This presentation offers a brief look scale in ontological (semantic) systems, tradeoffs in expressivity and data scale, and both information and systems architectural concerns for very large data scales.
Presentation outline

• “Big Data” and the Semantic Web
• Bigdata® Architecture
• RDF Database
• Use Cases
We focus on the backend semantic web database architecture and offer support and other services around that. Because we deal with open source, working with us is a bit different. Our revenues are mostly from consulting and support services. We try to get engaged with customers early in their development cycle so we can help them understand how to achieve their goals and work with them to get the most out of the database platform.
High level take aways

• Fast, scalable, open source, standards compliant database
  - Single machine to 50B triples or quads
  - Scales horizontally on a cluster
  - Built in RDFS+ reasoning
  - Compatible with 3rd party inference and middle tier solutions

• Reasons to be interested
  - You need to embed a high performance semantic web database.
  - You need to integrate, query, and analyze large amounts information.
  - You need fast turn around on support, new features, etc.

Our focus is one the database, application and information architecture, and application delivery. We help people to manage the challenges and tradeoffs to achieve scalable semantic applications.
What is “big data?”

• Big data is a way of thinking about and processing massive data.
  – Petabyte scale
  – Distributed processing
  – Commodity hardware
  – Open source

And with the emergence of big data analytics, many core and massively parallel processing.
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

As a historical note, provenance was left out of the original semantic web architecture because it requires statements about statements which implies second order predicate calculus. At the Amsterdam 2000 W3C Conference, TBL stated that he was deliberately staying away from second order predicate calculus. Provenance mechanisms are slowly emerging for the semantic web due to their absolute necessity in many domains. The SGML / XML Topic Maps standardization was occurring more or less at the same time. It had a focus which provided for provenance and semantic alignment with very little support for reasoning. Hopefully these things will soon be reconciled as different “profiles” reflecting different application requirements.
Provenance is huge concern for many communities. However, support for provenance, other than digitally signed proofs, was explicitly off the table when the semantic web was introduced in 1999. TBL’s reason for not tackling provenance was that it requires second order predicate calculus, while the semantic web is based on first order predicate calculus. However, it is possible to tackle provenance without using a highly expressive logic. The ISO and XML Topic Maps communities have lead in this area and, recently, there has been increasing discussion about provenance within the W3C.
The data – it’s about the data

English: The […] diagram [above] visualizes the data sets in the LOD cloud as well as their interlinkage relationships. Each node in this cloud diagram represents a distinct data set published as Linked Data. The arcs indicate that RDF links exist between items in the two connected data sets. Heavier arcs roughly correspond to a greater number of links between two data sets, while bidirectional arcs indicate the outward links to the other exist in each data set.

Image by Anja Jentzsch. The image file is licensed under the Creative Commons Attribution-Share Alike 3.0 Unported license.
Hunter Lab
University of Colorado
Larray Hunter (PI)
POC Kevin Livingston
kevin.livingston@ucdenver.edu
10 years ago everyone knew that the database was a frozen market. Today, nothing could be further from the truth. This is a time of profound upheaval in the database world.

There are a number of trends which are driving this. One is the continued pressure on commodity hardware prices and performance. Another is the growth in open source software. Today, all “big data” systems leverage open source software and many federal contracts require it. At the same time the speed of processors has hit the wall so applications are forced to parallelize and use distributed architectures. SSD has also changed the playing field, virtually eliminating disk latency in critical systems.

The central focus for the bigdata® platform central focus is a parallel semantic web database architecture. There are other distributed platforms, including those based on custom hardware solutions, for main memory graph processing. These tend to lack core features of a database architecture such as durability and isolation.
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, and data at rest.
  - Opportunistic exploitation of data, including data which can not be integrated quickly enough today to derive its business value.
  - Maintain fine-grained provenance of federated data.

