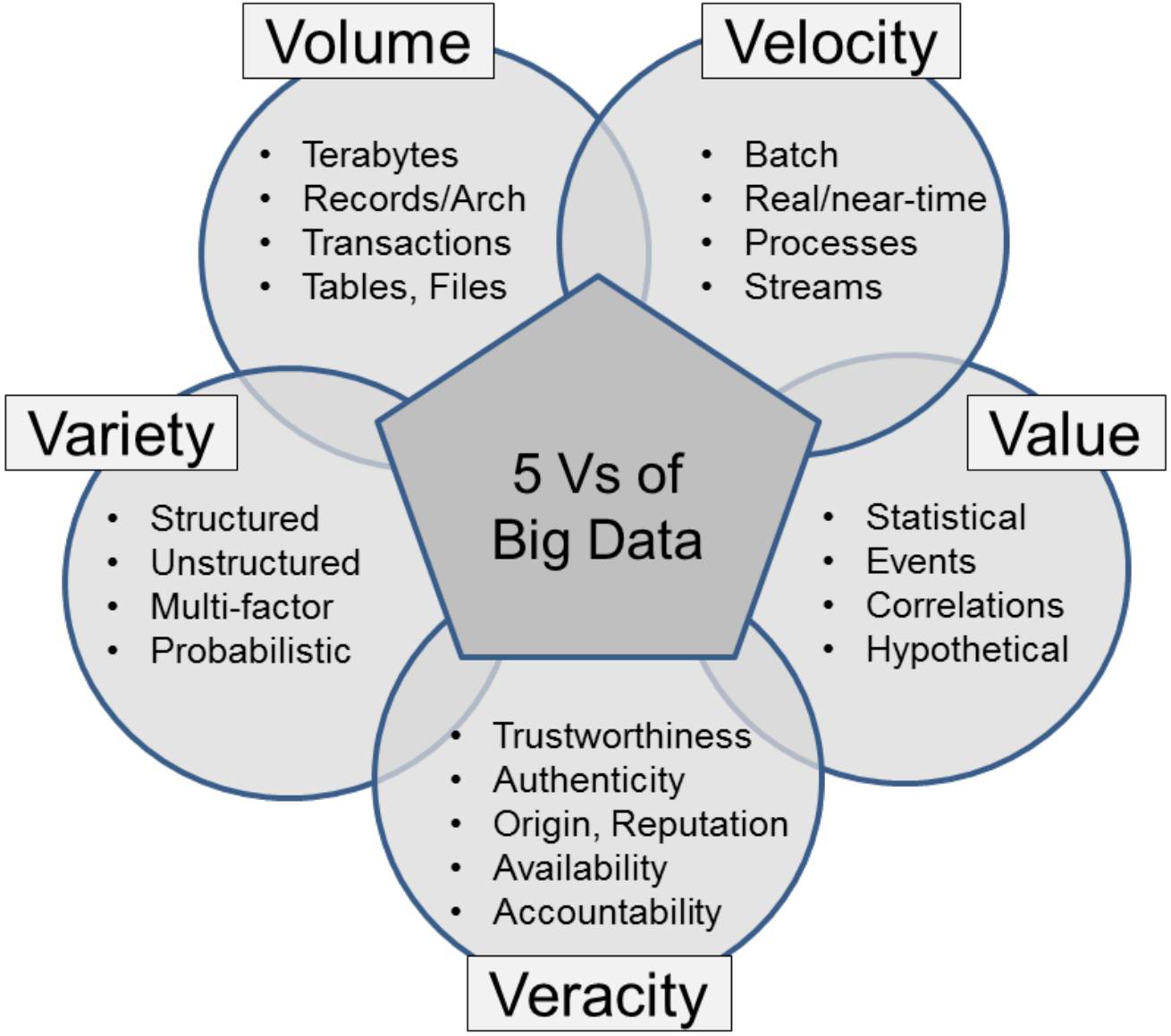
Scalable Data Management
Cloud Data Management

Vera Goebel

Thomas Plagemann

Big Data Management & Processing
Cloud Computing
CAP Theorem
NoSQL, NewSQL, PolyStores
Scalable DSP
Lazy Migration

