ANALYSIS OF BIG DATA TECHNOLOGIES AND METHODS:

QUERY LARGE WEB PUBLIC RDF DATASETS ON AMAZON CLOUD USING HADOOP AND OPEN SOURCE PARSERS

A thesis submitted in partial fulfillment of the requirements

For the degree of Master of Science in Computer Science

By

Ted Garcia

May 2013
The thesis of Ted Garcia is approved:

___________________________________  ____________________
Dr. Richard Lorentz  ____________________
Date

___________________________________  ____________________
Dr. Diane Schwartz  ____________________
Date

___________________________________  ____________________
Dr. George Wang, Chair  ____________________
Date

California State University, Northridge
## Table of Contents

Signature Page ii  
Dedication v  
List of Figures vi  
List of Tables vii  
Abstract viii

1. Introduction 1  
   1.1. Survey of Big Data 1  
   1.2. Semantic Web 5  
2. Big Data Technologies 7  
   2.1. Amazon Cloud Technologies 7  
   2.2. Hadoop’s Map/Reduce 8  
3. Semantic Web Technologies 14  
   3.1. Ontologies 15  
   3.2. Rich Data Format (RDF) 18  
   3.3. Parsers 20  
4. Related Works 24  
5. Analysis and Design of QUERY Program 26  
   5.1. Process Steps 28  
   5.2. Common Crawl Corpus 28
Dedication

This work is dedicated to the caring computer science professors that entice our minds and help us explore extremely complex concepts in order to evolve this useful discipline.

I am also indebted to the pioneers of the semantic web, without whom we may never have experienced intelligent search. When everyone said it was impossible, they continued to climb the mountain.

And thanks to another set of courageous fellows that confronted the cataclysmic growth and rising complexity of data. They threw away most existing technologies to wrestle the dragon and cobbled together some techniques from the past with incredible ingenuity. Thanks for Hadoop.

Finally, and most importantly, my family has supported me through this challenging endeavor with compassion and patience in which I am very grateful.
List of Figures

Figure 2.1 Map/Reduce Directed Acyclic Graph ................................................................. 8
Figure 2.2 Map/Reduce Directed Acyclic Graph ................................................................. 9
Figure 2.3 Map/Reduce Chaining ...................................................................................... 10
Figure 2.4 HDFS Architecture (Apache Hadoop) .............................................................. 11
Figure 2.5 Hadoop Job Management .................................................................................. 13
Figure 3.1 The Semantic Web Stack .................................................................................. 15
Figure 3.2 Main Entities and Relationships of the Innovation ........................................... 17
Figure 3.3 Main Example Ontology (Innovation Ontology) ................................................ 17
Figure 3.4 RDF Graph Representing Eric Miller ............................................................... 19
Figure 5.1 Query Program Architecture ........................................................................... 27
Figure 6.1 Parser Performance ........................................................................................... 34
Figure 6.2 Parser Performance - Any23 ............................................................................ 35
Figure 6.3 Parser Performance - NXParser ........................................................................ 36
Figure 6.4 Parser Performance - Jena ................................................................................ 37
Figure 6.5 Parser CPU Cost by File Size ........................................................................... 38
Figure 6.6 Parser CPU Cost by CPU Count ....................................................................... 39
Figure G.1 Amazon Computing Cost ................................................................................ 68
List of Tables

Table 5.1 Component Descriptions ................................................................. 26
Table 5.2 Basic Process Steps ........................................................................ 28
Table B.1 Test Data - Run Duration ................................................................. 49
Table B.2 Test Data - CPU Usage ................................................................. 50
Table B.3 Test Data - Memory Usage ............................................................. 51
Table H.1 Amazon Pricing ........................................................................... 72
Abstract

ANALYSIS OF BIG DATA TECHNOLOGIES AND METHODS:

QUERY LARGE WEB RDF DATASETS ON AMAZON CLOUD USING HADOOP AND OPEN SOURCE PARSERS

By

Ted Garcia

Master of Science in Computer Science

Querying large datasets has become easier with Big Data technologies such as Hadoop's MapReduce. Large public datasets are becoming more available and can be found on the Amazon Web Service (AWS) Cloud. In particular, Web Data Commons (Web Data Commons, 2012) has extracted and posted RDF Quads from the Common Crawl Corpus (Common Crawl, 2012) found on AWS which comprises over five billion web pages of the Internet. Technologies and methods are in their infancy when attempting to process and query these large web RDF datasets. For example, within the last couple of years, AWS and Elastic MapReduce (EMR) have provided processing of large files with parallelization and a distributed file system. RDF technologies and methods have existed for some time and the tools are available commercially and open source. RDF Parsers and databases are being used successfully with moderately sized...
datasets. However, the use and analysis of RDF tools against large datasets, especially in a distributed environment, is relatively new.

