
Bigdata[®]

Overview

Round table topics

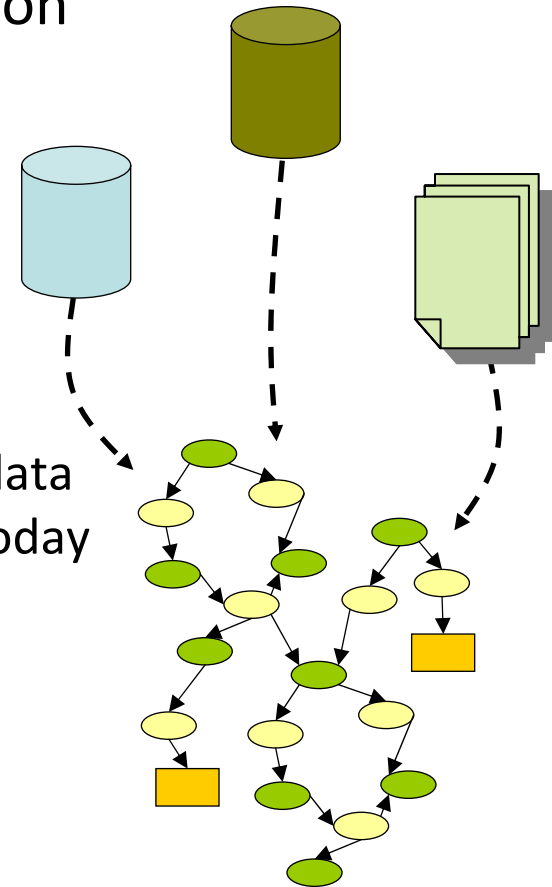
• Distribution, interoperability, and benchmarking	➤ Distributed database.
• Virtualized semantic repositories	➤ Not a current focus.
• Semantic data bus	➤ Exploring cloud scale approaches; Transactional remote object models.
• Embedded data processing	➤ Map Java code against data; Execute filters close to shards.
• Adaptive indexing and multi-modal retrieval	➤ Spatial indices; maintaining and querying custom indices.

What is “big data?”

- Big data is a new way of thinking about and processing massive data.
 - Petabyte scale
 - Commodity hardware
 - Distributed processing

The killer big data application

- Clouds + “Open” Data = Big Data Integration
- Critical advantages
 - *Fast* integration cycle
 - Open standards
 - Integrates heterogeneous data, linked data, structured data.
 - Opportunistic exploitation of data, including data which can not be integrated quickly enough today to derive its business value.



bigdata[®]

- Petabyte scale
- Dynamic sharding
- Commodity hardware
- Open source, Java
- HA Architecture
- High performance
- High concurrency (MVCC)
- Temporal database

Semantic web database

Key Differentiators

- Dynamic sharding
 - Incrementally scale from 10s, to 100s, to 1000s of nodes.
- Temporal database
 - Fast access to historical database states.
- HA Architecture
 - Built in design for high availability.
- Open source
 - Active development team including OEMs and Fortune 500 companies.

Petabyte scale

- Metadata service (MDS) used to locate shards.
- Maps a key range for an index onto a shard and the logical data service hosting that shard.
- MDS is heavily cached and replicated for failover.
- Petabyte scale (tera-triples) is easily achieved.
- Exabyte scale is much harder; breaks the machine barrier for MDS.

MDS scale	Data scale	Triples
71MB	terabyte	18,000,000,000
72GB	petabyte	18,000,000,000,000
74TB	exabyte	18,000,000,000,000,000

HA Architecture

- No single point of failure
 - Master election protocol uses zookeeper
 - Service registry uses jini
- MDS with replication scales to petabytes
- Data services
 - Writes and index builds are replicated
 - Guarantee consistent read on any node
 - Shard affinity
- Point in time rollback
 - Actually, you fork a version so old data remains accessible.
- Hot spares

