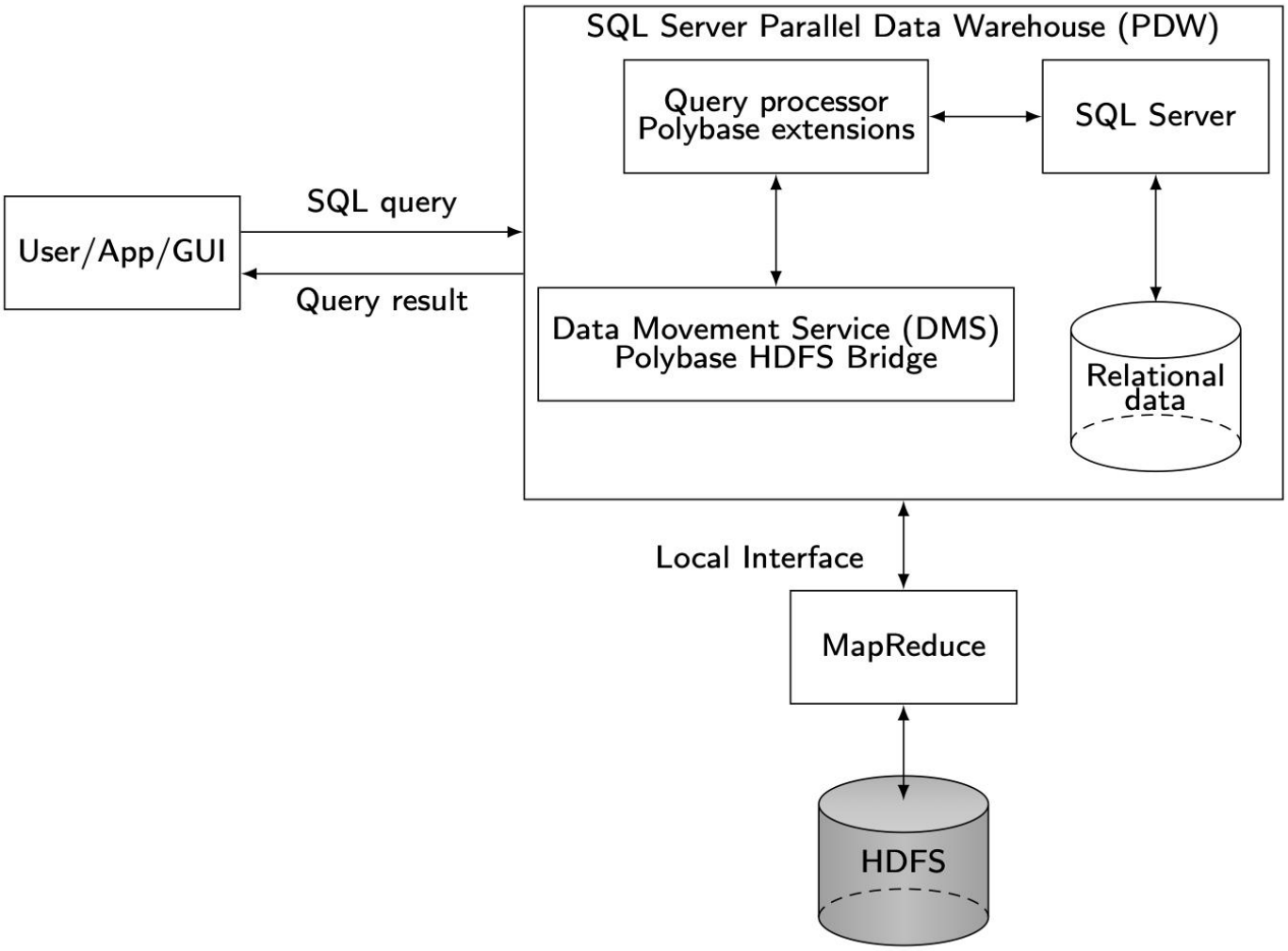
Architecture



Polybase

- Feature of the SQL Server Parallel Data Warehouse (PDW) to query and integrate HDFS through SQL
- HDFS data can be referenced in Polybase as external tables
- Using the PDW query optimizer, SQL operators on HDFS data are translated into MapReduce jobs to be executed directly on the Hadoop cluster

