BigData

by: Brandon McKune

Overview

- Problem
- Introduction
- Architecture
- BigData RDF

Problem

- Consider large amounts of heterogeneous data:
    - Different sources
    - Different formats
    - Data update rates

Problem cont.

- Combining this data to analyze can produce
  - new insights
  - interesting cross connections
  - better operational decision making

Addressing the Problem

- Need to be able to load and query these very large datasets
- Heterogeneous datasets, interesting data isn't usually stored at deployment
- dynamic alignment during continuous integration of new data
- Answer? BigData

Intro - What is BigData?

- What?
  - Scale-out Storage & Computing Fabric
- Supports
  - Optional Transactions
  - Very High Concurrency
  - Very High Aggregate I/O Rates

Intro - What is BigData

- What? cont.
  - Open source licensing and support
  - Implemented in Java
  - Supports single instance and clusters

Intro - What is BigData

- How?
  - Ordered Data (B+Trees)
  - Operates on clusters comprised of commodity hardware
  - Uses dynamically partitioned key-range shards

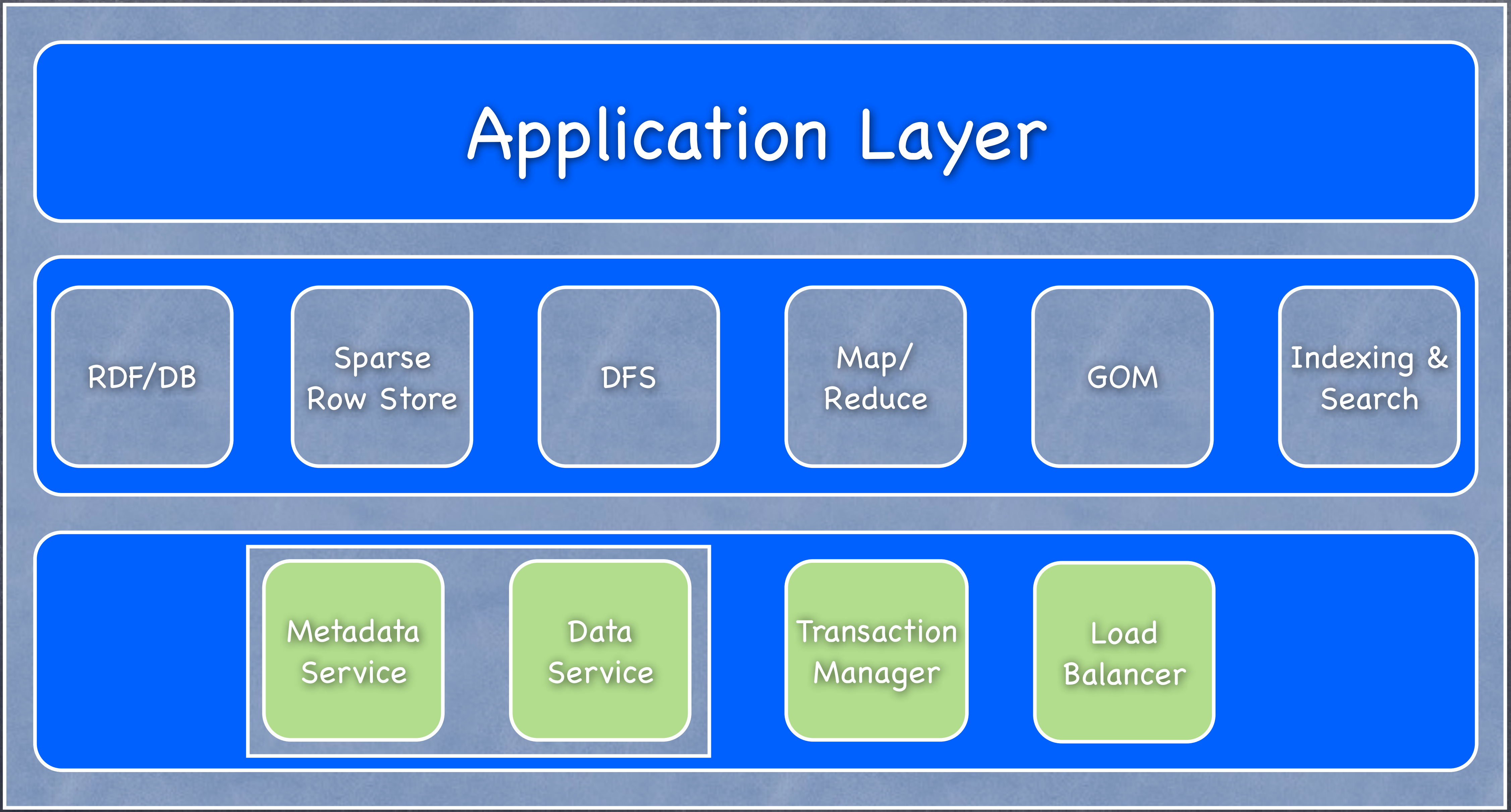
Architecture - B+Trees

- Why?
  - Large datasets - Stored on disk
  - Provides search, insert, and update in logarithmic amortize time
    - average time taken; not worst or best
  - Ability to perform balance operations and remains balanced during insert/delete.

