P2P and Distributed DB

Carlos Bermejo Fernández
Table of Contents

• Distributed Database System
• Database Management System
  • Architecture
  • Distributed Database Systems
  • Peer-to-Peer Systems
  • Multidatabase Systems
  • Design
• Peer-to-Peer Databases
  • Architecture
  • Infrastructure
• Implementations, classification
  • Distributed Database Systems and Peer-to-Peer
    • Edutella
    • Mariposa
  • NoSQL
Distributed Database System
DDBS
DDBS - Definition

• Distributed databases system (DDBS):
  • Database system
    • Logically interrelated databases
  • Computer network
• Objective: Integration
• Why?
  • Organizational structure of distributed enterprises
  • Large-scale data management
DDBS - Definition

- It is not a “collection of files”
- Physical distribution of data is important
- Heterogeneity of OS and hardware environments

- DBs that run on multiprocessor:
  - “parallel database systems”
DDBS – Data Delivery Alternatives

• Data delivery alternatives along three orthogonal dimensions:
  • Delivery mode
    • Pull-only
    • Push-only
    • Hybrid
  • Frequency
    • Periodic
    • Conditional, client profiles
    • Ad-hoc
    • Irregular
  • Communication methods
    • Unicast
    • One-to-many
DDBS - Promises

• Data Independence
  • Logical data independence (i.e. schema of the DB)
  • Physical data independence

• Network Transparency or Distribution Transparency
  • Location transparency
  • Naming transparency

• Replication Transparency

• Fragmentation Transparency
DDBS - Promises

• Reliability Through Distributed Transactions
• Improved Performance
  • Portion of the database, contention for CPU and I/O services
  • Localization reduces remote access delays
• Easier System Expansion
DDBS - Complications

• Problems influenced mainly by:
  • The DDBS is responsible for:
    • Choosing one of the stored copies of the requested data
    • The effect of an update is reflected on each and every copy
  • Communication link failures
  • Each site cannot have instantaneous information
  • Synchronization of transactions on multiple sites
DDBS – Design Issues

• Alternatives to placing data:
  • Partitioned (or non-replicated)
  • Replicated
    • Fully replicated
    • Partially replicated
  • Fragmentation and Distribution
DDBS – Design Issues

- Distributed Directory Management
  - Directory: location, descriptions
- Distributed Query Processing
- Distributed Concurrency Control
  - Synchronization of accesses
  - Integrity of the DB
- Distributed Deadlock Management
- Reliability
  - Consistency
  - Detect Failures
  - Recovery
- Replication
Database Management System
DBMS
Distributed DBMS - Definition

• Distributed Database Management System (DDBMS)
  • Software system that permits management of distributed databases
  • Distributed system transparent to users/clients
DBMS – Generic Centralized Architecture

• DBMS is interfaced with:
  • Communication subsystem
    • Interface the DBMS with other subsystems
      • For example: terminal monitor DMBS to run interactive transactions
  • Operating system
    • Interface between the DBMS and computer resources (CPU, memory)
DDBMS – Architectural Model

- **Autonomy**
  - Distribution of control not data
  - Degree individual DBMSs can operate independently

- **Distribution**
  - Physical distribution of data over multiple sites
    - Client/server
    - Peer-to-Peer distribution

- **Heterogeneity**
  - Data models
  - Query languages
  - Transaction management protocols
DBMS – Client/Server

• Entered the computing scene at the beginning of 1990’s.
DBMS – Data Server Approach
DBMS – Distributed Database Servers
Peer-to-Peer Systems

• Early works on DDBMSs all focused on P2P architectures
• Handle data fragmentation and replication. Third layer:
  • Logical organization of data
    • Local conceptual schema (LCS)
    • Global conceptual schema (GCS)
    • External schema (ES)
      • User applications and access
Distributed DBMS (DDBMS)
Multidatabase System (MDBS)

• Individual DBMS are fully autonomous.
  • No cooperation

• Federated DBMS
  • Maps multidatabases into a single federated database
  • collection of cooperating component systems

