SYSTAP, LLC

Graphs

Graph Databases

Graph Analytics on GPUs
Graph Database

- High performance, Scalable
  - 50B edges/node
  - High level query language
  - Efficient Graph Traversal
  - High 9s solution
- Open Source (Subscriptions)
  - Autodesk, EMC, market data, genomics and personalized medicine, etc.

GPU Analytics

- Extreme Performance
  - 5-100x faster than graphlab
  - 10,000x faster than graphdbs
- DARPA funding
- Disruptive technology
  - Early adopters
  - Huge ROIs
- Open Source
Related “Graph” Technologies

Redpoint repositions existing technology.

MPGraph compares favorably with high end hardware solutions from YARC, Oracle, and SAP, but is open source and uses commodity hardware.
Similar models, different problems

- Graph query and graph analytics (traversal/mining)
  - Related data models
  - *Very* different computational requirements
- Many technologies are a bad match or limited solution
  - Key-value stores (bigtable, Accumulo, Cassandra, HBase)
  - Map-reduce
- Anti-pattern
  - Dump all data into “big bucket”
Similar models, different problems

- Graph query and graph analytics (traversal/mining)
  - Related data models
  - Very different computational requirements
- Many technologies are a bad match or limited solution
  - Key-value stores (bigtable, Accumulo, Cassandra, HBase)
  - Map-reduce
- Anti-pattern
  - Dump all data into “big bucket”

Storage and computation patterns must be correctly matched for high performance.
Optimize for the right problem

- **Graph Query**
  - Declarative Query Language (SPARQL)
    - Query optimization is critical for performance.
  - Index Locality (1D partitioning, multiple indices)
    - Get everything about a subject on one page of the index.
  - Scale-out must flow queries over the data
    - Otherwise slams the network and the client
  - Must order and constrain joins to read as little data as possible
    - As-bound vectored nested index joins (bigdata)
    - Sideways information passing and merge joins (RDF3X)
Vectored Query in Scale-Out

1. The initial read on the POS index provides bindings for ?s.

2. The bindings for ?s are plugged into the next triple pattern and mapped across the index for that JOIN (SPO).

3. The intermediate solutions are evaluated against the SPO index partitions in parallel.
Optimize for the right problem

- Storage and computation patterns must be correctly matched for high performance.
- Graph analytics:
  - Parallelism – work must be distributed and balanced.
  - Memory bandwidth – memory, not disk, is the bottleneck
  - 2D partitioning – $O(N)$ communications pattern (versus $O(N^2)$)
Accelerated Graph Analytics

1. Graph data in Accumulo
2. Parallel reduce writes 2D edge partitioning
3. Parallel read of 2D data GPU DRAM
4. Extreme speed for graph analytics.
5. Parallel reduction writes output to HDFS or other target.

Parallel File System (HDFS)
The Semantic Web

- The Semantic Web is a stack of standards developed by the W3C for the interchange and query of metadata and graph structured data.
  - Open data
  - Linked data
  - Multiple sources of authority
  - Self-describing data and rules
  - Federation or aggregation
  - And, increasingly, provenance
The Standards or the Data?


S. Bratt, 2006.
The data – it’s about the data
The killer “big data” app

- Clouds + “Open” Data = Big Data Integration
- Critical advantages
  - Fast integration cycle
  - Open standards
  - Integrate heterogeneous data, linked data, structured data, data at rest, and streams.
    - Maintain fine-grained provenance of federated data.
  - Opportunistic exploitation of data
    - Fragmented information
    - Dynamic information
    - Latent Information (graph mining)
Unified Architecture (example)

Heterogeneous Data Sources as Input

- Streams
- Unstructured
- Semi-structured
- Structured

Unified Data Model

Resource Centric (Linked Data)

- Discover
- Federate
- Aggregate

Unified Compute and Storage Model

Data Bus
- Business Logic
- Web Clients
- Peer Systems

Data Cache
- Key Value Stores

Database (SSD)
- Aggregated –or– federated
- High-level Query (SPARQL)