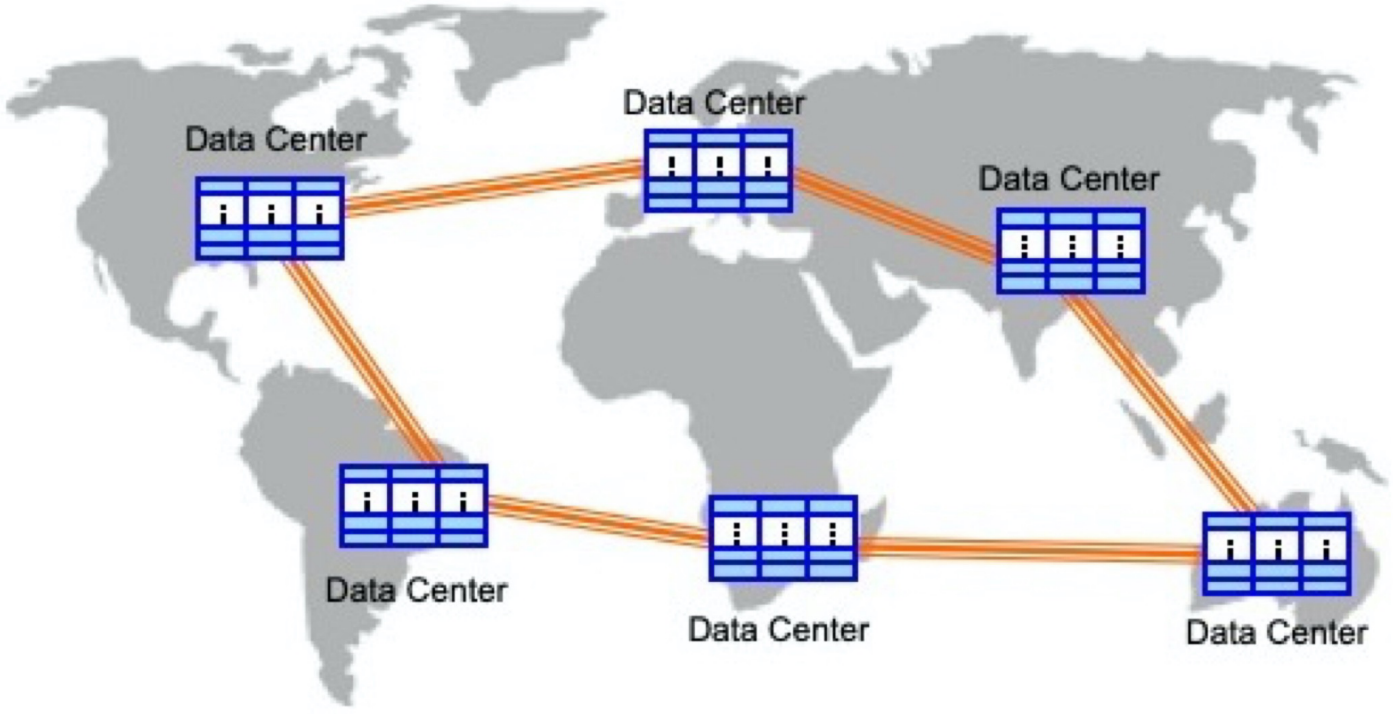
Big Data – Main Characteristics



Five Vs

- Volume
  - Increasing data size: petabytes (10^{15}) to zettabytes (10^{21})
- Variety (heterogeneity)
  - Multimodal data: structured, images, text, audio, video
  - 90% of currently generated data unstructured
- Velocity
  - Streaming data at high speed
  - Real-time processing
- Veracity
  - Data quality, consistency, data trustworthiness
- Value
  - Data analysis
  - Added-value to data