• Difference with Distributed DMBS
  • Definition of the Global Conceptual Schema (GCS)
Multidatabase System (MDBS) Architecture
Distributed Database Design

• Three orthogonal dimensions [Levin and Morgan, 1975]
  • Level of sharing
    • No sharing: each application and its data at one site
    • Data sharing: all programs are replicated at all sites, data files not
    • Data-plus-program sharing: both data and programs
  • Behaviour of access patterns
    • Static
    • Dynamic
  • Level of knowledge on access pattern behaviour
    • Complete information
    • Partial information
• Two major strategies
  • Top-down approach
  • Bottom-up approach
DDBS – Top-Down Design Process

• Requirement analysis
  • Elicits both the data and processing needs [Yao et al. 1982a]
  • Tightly integrated, homogeneous distributed DBMSs
Distributed Database Design - Fragmentation

- Decomposition of a relation into fragments
  - Transactions execute concurrently
  - Parallel execution of a single query into sets of subqueries

- Fragmentation alternatives
  - Dividing a table into smaller ones
    - Horizontally
    - Vertically

- Correctness Rules of Fragmentation
  - Completeness
  - Reconstruction
  - Disjointness

- Information Requirements
  - Database information
  - Application information
  - Communication information
  - Communication network information
  - Computer system information

<table>
<thead>
<tr>
<th>QUERY PROCESSING</th>
<th>Full replication</th>
<th>Partial replication</th>
<th>Partitioning</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Easy</td>
<td></td>
<td>Same difficulty</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DIRECTORY MANAGEMENT</th>
<th>Easy or nonexistent</th>
<th>Same difficulty</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>CONCURRENCY CONTROL</th>
<th>Moderate</th>
<th>Difficult</th>
<th>Easy</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>RELIABILITY</th>
<th>Very high</th>
<th>High</th>
<th>Low</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>REALITY</th>
<th>Possible application</th>
<th>Realistic</th>
<th>Possible application</th>
</tr>
</thead>
</table>
DDBS – Horizontal Fragmentation

• Partitions of a relation along its tuples.

• Two versions:
  • Primary horizontal fragmentation: using predicates that are defined on that relation
    • defined by a selection operation on the owner relations of a database schema
  • Derived horizontal fragmentation: partition of relation that results from predicates defined on another relation
    • defined on a member relation of a link according to a selection operation specified on its owner
    • Reduces network communication

• Information requirements
  • Database information
  • Application information
    • Qualitative: guides the fragmentation activity
    • Quantitative: incorporated into allocation models
DDBS – Horizontal Fragmentation

- Checking for Correctness
  - Completeness
    - Based on the selection predicates used
    - Derived horizontal fragmentation is more difficult to define
  - Reconstruction
    - Performed by the union operator in both the primary and derived horizontal fragmentation
  - Disjointness
    - Is guaranteed as long as the minterm predicates are mutually exclusive
    - Disjointness can be guaranteed if the join graph is simple
DDBS – Vertical Fragmentation

• Inherently more complicated than horizontal partitioning

• Two types of heuristic approaches:
  • Grouping: starts by assigning each attribute to one fragment, and at each step, joins some of the fragments until some criteria is satisfied [Sacca and Wiederhold, 1985]
  • Splitting: starts with a relation and decides on beneficial partitioning based on the access behavior of applications to attributes [Navathe et al., 1984]

• Information requirements:
  • Application information
    • Access frequencies
DDBS – Vertical Fragmentation

• Checking for Correctness
  • Completeness
    • Guaranteed by the Partition algorithm
      • Clustering Algorithm
        • Find some means of grouping the attributes of a relation
      • Partitioning Algorithm
        • Find sets of attributes that are accessed solely or by distinct sets of applications
  • Reconstruction
    • Possible by the join operation
  • Disjointness
    • Tuples IDs are used and replicated
    • Key attribute replicate, no disjoint in strict senses
DDBS – Hybrid Fragmentation

• Most cases horizontal or vertical will be not sufficient
• Vertical fragmentation -> horizontal one, or vice versa
  • Producing a tree-structured partitioning
DDBS – Allocation