Graph Mining (GPUs)
- Graph traversal / mining
  - “Think like a vertex”

Data Bus
- Business Logic
- Web Clients
- Peer Systems

Data Cache
- Key Value Stores

Database (SSD)
- Aggregated –or– federated
- High-level Query (SPARQL)

Graph Mining (GPUs)
- Graph traversal / mining
  - “Think like a vertex”
Road Map

• Column-wise
  – Faster load and query.
  – Increased data density and scaling.
  – Integration point with GPU (shared data).

• Multi-node GPU
  – 2D decomposition (DARPA STTR)

• Performance optimization for scale-out
  – Reducing latency and increasing throughput
  – Integration point for SPARQL acceleration and 2D GPU cluster.

• SPARQL on GPU
  – Query at 3 billion edges/second
  – Same underlying library, but horizontal scaling is NOT 2D.
Customers and Use Cases
Autodesk PLM360

Manufacturing Product Data is Heterogeneous

...and difficult to find, re-use, and share
Hadoop / bigdata® pipeline

Map/Reduce Layer

Inference Cloud

BD Journals on PFS / HDFS

HA Query Cloud

Scalable inference workload

Linear scaling on query throughput

Can be used for custom quads-mode inference strategies
Federated Query & Custom Services

• Language extension for remote services
  SERVICE uri { graph-pattern }

• Integrated into the vectored query engine
  – Solutions vectored into, and out of, remote end points.
  – Control evaluation order query hints:
    • runFirst, runLast, runOnce, etc.

SERVICE { .... } hint:Prior hint:runFirst “true”.

• ServiceRegistry
  – Configure service end point behavior
  – Embed custom “services” (custom indices, monitor transactions, etc).
High Availability

• Shared nothing architecture
  – Same data on each node
  – Coordinate only at commit

• Scaling
  – 50 billion triples or quads
  – Query throughput scales linearly

• Self healing
  – Automatic failover
  – Automatic resync after disconnect
  – Online single node disaster recovery

• Online Backup
  – Online snapshots (full backups)
  – HA Logs (incremental backups)

• Point in time recovery (offline)
EMC ProSphere

- Host-to-storage management solution
  - No single model. Information must be combined from many device vendors. RDF is a natural solution.
  - Application deployed as appliance into data centers everywhere.
    - Bundles the bigdata platform.
  - 220+ Engineers in US and India.

- SYSTAP
  - provides support, custom services, feature development, training.
bigdata® usage: EMC ProSphere

Topology Service JVM
- REST API
- Topology Service
- SAIL/bigdata API
- bigdata RDF store (journal)

Maps Service JVM
- Maps Service
- REST API
- Temp store
- GraphML
- SAN topology view

SAN topology view
Knowledge Base of Biology (KaBOB)

17 databases
- Entrez Gene
- DIP
- UniProt
- GOA
- GAD
- HGNC
- InterPro
- ... 12 ontologies
- Gene Ontology
- Sequence Ontology
- Cell Type Ontology
- ChEBI
- NCBI Taxonomy
- Protein Ontology
- ...
Kepler GK110 Die Photo

- Most complex commercial IC
  - 7.1 billion transistors.
  - 3x gain in power efficiency.
  - 2,496 CUDA cores.
  - 1.5 MB L2 Cache.
Graph Processing

GPUs
Graphs
and
Graph Data Mining
GPU Graph Processing

• Motivation – *speed*
  – 3 out of the top 5 super computers are GPU clusters
  – 3.3 B traversed edges per second (one GPU : Merrill, 2011)
  – 8.3 B traversed edges per second (quad GPU configuration : ibid)

• Goal
  – Blindingly fast SPARQL QUERY and graph data mining on GPU clusters
  – 20 minutes on Accumulo => 27 milliseconds on a GPU.