Polybase Architecture



HadoopDB

- Tightly couples the Hadoop framework, including MapReduce and HDFS, with multiple single-node RDBMS (e.g. PostgreSQL or MySQL) deployed across a cluster
- Extends the Hadoop architecture with four components
  - Database connector
  - Catalog
  - Data loader
  - SQL-MapReduce-SQL (SMS) planner

Estocada

- Self-tuning polystore
  - Automatic data distribution and partitioning across the different data stores
  - Each data partition is internally described as a materialized view over one or several data collection
- Query processor
  - Deals with single model queries only, each expressed in the query language of the corresponding data source
    - To integrate various data sources, one would need a common data model and language on top of Estocada
  - Query processing involves view-based query rewriting and cost-based optimization

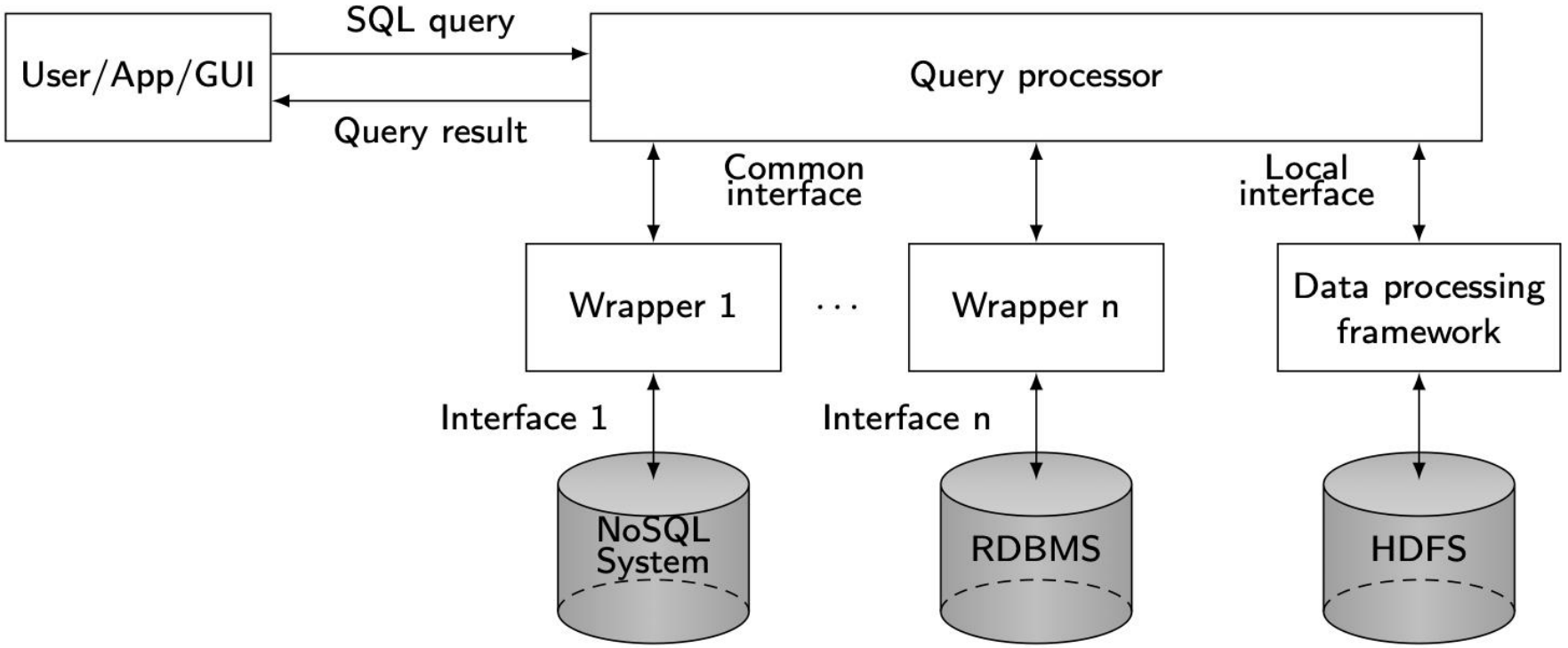
© 2020, M.T. Özsu & P. Valduriez

Hybrid Polystores

- Support data source autonomy as in loosely-coupled systems
- Exploit the local data source interface as in tightly-coupled systems
- Examples
  - SparkSQL (Databricks & UC Berkeley)
  - CloudMdsQL (Inria & LeanXcale)
  - BigDaWG (MIT, U. Chicago & Intel)