- Allocation of resources across the nodes
- Allocation problem (Allocation strategy)
  - Minimal cost: cost of queryinh, updating and data communication
  - Performance: maintain a performance metric
- Information requirements
  - Database information
  - Application information
  - Site information, each compute site
    - Knows storage and processing capacity
  - Network information
DDBS – Data Directory

• Schema information is stored in a data dictionary/directory (catalog)
  • Directory: meta-database that stores a number of information
    • Global directory (GD), describes the database schema as users see it
    • Local directory (LD), describes the local mapping and schema at each site

• Issues with directories
  • Distributed database techniques apply to directory management
  • Location issues
    • GD centrally or distributed
  • Replication
DDBS – Database Integration

• Appropriate in multidatabase systems

• Integrating local conceptual schemas (LCS) into global database schema (GCS)

• Database integration [Jhingtan et al. 2002]
  • Logical
  • Physical

• Integrated databases (materialized), data warehouse
  • ETL (Extract-Transform-Load)
DDBS – Physical Integration, Data warehouses

• On-line Analytical Processing (OLAP)
  • Trend analysis
  • Forecasting

• On-Line Transaction Processing (OLTP)
  • Banking systems
  • Airline reservation

• Gather data from number of operational DBs
DDBS – Logical Integration

• Virtual integration
  • No materialized global DB
• Data resides in operational databases
• GCS provides a virtual integration for querying
DDBS – Bottom-Up Design Process

- Appropriate in multidatabase systems
- Integrating local conceptual schemas (LCS) into global database (GCS)
- Logical or physical integration
- Local as View (LAV) or Global as View (Gav) [Lenzerini, 2002]
- [Koch, 2001]
  - GAV, query results constrained by objects defined in GCS
  - LAV, query results constrained by objects in local DBMS
DDBS – Bottom-Up Design Process

• Schema translation
  • Common intermediate canonical representation
  • Necessary only if component databases heterogeneous
  • Local schemas different data models

• Schema generation
  • Schema matching, semantic correspondences among the translated LCS elements
  • Integration of common schema elements into GCS
  • Schema mapping, how to map elements of each LCS to GCS
DDBS - Data and Access Control

• Important requirement in centralized or distributed DBMS
  • Support semantic data control, data and access control using high-level semantics
  • Semantic data includes:
    • View management
    • Security control
    • Semantic integrity control
      • Language for manipulating assertions
      • Enforcement mechanism to enforce database integrity
      • Generally not supported by DDBMS
Distributed Query Execution

- Input query is expressed in relational calculus
- Complexity of the problem is proportional to:
  - Expressive power
  - Abstraction capability of the query language
- Execution strategy minimizes cost function (I/O, CPU, communication costs)
- Query processors may differ in various aspects such as:
  - Type of algorithm
  - Optimization granularity
  - Optimization timing
  - Use of statistics
  - Decision site
  - Network topology
  - Replication fragments
  - Use of semijoins
Query Decomposition

• Query decomposition maps a distributed calculus query into an algebraic query on global relations:

\[
\Pi_{\text{ENAME}} \\
\sigma_{\text{DUR}=12 \lor \text{DUR}=24} \\
\sigma_{\text{PNAME} = \text{"CAD/CAM"}} \\
\sigma_{\text{ENAME} \neq \text{"J. Doe"}} \\
\bowtie_{\text{PNO}} \\
\text{project} \\
\text{select} \\
\text{join}
\]
Data Localization of Distributed Data

- Data localization takes as input the decomposed query on global relations and applies data distribution information to the query.
- Fragmentation is defined through fragmentation rules, which can be expressed as relational queries.
- Localization uses information stored in the fragment schema.
Optimization of Distributed Queries

• Selecting optimal execution strategy for a query is NP-hard in the number of relations [Ibaraki and Kameda, 1984]

• The selection of optimal strategy requires:
  • Prediction of execution costs of alternative candidate orderings