• Open source
  – Deploy in workstations, HPC clusters, EC2, or your own data center
Many Core is the Future

- Top 500 Super Computer Sites:
  - 3 out of top 5 are GPU clusters (11/2011)
  - #1 and #8 (11/2012)
- CPU Clock Rates are stagnant.
- Simple compute units + parallelism => Increased performance.

Historical Single-/Double-Precision Peak Compute Rates

<table>
<thead>
<tr>
<th>Date</th>
<th>GFLOPS</th>
<th>Precision</th>
<th>Vendor</th>
</tr>
</thead>
<tbody>
<tr>
<td>2002</td>
<td></td>
<td>SP</td>
<td>AMD (GPU)</td>
</tr>
<tr>
<td>2004</td>
<td></td>
<td>DP</td>
<td>NVIDIA (GPU)</td>
</tr>
<tr>
<td>2006</td>
<td></td>
<td>SP</td>
<td>Intel (CPU)</td>
</tr>
<tr>
<td>2008</td>
<td></td>
<td>DP</td>
<td>Intel Xeon Phi</td>
</tr>
<tr>
<td>2010</td>
<td></td>
<td>SP</td>
<td>AMD (GPU)</td>
</tr>
<tr>
<td>2012</td>
<td></td>
<td>DP</td>
<td>NVIDIA (GPU)</td>
</tr>
</tbody>
</table>

TOP 10 Systems - 11/2011

1. K computer, SPARC64 VIIIfx 2.0GHz, Tofu interconnect
2. NUDT YH MPP, Xeon X5670 6C 2.93 GHz, NVIDIA 2050
3. Cray XT5-HE Opteron 6-core 2.6 GHz
4. Dawning TC3600 Blade, Intel X5650, NVidia Tesla C2050 GPU
5. HP ProLiant SL390s G7 Xeon 6C X5670, Nvidia GPU, Linux/Windows
GPUs – A Game Changer for Graph Analytics?

- Graphs are everywhere in data, also a powerful data model for federation
- GPUs may be the technology that finally delivers real-time analytics on large graphs
  - 10x speedup over CPU
  - 10x DRAM bandwidth
- This is a hard problem
  - Data dependent parallelism
  - Non-locality
  - PCIe bus is bottleneck
- Significant speed up over CPU on BFS
  - 3 billion edges per second on one GPU (see chart).
- Roadmap
  - GPU accelerated vertex-centric graph mining platform.
  - GPU accelerated graph query
GAS – a Graph-Parallel Abstraction

- Graph-Parallel Vertex-Centric API ala GraphLab
- “Think like a vertex”
- Gather: collect information about my neighborhood
- Apply: update my value
- Scatter: signal adjacent vertices
- Can write all sorts of graph algorithms this way
  - BFS, PageRank, Connected Component, Triangle Counting, Max Flow, etc.
**Triangle Counting on Twitter Graph**

34.8 Billion Triangles

- **Hadoop**
  - [WWW’11]
  - 1636 Machines
  - 423 Minutes

- **PowerGraph**
  - 64 Machines
  - 15 Seconds

**Why? Wrong Abstraction →**

Broadcast $O(\text{degree}^2)$ messages per Vertex

---

S. Suri and S. Vassilvitskii, “Counting triangles and the curse of the last reducer,” WWW’11
GPU Speedups vs GraphLab (SSSP)
Graph Mining on GPU Clusters

- 2D partitioning (aka vertex cuts)
- Minimizes the communication volume.
- Batch parallel Gather in row, Scatter in column.
RDF Database

Overview
New features
Integration Points
REST API
Bigdata 1.3

- Fast, scalable, open source, standards compliant database
  - Single machine to 50B triples or quads
    - Plus a dedicated provenance mode.
  - Scales horizontally on a cluster
  - SPARQL 1.1 Query, Property Paths, Update, Federated Query, etc.
  - Native RDFS+ inference.
    - Vectored query engine.
  - High Availability
  - RDF Graph Mining
Sesame Sail & Repository APIs

- Java APIs for managing and querying RDF data
- Extension methods for bigdata:
  - Non-blocking readers
  - RDFS+ truth maintenance
  - Semantic / graph search
  - Change history
  - Custom services
  - Etc.
NanoSparqlServer

- Easy deployment
  - Embedded or Servlet Container
  - Java client encapsulates remote operations.