In order to assist the BigData developer, this work explores several open source parsing tools and how they perform in Hadoop on the Amazon Cloud. Apache Any23 (Apache Any23, 2012), Apache Jena RIOT/ARQ (Apache Jena, 2013), and SemanticWeb.com's NxParser (NX Parser, 2012) are open source parsers that can process the RDF quads contained in the Web Data Commons files. In order to achieve the highest performance, it is essential to work with large datasets without preprocessing or importing them into a database. Therefore, parsing and querying will be done on the raw Web Data Commons files. Since the parsers do not all have query support, they will be analyzed with extract and parse functionality only. This work includes challenges and lessons learned from using these parsing tools in the Hadoop and Amazon Cloud environments and suggests future research areas.
1. INTRODUCTION

This work attempts to illuminate the currently evolving state of two fairly new topics, i.e. Big Data and the Semantic Web. Even though they both stand on their own as significant movements, this work will show how Big Data techniques and technologies can come together to solve some of the Semantic Web challenges.

1.1. Survey of Big Data

Big Data has three major components, i.e. data transformation and analysis, storage, and visualization. We will be taking care of data analysis and not doing any transformation, storage nor visualization.

Transformation can be done by the parsing tools in this paper but this paper's application does not require it. There are many transformation tools that can be used to move data from one format to another. For example XML is often the source of or a destination of transformed data. Transformation tools are similar to those found in ETL (extract-transform-load) tools found in business intelligence solutions.

Storage is also an area of evolution since current solutions are not adequate to house and manage huge amounts of data. There are many databases and special file-systems being used for big data. It this paper, storage is being done by the Amazon cloud S3 service and it is only a temporary file-system - there is no database.

Visualization is its own subject domain and deals with how to show the results of Big Data analytics in reports, graphs, and diagrams. IEEE has big data visualization as number three on its top ten trends of 2013. We've entered a data-driven era, in which data are continuously acquired for a variety of purposes. The ability to make timely
decisions based on available data is crucial to business success, clinical treatments, cyber and national security, and disaster management. Additionally, the data generated from large-scale simulations, astronomical observatories, high-throughput experiments, or high-resolution sensors will help lead to new discoveries if scientists have adequate tools to extract knowledge from them.

However, most data have become simply too large and often have too short a lifespan. Almost all fields of study and practice sooner or later will confront this big data problem. Government agencies and large corporations are launching research programs to address the challenges presented by big data. Visualization has been shown to be an effective tool not only for presenting essential information in vast amounts of data but also for driving complex analyses. Big data analytics and discovery present new research opportunities to the computer graphics and visualization community. This 2013 theme issue of *IEEE Computer Graphics and Applications* aims to highlight the latest advancements in solving the big data problems via visual means. *Computer* magazine will also be publishing a special issue on big data in June 2013. (Top Trends for 2013, 2013)

Due to the explosion of data sizes, complexity, and variety in the last few years, better approaches are being sought to retrieve and analyze this data.

Big Data is a continuation of computer science's attempt to extract answers from data. But in this case, the data is every growing in size and complexity. Big Data is the next evolutionary step after Business Intelligence. Business Intelligence provides a
sophisticated capability to analyze transactional data. But Business Intelligence falls short in tackling Big Data challenges.

Definition: Big data usually includes data sets with sizes beyond the ability of commonly-used software tools to capture, manage, and process the data within a tolerable elapsed time. Big data sizes are a constantly moving target, as of 2012 ranging from a few dozen terabytes to many petabytes of data in a single data set. Big data requires exceptional technologies to efficiently process large quantities of data within tolerable elapsed times (WikiPedia: Big Data, 2013).