See http://www.bigdata.com/whitepapers/bigdata_ha_whitepaper.pdf

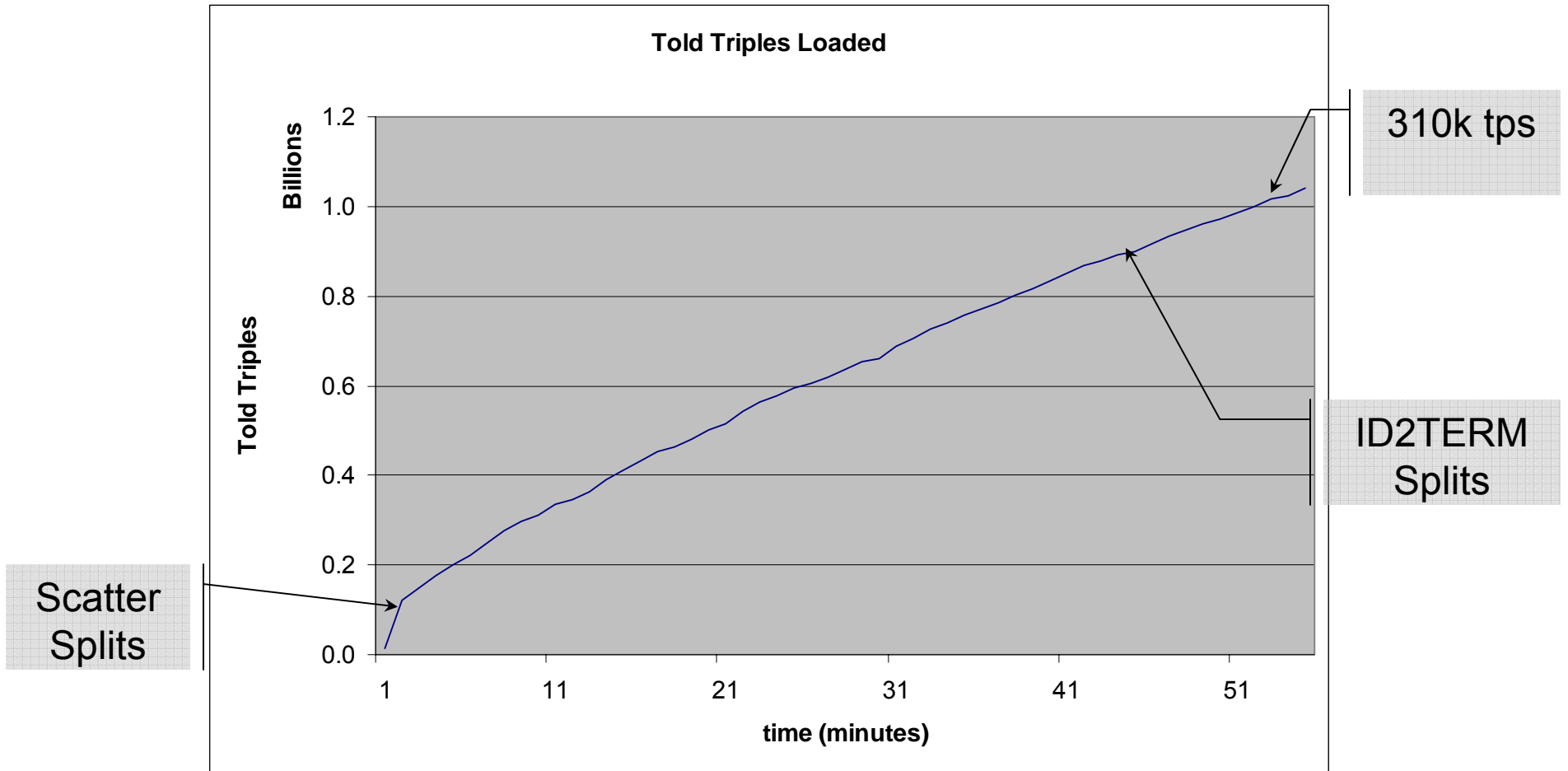
RDF Database Modes

- Triples
 - 2 lexicon indices, 3 statement indices.
 - RDFS+ inference.
 - All you need for lots of applications.
- Provenance
 - Datum level provenance.
 - Query for statement metadata using SPARQL.
 - No complex reification.
 - No new indices.
 - RDFS+ inference.
- Quads
 - Named graph support.
 - Useful for lots of things, including some provenance schemes.
 - 6 statement indices, so nearly twice the footprint on the disk.

Bulk Data Load

- Very high data load rates
 - 1B triples in under an hour (better than 300,000 triples per second on a 16 node cluster).
- Executed as a distributed job
 - Read data from a file system, the web, HDFS, etc.
- Database remains available for query during load
 - Read from historical commit points.

Bigdata[®] U8000 Data Load



Query evaluation

- Nested subquery
 - Clients demand data from the shards, process joins locally.
 - Can generate a huge number of RMI requests.
- Pipeline joins
 - Map *binding sets* over the shards, executing joins close to the data.
 - Faster for single machine and *much* faster for distributed query
- New join algorithms
 - E.g., push statement patterns
 - Latency and resource requirements
 - Etc.