• Main components
  • Search space
    • Equivalents execution plans for the input query, based on execution order and implementation
  • Cost model
    • Uniform distributions of attribute values
    • Skewed data distributions: histograms
  • Search strategy
    • Explodes search space
    • Selects best plan using cost model
    • Dynamic programming: enumeration of plans with some pruning
The CAP-Theorem

- **Consistency** system is in a consistent state after the execution of an operation.
  - Distributed system is consistent after update operation all readers see his updates in some shared data source

- **Availability** system is designed and implemented in a way that allows it to continue operation

- **Partition Tolerance** the ability to continue operation in the presence of network partitions
  - network nodes cannot connect to each other
Distributed Concurrency Control

- It ensures that the consistency of a database is maintained in a multiuser distributed environment
Distributed Concurrency Control – D2PL

• Distributed 2PL (D2PL)
  • The two-phase locking rule simply states that no transaction should request a lock after it releases one of its locks
  • Transaction Manager (TM)
  • Database Manager (DM)
Distributed DBMS Reliability

• Reliable distributed DBMS can continue to process user request even when the underlying system is unreliable
  • Refers to the atomicity and durability properties of transactions
Distributed DBMS Reliability

• System, State and Failure
• Distributed Systems, four type of failures:
  • Transaction failures (aborts)
  • Site (system)
  • Failures
  • Media (disks) failures
  • Communication line failures

• Local Reliability Protocols
  • Atomicity and durability properties of local transactions
  • Checkpointing

• Distributed Reliability Protocols
  • Two-Phase Commit Protocol

Sources of System Failure (Based on Siewiorek and Swarz, 1982)
Data Replication

• Purposes:
  • System availability
  • Performance
  • Scalability
  • Application requirements

• Decisions and factors that impact the design of replication protocols
  • Database design
  • Database consistency
  • Where updates are performed
  • Update propagation
  • Degree of replication transparency
Data Replication

• Consistency of Replicated Databases
  • Mutual consistency
    • Strong, all copies of a data item have the same value at the end of the execution of an update
    • Weak, values of replicas not required to be identical after update. It will become with time
  • Update Management Strategies
    • Eager Update Propagation, apply changes to all replicas within the context of update (2PC commit point)
    • Lazy Update Propagation, transaction not wait to replicas all identical
  • Replication Protocols
    • Eager Distributed Protocols, updates anywhere, first applied to local replica, propagates to other replicas. Context of the update transaction
    • Lazy Distributed Protocols, propagation of updates lazily
      • Changes (conflicts) need to be reconciled
      • Changes in timestamp order
      • Update preferences from certain sites
Peer-to-Peer Databases
P2P Databases
Peer to Peer Data Management

• Modern P2P systems differ from the old systems
  • Massive distribution, geographically distributed
  • Heterogeneity of every aspect and autonomy
  • Volatility of these modern systems

• Requirements:
  • Autonomy. Ability to join and leave the system without restriction
  • Query expressiveness
  • Efficiency
  • Quality of service
  • Fault-tolerance
  • Security

• Different P2P systems have been developed:
  • Gnutella
  • Kazaa
  • SETI
  • BitTorrent
P2P Architecture

Fig. 16.1 Peer Reference Architecture
P2P Infrastructure – Unstructured Networks

• No restriction on data placement
• Non deterministic network creation (ad-hoc)
• Data placement is unrelated to the overlay topology
• Indices
  • Distributed, each peer maintains metadata
  • Centralized, one peer maintains metadata of the entire P2P
P2P Infrastructure – Unstructured Networks

• Distributed Index
  • Flooding issues
  • TTL limit
  • Other approaches:
    • Each peer choose a subset of its neighbors [Kalogeraki et al., 2002]
    • Random walks [Lv et al., 2002]
    • List neighbors most likely to be in the direction of the requested peer [Crespo and Garcia-Molina, 2002]
    • Gossip protocols (epidemic) [Kermarrec and van Steen, 2007]. Used for data dissemination
P2P Infrastructure – Structured Networks

• Structured P2P networks have emerged to address the scalability issues faced by unstructured P2P networks [Ritter, 2001; Ratnasamy et al., 2001b; Stoica et al., 2001a]
  • Higher scalability
  • Lower autonomy of peers