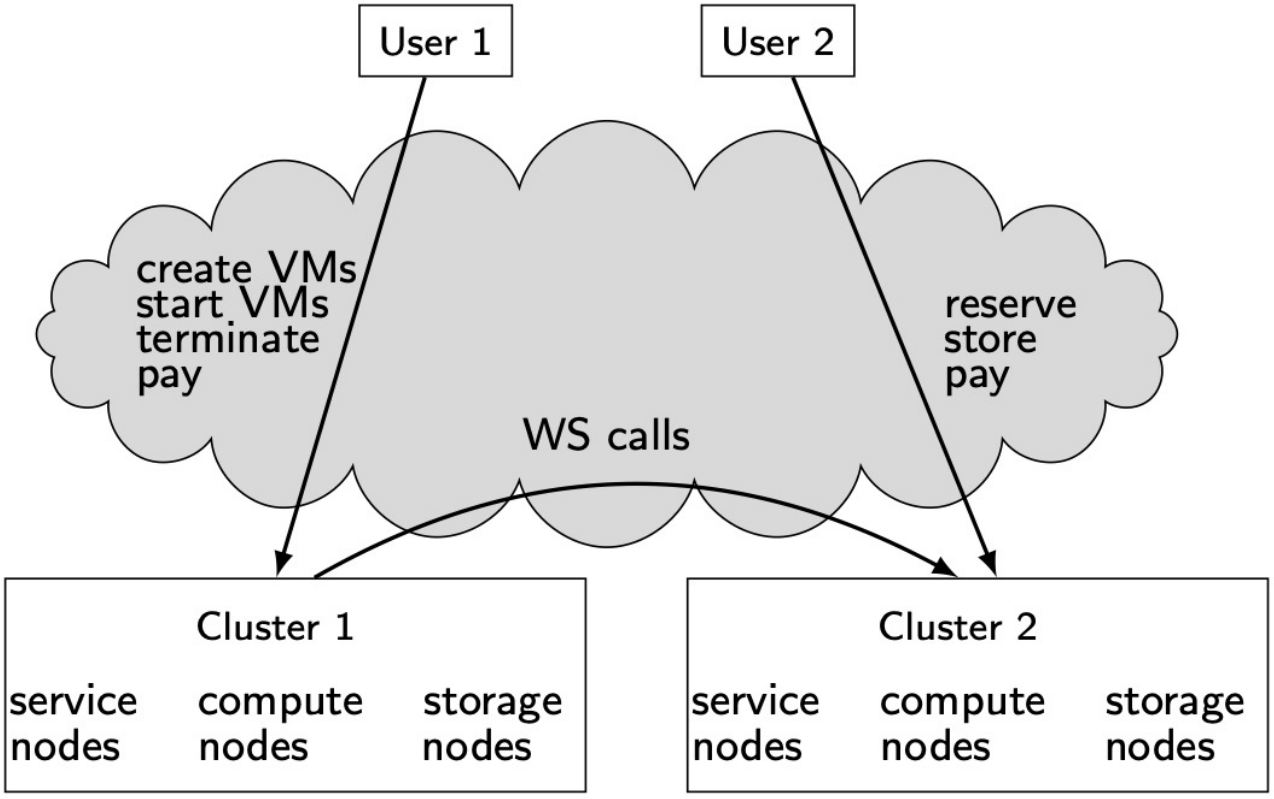
Geo-Distributed Data Centers



Cloud Computing & Architecture

On-demand, reliable services provided over the Internet in a cost-efficient manner

- Cost savings: no need to maintain dedicated compute power
- Elasticity: better adaptivity to changing workload



Cloud Computing

On-demand, reliable services provided over the Internet in a cost-efficient manner (use Data Centers)

- IaaS – Infrastructure-as-a-Service
- PaaS – Platform-as-a-Service
- SaaS – Software-as-a-Service
- DaaS – Database-as-a-Service

Scalability

- Issue is database scaling and workload scaling
- Adding processing and storage power
- Scale-out: add more servers
  - Scale-up: increase the capacity of one server → has limits

Cloud Taxonomy

- Infrastructure-as-a-Service (IaaS)
  - Computing, networking and storage resources, as a service
  - Provides elasticity: ability to scale up (add more resources) or scale down (release resources) as needed
  - Example: Amazon Web Services: Virtual machine EC2
- Software-as-a-Service (SaaS)
  - Application software as a service
  - Generalizes the earlier ASP model with tools to integrate other applications, e.g. developed by the customer (using the cloud platform)
  - Hosted applications: from simple (email, calendar) to complex (CRM, data analysis or social network)
  - Example: Safesforce CRM system
- Platform-as-a-Service (PaaS)
  - Computing platform with development tools and APIs as a service
  - Enables developers to create and deploy custom applications directly on the cloud infrastructure and integrate them with applications provided as SaaS
  - Example: Google Apps

Cloud Characteristics

- Compute power is elastic, but only if workload is parallelizable! -> Applications must be designed to run on shared-nothing architecture
- Data is stored at untrusted hosts! Security and Privacy?
- Data is replicated, often across large geographic distances -> Transactions? Consistency? Security and Privacy?

Main Issue: Security and Privacy

- Current solutions
  - Internal cloud (or private cloud): the use of cloud technologies but in a private network behind a firewall
    - Much tighter security
    - 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 and requires talented security administrators*