B+Tree cont.

- Index
  - Maps byte[] keys to byte[] values
- "Tuple" = key/value, delete flag, and timestamp
- Copy-on-Write

Architecture



Dynamic Partitioning

- Index Partition
  - Collection of local resources
  - Made up of indices dynamically aggregated into key-range shards
  - Id, boundary, and location
  - also called scale out index

Dynamic Partitioning

- Three basic operations
  - Split - divides an index partition into two index partitions over same key-range
  - Move - moves an index partition from one data service to another
  - Join - two sibling index partitions into one over same key-range

Metadata Service

- Index Partition Locator (DNS for BigData)
  - Maps Index Partition Id to Key-Range
- Specialized Data Service
- Upper bound ~200 Petabytes
- Uses Hadoop's Zookeeper

Data Service

- Maintains a append-only write buffer (Journal)
- Any number of read-only, optimized index segments.
- Remember Index Partition?
  - View onto the Journal and historical data

Concurrency Control

- Some DB architectures use two phase locking (2PL)
- BigData uses Multi-Version Concurrency Control (MVCC)
  - Readers never block
  - Writers run concurrently; even on shared resources

Concurrency - Cont.

- MVCC Explained
  - use of timestamped transactions
  - copy-on-write mechanisms in B+Tree
  - Immortal Store architecture of the Journal
  - History Retention required on data services