- High performance SPARQL end point
  - Optimized for bigdata MVCC semantics (queries are non-blocking)
  - Built in resource management
  - Scalable!

- Simple REST API
  - SPARQL 1.1 Query and Update
  - Simple and useful REST-ful INSERT/UPDATE/DELETE methods
  - ESTCARD exposes super fast range counts for triple patterns
  - “Explain” a query.
  - Monitor / cancel running queries.
SPARQL Query

- High level query language for graph data.
  - Standard (W3C)
- Based on graph pattern matching.
  - SELECT vars WHERE graph-pattern
    - Returns result set (table)
  - CONSTRUCT template WHERE graph-pattern
    - Returns sub-graph
  - DESCRIBE uri
    - Returns graph for that object.
- Database can optimize physical storage and joins
  - Versus navigation-only APIs such as blueprints

```sparql
SELECT ?x ?z
WHERE {
}
```
BSBM 100M (Single Server)

- Graph shows series of trials for the BSBM reduced query mix (w/o Q5).
- Metric is Query Mixes per Hour (QMph). Higher is better.
- 8 client curve shows JVM and disk warm up effects. Both are hot for 16 client curve.
- Occasional low points are GC.
- Apple mini (4 cores, 16G RAM and SSD). Machine is CPU bound at 16 clients. No IO Wait.
JVM Heap Pressure

JVMs provide fast evaluation (rivaling hand-coded C++) through sophisticated online compilation and auto-tuning.

However, a non-linear interaction between the application workload (object creation and retention rate), and GC running time and cycle time can steal cycles and cause application throughput to plummet.
Choose standard or analytic operators

- Easy to specify which
  - URL query parameter or SPARQL query hint

- Java operators
  - Use the managed Java heap.
  - Can sometimes be faster or offer better concurrency
    - E.g., distinct solutions is based on a concurrent hash map
  - BUT
    - The Java heap can not handle very large materialized data sets.
    - GC overhead can steal your computer

- Analytic operators
  - Scale up gracefully
  - Zero GC overhead.
SPARQL Update

• Graph Management
  – Create, Add, Copy, Move, Clear, Drop

• Graph Data Operations
  – LOAD uri
  – INSERT DATA, DELETE DATA
  – DELETE/INSERT

( WITH IRIref )?
  ( ( DeleteClause InsertClause? ) | InsertClause )
  ( USING ( NAMED )? IRIref )* WHERE GroupGraphPattern

• Can be used as a RULES language, update procedures, etc.
SPARQL UPDATE Extension

- **Language extension for SPARQL UPDATE**
  - Adds durable solution sets.

  ```sql
  INSERT INTO %solutionSet1
  SELECT ?product ?reviewer WHERE { … }
  ```

- **Easy to slice result sets:**

  ```sql
  SELECT ... { INCLUDE %solutionSet1 } OFFSET 0 LIMIT 1000
  SELECT ... { INCLUDE %solutionSet1 } OFFSET 1000 LIMIT 1000
  SELECT ... { INCLUDE %solutionSet1 } OFFSET 2000 LIMIT 1000
  ```

- **Re-group or re-order results:**

  ```sql
  SELECT ... { INCLUDE %solutionSet1 } GROUP BY ?x
  SELECT ... { INCLUDE %solutionSet1 } ORDER BY ASC(?x)
  ```

- **Expensive JOINs are NOT recomputed.**
Semantic Search

- Enable in your KB properties:
  - com.bigdata.rdf.store.AbstractTripleStore.textIndex=true.