• Most popular indexing and data location mechanism is dynamic hash table (DHT)

• The key is hashed to generate a peer id, which stores the data corresponding to object contents
P2P Infrastructure – Structured Networks

• More important existing DHT-based overlays:
  • Tree. Tree approach
    • Leaf nodes: node identifiers
      • Store keys to be searched
    • Search longest prefix match at each intermediate node. Tapestry [Zhao et al., 2004]
  • Hypercube. The hypercube routing geometry is based on d-dimensional Cartesian coordinate space
    • partitioned into an individual set of zones
      • each node maintains a separate zone of the coordinate space
    • Content Addressable Network (CAN) [Ratnasamy et al., 2001a]
  • Ring. One dimensional circular identifier space
    • Nodes placed different locations on the circle
    • Chord [Stoica et al., 2001b]
• Implementations
  • Cassandra
  • Voldemort
P2P Infrastructure – Super P2P

- Hybrid between pure P2P and traditional server/client architecture
P2P Infrastructure - Comparison

<table>
<thead>
<tr>
<th>Requirements</th>
<th>Unstructured</th>
<th>Structured</th>
<th>Super-peer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Autonomy</td>
<td>Low</td>
<td>Low</td>
<td>Moderate</td>
</tr>
<tr>
<td>Query expressiveness</td>
<td>High</td>
<td>Low</td>
<td>High</td>
</tr>
<tr>
<td>Efficiency</td>
<td>Low</td>
<td>High</td>
<td>High</td>
</tr>
<tr>
<td>QoS</td>
<td>Low</td>
<td>High</td>
<td>High</td>
</tr>
<tr>
<td>Fault-tolerance</td>
<td>High</td>
<td>High</td>
<td>Low</td>
</tr>
<tr>
<td>Security</td>
<td>Low</td>
<td>Low</td>
<td>High</td>
</tr>
</tbody>
</table>
P2P – Querying Over P2P systems

• Supporting more complex queries in P2P subject of much recent research

• Top-k Queries
  • Used monitoring, information retrieval, and multimedia databases [Ilyas et al., 2008]
  • User requests k most relevant answers to be returned
  • Basic techniques
    • Threshold Algorithm (TA)
    • TA-Style Algorithm
    • Best Position Algorithm (BPA)

• Join Queries
  • Most efficient Hash based

• Range Queries
  • Structured P2P and DHTs have difficulties
    • Ordering of data after hashing
Existing DDBS and P2P Databases
## Existing DDBS and P2P Databases

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Edutella</td>
<td>Loos. with SP</td>
<td>Cluster of Peers</td>
<td>Semantic</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>Galanis et al.</td>
<td>Loos. coupled</td>
<td>DHT</td>
<td>Semantic</td>
<td>Logarithmic</td>
<td>NA</td>
</tr>
<tr>
<td>Pier</td>
<td>Loos. coupled</td>
<td>DHT</td>
<td>Forward-progress</td>
<td>Logarithmic</td>
<td>NA</td>
</tr>
<tr>
<td>GridVine</td>
<td>Loos. coupled</td>
<td>DHT</td>
<td>Semantic Gossiping</td>
<td>Logarithmic</td>
<td>NA</td>
</tr>
<tr>
<td>PeerDB</td>
<td>Loos. with LigLO</td>
<td>Cluster of Peers</td>
<td>Agent-assisted</td>
<td>NA</td>
<td>128-bit enc.</td>
</tr>
<tr>
<td>Piazza</td>
<td>Loos. with Virtual Peers</td>
<td>Mapping-based</td>
<td>Semantic</td>
<td>Nr. of peer attributes</td>
<td>NA</td>
</tr>
<tr>
<td>PIERSearch</td>
<td>Loos. with DHT</td>
<td>Gnutella-graph/DHT</td>
<td>Semantic</td>
<td>DHT for rare items</td>
<td>NA</td>
</tr>
<tr>
<td>Mariposa</td>
<td>Tightly coupled</td>
<td>NA</td>
<td>Economic-based</td>
<td>WAN-based</td>
<td>NA</td>
</tr>
<tr>
<td>R*</td>
<td>Tightly coupled</td>
<td>Master/Appr.</td>
<td>Catalog-based</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>SDD-1</td>
<td>Tightly coupled</td>
<td>DM/TM</td>
<td>Reducer-based</td>
<td>NA</td>
<td>NA</td>
</tr>
</tbody>
</table>
Example P2P - Edutella