© 2020, M.T. Özsu & P. Valduriez

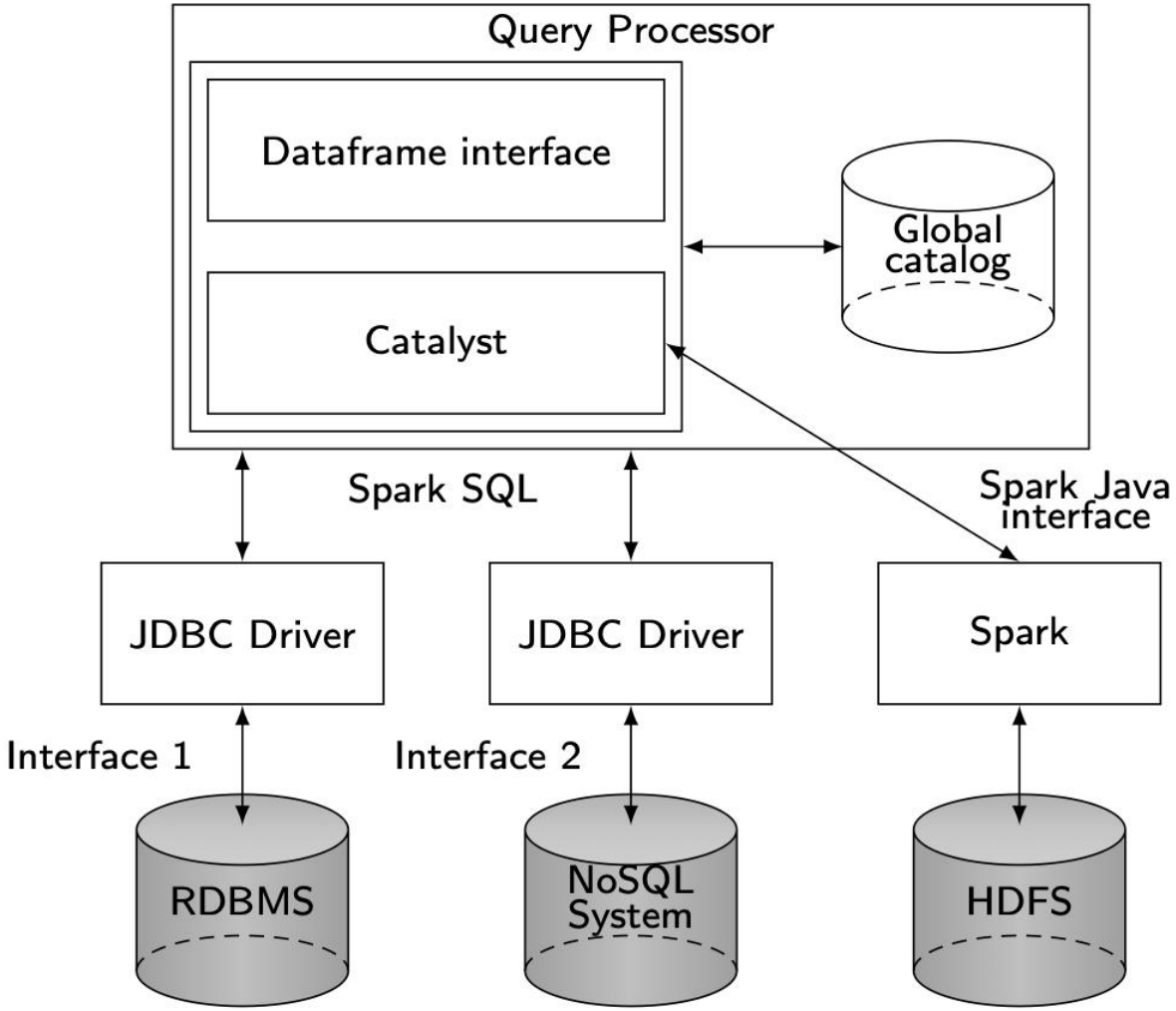
Architecture



SparkSQL

- Runs as a library on top of Spark
- The query processor
  - Directly accesses the Spark engine through the Spark Java interface
  - Accesses external data sources (e.g. an RDBMS or a key-value store) through the Spark SQL common interface supported by wrappers (JDBC drivers)
- Extensible query optimizer
- In-memory caching using columnar storage

SparkSQL Architecture



© 2020, M.T. Özsu & P. Valduriez

CloudMdsQL

