The value and challenges of Large Scale Entity Analysis for National Security

Mrs. Mary Galvin, LexisNexis Special Services
Dr. Flavio Villanustre, LexisNexis Risk Solutions
VP of Technology & Open Source Software Lead for the HPCC Project

March 6th, 2013
Risk Solutions
Big Data in the National Security Mission

1. Rapid Prototyping Projects
   • The Open Source Nature of the HPCC Combined with its Immediate Availability on AWS Enable **Big Data Analysis to Keep Pace with Mission**

2. Fraud Detection Pertaining to:
   • Counterfeit Parts/Supply Chain
   • Visas & Passports

3. Identity Resolution & Graph Analysis
History of the open source HPCC Systems platform

- Designed and Developed from the Ground-Up to Meet LexisNexis’ Internal Big Data Needs.
- Google’s MapReduce Paper is Published.
- The Idea of Releasing the HPCC to the OSS Community was Presented to LexisNexis Corporate Management.
- The Spread of HPCC Users has Gone Global, and as a Result, Innovation Ignites.

- Late 90s/Early 2000s
- 2001
- 2004
- 2007
- 2009
- 2011
- 2013

- United States Government Sought After Getting LexisNexis’ Data Capabilities In-House for their Internal Data Mining Needs.
- First Release of Hadoop Available (designed after Map Reduce Papers).
- The HPCC is Officially Released to the Open Source Community!
The open source LexisNexis HPCC Systems platform
• The HPCC Systems platform includes:
  • Thor: batch oriented data manipulation, linking and analytics engine
  • Roxie: real-time data delivery and analytics engine
• A high level declarative data oriented language: ECL
  • Implicitly parallel
  • No side effects
  • Code/data encapsulation
  • Extensible
  • Highly optimized
  • Builds graphical execution plans
  • Compiles into C++ and native machine code
  • Common to Thor and Roxie
• An extensive library of ECL modules, including data profiling, linking and Machine Learning
A comparison of the HPCC and Hadoop stacks
Beyond ECL: SALT

- The acronym stands for “Scalable Automated Linking Technology”
- Template based ECL code generator
- Provides for automated data profiling, QA/QC, parsing, cleansing, normalization and standardization
- Sophisticated specificity and relatives based linking and clustering

![Diagram of data preparation and linkage processes]

- 42 Lines of SALT
- 3,980 Lines of ECL
- 482,410 Lines of C++
Beyond SALT: Smart View

- 0 Lines of Code!
- 42 Lines of SALT
- 3,980 Lines of ECL
- 482,410 Lines of C++
1. Rapid Prototyping Projects
   • The HPCC’s ETL Capabilities on the Amazon Cloud Helped IC Client Meet 60 Day Req’t for Turnaround

2. Fraud Detection Pertaining to:
   • Medicaid
   • Property Flipping & Equity Stripping
   • Medical Prescriptions

3. Identity Resolution & Graph Analysis:
   • Wikipedia Topics
   • Legal Entities
The Future: Knowledge Engineering

Knowledge Engineering through Graph Processing and Machine Learning

At the end of the day, as a Data Analyst

- Do I really care about the format of the data?
- Do I even care about the placement of the data?
- I do care (a lot!) about what can be inferred from the data
- The context is important as it affects my inference process
- I want to be able to leverage existing algorithms

<table>
<thead>
<tr>
<th>ECL</th>
<th>KEL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generates C++ (1-&gt;100)</td>
<td>Generates ECL (1-&gt;12)</td>
</tr>
<tr>
<td>Files and Records</td>
<td>Entities and associations</td>
</tr>
<tr>
<td>Detailed control of data format</td>
<td>Loose control of input format; none of processing</td>
</tr>
<tr>
<td>Can write graph and statistical algorithms</td>
<td>Major algorithms built in</td>
</tr>
<tr>
<td>Thor/Roxie split by human design</td>
<td>Thor/Roxie split by system design</td>
</tr>
<tr>
<td>Solid, reliable and mature</td>
<td>R&amp;D</td>
</tr>
</tbody>
</table>
KEL by example (WIP!)

• Define Data Sources
  • USE IMDB.File_Actors(FLAT,Actor,Movie,Appearance)

• Define Entities
  • Actor := ENTITY( FLAT(UID(ActorName),Actor=ActorName) )
  • Movie := ENTITY( FLAT(UID(MovieName),Title=MovieName) )
  • Appearance := ASSOCIATION( FLAT(Actor Who,Movie What) )

• Define Associations
  • CoStar := ASSOCIATION( FLAT(Actor Who,Actor WhoElse) )
  • GLOBAL: Appearance(#1,#2) Appearance(#3,#2) => CoStar(#1,#3)

• Define Queries
  • QUERY:FindActors(_Actor) <= Actor(_Actor)
  • QUERY:FindMovies(_Actor) <= Movie(UID IN Appearance(Wo IN
    Actor(_Actor){UID}){What})
  • QUERY:FindCostars(_Actor) <= Actor(UID IN CoStar(Who IN
    Actor(_Actor){UID}){WhoElse})
  • QUERY:FindAll(_Actor) <= Actor(_Actor),Movie(UID IN Appearance(Who IN _1{UID}){What}),Actor(UID IN CoStar(Who IN _1{UID}){WhoElse})
Useful links

- LexisNexis open source HPCC Systems Platform: http://hpccsystems.com
- Machine Learning portal: http://hpccsystems.com/ml
- The HPCC Systems blog: http://hpccsystems.com/blog
- The HPCC source code GitHub portal: https://github.com/hpcc-systems
- Community forums: http://hpccsystems.com/bb