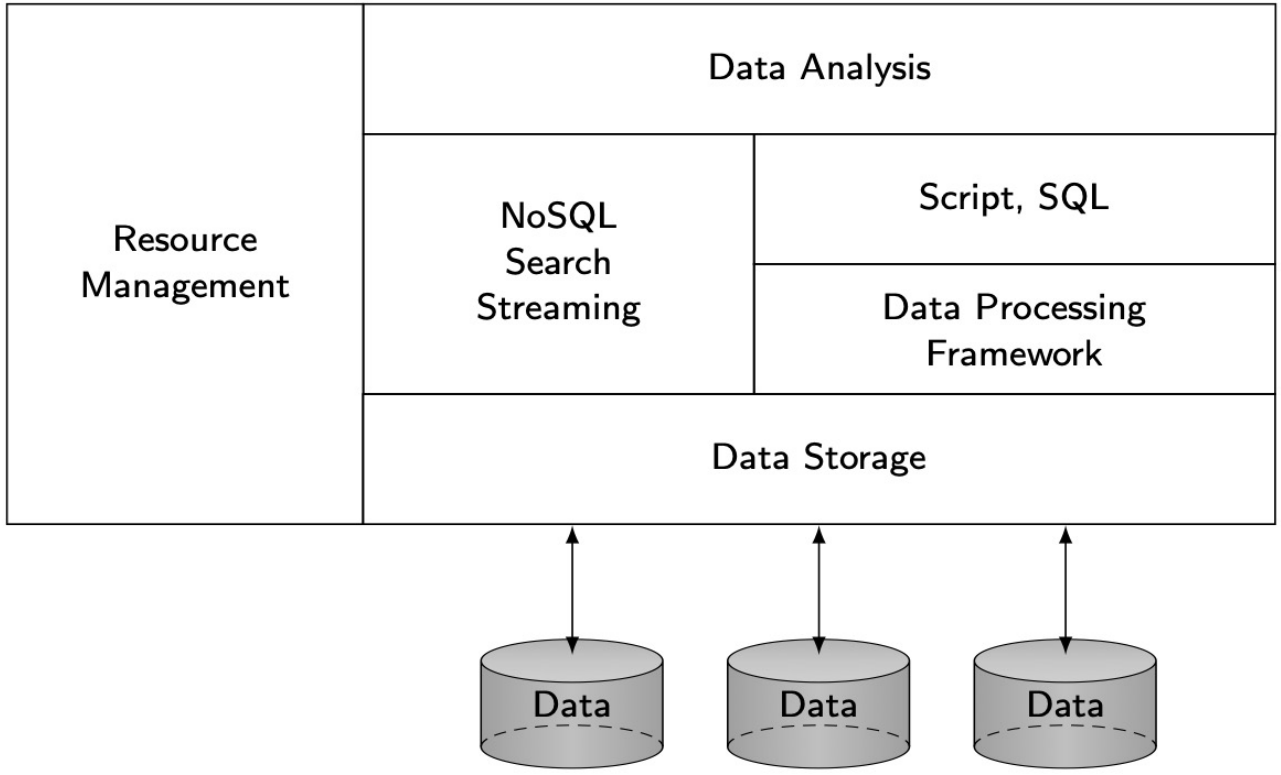
Data Management in the Cloud

- Data management applications are potential candidates for deployment in the cloud
  - **industry:** enterprise database system have significant up-front cost that includes both hardware and software costs
  - **academia:** manage, process and share mass-produced data in the cloud
- Many “Cloud Killer Apps” are in fact data-intensive
  - Batch Processing as with map/reduce
  - Online Transaction Processing (OLTP) as in automated business applications
  - Online Analytical Processing (OLAP) as in data mining or machine learning (Data Warehouses)