Preparing a query

Original query:

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

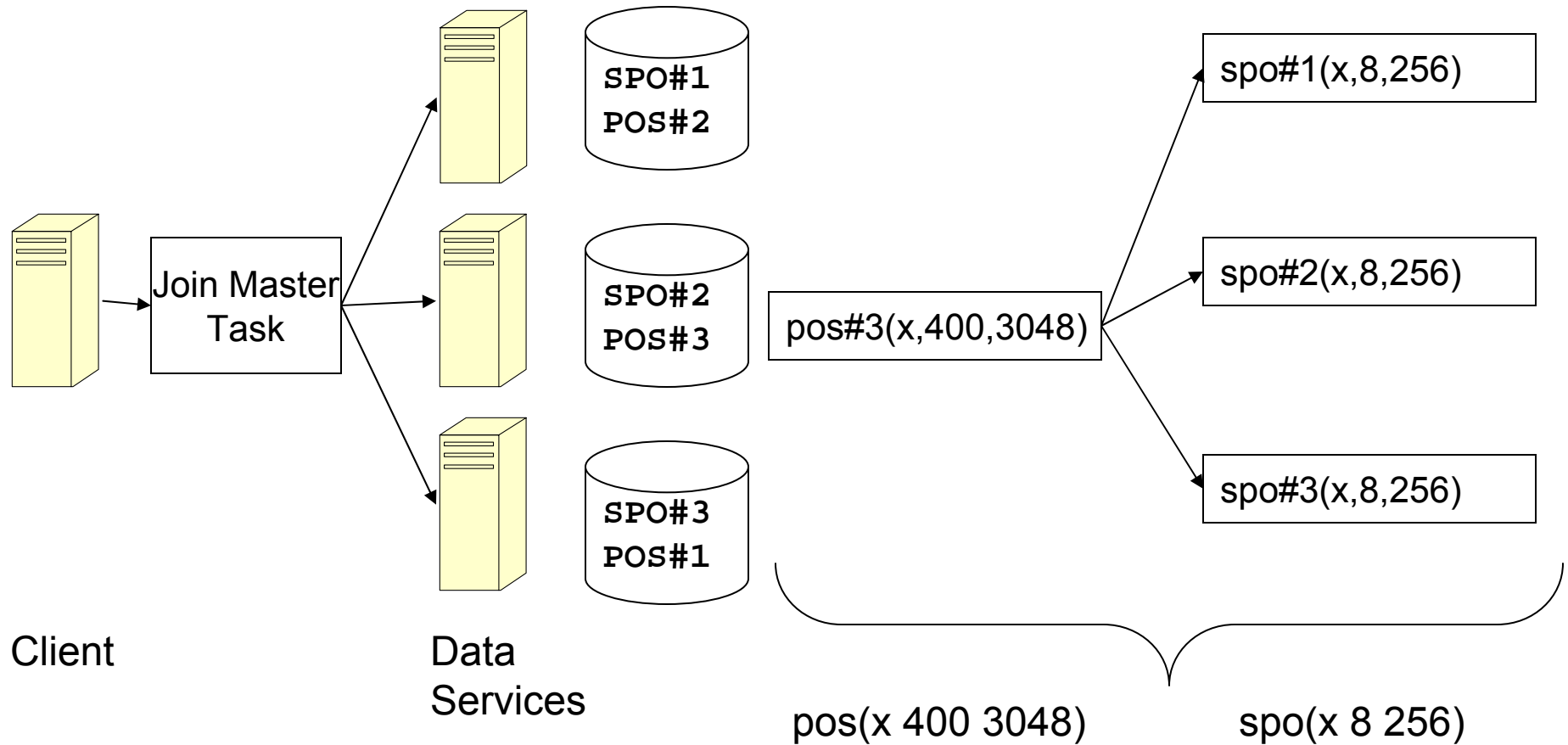
Translated query:

```
query :- (x 8 256) ^ (x 400 3048)
```

Query execution plan (access paths selected, joins reordered):

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

Pipeline join execution



Berlin SPARQL Benchmark (BSBM)

- This table reports the QMpH and QpS for concurrent clients against the 100M triple data set using the reduced query mix. Higher is better.

		Best SUT	bigdata	
		cold cache	cold cache	hot cache
	#of clients	4	4	4
QMpH	all queries	2,822.40	2,317.08	28,709.04
QpS	Q1	37.20	23.11	197.86
	Q2	58.10	78.24	208.20
	Q3	33.50	n/a	n/a
	Q4	15.80	14.13	140.00
	Q5	n/a	n/a	n/a
	Q6	n/a	n/a	n/a
	Q7	7.90	4.42	94.00
	Q8	12.00	11.65	185.88
	Q9	55.80	54.81	280.20
	Q10	21.60	11.48	215.14
	Q11	39.30	46.09	285.44
	Q12	31.30	55.39	250.50

- Q5 and Q6 are excluded in the BSBM reduced query mix.
- Q3 was excluded; it is handled by Sesame, which is painfully slow.
- Best SUT is highest score reported for any triple store for each metric in the Nov 2009 follow up to the BSBM study.
- Main outstanding bottleneck is concurrent disk I/O, which we will have resolved shortly.