- P2P Networking Infrastructure based on Resource Description Framework (RFD)
- Gnutella to educational domain
  - Metadata in Gnutella is limited (Mp3: Artist-Song)
- Services
  - Querying
  - Mapping
    - Translating between different metadata vocabularies
  - Replication
    - Data persistence
    - Data integrity
    - Availability
    - Workload balancing
Example P2P - Edutella

- **Edutella Query Service**
  - Peers register the queries they may be asked through the query service

- **Edutella Replication**
  - Complements local storage by replication
  - Replication of metadata

- **Edutella Mapping, Meditation, Clustering**
  - Manage mappings between different schemas
  - Schema translation
Example DDBS - Mariposa

• Architecture DDBMS
  • Move the query to the data [Stonebraker, 1986]
  
• Approaches
  • Data allocate in sites of computer network
  • client-server file systems [Howarts, 88]
  
• Tertiary memory or deep store
  • Epoch
  • UniTree

• Object-Oriented DBMS
  • Data objects has specific home
  • Move the data to the query

<table>
<thead>
<tr>
<th></th>
<th>Distributed DBMS</th>
<th>Distributed file system</th>
<th>Deep store file system</th>
<th>OODB</th>
<th>Mariposa</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unit of storage</td>
<td>fragment</td>
<td>file</td>
<td>file</td>
<td>class</td>
<td>fragment</td>
</tr>
<tr>
<td>(object)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fixed object home</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>Site control</td>
<td>human</td>
<td>human</td>
<td>human</td>
<td>human</td>
<td>internal rule system</td>
</tr>
<tr>
<td>Object cached</td>
<td>fragment</td>
<td>block</td>
<td>segment</td>
<td>block</td>
<td>fragment</td>
</tr>
<tr>
<td>Length of caching</td>
<td>one query</td>
<td>procedurally controlled</td>
<td>procedurally controlled</td>
<td>procedurally controlled</td>
<td>internal rule system</td>
</tr>
<tr>
<td>Caching policy</td>
<td>n/a</td>
<td>LRU</td>
<td>typically weighted LRU</td>
<td>LRU</td>
<td>defined with rules</td>
</tr>
<tr>
<td>Object changes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>yes or no</td>
</tr>
<tr>
<td>representation when</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cached?</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Entity moved</td>
<td>query</td>
<td>data</td>
<td>data</td>
<td>data</td>
<td>query or data</td>
</tr>
</tbody>
</table>
Example DDBS - Mariposa

• Distributes data over a collection of sites that are connected in a WAN

• Data model
  • Instances of objects named classes
    • Each contains a collection of attributes
  • Each class is divided into a collection of fragments
    • Fragments can have distribution criteria
      • Controls logical composition of instances
      • Not specific home

• Major aspects
  • Rule system and engine rule set
    • On event do action
  • Fragment movement algorithm
  • Query optimizer and execution engine
  • Approach to multiple copies for this environment

• Unifies best features of DDBMS
NoSQL – Definition

• The term NoSQL was first used in 1998 for a relation database that omitted the use of SQL [Strozzi 1998]. The term was picked up again in 2009 and used for conferences of advocates of non-relational databases.