- Simple full text search:
  ```
  prefix bd: <http://www.bigdata.com/rdf/search#>
  select ?s, ?o {
    ?o bd:search "mike" . # all literals with "mike" token.
    ?s ?p ?o . # all subjects for those literals.
  }
  ```

- Lots of options (cosine relevance, rank, match all terms, etc.).

- Low latency, web facing applications built by “slicing” the search index.

Federated Query & Custom Services

• Language extension for remote services
  
  SERVICE uri { graph-pattern }

• Integrated into the vectored query engine
  – Solutions vectored into, and out of, remote end points.
  – Control evaluation order query hints:
    • runFirst, runLast, runOnce, etc.

  SERVICE { .... }
  hint:Prior hint:runFirst “true”.

• ServiceRegistry
  – Configure service end point behavior
  – Embed custom “services” (custom indices, monitor transactions, etc.)
RDF Graph Mining

- “GAS” SERVICE – 2x faster than neo4j, 4x faster than titan.
- Handles typed linked and link attributes efficiently

PREFIX gas: <http://www.bigdata.com/rdf/gas#>
SELECT ?depth (count(?out) as ?cnt) {
    SERVICE gas:service {
        gas:program gas:gasClass "com.bigdata.rdf.graph.analytics.BFS" .
        gas:program gas:in <ip:/112.174.24.90> . # one or more times, specifies the initial frontier.
        gas:program gas:out ?out . # exactly once - will be bound to the visited vertices.
        gas:program gas:out1 ?depth . # exactly once - will be bound to the depth of the visited vertices.
        gas:program gas:maxIterations 4 . # optional limit on breadth first expansion.
    }
}
group by ?depth
order by ?depth

SPARQL Query

<table>
<thead>
<tr>
<th>depth</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>207</td>
</tr>
<tr>
<td>2</td>
<td>1,985</td>
</tr>
<tr>
<td>3</td>
<td>9,861</td>
</tr>
<tr>
<td>4</td>
<td>29,366</td>
</tr>
</tbody>
</table>
Highly Available Replication Cluster

(HAJournalServer)

(bigdata 1.3 release)
HA Replication Cluster

• Same backend database (Journal + RWStore)
  – Same data scale as Journal.
  – Same IO profile as Journal.
  – Same low latency query profile as Journal.

• REST API is mandatory
  – SPARQL Query
  – SPARQL UPDATE
  – SPARQL Federated Query
    • Can be used to cross tenant or machine boundaries.

• Writes on leader
  – Identified in zookeeper or using REST API.

• Query on leader or followers
  – Each query is answered 100% locally.
    • Zero coordination overhead.
    • Linear scaling in query throughput
High Availability

- Shared nothing architecture
  - Same data on each node
  - Coordinate *only* at commit

- Scaling
  - 50 billion triples or quads
  - Query throughput scales linearly

- Self healing
  - Automatic failover
  - Automatic resync after disconnect
  - Online single node disaster recovery

- Online Backup
  - Online snapshots (full backups)
  - HA Logs (incremental backups)

- Point in time recovery (offline)
High Availability

- Write on the leader. Read on any service.
- Writes replicated using low-level transfers.
- Quorum fully consistent at each commit.
High Availability

- Write on the leader. Read on any service.
- Writes replicated using low-level transfers.
- Quorum fully consistent at each commit.
Self-Healing

Service can fail for a variety of reasons:
- JVM down
- Machine down
- Network partition
- Zookeeper timeout
- Discovery failure
- Wrong commit point
- Severe clock skew

- Goal is to guarantee eventual consistency without allowing intermediate illegal states.
- Persistent state of the service must remain self-consistent
HA Load Balancer

- Transparent Proxy
- Writes proxied to the leader.
- Reads load balanced over quorum.
  - Host metrics (CPU, IO Wait)
  - Service metrics (GC Time)
- Custom policies
  - Tenant aware
  - Low & high latency pools
HA Deployment