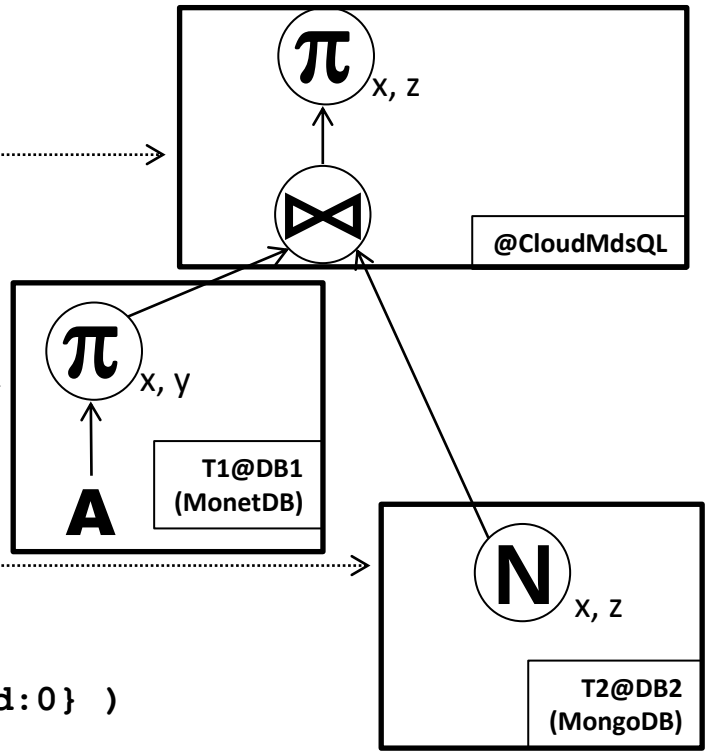
- JSON-based data model
  - With rich data types
  - To allow computing on typed values
  - No global schema and schema mappings to define
- Functional-style SQL
  - Can represent all query building blocks as functions
  - A function can be expressed in one of the DB languages
  - Function results can be used as input to subsequent functions
- Fully distributed architecture exploited by the query processor
  - Select pushdown and bind join optimization
  - Operator ordering
  - Reducing data transfers between nodes