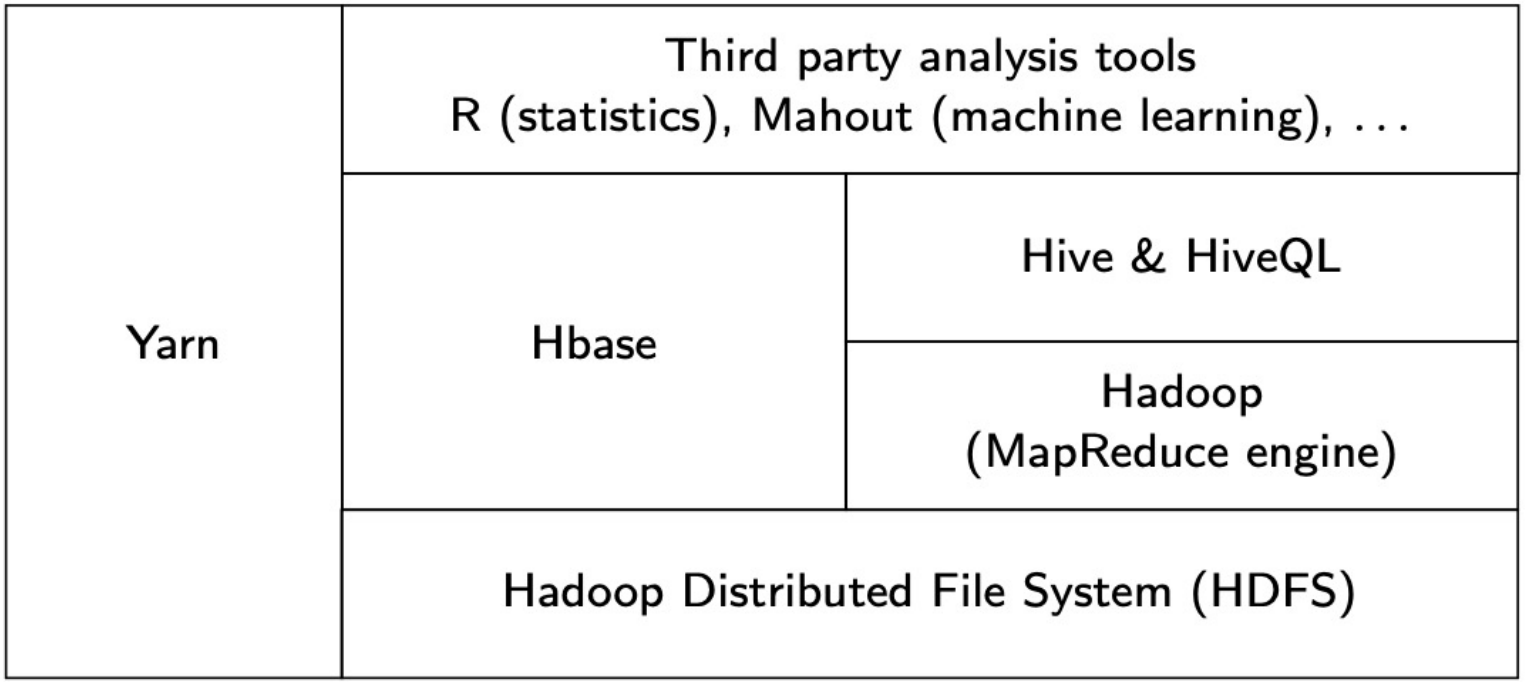
Big Data Processing Platforms

- Applications that do not need full DBMS functionality
  - Data analysis of very large data sets
  - Highly dynamic, irregular, schemaless, …
- Parallelization problems
- MapReduce/Spark
- Advantages
  - Flexibility
  - Scalability
  - Efficiency
  - Fault-tolerance
- Disadvantage
  - Reduced functionality
  - Increased programming effort

Big Data Software Stack



Hadoop Stack



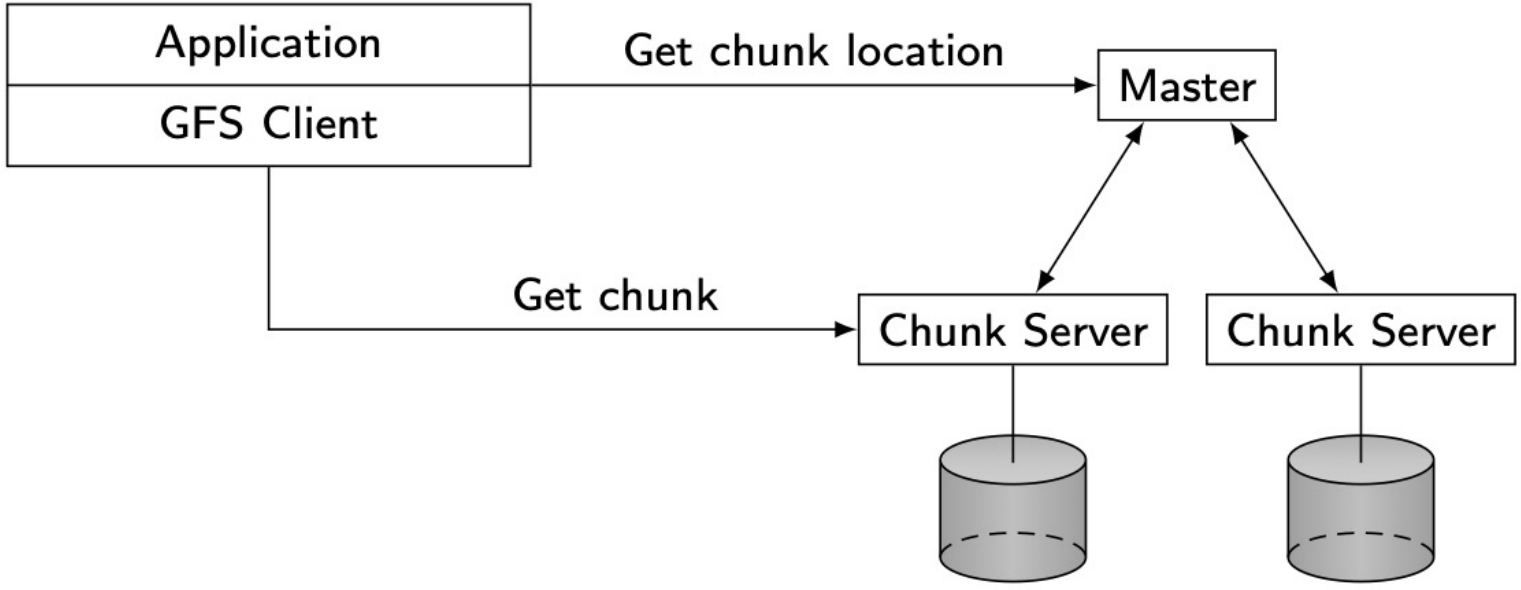
Distributed Storage System

Storing and managing data across the nodes of shared-nothing cluster

- Object-based
  - Object = ⟨oid, data, metadata⟩
  - Metadata can be different for different object
  - Easy to move
  - Flat object space → billions/trillions of objects
  - Easily accessed through REST-based API (`get`/`put`)
  - Good for high number of small objects (photos, mail attachments)
- File-based
  - Data in files of fixed- or variable-length records
  - Metadata-per-file stored separately from file
  - For large data, a file needs to be partitioned and distributed