Query Performance (LUBM U8000)

10/22/2009	#trials=10	#parallel=1		
Query	Time	Result#	delta-t	% change
query1	254	4	24	10%
query2	8,212,149	2,528	(10,227,868)	-55%
query3	194	6	(67)	-26%
query4	876	34	(422)	-33%
query5	1932	719	(75)	-4%
query6	713,445	69,222,196	(2,477,634)	-78%
query7	838	61	(29)	-3%
query8	3239	6463	(2,539)	-44%
query9	2,851,182	1,379,952	(2,699,119)	-49%
query10	121	0	(11)	-8%
query11	261	0	(88)	-25%
query12	1709	0	(227)	-12%
query13	47	0	9	24%
query14	646,517	63,400,587	(2,426,916)	-79%
Total	12,432,764	134,012,550	(17,834,962)	-59%

- Cluster of 10 nodes.
- 60% improvement in one week.

Bigdata[®] Roadmap – 2010

- Q2
 - Eliminate remaining bottlenecks.
 - Distributed query optimizations.
 - High-Availability.
- Q3
 - Simplified deployment, configuration, and administration.
 - 100 node operational deployment.
- Q4
 - Spatial indices

Bryan Thompson
Chief Scientist
SYSTAP, LLC
bryan@systap.com

bigdata®

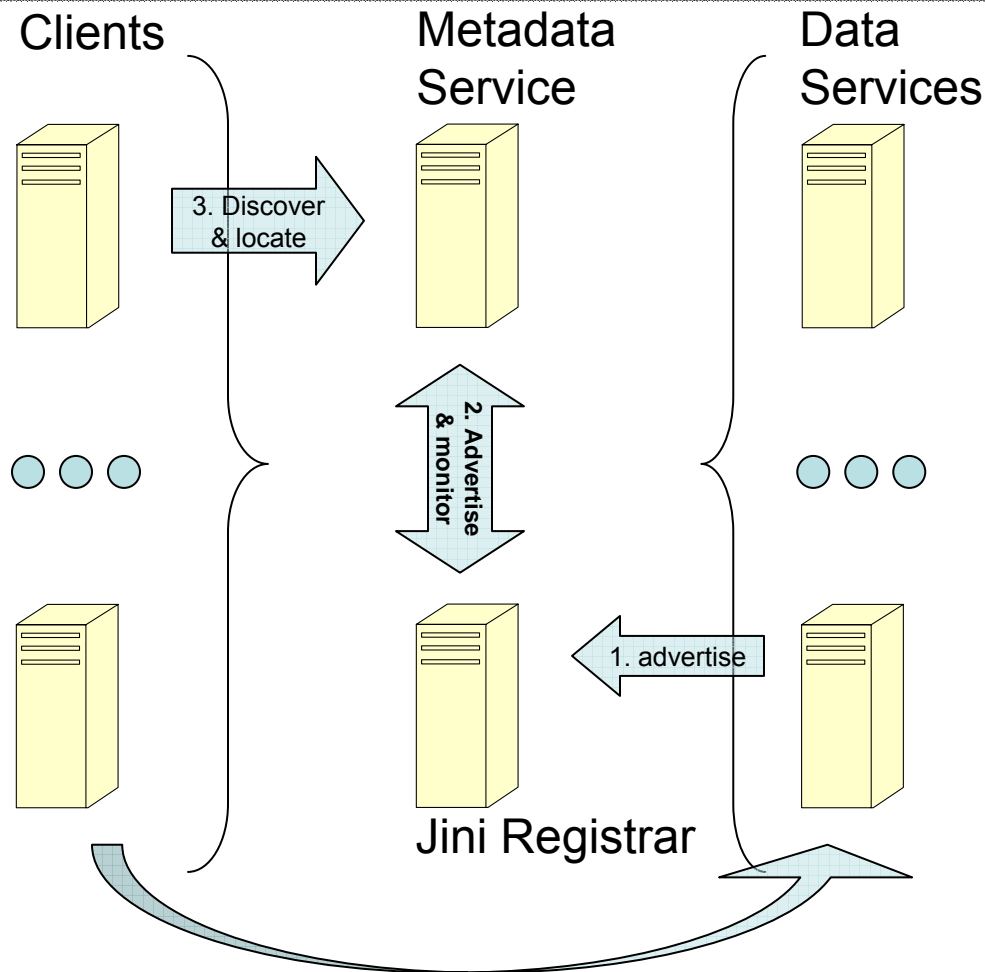
**Flexible
Reliable
Affordable
Web-scale computing.**