This is our take on where this is all heading. We tend to focus on high data scale and low expressivity with rapid data exploitation cycles.
bigdata®

- Petabyte scale
- Dynamic sharding
- Commodity hardware
- Open source, Java

- High performance
- High concurrency (MVCC)
- High availability
- Temporal database

Semantic web database
bigdata® 1.1

- Dual deployment models
  - Scales to 50B triples/quads on a single machine
  - Scales horizontally on a cluster
- Three database modes:
  - Triples, Provenance, or Quads.
  - Native RDFS+ inference
- SPARQL 1.1 support (except property paths and update)
  - New query model
  - Faster query plans.
  - Support for 3rd party SERVICEs.
  - Query hints based on virtual triples
- New analytics package
- Storage layer improvements:
  - New index for BLOBs.
  - Inline common vocabulary items in 2-3 bytes.

- Bigdata 1.1 largely focused on query performance.
- Put all of our effort into something that really scales up.
Query Performance

- Graph shows series of trials for the BSBM 100M data set and the 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. IO Wait 5% - 6%. 

---

<table>
<thead>
<tr>
<th>BSBM 100M</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trials</td>
</tr>
<tr>
<td>1 1 1 2 2</td>
</tr>
<tr>
<td>3 3 4 4 4</td>
</tr>
<tr>
<td>5 5 6 6 6</td>
</tr>
</tbody>
</table>

---

<table>
<thead>
<tr>
<th>QMpH (Qs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 3 4 5</td>
</tr>
<tr>
<td>6 7 8 9 10</td>
</tr>
</tbody>
</table>

---

Graph: BSBM 100M Trials vs QMpH

- 8 clients 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. IO Wait 5% - 6%.
bigdata® is a federation of services

Data services (DS) manage shards
Shard locator service (SLS) maps shards to their data services
Transaction service (TS) coordinates transactions
Load balancer service (LBS) centralizes load balancing decisions
Client services (CS) provide execution of distributed jobs
Jini lookup services (LUS) provide service discovery
Zookeeper quorum servers (ZK) provide for master election, configuration management, and global synchronous locks
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.

<table>
<thead>
<tr>
<th>MDS scale</th>
<th>Data scale</th>
<th>Triples</th>
</tr>
</thead>
<tbody>
<tr>
<td>71MB</td>
<td>terabyte</td>
<td>18,000,000,000</td>
</tr>
<tr>
<td>72GB</td>
<td>petabyte</td>
<td>18,000,000,000,000</td>
</tr>
<tr>
<td>74TB</td>
<td>exabyte</td>
<td>18,000,000,000,000,000</td>
</tr>
</tbody>
</table>
Cloud Architecture

- Hybrid shared nothing / shared disk architecture
  - Compute cluster
    - Spin compute nodes up or down as required
  - plus
    - Managed cloud storage layer
      - S3, openstack, parallel file system, etc

Large scale systems need to think about deployment architectures and managing the raw resources that they will consume. Bigdata® has evolved over time into a hybrid architecture in which there is a separation of concerns between the compute and persistence layers. The architecture still leverages local computation (including local disk access) whenever possible. We do this now through caching shards on the instances nodes while leaving read-only files on long term durable storage. This architecture makes it possible to deploy to platforms like Amazon EC2 (compute) + Amazon S3 (high 9s durable storage).
Timeliness vs. Completeness

- Rapidly exploit fusion of data sources.
  - Exploitation cycle can be just a few hours.
- High level reasoning over curated information
  - Careful, detailed, and length period of ontology development;
  - In depth reconciliation of data sources and their semantics.
  - Exploitation cycle can be six months to several years.

There are two very different ways in which semantic technology can unlock value from data. It is important to recognize which world fits your problem and how to best invest your resources.

**Rapid information exploitation**

In this world, the benefit is derived from the rapid pace at which new data and new data sources can be combined and exploited.

**High level reasoning over curated information**

In this world, the benefit is derived from non-trivial inferences drawn over highly vetted data.
Expressivity vs. Scale

- Expressivity, Scale, Speed, Completeness
  - Either partition the data or partition the reasoning.
  - Don’t be seduced by expressivity for its own sake.
- A little ontology goes a long way
  - Avoid constructs that tell you things you probably already know (e.g. domain/range)

Many times people try to have both expressivity and scale. This is very expensive

Don’t be seduced by expressivity

Just because you CAN say it doesn’t mean you SHOULD say it. Stick to things that are strictly useful to building your big data application.