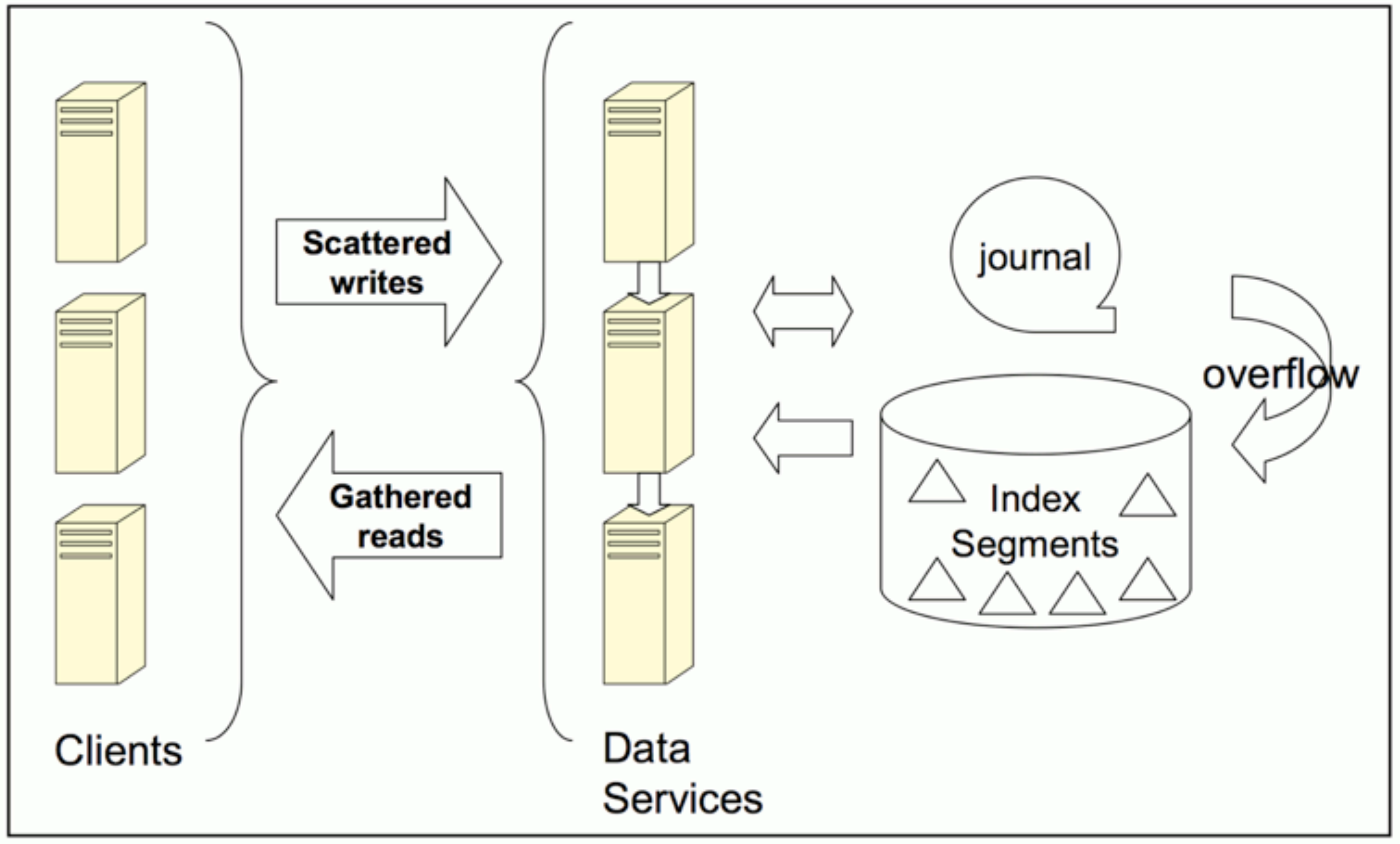
Database History

- Immortal database architecture
    - Has configurable history retention based on users needs
    - ex.
        - History Retention = 2 days
        - Write tuple to DB, then delete it at some point P
        - You have 2 days from P to access tuple before that tuple is removed

High Availability

- Replicate State
- Scatter Reads/Gather Writes across the cluster
- Multiple physical instances for each local data service

High Availability



BigData RDF

- Distributed Operations
- Footprint grows incrementally
  - does not require reload of data on new hardware additions

BigData RDF

- 3 distinct modes:
  - triples
  - triples with provenance
  - quads
- Can abstract to: Lexicon and Statement
- Concentration on triples (S,P,O)

BigData RDF

- Lexicon
  - maps RDF values (URIs, literals, & blank nodes) to unique 64 bit internal Ids
- Statement
  - models the Subject, Predicate, and Object for each statement.
  - used during querying statement patterns (S,P,O); (O,P,S); (P,O,S)