Backing material

Remaining Bottlenecks

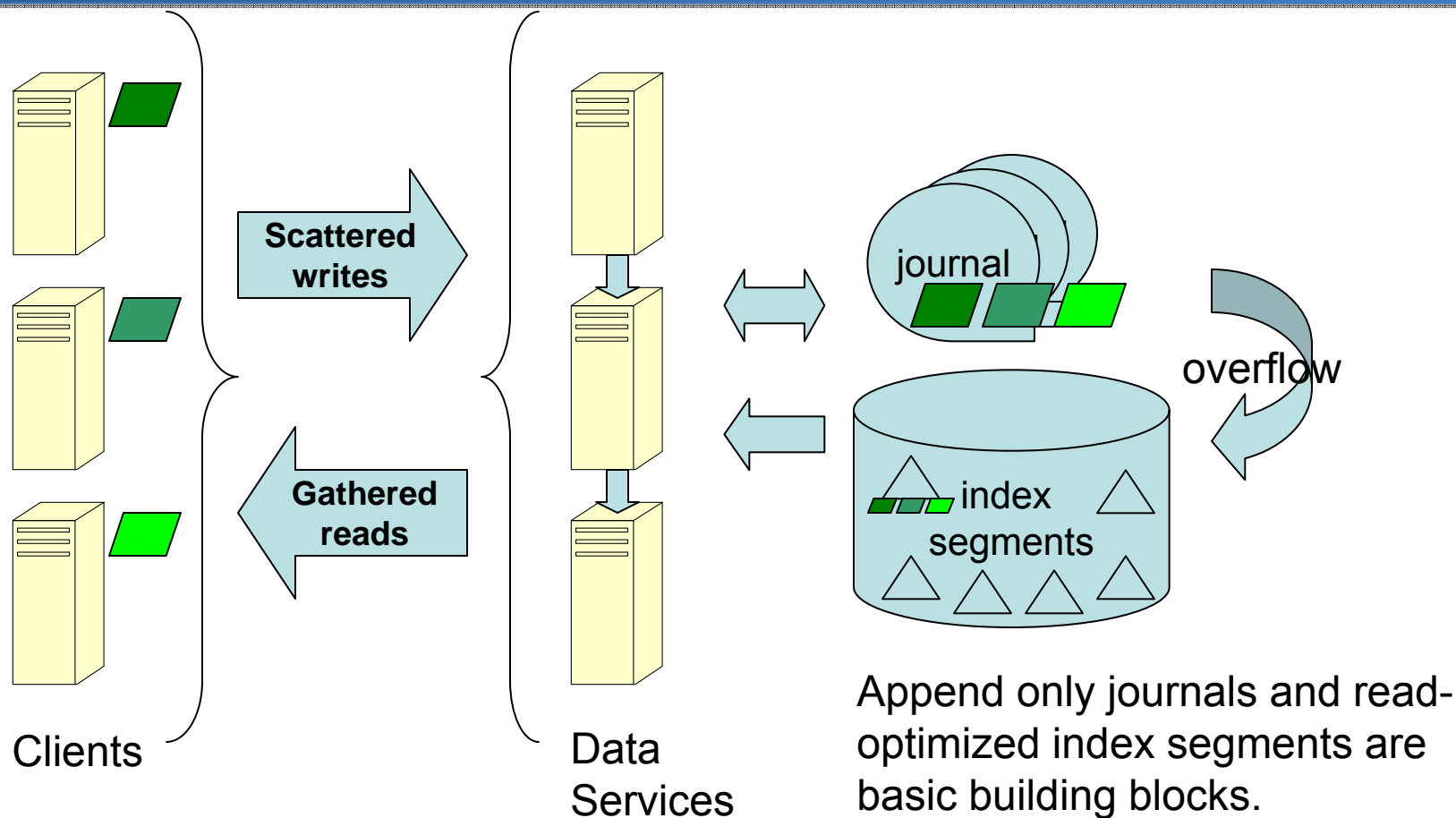
- Eliminate various hotspots
 - Non-blocking LIRS cache (with the infinispan project).
- Improved resource management
 - Joins buffers and threads.
 - Separate payload from RMI messages.
 - Schedule index partition splits proactively.
 - Buffer index writes for the target host, not the target shard.
- We expect to more than double performance
 - Already substantially improved by faster index builds.