Computationally expensive

Expressivity is not free. It must be paid for either with load throughput or query latency, or both.

Not easily partitioned

Higher expressivity often involves more than one piece of information from the abox – meaning you have to cross server boundaries. With lower expressivity you can replicate the ontology everywhere on the cluster and answer questions LOCALLY.

A little ontology goes a long way

There can be a lot of value just getting the data federated and semantically aligned.

Avoid constructs that tell you things you probably already know (e.g. domain/range)

Domain and range are interesting. People think they are supposed to be for describing what properties and reverse properties a particular type of resource should have, but in reality they are used to infer a resource’s type based on its properties and reverse properties. This type of inference is almost never useful in real systems (classification of resources are almost always known in advance).
Information Architecture

- Benefits of micro ontologies
  - Separate out system architecture, application architecture, and domain architecture.

- Provenance
  - Bigdata® has a dedicated mode for datum level provenance.
  - Fast, inline representation.
  - Smaller disk footprint.
  - SPARQL query.

- Modeling relationships
  - Provenance model allows dual modeling of relationships as entities.

Provenance is vital in many applications, but support for provenance is non-standard and varies across platforms.

Our approach is a custom provenance mode in which each statement can be used as-if it were a resource. You can create multiple levels of statements about statements, interchange the data using an RDF/XML extension, and query it using SPARQL. We also provide truth maintenance for the statements about statements.

Dedicated handling for provenance is a necessity, not just a “shortcut” for reification. The cost of reifying every assertion is enormous at scale. There is also a high cognitive cost for always reifying the data. Only reifying some data makes matters worse since (a) you could find out that you need to reify something else later; and (b) you need to keep track of what is reified and what is not reified.

In the “ontology light, data heavy” world we tend to recommend the use of micro ontologies. This helps to partition the different aspects of the application while making only the domain distinctions which are critical for a given application. The more ontological distinctions you create, the more distinctions you have to maintain and the harder it is to ensure that the semantics of the ontological distinctions are followed correctly throughout the data and the application.
Statement Level Provenance

- Important to know where data came from in a mashup

- `<mike, memberOf, SYSTAP>`
- `<http://www.systap.com, sourceOf, ....>`

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

• Creates a “model” of the statement.

  <_s1, subject, mike>
  <_s1, predicate, memberOf>
  <_s1, object, SYSTAP>
  <_s1, type, Statement>

• Then you can say,

  <http://www.systap.com, sourceOf, _s1>
Statement Identifiers (SIDs)

• Statement identifiers let you do exactly what you want:
  
  `<mike, memberOf, SYSTAP, _s1>`
  `<http://www.systap.com, sourceOf, _s1>`

• SIDs look just like blank nodes

• And you can use them in SPARQL
  
  `construct { ?s <memberOf> ?o . ?s1 ?p1 ?sid . }`
  `where {`
  `  GRAPH ?sid { ?s <memberOf> ?o }`
  `}`
RDF Database “Schema”

This is slightly dated. We now inline numeric data types into the statement indices for faster aggregation and use a BLOBS index for large literals.

The table and provenance indices use three statement indices.

The quad store uses six indices whose keys are based on permutations of (s,p,o,v).
Bigdata Query Engine

- Vectored query engine
  - High concurrency and vectored operator evaluation.

- Physical query plan (BOPs)
  - Supports pipelined, blocked, and at-once operators.
  - Chunks queue up for each operator.
  - Chunks transparently *mapped* across a cluster.

- Query engine instance runs on:
  - Each data service; *plus*
  - Each node providing a SPARQL endpoint.
As part of our 1.1 release, we had to take over more control of query evaluation from the Sesame platform. We found that the Sesame operator tree was discarding too much information about the structure of the original SPARQL query. Also, bigdata query evaluation has always been vectored and operators on IVs rather than RDF Values. The Sesame visitor pattern was basically incompatible with bigdata query evaluation. Bigdata is now integrated with the Sesame platform directly at the SPARQL parser. All query evaluation from that point forward is handled by bigdata.

