Lecture NoSQL Database Systems

- What is NoSQL?
- CAP Theorem
- Types of NoSQL
- Data Model
- Frameworks

What is NoSQL?

- NoSQL:
  - Not Only SQL (not No-SQL) database
  - Non relational data storage system
  - No ACID property
- A **NoSQL** database provides a mechanism for storage and retrieval of data that
  - Uses looser consistency model than traditional relational databases
  - is highly scalable; and
  - Has higher availability.

Why NoSQL?

- A large collection of structured or semi structure data
  - Social media sites such as Facebook, Twitter
  - Lack of scalability of relational databases
- High availability,
  - Not ACID
- Dynamic schemas
  - No fixed table schemas
- Frequent simpler queries
  - No joins
  - Nested queries

Scaling Up

- RDBMS were not designed to be distributed and/or grided
  - Distribution and ACID property
  - Distribution and join operations
- Multiple node relational database systems
  - Master-slave
  - Sharding

Scaling RDBMS – Master/Slave

- Master-Slave
  - All writes are written to the master. All reads performed against the replicated slave databases
  - Critical reads may be incorrect as writes may not have been propagated down
  - Large data sets can pose problems as master needs to duplicate data to slaves

Scaling RDBMS - Sharding

- Partition or sharding
  - Scales well for both reads and writes
  - Not transparent, application needs to be partition-aware
  - Can no longer have relationships/joins across partitions
  - Loss of referential integrity across shards

Other ways to scale RDBMS

- Multi-Master replication
- INSERT only, not UPDATES/DELETES
- No JOINs, thereby reducing query time
  - This involves de-normalizing data
- In-memory databases

How did we get here?

- Explosion of social media sites
  - (Facebook, Twitter) with large data needs
- Rise of cloud-based solutions
  - Amazon S3 (simple storage solution)
- Just as moving to dynamically-typed languages
  - a shift to dynamically-typed data with frequent schema changes
  - Ruby/Groovy
- Open-source community

Some influential events

- Some major papers/systems were the seeds of the NoSQL movement
  - BigTable (Google)
  - Dynamo (Amazon)
    - Gossip protocol (discovery and error detection)
    - Distributed key-value data store
    - Eventual consistency
  - CAP Theorem
  - MapReduce and Hadoop

The Perfect Storm

- Large datasets, acceptance of alternatives, and dynamically-typed data has come together in a perfect storm
- Not a backlash/rebellion against RDBMS
- SQL is a rich query language that cannot be rivaled by the current list of NoSQL offerings

Brewer's CAP Theorem

A distributed system can support only two of the following characteristics:

- Consistency
- Availability
- Partition tolerance

Consistency

- all nodes see the same data at the same time
- client perceives that a set of operations has occurred all at once
- More like Atomic in ACID transaction properties

Availability

- node failures do not prevent survivors from continuing to operate
- Every operation must terminate in an intended response

Partition Tolerance

- the system continues to operate despite arbitrary message loss
- Operations will complete, even if individual components are unavailable

Availability

- Traditionally, thought of as the server/process available five 9's (99.999 %).
- However, for large node system, at almost any point in time there's a good chance that a node is either down or there is a network disruption among the nodes.
  - Want a system that is resilient in the face of network disruption

Consistency Model

- A consistency model determines rules for visibility and apparent order of updates.
- For example:
  - Row X is replicated on nodes M and N
  - Client A writes row X to node N
  - Some period of time t elapses.
  - Client B reads row X from node M
  - Does client B see the write from client A?
  - Consistency is a continuum with tradeoffs
  - For NoSQL, the answer would be: maybe
  - CAP Theorem states: Strict Consistency can't be achieved at the same time as availability and partition-tolerance.