© 2020, M.T. Özsu & P. Valduriez

CloudMdsQL Example

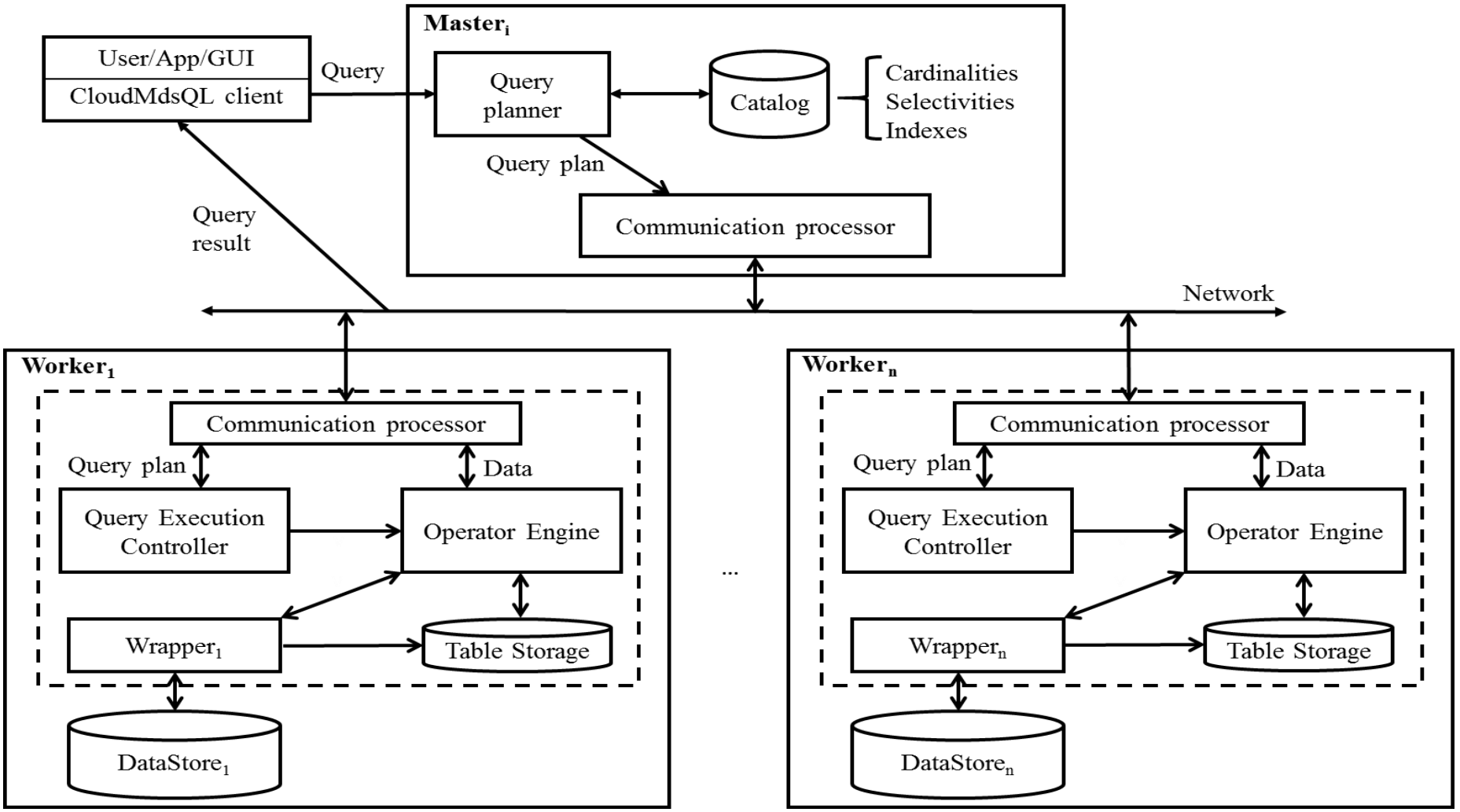
- A query that integrates data from:
  - DB1 – relational (MonetDB)
  - DB2 – document (MongoDB)

```
/* Integration query */
SELECT T1.x, T2.z
FROM T1 JOIN T2
  ON T1.x = T2.x

/* SQL sub-query */
T1(x int, y int)@DB1 =
( SELECT x, y FROM A )

/* Native sub-query */
T2(x int, z string)@DB2 =
{*
  db.B.find( {$lt: {x, 10}}, {x:1, z:1, _id:0} )
*}
```