Most of the work is now done using the Abstract Syntax Tree (AST). The AST is much closer to the SPARQL grammar. AST optimizers annotate and rewrite the AST. AST then translated to Bigdata Operators (BOPs). BOPs run on the query engine. The AST ServiceNode provides for 3rd party operators and services.

The bigdata query engine is vectored and highly concurrent. Most bigdata operators support concurrent evaluation and multiple instances of the same operator can be running in parallel. This allows us to fully load the disk queue. With the new memory manager and analytic query operators, having large solutions sets materialized in memory no longer puts a burden on the JVM.
Pipeline Join Execution
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.

Go and get sesame, it will fall over because of this.

Several commercial and open source Java grid cache products exist, including Oracle’s Coherence, inﬁnispan, and Hazelcast. However, all of these products share a common problem – they are unable to exploit large heaps due to an interaction between the Java Garbage Collector (GC) and the object creation and object retention rate of the cache. There is a non-linear interaction between the object creation rate, the object retention rate, and the GC running time and cycle time. For many applications, garbage collection is very efﬁcient and Java can run as fast as C++. However, as the Java heap begins to ﬁll, the garbage collector must run more and more frequently, and the application is locked out while it runs. This leads to long application level latencies that bring the cache to its knees and throttles throughput. Sun and Oracle have developed a variety of garbage collector strategies, but none of them are able to manage large heaps efﬁciently. A new technology is necessary in order to successfully deploy object caches that can take advantage of servers with large main memories.
Analytic Package

- **Bigdata® Memory Manager**
  - 100% Native Java
  - Application of the bigdata® RWStore technology
  - Manages up to 4TB of the native process heap
  - Relieves heap pressure, freeing JVM performance.

- **Extensible Hash Index (HTree)**
  - Fast, scalable hash index
  - Handles key skew and bucket overflow gracefully
  - Use with disk or the memory manager (zero GC cost up to 4TB!)
  - Used for analytic join operators, named solution sets, caches, etc.
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.
Bigdata® Use Cases

Some Examples
KaBOB (Knowledge Base of Biology)

Hunter Lab
University of Colorado
• Kevin Livingston
  – kevin.livingston@ucdenver.edu
• Mike Bada
• Bill Baumgartner
• Yuriy Malenkiy
• Larry Hunter (PI)

Open Biomedical Ontologies

biomedical data & information
biomedical knowledge
application data

KaBOB (Knowledge Base of Biology)
Motivation for KaBOB

• Biomedical researchers face a profound challenge in keeping track and making sense of
  – numerous curated databases
  – rapidly growing literature
  – data from high-throughput experiments

• Semantic integration is necessary to effectively unify and reason over all this information
Hunter Lab
University of Colorado
Larray Hunter (PI)
POC Kevin Livingston
kevin.livingston@ucdenver.edu
KaBOB Current State

- Unified representations under Open Biomedical Ontologies (OBO)
- 8.7 Billion triples and counting
  - Database records
  - Unified biomedical entities (gene, proteins, etc.)
- Used to drive applications in
  - NLP / Text Mining
  - Visualization
  - Reasoning
- Working toward more and more unified biomedical representation beyond entities
  (protein-protein interactions, pathways, drug targets, ...)

http://www.bigdata.com/blog

Presented to CSHALS 2012
• High-performance knowledge discovery & collaboration web application
• Used by Pharmaceuticals, Biotechnology Companies, Healthcare Organizations
• A preferred standards compliant triple store is bigdata®
Manufacturing Product Data is Heterogeneous

…and difficult to find, re-use, and share
Inforbix Product Data Applications

- Using RDF and bigdata® to tackle the manufacturing data problem
- Federate and index disparate product data sources
- Fast Google-style search experience promoting re-use of high-value product data
bigdata

Flexible
Reliable
Affordable
Web-scale computing.