Google File System (GFS)

- Targets shared-nothing clusters of thousands of machines
- Targets applications with characteristics:
  - Very large files (several gigabytes)
  - Mostly read and append workloads
  - High throughput more important than low latency
- Interface: `create`, `open`, `read`, `write`, `close`, `delete`, `snapshot`, `record append`



Scalable Data Management

- Horizontal Scaling
  -> performance and scalability – throughput (OLTP) / data analysis (DW)
  -> parallel data access/storage/processing
- Shared Nothing architectures
- Applications: big web players, IoT, Data Analysis (DW)
- Big data: (often) unstructured data
- Data interconnexion: Hyperlinks, tags, blogs, …
- 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
- CAP Theorem - a shared-data system can only choose at most 2 out of 3 properties: Consistency, Availability, and Tolerance to Partitions

CAP Theorem

- Polemical topic
  - “A database cannot 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

What is the CAP Theorem?

- 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

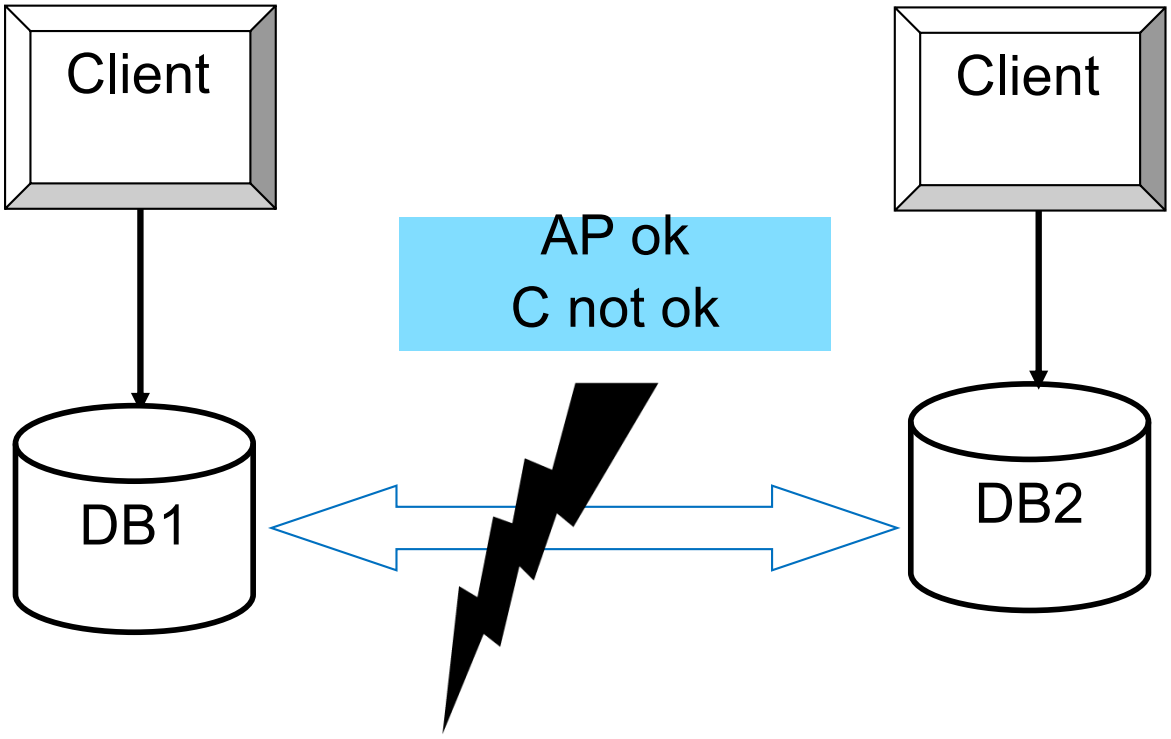
System	Data Model	Query Interface	Consistency	CAP Options	License
BigTable	Column Families	Low-Level API	Eventually Consistent	CP	Internal at Google
Google AppEng	Objects Store	Python API - GQL	Strictly Consistent	CP	Commercial
PNUTS	Key-Value Store	Low-Level API	Timeline Consistent	AP	Internal at Yahoo
Dynamo	Key-Value Store	Low-Level API	Eventually Consistent	AP	Internal at Amazon
S3	Large Objects Store	Low-Level API	Eventually Consistent	AP	Commercial
SimpleDB	Key-Value Store	Low-Level API	Eventually Consistent	AP	Commercial
RDS	Relational Store	SQL	Strictly Consistent	CA	Commercial
SQL Azure	Relational Store	SQL	Strictly Consistent	CA	Commercial
Cassandra	Column Families	Low-Level API	Eventually Consistent	AP	Open source - Apache
Hypertable	Multi-dimensional Table	Low-Level API, HQL	Eventually Consistent	AP	Open source - GNU
CouchDB	Document-Oriented Store	Low-Level API	Optimistically Consistent	AP	Open source - Apache