BigData RDF

- Supports
  - SPARQL
  - RDFS+ inference
  - Fast load & queries

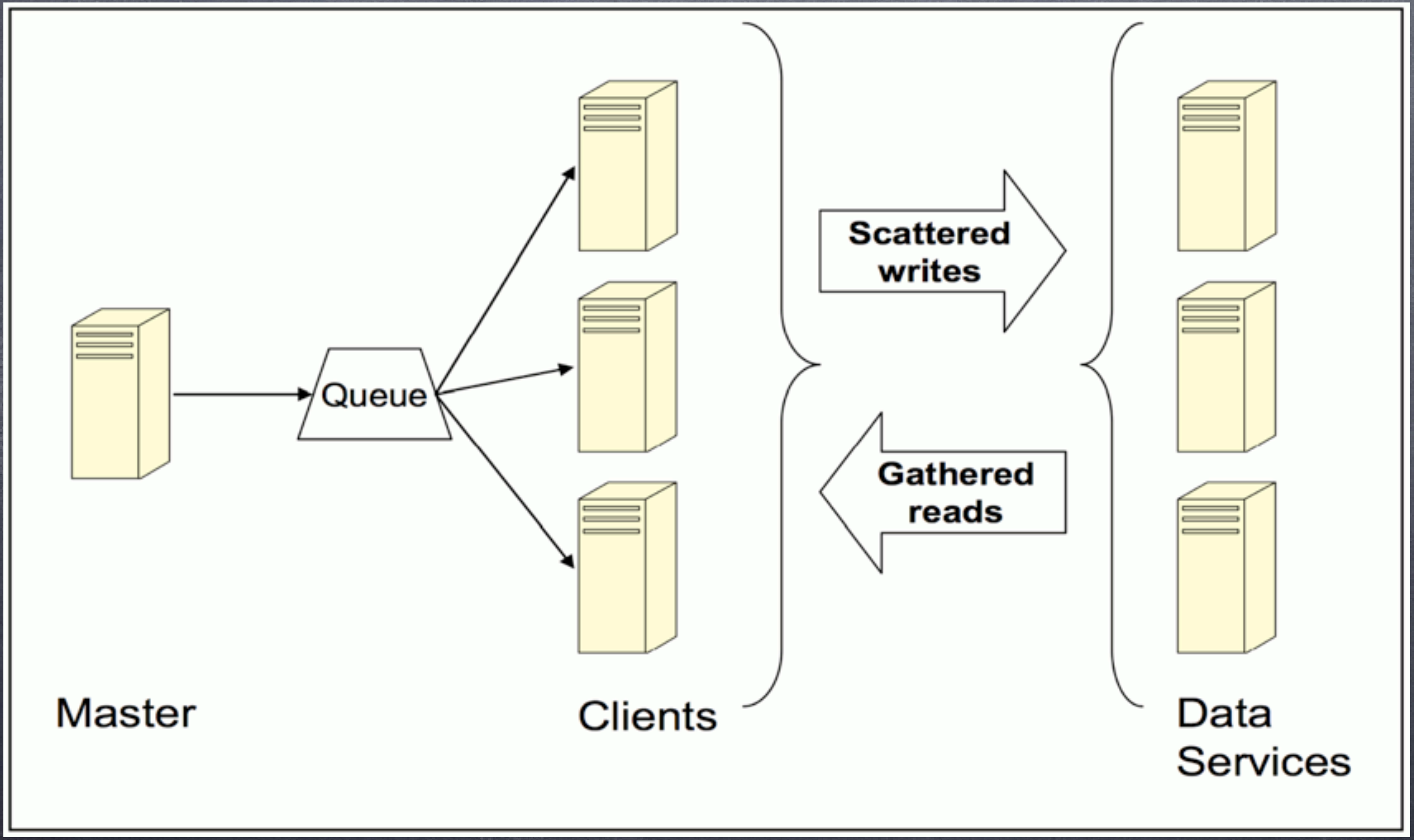
Distributed Jobs & Data Loading

- Defines distributed execution model for processing ordered data
- Distributed BigData program (a “master”)
  - Creates subtasks (“Clients”) that execute within the service containers in the cluster

Distributed Jobs cont.