• EC2
  • SSD instance types (IOPS, ephemeral)
  • Snapshots and logs on EBS (durable)
  • Restore from EBS on instance restart
  • Coming soon
    • Click start HA clusters on EC2

• Private clouds
  • OpenStack
  • chef/puppet
HA Service Architecture

Unified API

REST API (SPARQL, GAS, etc.)
- SPARQL XML
- RDF/XML
- Turtle
- SPARQL JSON
- N-Triples
- TriG
- N-Quads
- RDF/JSON

Zookeeper

Application Client

ServiceStarter
- HAJournalServer
  - jetty
  - Load Balancer
  - REST API
  - HAGlue API
  - Journal
  - Lookup Service
  - Class Server

HA Replication Cluster

Ensemble
- Zookeeper
- Zookeeper
- Zookeeper
BSBM 100M (3-Node HA Cluster)

- 3-Node, Shared-Nothing Replication Cluster
  - 3x 2011 Mac Mini (4 cores, 16G RAM and SSD).
- Query Scales Linearly
- CPU bound
  - 70-90k QMnP on newer servers.

Query Performance Scales Linearly with Cluster Size
BSBM 100M, 3-node replication cluster using Intel Mac Minis

Aggregate Throughput

Per-node Throughput
HA DEMO

- Brief demonstration of an HA-3 cluster
- Start 3 services (A,B,C)
- Load some data (TBL foaf crawl)

```sql
DROP ALL;
LOAD <file:/Users/bryan/Documents/workspace/BIGDATA_RELEASE_1_2_0/bigdata-rdf/src/resources/data/foaf/data-0.nq.gz>;
LOAD <file:/Users/bryan/Documents/workspace/BIGDATA_RELEASE_1_2_0/bigdata-rdf/src/resources/data/foaf/data-1.nq.gz>;
LOAD <file:/Users/bryan/Documents/workspace/BIGDATA_RELEASE_1_2_0/bigdata-rdf/src/resources/data/foaf/data-2.nq.gz>;

- Identical data on all services
  SELECT (count(*) as ?c) { ?s ?p ?o } LIMIT 1

- Self-healing
  - Shutdown C
  - Load more data on A + B. Commit.
    LOAD <file:/Users/bryan/Documents/workspace/BIGDATA_RELEASE_1_2_0/bigdata-rdf/src/resources/data/foaf/data-3.nq.gz>;
  - Restart C. Resyncs and joins quorum.

- Snapshot service.
```

<table>
<thead>
<tr>
<th>File</th>
<th>Quads</th>
<th>Size (gz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>timbl/data-0.nq.gz</td>
<td>89</td>
<td>2.5K</td>
</tr>
<tr>
<td>timbl/data-1.nq.gz</td>
<td>16516</td>
<td>293K</td>
</tr>
<tr>
<td>timbl/data-2.nq.gz</td>
<td>87250</td>
<td>1.2M</td>
</tr>
<tr>
<td>timbl/data-3.nq.gz</td>
<td>388412</td>
<td>5.1M</td>
</tr>
<tr>
<td>timbl/data-4.nq.gz</td>
<td>9405528</td>
<td>113M</td>
</tr>
<tr>
<td>timbl/data-5.nq.gz</td>
<td>93898523</td>
<td>1016M</td>
</tr>
<tr>
<td>timbl/data-6.nq.gz</td>
<td>101010423</td>
<td>1.2G</td>
</tr>
</tbody>
</table>
Bigdata®

Services and dynamics
Related “Graph” Technologies

Redpoint repositions existing technology.