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

Symmetric, Asynchronous Replication



But we have eventual consistency

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

NoSQL (Not Only SQL): definition

- Specific “DBMS” for web-based data (Data Store)
  - Specialized data model
    - Key-value, table, document, graph, multimodal
  - 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: 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

Key-value Stores

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

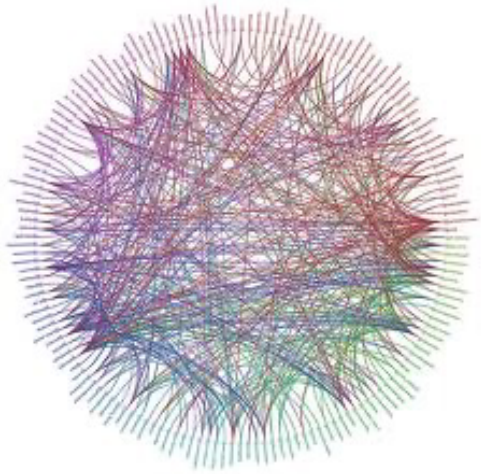
Document Stores

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

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+, ...
- 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

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, ...
  - Enterprise networks
    - Impact analysis, QoS
  - Identity management
    - Group management, provenance

Multimodel Stores: OrientDB

- Integration of key-value, document and graph
  - Extension of object 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