Service Discovery



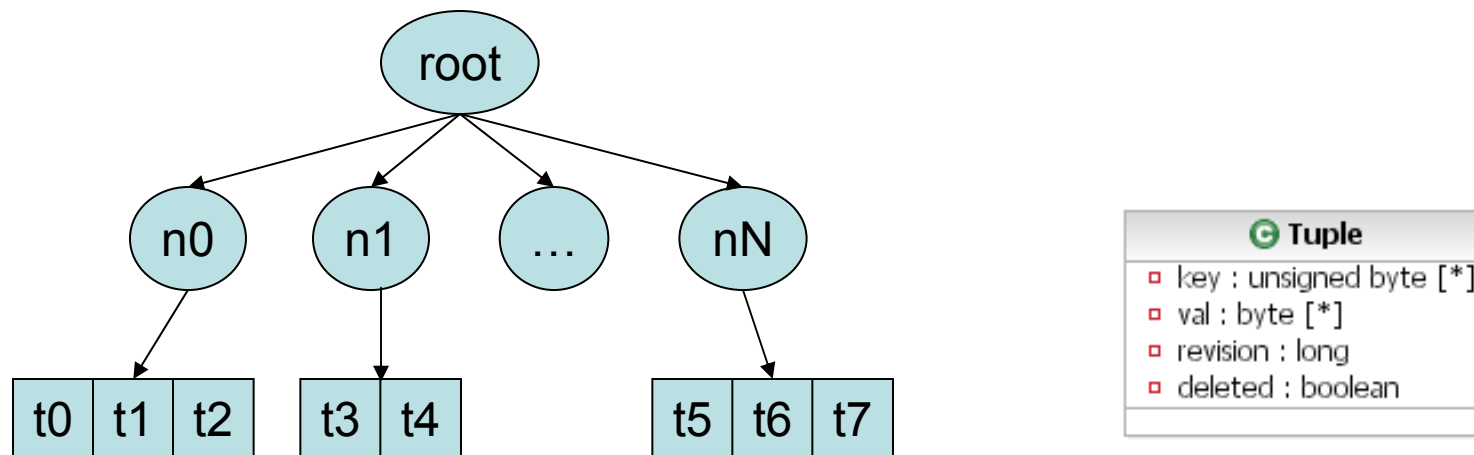
1. Services discover service registrars and advertise themselves.
2. Clients discover registrars, lookup the metadata service, and use it to obtain locators spanning key ranges of interest for a scale-out index.
3. Clients resolve locators to data service identifiers, then lookup the data services in the service registrar.
4. Clients talk directly to data services.
5. Client libraries encapsulate this for applications.

The Data Service

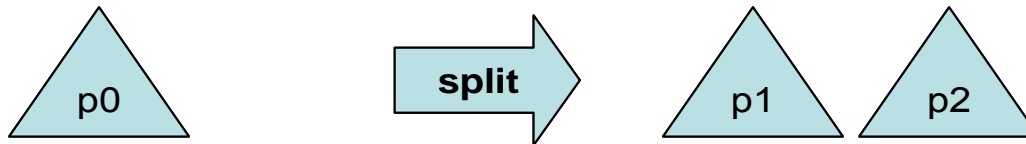


Bigdata[®] Indices

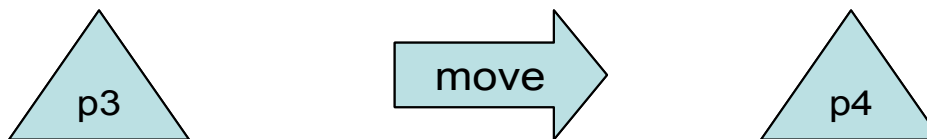
- Dynamically key-range partitioned B+Trees for indices
 - Index entries (tuples) map unsigned byte[] keys to byte[] values.
 - Tuples also have “delete flag” and timestamp
- Index partitions distributed across data services on a cluster
 - Located by centralized metadata service



Dynamic Key Range Partitioning



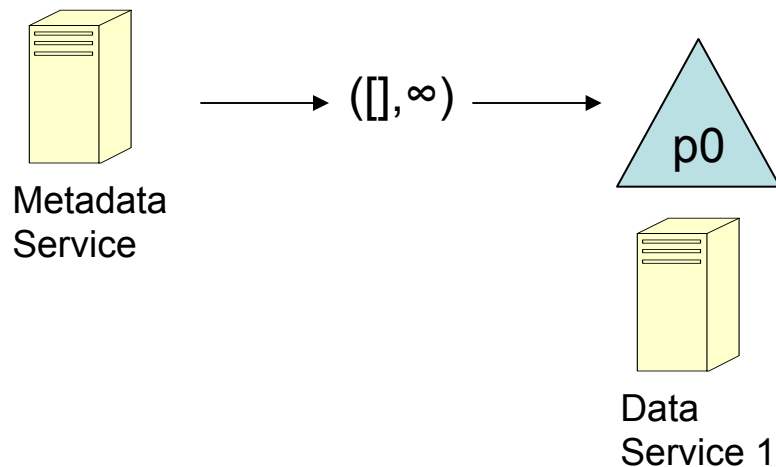
Splits break down the indices dynamically as the data scale increases.



