Clause-Iteration with Map-Reduce to Scalably Query Data Graphs: The SHARD Triple-Store

Kurt Rohloff    Rick Schantz
krohloff@bbn.com schantz@bbn.com
@avometric

Many thanks to:
Prakash Manghwani, Mike Dean,
Ian Emmons, Gail Mitchell, Doug Reid,
Chris Kappler from BBN
Hanspeter Pfister from Harvard SEAS
Phil Zeyliger from Cloudera
Outline

- Challenge Problem: Scalably Query Graph Data
- Large-Scale Computing and MapReduce
- SHARD
- Design Insights
A Preface

SHARD is a cloud based graph store.
- High-performance scalable query processing.

SHARD released open-source.
- BSD license.

More information and code at:
  - My webpage
  - Sourceforge (SHARD-3store)

- Use svn to get code:
  
  `svn co https://shard-3store.svn.sourceforge.net/svnroot/shard-3store shard-3store`

  - Don’t worry - this command is on SourceForge!
Scalable Graph Data Querying

- Emerging commercially
  - Use by NYTimes, BBC, Pharma, ...
  - Numerous startups.
  - Oracle, MySQL have SemWeb support.

- Government use...

- See the SemWeb.
SPARQL-like Queries

SPARQL Query to find all people who own a car made in Detroit:

SELECT ?person
WHERE {
  ?car a :Car .
  ?car :madeIn :Detroit .
}
Answering Queries

Variables bindings:
?person to Kurt
?car to car0

Kurt owns car0

car0 madeBy Ford

Kurt livesIn Cambridge

Cambridge madeIn Detroit

Car madeIn Detroit

City a Detroit

?person owns ?car

?car madeIn Detroit

krohloff@bbn.com
Design Considerations

- Scalable – web-scale?
- High Assurance.
- Cost Effective – commodity hardware?
- Modular inferred data separation.
- Robustness.

- Considerations as endless as applications.
Scale Limitations!

• Triple-Store Study:
  – “An Evaluation of Triple-Store Technologies for Large Data Stores”, SSWS '07 (Part of OTM).

• What about cloud computing?
  – Economic scalability…
General Programming for Scalable Cloud Computing

From Experience:

• Inherently multi-threaded.

• Toolsets still young.
  – Not many debugging tools.

• Mental models are different...
  – Learn an algorithm, adapt it to chosen framework.
  – Ex: try to fit problem into PageRank design pattern.
    • (This isn’t what we do, but this approach seems common.)
Scalable Distributed System (Cloud) Design Concept

Abstraction of parallelization enables much easier scaling.

• We use maturing MapReduce framework in Hadoop to bulk process graph edges.
• This provides services layer to scale our graph query processing techniques.

• Innovation:
  – Iterative clause-based construction of queries.
  – Join partial query responses over multiple Map-Reduce jobs using flagged keys.

krohloff@bbn.com
SHARD Triple-Store Built on Hadoop

Prioritized goals:

• Commodity hardware, ONLY
• Web scalable
• Robust

What is good:

Design Considerations:

• Large query responses
• Complex queries
In first Map Step, first query clause is used to find partial query matches that satisfy first clause
• Keys are variable bindings
• Values are set to null

Source data:
John owns dog0
Kurt livesIn Cambridge
Kurt owns car0
dog0 a Dog
car0 a Car
...


1st Map Key-Val
Output:
{John dog0} - null
{Kurt car0} - null
...

In first Reduce Step, repeated partial matches are removed
Map partial query matches from 2\textsuperscript{nd} query clause.

- Keys are variable bindings previously observed.
- Values are set to new variable bindings.

Map matches from previous clause for reordering.

- Keys are variable bindings common with current clause
- Values are previous non-common bindings

Source data:

John owns dog0
Kurt livesIn Cambridge
Kurt owns car0
dog0 a Dog
car0 a Car
...

1\textsuperscript{st} Map Key-Val
Output:

\{John dog0\} - null
\{Kurt car0\} - null
...