• Open source distributed, non relational databases

• Markets which need specialized DBMSs
  • Data warehouses
  • Stream processing
  • Text processing
  • Scientific-oriented databases
  • Semi-structured data
NoSQL Architecture

• Partitioning
  • Memory Caches, like memcached
    • Partitioned in-memory databases
  • Clustering of database servers
  • Separating Reads from Writes
    • Separated servers to dedicated operations
  • Sharding partition the data updated and requested on same node
    • Mapping between data partitions (shards)
NoSQL Architecture

• Storage Layout
  • Row-based Storage Layout
    • Table of relational model serialized and flushed to disk
  • Columnar Storage Layout
    • Serializes tables by appending columns
  • Columnar Storage Layout with Locality Groups
    • similar to column-based
    • feature of defining so called locality groups that are groups of columns expected to be accessed together by clients
  • Log Structured Merge Trees (LSM-trees)
    • Describe how to efficiently use memory and disk storage in order to satisfy read and write requests in an efficient, performant and still safely manner
NoSQL Architecture

• Query Models
  • Companion SQL-database
    • Approach in which searchable attributes are copied to a SQL or text database
  • Scatter/Gather Local Search
    • Used if the NoSQL store allows querying and indexing within database server nodes
  • Distributed B+Trees
    • Hash the searchable attribute to locate the root node of a distributed B+tree
  • Prefix Hash Table
    • Tree-datastructure where every path from the root-node to the leafs contains the prefix of the key and every node in the tree contains all the data whose key is prefixed by it
NoSQL Classification

• Taxonomy by Stephen Yen 2009

<table>
<thead>
<tr>
<th>Term</th>
<th>Matching Databases</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key-Value-Cache</td>
<td>Memcached</td>
</tr>
<tr>
<td></td>
<td>Re�cached</td>
</tr>
<tr>
<td></td>
<td>Coherence</td>
</tr>
<tr>
<td></td>
<td>Infinispan</td>
</tr>
<tr>
<td></td>
<td>EXtreme Scale</td>
</tr>
<tr>
<td></td>
<td>Jboss Cache</td>
</tr>
<tr>
<td></td>
<td>Velocity</td>
</tr>
<tr>
<td></td>
<td>Terracq</td>
</tr>
<tr>
<td>Key-Value-Store</td>
<td>keyspace</td>
</tr>
<tr>
<td></td>
<td>Flare</td>
</tr>
<tr>
<td></td>
<td>Schema Free</td>
</tr>
<tr>
<td></td>
<td>RACloud</td>
</tr>
<tr>
<td>Eventually-Consistent Key-Value-Store</td>
<td>Dynamo</td>
</tr>
<tr>
<td></td>
<td>Voldemort</td>
</tr>
<tr>
<td></td>
<td>Dynomite</td>
</tr>
<tr>
<td></td>
<td>SubRecord</td>
</tr>
<tr>
<td></td>
<td>MolionDb</td>
</tr>
<tr>
<td></td>
<td>Dovetaileb</td>
</tr>
<tr>
<td>Ordered-Key-Value-Store</td>
<td>Tokyo Tyrant</td>
</tr>
<tr>
<td></td>
<td>Lightcloud</td>
</tr>
<tr>
<td></td>
<td>NMDB</td>
</tr>
<tr>
<td></td>
<td>Lasso</td>
</tr>
<tr>
<td></td>
<td>Memcached</td>
</tr>
<tr>
<td></td>
<td>Actored</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Term</th>
<th>Matching Databases</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data-Structures Server</td>
<td>Redis</td>
</tr>
<tr>
<td>Tuple Store</td>
<td>GigaSpaces</td>
</tr>
<tr>
<td></td>
<td>Coord</td>
</tr>
<tr>
<td></td>
<td>Apache River</td>
</tr>
<tr>
<td>Object Database</td>
<td>ZopeDB</td>
</tr>
<tr>
<td></td>
<td>DB4O</td>
</tr>
<tr>
<td></td>
<td>Shoal</td>
</tr>
<tr>
<td>Document Store</td>
<td>CouchDB</td>
</tr>
<tr>
<td></td>
<td>Mango</td>
</tr>
<tr>
<td></td>
<td>Jackrabbit</td>
</tr>
<tr>
<td></td>
<td>XML Databases</td>
</tr>
<tr>
<td></td>
<td>ThruDB</td>
</tr>
<tr>
<td></td>
<td>CloudKit</td>
</tr>
<tr>
<td></td>
<td>Perservere</td>
</tr>
<tr>
<td></td>
<td>Risk Basho</td>
</tr>
<tr>
<td></td>
<td>Scalans</td>
</tr>
<tr>
<td>Wide Columnar Store</td>
<td>Bigtable</td>
</tr>
<tr>
<td></td>
<td>Hbase</td>
</tr>
<tr>
<td></td>
<td>Cassandra</td>
</tr>
<tr>
<td></td>
<td>Hypertable</td>
</tr>
<tr>
<td></td>
<td>KAI</td>
</tr>
<tr>
<td></td>
<td>OpenNeptune</td>
</tr>
<tr>
<td></td>
<td>Qbase</td>
</tr>
<tr>
<td></td>
<td>KDI</td>
</tr>
</tbody>
</table>
NoSQL Classification