Eventual Consistency

- When no updates occur for a long period of time, eventually all updates will propagate through the system and all the nodes will be consistent
- For a given accepted update and a given node, eventually either the update reaches the node or the node is removed from service

BASE Transactions

- **B**asically **A**vailable,
- **S**oft state
  - consistency is the developer's problem and should not be handled by the database
- **E**ventually Consistent

BASE Transactions

- Characteristics
  - Weak consistency
    - stale data OK
  - Availability first
  - Best effort
  - Approximate answers OK
  - Aggressive (optimistic)
  - Simpler and faster

What kinds of NoSQL databases

- NoSQL solutions fall into two major areas:
  - Key/Value or 'the big hash table'.
    - Amazon S3 (Dynamo)
    - Voldemort
    - Scalaris
  - Schema-less which comes in multiple flavors
    - column-based (Cassandra, HBase)
    - document-based (CouchDB)
    - graph-based (Neo4J)

Key/Value

Pros:

- very fast
- very scalable
- simple model
- able to distribute horizontally

Cons:

- many data structures (objects) can't be easily modeled as key value pairs

Schema-Less

Pros:

- Schema-less data model is richer than key/value pairs
- eventual consistency
- many are distributed
- still provide excellent performance and scalability

Cons:

- typically no ACID transactions or joins

Common Advantages

- Cheap, easy to implement (open source)
- Data are replicated to multiple nodes (therefore identical and fault-tolerant) and can be partitioned
  - Down nodes easily replaced
  - No single point of failure
- Easy to distribute
- Don't require a schema
- Can scale up and down
- Relax the data consistency requirement (CAP)

List of NoSQL systems (www.nosql-database.org)

- Cassandre
- Haddop & Hbase
- CouchDB
- MongoDB
- StupidDB
- MapReduce
- Trift
- Couddata
- Etc.

What are we giving up?

- joins
- group by
- order by
- ACID transactions
- SQL as a sometimes frustrating but still powerful query language
- easy integration with other applications that support SQL

Cassandra

- Originally developed at Facebook
- Follows the BigTable data model
  - column-oriented
- Uses the Dynamo Eventual Consistency model
- Written in Java
- Open-sourced and exists within the Apache family
- Uses Apache Thrift as it's API

Thrift

- Created at Facebook along with Cassandra
- Is a cross-language, service-generation framework
- Binary Protocol (like Google Protocol Buffers)
- Compiles to: C++, Java, PHP, Ruby, Erlang, Perl, ...

Searching

- Relational
  - SELECT `column` FROM `database`,`table` WHERE `id` = key;
  - SELECT product_name FROM rockets WHERE id = 123;
- Cassandra (standard)
  - keyspace.getSlice(key, “column_family”, "column")
  - keyspace.getSlice(123, **new** ColumnParent(“rockets”), *getSlicePredicate*());

Typical NoSQL API

- Basic API access:
  - get(key) -- Extract the value given a key
  - put(key, value) -- Create or update the value given its key
  - delete(key) -- Remove the key and its associated value
  - execute(key, operation, parameters) -- Invoke an operation to the value (given its key) which is a special data structure (e.g. List, Set, Map .... etc).

Data Model

- Within Cassandra, you will refer to data this way:
  - **Column:** smallest data element, a tuple with a name and a value

    :Rockets, '1' might return:

    {'name' => ‘Rocket-Powered Roller Skates',

    ‘toon' => ‘Ready Set Zoom',

    ‘inventoryQty' => ‘5‘,

    ‘productUrl’ => ‘rockets\1.gif’}

Data Model Continued

- **ColumnFamily**: There's a single structure used to group both the Columns and SuperColumns. Called a ColumnFamily (think table), it has two types, Standard & Super.
  - Column families must be defined at startup
- **Key**: the permanent name of the record
- **Keyspace**: the outer-most level of organization. This is usually the name of the application. For example, 'Acme' (think database name).

Cassandra and Consistency

- Talked previous about eventual consistency
- Cassandra has programmable read/writable consistency
  - One: Return from the first node that responds
  - Quorom: Query from all nodes and respond with the one that has latest timestamp once a majority of nodes responded
  - All: Query from all nodes and respond with the one that has latest timestamp once all nodes responded. An unresponsive node will fail the node