- Master assigns work to the client
- Client runs parser/loader task
- Parser/Loader task
  - consumes pathnames
  - reads from files
  - parses RDF/XML
  - prepares & writes to the asynchronous buffer

BigData RDF


Queue
Scattered writes
Gathered reads
Master
Clients
Data Services

Asynchronous Buffer Writes

- All tasks share one buffer per client
- Plays several roles:
  - decouples the client tasks
  - breaks down tuples based on index
  - handles duplicate tuples
  - insures large data writes for efficient disk usage

RDF - Querying

- Uses
  - Sesame 2 (OpenRDF.org) backend but overrides query evaluation for efficiency
  - Uses “Pipeline Join” evaluation

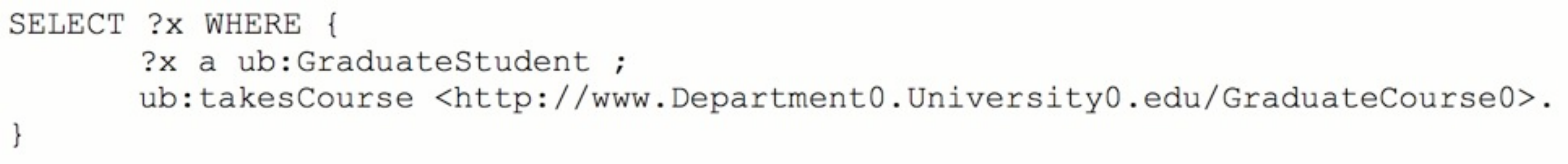
Sesame 2 (OpenRDF)

- Framework for Storing and Querying RDF data
- Includes
  - Various storage backends (memory, file, database (MySQL, etc.))
  - query languages (SPARQL, SeRQL)
  - Inferencers
  - Client/server protocols.

Pipeline Join

- Client submits a query
- Join Master is executed for that query
    - coordinates work to be performed
- Once rule is evaluated, results streamed back to join master task

Pipeline Join



```
SELECT ?x WHERE {
        ?x a ub:GraduateStudent ;
        ub:takesCourse <http://www.Department0.University0.edu/GraduateCourse0>.
}
```

- Two Patters:
  - (?x, rdf:type, ub:GraduateStudent)
  - (?x, ub:takesCourse, “DBandWeb”)
- Lexicon resolves into:
  - (?x, 8, 256)
  - (?x, 400, 3048)