MPGraph compares favorably with high end hardware solutions from YARC, Oracle, and SAP, but is open source and uses commodity hardware.
bigdata® is a federation of services

Unified API

**RDF Data and SPARQL Query**
- SPARQL XML
- RDF/XML
- Turtle
- SPARQL JSON
- N-Triples
- TriG
- N-Quads
- RDF/JSON

**Client Service**

**Data Service**

**Distributed Index Management and Query**

**Management Functions**
- Registrar
- Zookeeper
- Shard Locator
- Transaction Mgr
- Load Balancer
Typical Software Stack

- Application Layer
  - Unified API
  - Unified API Implementation
  - API Frameworks (Spring, etc.)
  - Sesame Framework
    - SAIL
    - RDF
    - SPARQL
  - Bigdata RDF Database
  - Bigdata Component Services
  - OS (Linux)
  - Cluster and Storage Management

- Java
- HTTP
- Jini
- Zookeeper
Service Discovery (1/2)

- Services discover registrars. Registrar discovery is configurable using either unicast or multicast protocols.

- Services advertise themselves and lookup other services (a).

- Clients use the shard locator to locate key-range shards for scale-out indices (b).
Service Discovery (2/2)

- Clients resolve shard locators to data service identifiers (a), then lookup the data services in service registrar (b).

- Data moves directly between clients and services (c).

- Service protocols not limited to RMI. Custom NIO protocols for data high throughput.

- Client libraries encapsulate this for applications, including caching of service lookup and shard resolution.
Persistence Store Mechanisms

• **Read/Write (RW) store**
  – Efficiently recycles allocation slots on the backing file
  – Used by services that need persistence without sharding, HA, etc.
  – Also used in the scale-up single machine database (~50B triples)

• **Write Once, Read Many (WORM) store**
  – Append-only, log structured store (aka “journal”)
  – Used by the data services used to absorb writes
  – Plays an important role in the scale-out database architecture

• **Index segment (SEG) store**
  – Read-optimized B+Tree files
  – Generated by bulk index build operations on a data service
  – Plays an important role in dynamic sharding and analytic query
The Data Service (1/2)

Append only journals and read-optimized index segments are basic building blocks.
The Data Service (2/2)

Behind the scenes on a data service.

**Journal**
- Block append for ultra fast writes
- Target size on disk ~ 200MB.

**Overflow**
- Fast synchronous overflow
- Asynchronous index builds
- Key to Dynamic Sharding

**Index Segments**
- 98%+ of all data on disk
- Target shard size on disk ~200M
- Bloom filters (fast rejection)
- At most one IO per leaf access
- Multi-block IO for leaf scans

Data Service

index segments

journal

overflow
Bigdata® Indices

Scale-out B+Tree and Dynamic Sharding
Bigdata® Indices

• Dynamically key-range partitioned B+Trees for indices
  – Index entries (tuples) map unsigned byte[ ] keys to byte[ ] values.
  – “deleted” flag and timestamp used for MVCC and dynamic sharding.

• Index partitions distributed across data services on a cluster
  – Located by centralized metadata service

<table>
<thead>
<tr>
<th>nodes</th>
<th></th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>leaves</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>keys</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>a</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>values</td>
<td>v1</td>
<td>v2</td>
<td>v4</td>
<td>v5</td>
<td>v7</td>
</tr>
</tbody>
</table>

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>v2</td>
<td>v3</td>
<td>v5</td>
<td>v6</td>
<td>v8</td>
<td>v10</td>
</tr>
</tbody>
</table>

Tuple:
- key: unsigned byte [*]
- val: byte [*]
- revision: long
- deleted: boolean
**Dynamic Key Range Partitioning**

- **Splits** break down the shards as the data scale increases.
- **Joins** merge shards when data scale decreases.
- **Moves** redistribute shards onto existing or new nodes in the cluster.

Diagram:

- **p0** split to **p1** and **p2**
- **p1** and **p2** join to **p3**
- **p3** move to **p4**
Dynamic Key Range Partitioning

Initial conditions place the first shard on an arbitrary data service representing the entire index key range.

Shard Locator Service

 DataService1

(region, \infty)

p0
Dynamic Key Range Partitioning

Writes cause the shard to grow. Eventually its size on disk exceeds a preconfigured threshold.