Classification of NoSQL databases

- Physical layout
- Consistency model

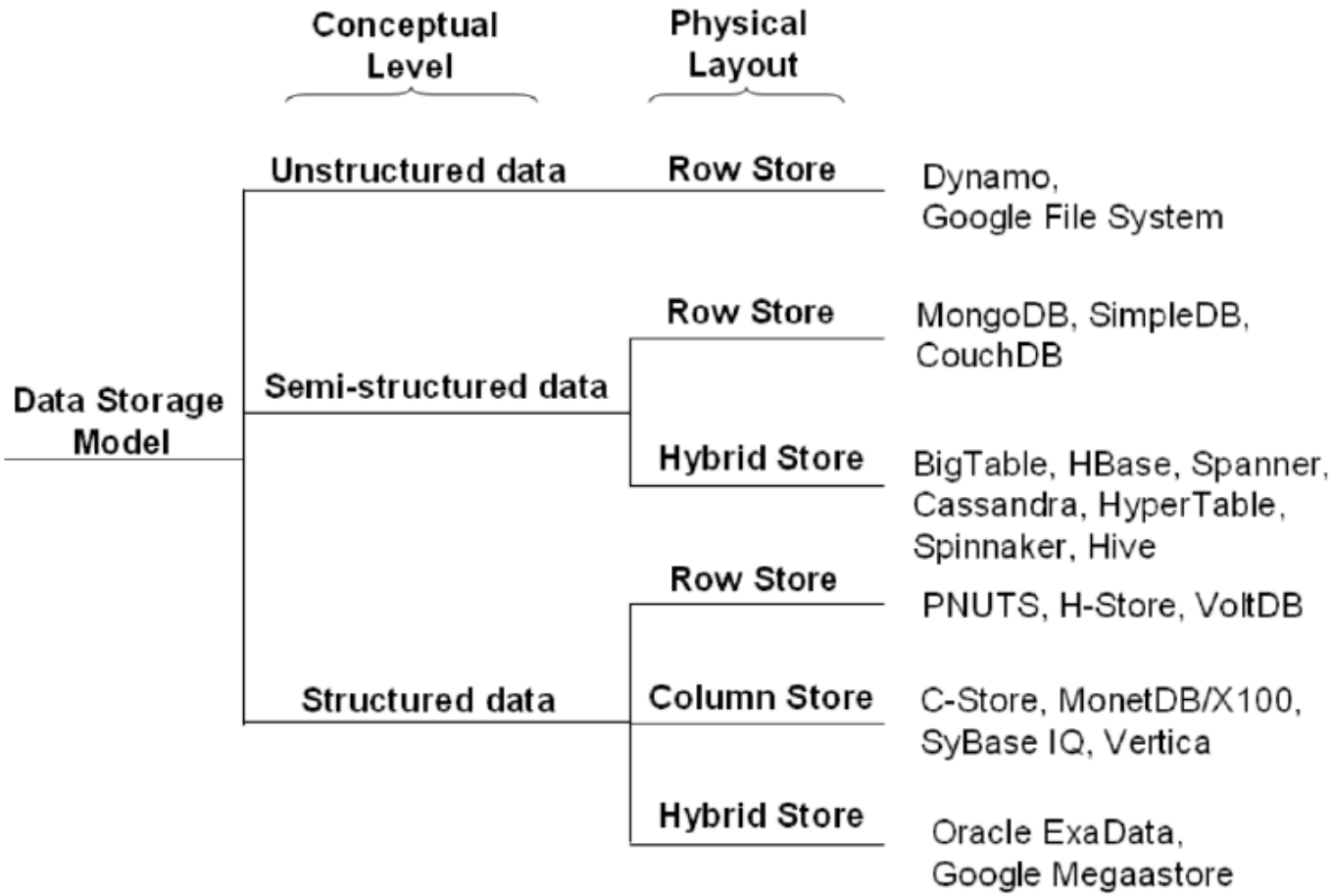
Physical Layout

- Row storage
  - Dominant in transitional DBMS
- Column storage
  - Efficient scan-level access
  - Easy compression
  - Optimized analytical workloads
  - SyBase IQ, C-Store, MonetDB-X100
- Hybrid
  - combination of row and column stores
  - Table: ne-gained hybrid
  - Block: partition attributes across

Conceptual mode

- Unstructured data
  - uninterrupted, isolated and stored in binary object
  - simplest format and maximum exibility
- Semi-structured data
  - inner structure without xed schema
  - document-oriented store, column-family store
- Structured data
  - replies on the relational calculus
  - structured entities with strict relationships according to schema
  - constraints: entity integrity and referential integrity