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

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

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

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

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

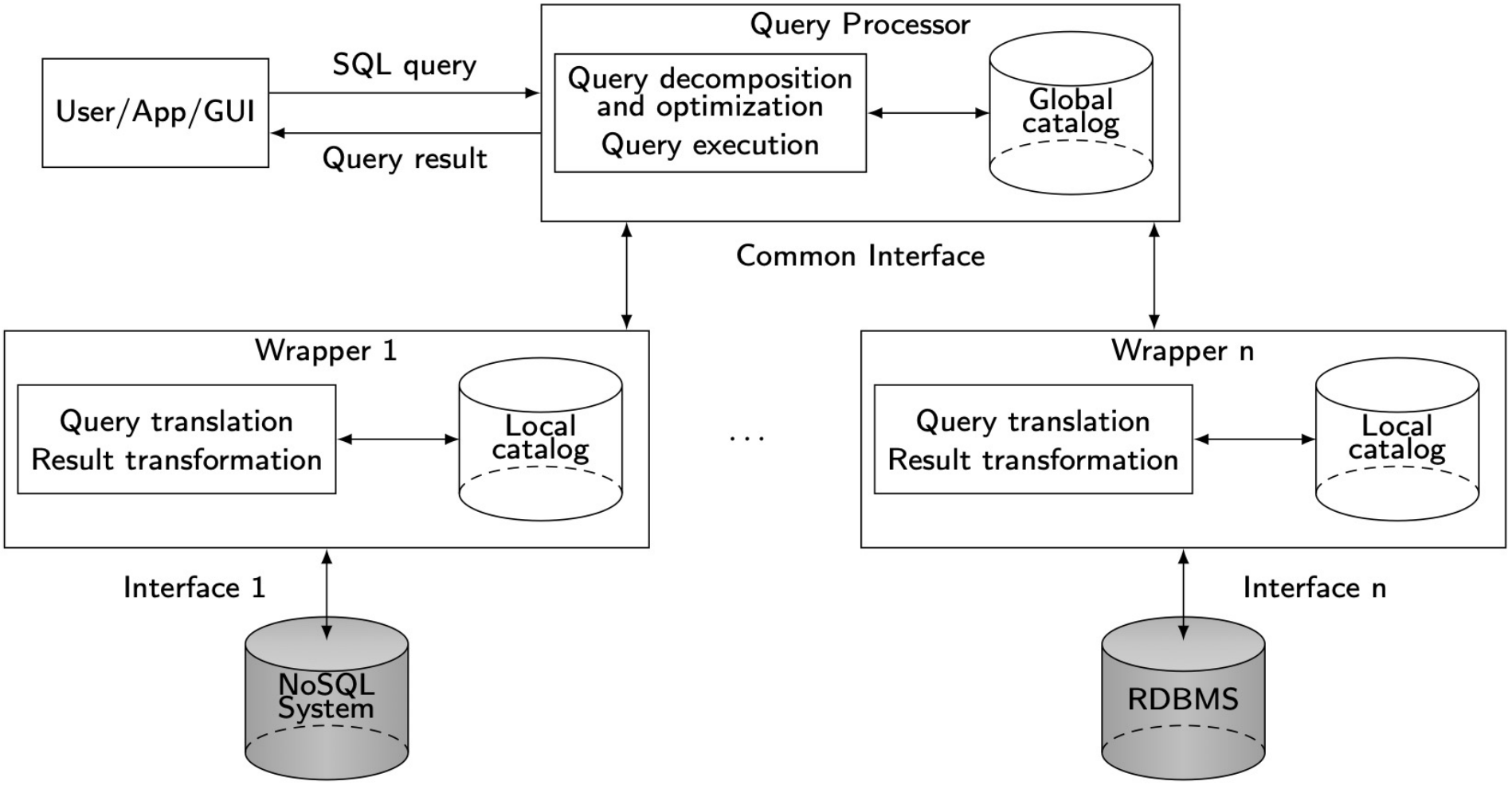
Differences with Distributed DBS

- Multidatabase systems
  - 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 efficient multistore architecture
  - No restriction to where mediator and wrapper components need be installed
- Classification of Polystores: 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)

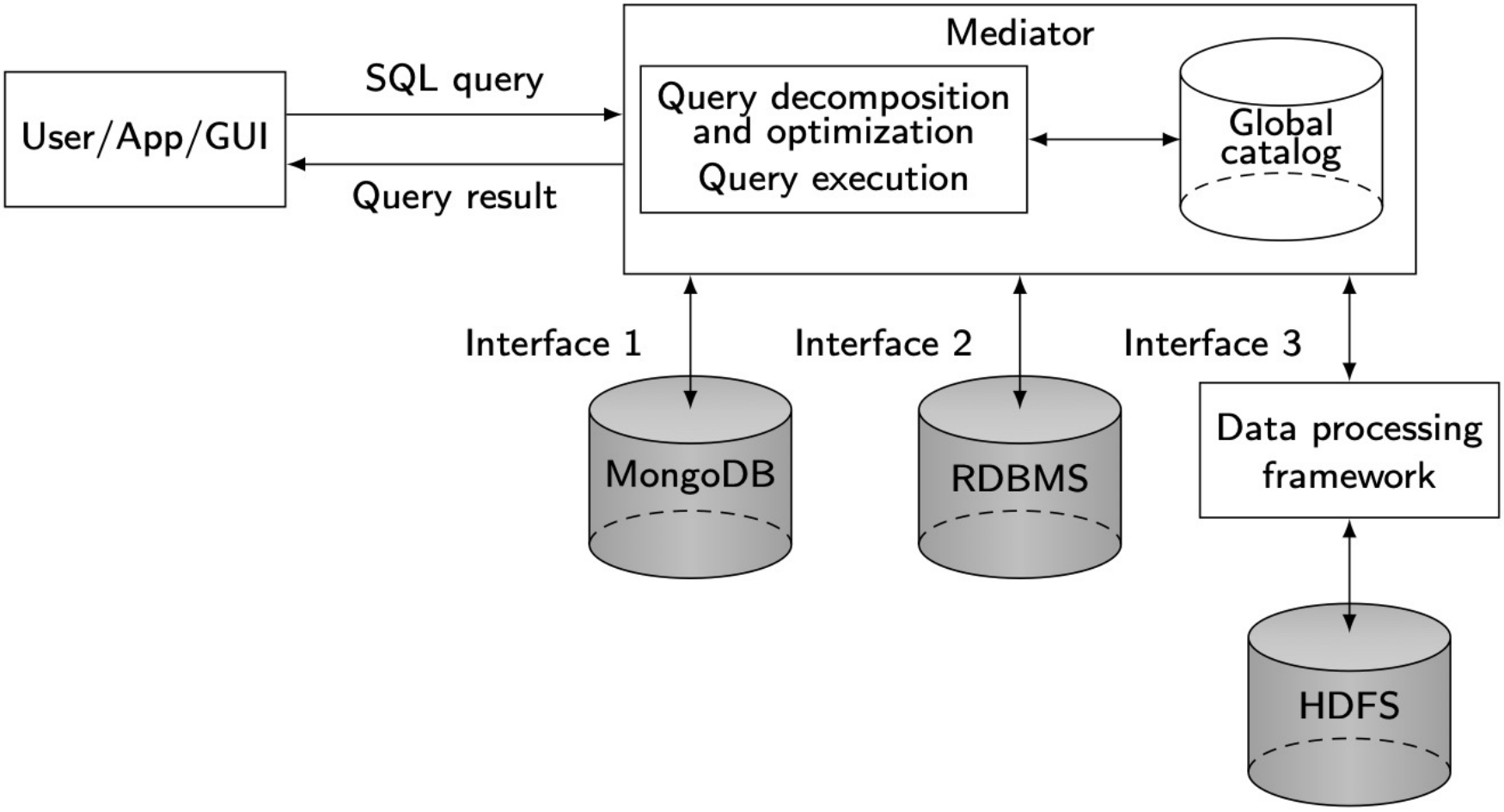
Architecture



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, France)

Architecture



Comparison - 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

Comparison - Implementation

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