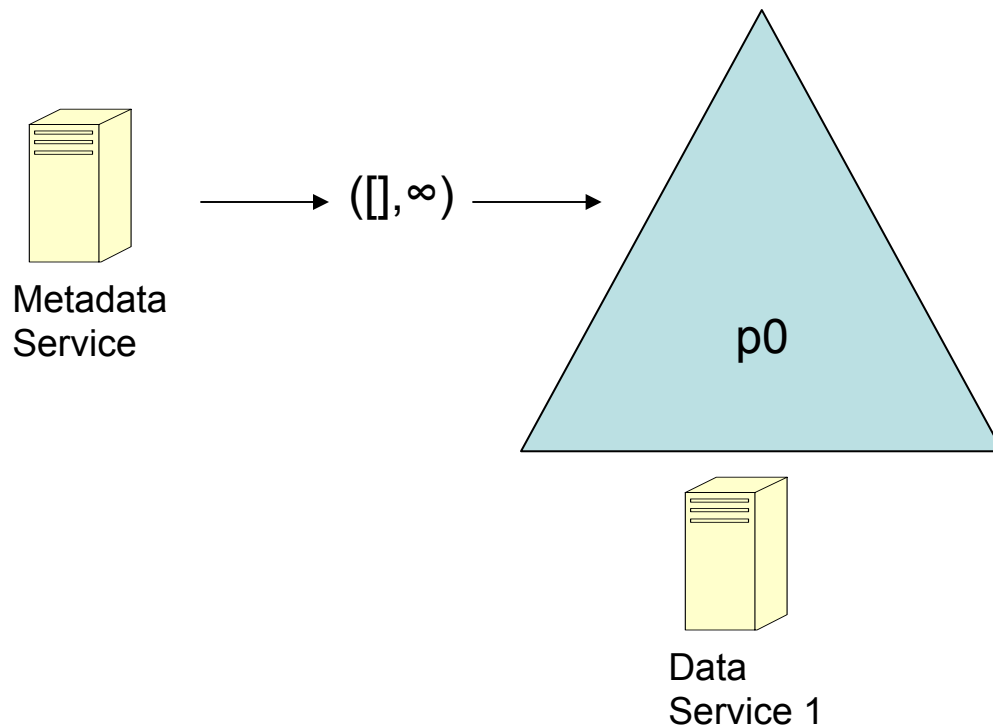
Moves redistribute the data onto existing or new nodes in the cluster.

Dynamic Key Range Partitioning

- Initial conditions place a single index partition on an arbitrary host representing the entire B+Tree.