2\textsuperscript{nd} Map Key-Val
Output:

\{car0\} – null
...
\{dog0\} – \{John\}
\{car0\} – \{Kurt\}
...

?car a Car .
Reduce joins partial mappings on common variable bindings with flagged keys.

2nd Map Key-Val Output:
{car0} – null
…
{dog0} – {John}
{car0} – {Kurt}
…

2nd Reduce Key-Val Output:
{car0} – {Kurt}
…

Process continues over all query clauses.
Graphs saved as flat-file in HDFS:
(Portions of file saved on each data node.)

Kurt owns car0 livesIn Cambridge
Car0 a Car madeBy Ford madeIn Detroit
Cambridge a City
Detroit a City
HDFS data partitioning

- Hash Partitioning by Default.
- Neighborhood partitioning would probably provide better performance.
  - R&D opportunity!
Query Processing Implementation

- BBN-developed query processor.
  - Starting integration with “standard” interfaces
    - Jena, Sesame.
- SHARD supports “most” of SPARQL.
  - Like most commercial triple-stores.
- Large performance improvements possible with improved query reordering.
Data Persistence Advice from SHARD

• Down to “bare metal” in HDFS for large-scale efficiency.
  – No Berkeley DB, no C-stores, …. Nothing.

• Simple data storage as flat files.
  – Lists of (predicate, object) pairs for every subject by line.
  – Ex: Kurt owns car0 livesin Cambridge

• Simple often really is better…
Test Data

• Deployed code on Amazon EC2 cloud.
  – 19 XL nodes.

• LUBM (Lehigh Univ. BenchMark)
  – Artificial data on students, professors, courses, etc… at universities.

• 800 million edge graph.
  – 6000 LUBM university dataset.

• In general, performed comparably to “industrial” monolithic triple-stores.

krohloff@bbn.com
<table>
<thead>
<tr>
<th>Query Type</th>
<th>SHARD</th>
<th>Parliament+Sesame</th>
<th>Parliament+Jena</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simple Query, Small Response: Triple Lookup (Query 1)</td>
<td>404 sec. (approx 0.1 hr.)</td>
<td>0.1hr</td>
<td>0.001hr</td>
</tr>
<tr>
<td>Triangular Query (Query 9)</td>
<td>740 sec. (approx 0.2 hr.)</td>
<td>1hr</td>
<td>1hr</td>
</tr>
<tr>
<td>Simple Query, Large Response: (Query 14)</td>
<td>118 sec. (approx 0.03 hr.)</td>
<td>1hr</td>
<td>5hr</td>
</tr>
</tbody>
</table>

krohloff@bbn.com
Insight from Query Performance

• SHARD is not optimal for edge look-ups.
  – This could be expected – SHARD (and MapReduce implementations) have no real indexing support.
• SHARD does well where large portions of dataset need to be processed.
  – Ex:
    • Multiple join operations
    • Return large datasets
  – This behavior is an artifact of parallel searching and joining operation native to Clause-Iteration.
Design Insights

• Abstraction is a big win.
  – Surprisingly economical for development.

• Lack of indexing limits look-up capabilities.
  – This may not be so bad for some applications
  – Index will also need to be continually updated as data added.
Design Insights – Data Partitioning

• Data linking may be a big win to reduce join overhead and reduce need for iterations over clauses.
  – A first step would be advanced data partitioning.
  – Done some in Cloud9, but still wide open for even basic R&D implementations.

• Advanced data partitioning would also minimize overhead of moving intermediate results between compute nodes.
  – This seemed to be biggest bottleneck.
Design Insights – Query Processing

• Query pre-processing may also be a big win.
  – Could also greatly reduce amount of data carried between nodes during join operations.

• Subject-Iteration may be an alternative approach for queries with strongly connected source nodes.
  – Iterate over query subject rather than clauses.
Thanks!
Questions?

Kurt Rohloff
krohloff@bbn.com
@avometric