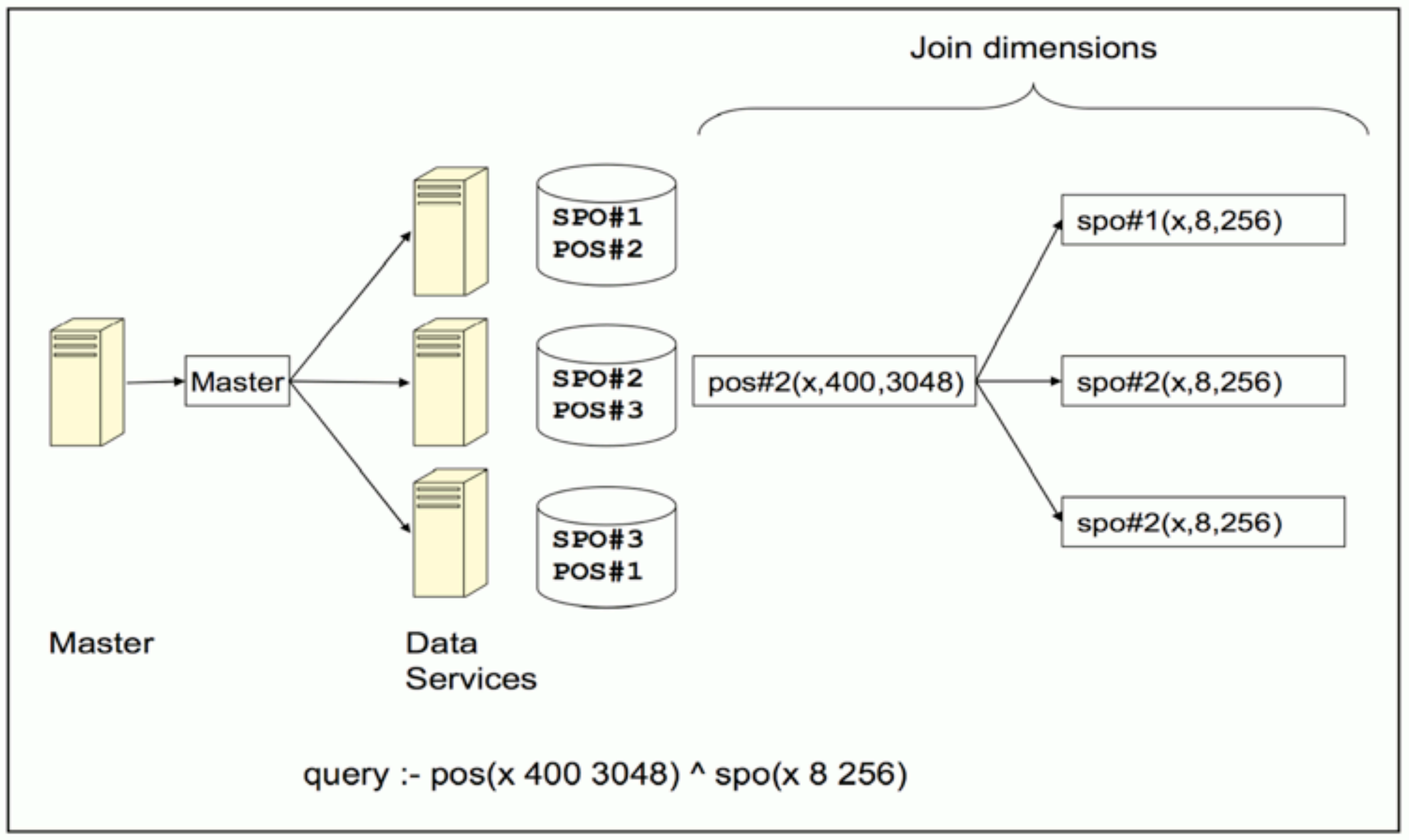
Pipeline Join cont.

- Join query considers the range counts, reorders them, and assigns them to specific paths
- ex.
  - query :- pos(x 400 3048) ^ spo(x 8 256)
- pos will evaluate against POS index
- spo will evaluate against SPO index

Pipeline Join cont.

- First join tuples are located at POS#2
- Second join tuples are located at SPO#1, SPO#2, and SPO#3

Pipeline Join cont.



query :- pos(x 400 3048) ^ spo(x 8 256)

Inference

- Two methods of inference
  - eager closure
  - materialize at query time

Eager Closure

- Computer the closure of the model over explicit triples
    - materializes all inferred data
- load those data items next to explicit triples
- Problem:
    - significant latency to compute
    - space requirements

Materialize @ Query

- Backward reasoning
  - Prolog systems
  - magic sets*
    - Lends itself to set at a time, not tuple at a time (RDF is tuple)

BigData Inferences

- Use Hybrid approach
- One big issue with eager
  - must keep inferred data up to date when more truth data is entered

References

- "BigData Scale-out Architecture Whitepaper." SYSTAP, LCC. 2010. April 15, 2013. http://www.bigdata.com/whitepapers/bigdata_whitepaper_10-13-2009_public.pdf

- Cloud Computing with BigData. "bigdata", 2012. PDF. April 16, 2013. http://assets.en.oreilly.com/1/event/12/Cloud%20Computing%20with%20bigdata%20Presentation.pdf

References

- SYSTAP, LLC. “BigData.” 2010. April 21, 2013. http://www.systap.com/bigdata.htm