Shard Locator Service

\([, \infty)\]

p0

DataService1
Dynamic Key Range Partitioning

Instead of a simple two-way split, the initial shard is “scatter-split” so that all data services can start managing data.
The newly created shards are then dispersed around the cluster.

Subsequent splits are two-way and moves occur based on relative server load (decided by Load Balancer Service).
Shard Evolution

Builds generate index segments from just the old journal.

Merge compacts the shard view into a single index segment.
Shard Evolution

- Initial journal on DS.
- Incremental build of new segments for a shard with each journal overflow.
- Shard periodically undergoes compacting merge.
- Shard will split at 200MB.
Bulk Data Load

High Throughput with Dynamic Sharding
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 with snapshot isolation.
Distributed Bulk Data Loader

Application client identifies RDF data files to be loaded.

One client is elected as the job master and coordinates the job across the other client.

Clients read directly from shared storage.

Writes are scattered across the data service nodes.
Input (2.5B triples)

16 node cluster loading at 180k triples per second
Throughput (2.5B triples)

16 node cluster, 2.5B triples.
Disk Utilization (~56 bytes/triple)
Shards over time (2.5B triples)
Dynamic Sharding in Action

Shard builds, merges, and splits for 2.5B triples

# of operations vs. minutes:
- Split
- Build
- Merge
Scatter Splits in Action
(Zoom on first 3 minutes)
Vectored Query in Scale-Out

Client issues SPARQL query

Data Service #1

Data Service #2

Data Service #3

(SPO)

(POS)

(?s rdf:type foaf:Person)

(?s foaf:name ?name)

1. The initial read on the POS index provides bindings for ?s.

2. The bindings for ?s are plugged into the next triple pattern and mapped across the index for that JOIN (SPO).

3. The intermediate solutions are evaluated against the SPO index partitions in parallel.
Bryan Thompson
SYSTAP, LLC
bryan@systap.com

http://www.systap.com/bigdata.htm
http://www.bigdata.com/blog
Backup Material
Statement Identifiers

RDF Graphs with efficient link attributes
Statement Level Metadata

• Important to know where data came from in a mashup

• :mike :memberOf :SYSTAP .


• But you CAN NOT say that in RDF.
RDF “Reification”

• Creates a “model” of the statement:
  
  _:_s1  rdfs:subject  :mike  .
  _:_s1  rdfs:_predicate  :memberOf  .
  _:_s1  rdfs:object  :SYSTAP  .
  _:_s1  rdfs:type  rdf:Statement  .

• Then you can say:
  
Statement Identifier Mode (SIDs)

- Special database mode
  - SIDs look just like blank nodes
  - Leverages the *named graph* of the statement

- SIDs let you do exactly what you want:
  ```
  :mike :memberOf :SYSTAP _:_sl .
  ```

- Use SIDs in SPARQL:
  ```
  select ?s ?o ?source
  where {
    GRAPH ?sid { ?s :memberOf ?o } .
  }
  ```
Reification Done Right

• Extends the concept to support quads.
  – Outcome from Dagstuhl 2012 Semantic Data Management workshop.
  – Collaborative effort with SYSTAP, Open Link, Humboldt University, Karlsruhe Institute of Technology.
  – W3C Member Submission in preparation.
  – Harmonized with RDF model theory & SPARQL algebra.
  – Efficient in index structures and queries.
    • Extensions for N3, TURTLE, and SPARQL are proposed.
    • Interchange and query for link attributes (graph databases).
Works with triples or quads

- *Inline* statements into statements.
  
  ```
  << :mike :memberOf :SYSTAP >>
  dc:source <http://www.systap.com>
  ```

- Same syntax works for query
  
  ```
  select ?s ?o ?source
  where {
  }
  ```

- Standardized approach for:
  - Link attributes (graph databases).
  - Confidence measures (entity / link extractors).
  - Datum level security models.