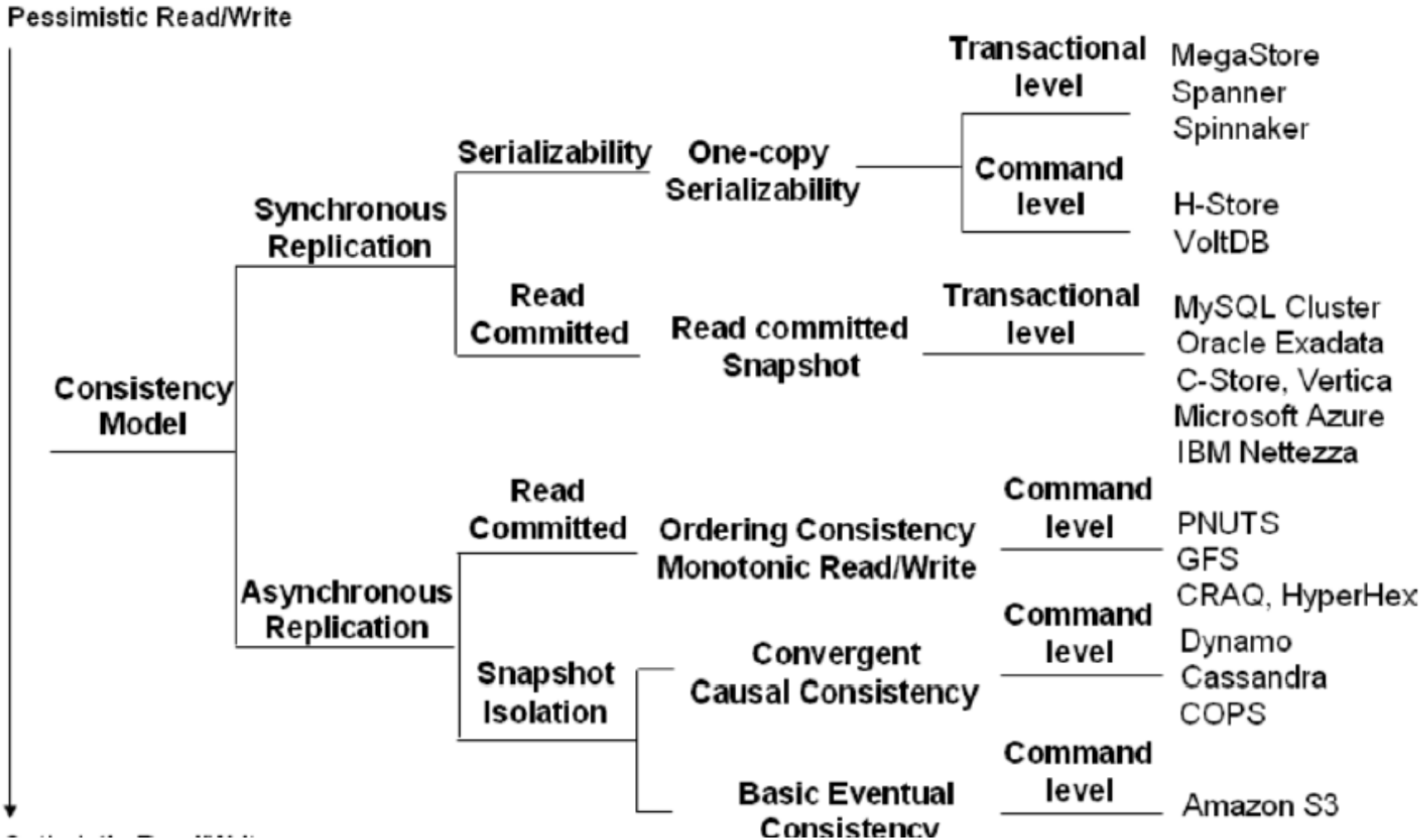
Data model Taxonomy



consistency

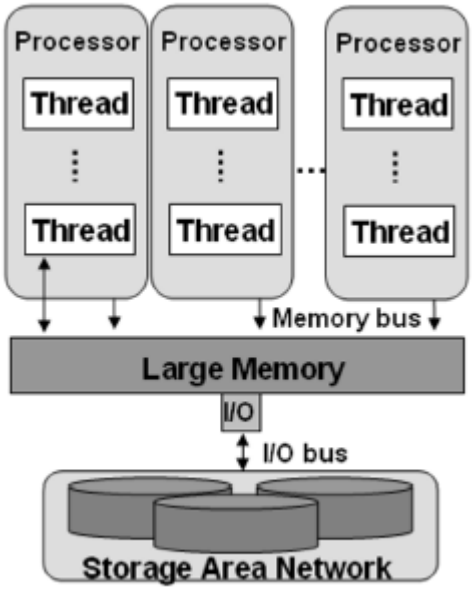
- Serializablility
- Snapshot isolation
  - Transaction sees a consistent committed snapshot of the database
  - Optimistic Read and Write
- Strict consistency
- Eventual consistence

Consistency mode taxonomy



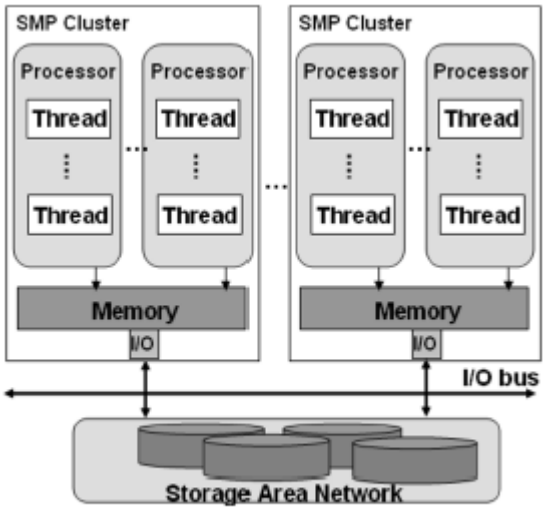
SMP on shared-memory architecture

- Processors with own cache
- Threads allocated to processors
- Coherent memory pool for sharing data
- Scale-up by adding processors
- Bounded by resource limit
  - Memory/IO bus bandwidth
  - L2 cache consistency
