Building Scalable Technologies for Semantic Analysis

JOHN FEO
HIGH PERFORMANCE DATA ANALYTICS PROJECT
PACIFIC NORTHWEST NATIONAL LABORATORY
The problem

- Data is no longer “owner produced,” but rather gathered from external sources on the web. *It is unstructured and heterogeneous.*

- The fixed schemas and table formats of relational databases are too rigid for web-gathered data.

- NoSQL databases have emerged, but their chosen approach of distributing data over many systems makes finding complex connections prohibitive.
• Flexible data model that supports structured and unstructured data in a single form
• In-memory datastore using local, remote, and flash memories
• General parallel programming model – *not record or vertex centric*
• Runs on commodity platforms from desktops to clouds – *no special system requirements*
Why do we perform better than others

- We store unstructured data as a graph
- We process graph data using graph methods
- We support a general parallel programming model allowing methods to be written naturally
- We have developed a multithreaded runtime system that scales out on commodity hardware
- We use standard languages (SPARQL, C++)
- We require no special systems (x86, Linux, MPI)

ADVANTAGES

- Larger data size
- Greater productivity
- Faster time to solution
- Lower cost of ownership
Mary called her sister Sally to discuss buying her 6-year daughter a pony for Christmas.

1) Mary called Sally
2) Mary has a sister named Sally
3) Sally has a sister named Mary
4) Either Mary or Sally has a daughter
5) The daughter is 6 years old
6) Mary wants to buy a pony

Sally rented Joe’s condo in Hawaii for a two week vacation. She paid $1200 rent.

1) Sally traveled to Hawaii
2) Sally vacationed in Hawaii
3) Joe owns a condo
4) Joe’s condo is in Hawaii
5) Sally rented Joe’s condo
6) Joe rented his condo for $600 per week
Use *graph algorithms* rather than table joins

**RETURN ALL PERSONS WHO HAVE SHARED 2 OR MORE ADDRESSES WITH JOHN**

```
NAME    ADDR
John    lived_at 100 Main
John    lived_at 243 Spruce
John    lived_at 212 Pine
Fred    lived_at 212 Pine
Mary    lived_at 212 Pine
```

```
NAME    ADDR
John    lived_at 100 Main
Fred    lived_at 212 Pine
Mary    lived_at 212 Pine
```

“*Everything you do at scale must be parallelized or it will run forever*”
- Michael Stonebraker
Use *memory* rather than disks

- Graph algorithms cannot take advantage of conventional storage hierarchies or locality-preserving, distributed data structures
  - *So keep everything in memory for fast random access*

- … but memory is very limited
  - *So use a cluster to expand available memory by adding nodes*

- … but distributed data incurs long latencies
  - *So use multithreading to tolerate latencies*
Use multithreading to hide latencies

- Generate hundreds of threads per core
- Rather than execute one thread at-a-time per core (conventional runtime), switch among active threads (multithreading runtime) such that …

**Gaps introduced by long latency operations in one thread are filled by instructions in other threads**
GEMS can scale up and scale out

- 512 GB, 32 cores, $8,000
- 1 TB, 80 cores, $85,000
- 1 TB, $10,000
- 2 TB, 80 cores, $85,000
- 100+ TB, 1024+ cores, $500,000
GEMS software stack

Manages communication, distributed data, parallel tasks
Makes parallel systems easy to use efficiently

Algorithms and data structures that are locality-(in)sensitive

Query interface with automatic optimization

<table>
<thead>
<tr>
<th>SPARQL</th>
<th>Hand-coded C++</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPARQL to C++ Compiler</td>
<td></td>
</tr>
<tr>
<td>Semantic Graph Library (SGLIB)</td>
<td></td>
</tr>
<tr>
<td>Multi-threaded Runtime System (GMT)</td>
<td></td>
</tr>
<tr>
<td>Commodity cluster</td>
<td>Special-purpose hardware</td>
</tr>
</tbody>
</table>
1B triples, 4TB memory systems

Same main memory size, but GEMS system had half the processors
Ran a data size that fit main memory to minimize Neo4j disk transfers
Rebooted Neo4j to use best mode for each query
Hired experienced Neo4j user to conduct test
GEMS vs. GraphLab and GraphX

Choose an algorithm studied heavily for both graph libraries
Worked closely with library development groups to insure best performance
GEMS is 4x faster than GraphLab and 16x faster than GraphX
Setup times – 1B triples

<table>
<thead>
<tr>
<th>FROM TRIPLES FILE</th>
<th>BUILD DICTIONARY, BUILD GRAPH, SAVE GZIP FILE</th>
</tr>
</thead>
<tbody>
<tr>
<td>16 P</td>
<td>1007 sec</td>
</tr>
<tr>
<td>32 P</td>
<td>555 sec</td>
</tr>
<tr>
<td>64 P</td>
<td>384 sec</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FROM GZIP FILE</th>
<th>RESTORE TRIPLES, RESTORE DICTIONARY, BUILD GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td>16 P</td>
<td>906 sec</td>
</tr>
<tr>
<td>32 P</td>
<td>432 sec</td>
</tr>
<tr>
<td>64 P</td>
<td>238 sec</td>
</tr>
</tbody>
</table>
SELECT ?resource ?location WHERE {
}

A path (+, *) is just a recursive call

```java
int DR_Node = dictionary.lookup(":DataResources");
forEach(ANY, ":subClassOf", DR_Node, Loop1);
forEach(ANY, ":type", DR_Node, Loop2);
......
......
 ......
// ?dataResource :subClassOf :DataResource
static void Loop1(subject, predicate, object) {
    forEach(ANY, ":subClassOf", subject, Loop1);
    forEach(ANY, ":type", subject, Loop2);
}
// ?resource :type :DataResource
static void Loop2(subject, predicate, object) {
    args_t args;
    args.resource = subject;
    forEach(subject, ":location", ANY, Loop3, args);
}
```
Attributed edges

In many problem domains, relationships have many attributes

Node ID: complex structure, two octets, A.B:P
Node label: internal/external
Edge ID: unique number
Edge label: application protocol
Edge attributes: # packets, # bytes, time interval, …

Creating “star patterns” wastes space and complicates query processing
Recognize the distinction between **relationships** and **attributes**
- Store relationships as a graph
- Store attributes in a table

Special predicates (UIDs) indicate record #
Can enrich with traditional RDF edges
Conclusions

- We are developing a scalable, in-memory triplestore capable of knowledge discovery on web-scale data warehouse
  - Scales with data size
  - Multiple programming entry points
  - Conventional cluster and cloud systems

- We are working with government agencies and early adopters on real world problems

- We seek partners in transitioning our platform from prototype to production