© 2020, M.T. Özsu & P. Valduriez

CloudMdsQL Distributed Query Engine



MFR Statement

- Sequence of Map/Filter/Reduce operations on datasets for big data frameworks (e.g. Spark)
  - Example: count the words that contain the string ‘cloud’

- A dataset is an abstration for a set of tuples, a Spark RDD
  - Consists of key-value tuples
  - Processed by MFR operations

© 2020, M.T. Özsu & P. Valduriez

MFR Example

Query: retrieve data from RDBMS and HDFS

```
/* Integration subquery*/
SELECT title, kw, count FROM T1 JOIN T2 ON T1.kw = T2.word
WHERE T1.kw LIKE '%cloud%'

/* SQL subquery */
T1(title string, kw string)@rdbms = ( SELECT title, kw FROM tbl )

/* MFR subquery */
T2(word string, count int)@hdfs = {*
       SCAN(TEXT,'words.txt')
       .MAP(KEY,1)
       .REDUCE(SUM)
       .PROJECT(KEY,VALUE)  *}
```

BigDAWG

- No common data model and language
- Key abstraction: island of information
  - A collection of data stores accessed with a single query language
  - Examples of islands: relational, array, NoSQL, DSMS
- Within an island, there is loose-coupling of the data stores, which need to provide a wrapper island language to their native one
- Query processing within an island using distributed techniques
  - Function shipping
  - Data shipping

© 2020, M.T. Özsu & P. Valduriez

Comparisons: functionality

Polystore	Objective	Data model	Query language	Data stores
Loosely-coupled				
BigIntegrator	Querying relational and cloud data	Relational	SQL-like	BigTable,RDBMS
Forward	Unyfing relational and NoSQL	JSON-based	SQL++	RDBMS, NoSQL
QoX	Analytic data flows	Graph	XML based	RDBMS, ETL
Tightly-coupled				
Polybase	Querying Hadoop from RDBMS	Relational	SQL	HDFS, RDBMS
HadoopDB	Querying RDBMS from Hadoop	Relational	SQL-live (HiveQL)	HDFS, RDBMS
Estocada	Self-tuning	No common model	Native QL	RDBMS, NoSQL
Hybrid				
SparkSQL	SQL on top of Spark	Nested	SQL-like	HDFS, RDBMS
BigDAWG	Unifying relational and NoSQL	No common model	Island query languages	RDBMS, NoSQL, Array DBMS, DSMSs
CloudMdsQL	Querying relational and NoSQL	JSON-based	SQL-like with native subqueries	RDBMS, NoSQL, HDFS

Comparisons: implementation techniques

Polystore	Objective	Data model	Query language	Data stores
Loosely-coupled				
BigIntegrator	Importer,absorber, finalizer	LAV	Access filters	Heuristics
Forward	Query processor	GAV	Data store capabilities	Cost-based
QoX	Dataflow engine	No	Data/function shipping	Cost-based
Tightly-coupled				
Polybase	HDFS bridge	GAV	Query splitting	Cost-based
HadoopDB	SMS planer, db conector	GAV	Query splitting	Heuristics
Estocada	Storage advisor	Materialzed views	Query rewriting	Cost-based
Hybrid				
SparkSQL	Dataframes	Nested	In-memory caching	Cost-based
BigDAWG	Island query	GAV within islands	Function/datashipping	Heuristics
CloudMdsQL	Query planner	No	Bind join	Cost-based

Conclusions for Polystores

1. The ability to integrate relational data (stored in RDBMS) with other kinds of data stores
2. The growing importance of accessing HDFS within Hadoop
3. Most systems provide a relational/ SQL-like abstraction
   - QoX has a more general graph abstraction to capture analytic dataflows
   - BigDAWG allows the data stores to be directly accessed with their native (or island) languages

© 2020, M.T. Özsu & P. Valduriez