  - Number of cores
  - Custom-built with high price



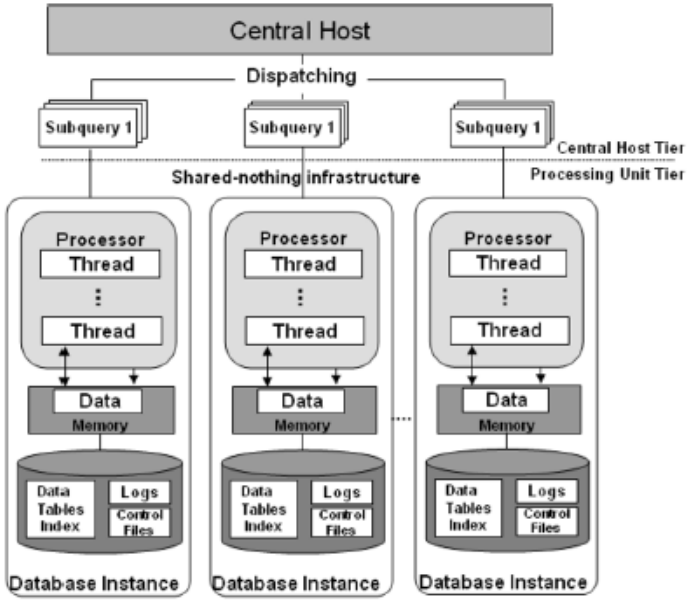
MPP on Shared-disk architecture

- Parallel SMP clusters
- Common storage by shared I/O
- Disk arrays in SAN
- Mitigate I/O bus bandwidth
  - Cache fusion protocol
    - Oracle Exadata
  - Query streaming preprocess
    - IBM Netezza



Dispatching on shared-nothing architecture

- Centeral coordinator
  - Devision
  - Dispatch
  - post-process
- Loosely coupled machines dedicated memory and disk
- Independent database instance
  - Operating on portion of the data
- Each to scale out



System architecture Taxonomy