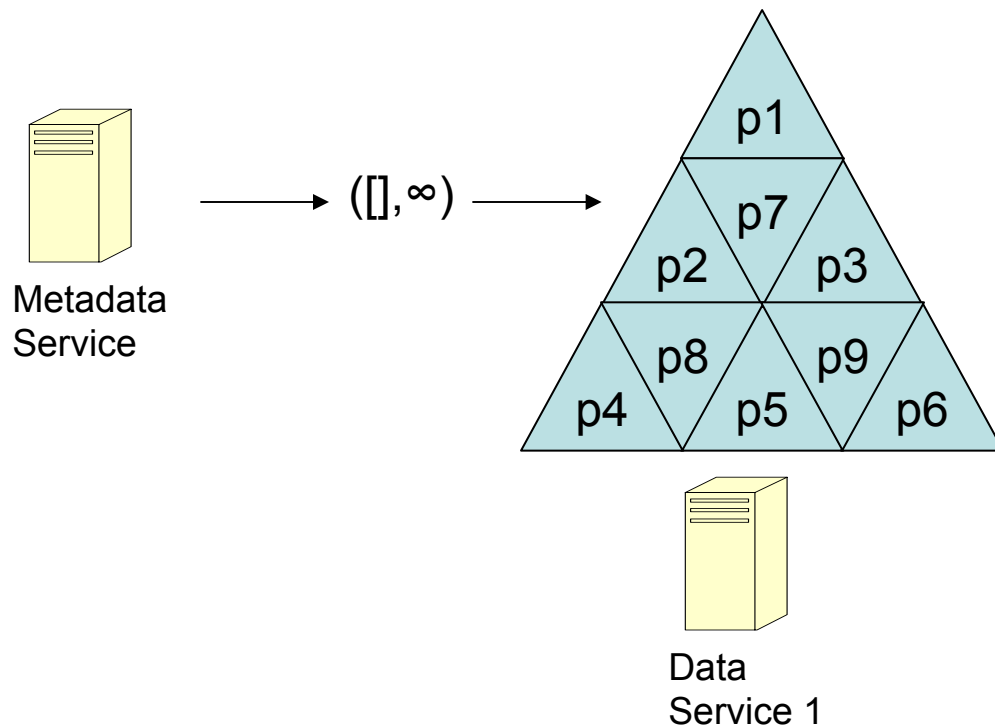


Dynamic Key Range Partitioning



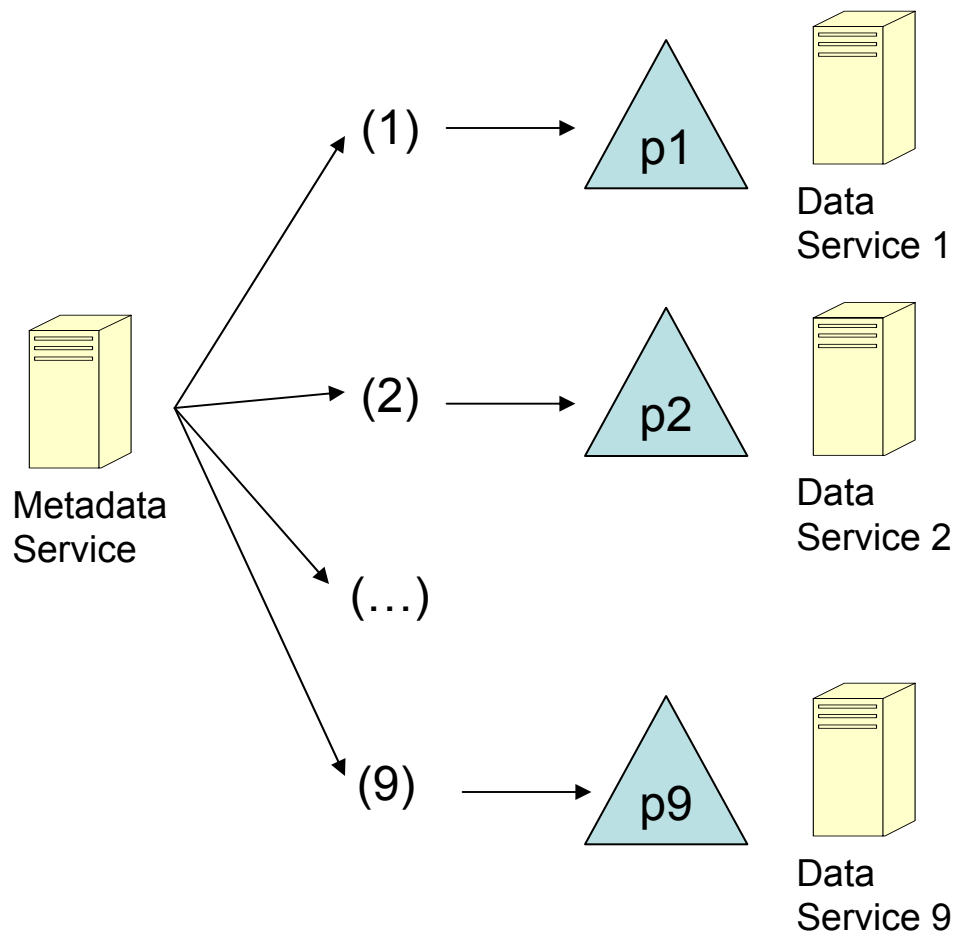
- Writes cause the partition to grow. Eventually its size on disk will exceed a preconfigured threshold.

Dynamic Key Range Partitioning



- Instead of a simple two-way split, the initial partition is “scatter-split” so that all data services can start managing data.
- Nine data services in this example.

Dynamic Key Range Partitioning



- The newly created partitions are then moved to the various data services.
- Subsequent splits are two-way and moves occur based on relative server load (decided by load balancer service).

What's in a shard?

- Data for a key-range of a named index
- An ordered view of:
 - One or more journal files (used to absorb writes).
 - Zero or more index segments (perfect, read-optimized B+Tree files).
 - Reads test each journal or segment in turn looking for a hit.
 - Delete markers prevent old data from being read.
- As the journal fills up with writes
 - Data are asynchronously migrated from the old journal onto index segment files.
- When the view gets complex
 - A compacting merge creates a single index segment file.
- When the view gets big, we split it.