• Categorization and Comparison by Scofield and Popescu 2010

<table>
<thead>
<tr>
<th></th>
<th>Performance</th>
<th>Scalability</th>
<th>Flexibility</th>
<th>Complexity</th>
<th>Functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key-Value Stores</td>
<td>high</td>
<td>high</td>
<td>high</td>
<td>none</td>
<td>variable (none)</td>
</tr>
<tr>
<td>Column stores</td>
<td>high</td>
<td>high</td>
<td>moderate</td>
<td>low</td>
<td>minimal</td>
</tr>
<tr>
<td>Document stores</td>
<td>high</td>
<td>variable (high)</td>
<td>high</td>
<td>low</td>
<td>variable (low)</td>
</tr>
<tr>
<td>Graph databases</td>
<td>variable</td>
<td>variable</td>
<td>high</td>
<td>high</td>
<td>graph theory</td>
</tr>
<tr>
<td>Relational databases</td>
<td>variable</td>
<td>variable</td>
<td>low</td>
<td>moderate</td>
<td>relational algebra</td>
</tr>
</tbody>
</table>
# NoSQL Classification

## Data and Query Model

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Model</th>
<th>Query API</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cassandra</td>
<td>Columnfamily</td>
<td>Thrift</td>
</tr>
<tr>
<td>CouchDB</td>
<td>Document</td>
<td><code>map/reduce views</code></td>
</tr>
<tr>
<td>HBase</td>
<td>Columnfamily</td>
<td>Thrift, REST</td>
</tr>
<tr>
<td>MongoDB</td>
<td>Document</td>
<td>Cursor</td>
</tr>
<tr>
<td>Neo4j</td>
<td>Graph</td>
<td>Graph</td>
</tr>
<tr>
<td>Redis</td>
<td>Collection</td>
<td>Collection</td>
</tr>
<tr>
<td>Riak</td>
<td>Document</td>
<td>Nested hashes</td>
</tr>
<tr>
<td>Scalaris</td>
<td>Key/Value</td>
<td>get/put</td>
</tr>
<tr>
<td>Tokyo Cabinet</td>
<td>Key/Value</td>
<td>get/put</td>
</tr>
<tr>
<td>Voldemort</td>
<td>Key/Value</td>
<td>get/put</td>
</tr>
</tbody>
</table>

## Persistence Design

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Design</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cassandra</td>
<td>Memtable / SSTable</td>
</tr>
<tr>
<td>CouchDB</td>
<td>Append-only B-tree</td>
</tr>
<tr>
<td>HBase</td>
<td>Memtable / SSTable on HDFS</td>
</tr>
<tr>
<td>MongoDB</td>
<td>B-tree</td>
</tr>
<tr>
<td>Neo4j</td>
<td>On-disk linked lists</td>
</tr>
<tr>
<td>Redis</td>
<td>In-memory with background snapshots</td>
</tr>
<tr>
<td>Riak</td>
<td>?</td>
</tr>
<tr>
<td>Scalaris</td>
<td>In-memory only</td>
</tr>
<tr>
<td>Tokyo Cabinet</td>
<td>Hash or B-tree</td>
</tr>
<tr>
<td>Voldemort</td>
<td>Pluggable (primarily BDB MySQL)</td>
</tr>
</tbody>
</table>
Thank you