Besides the increasing volume of data, Big Data also attempts to attack problems relating to velocity (speed of data in and out), variety (range of data types and sources), and complexity of virtual data. Querying and processing data sizes of hundreds of millions of records is what Big Data is aimed at. Variety of data, such as structured, semi-structured, and unstructured is the focus of Big Data. For example, correlating sales data for a transactional system with web content such as Twitter conversations on the same product sales data. Big Data also needs to process complex calculations to each row or row groupings for data mining and forecasting applications that currently cannot be done reasonably with conventional technologies.

Usage

The trend to larger data sets is due to the additional information derivable from analysis of a single large set of related data, as compared to separate smaller sets with the same total amount of data, allowing correlations to be found to "spot business trends, determine quality of research, prevent diseases, link legal citations, combat crime, and
determine real-time roadway traffic conditions.” Scientists regularly encounter limitations due to large data sets in many areas, including meteorology, genomics, connectomics, complex physics simulations, and biological and environmental research. The limitations also affect Internet search, finance and business informatics. Data sets grow in size in part because they are increasingly being gathered by ubiquitous information-sensing mobile devices, aerial sensory technologies (remote sensing), software logs, cameras, microphones, radio-frequency identification readers, and wireless sensor networks.

Examples include web logs, RFID, sensor networks, social networks, social data (due to the social data revolution), Internet text and documents, Internet search indexing, call detail records, astronomy, atmospheric science, genomics, biogeochemical, biological, and other complex and often interdisciplinary scientific research, military surveillance, medical records, photography archives, video archives, and large-scale e-commerce.¹

**Volume**

Due to data explosion, processing 100's of millions of records is becoming commonplace. Most, if not all, Fortune 500 companies have transactional data in this magnitude. Walmart is certainly a poster child for Big Data regarding volume. The largest retailer in the world can benefit from Big Data for marketing, logistics, and many other purposes.

**Variety**

In recent years, focus is on handling data types separately. Even unstructured data has been dealt with individually. Correlating different data types, even though desirable,
has been challenging at best. For example, the need to correlate transactional sales data and web content has been desirable but too much of a challenge for current technologies.

Complexity

Applying complex algorithms to even millions of records can take a very long time with current technologies. Grouping and using simple statistics has been done for some years with fast results. However, more difficult statistical methods such as standard deviation or other complex algorithms like recursion and pattern matching, cannot execute with acceptable performance.

1.2. Semantic Web

The semantic web was envisioned many decades ago but has only been realized more recently as part of the Internet. The semantic web allows the contextualization of content created in web pages to be searched more effectively than just keywords. In many ways, this is considered the new way to search the Internet and is requiring web content developers to add semantic web content to every page. Because of this, companies are crawling the Internet for these gems and providing insightful analytics for content on the Internet.

The semantic web has grown in popularity and has the attention of all domains of interest including industry, research, and academia. For example, the ISWC or International Semantic Web Conference is held every year and attracts many industry leaders to attend. They state on their 2013 conference web page:

Linked Data and Semantic Technologies are new disruptive technologies gaining traction in mainstream business providing access to rich analytics opportunity. The
Industry track at ISWC 2013 will provide a platform for key players in the field to share their insights and experience of using these technologies in the wild.

Topics of interest include, but are not limited to, the following:

- Applications of Semantic Technologies in various industrial domains (automotive, financial, healthcare and life sciences, energy industry, etc.)
- Applications of Semantic Technologies in community and governmental services
- Industrial trends related to the usage of Linked Data and Semantic Technologies
- Financial and strategic investments in Linked Data and Semantic Technologies
2. BIG DATA TECHNOLOGIES

Big Data is difficult to work with using relational databases, statistical software, and visualization software. Massively parallel software running on tens, hundreds, or even thousands of servers is more suitable for Big Data.

A 2011 McKinsey report suggests suitable technologies include A/B testing, association rule learning, classification, cluster analysis, crowd sourcing, data fusion and integration, ensemble learning, genetic algorithms, machine learning, natural language processing, neural networks, pattern recognition, predictive modeling, regression, sentiment analysis, signal processing, supervised and unsupervised learning, simulation, time series analysis and visualization. Additional technologies being applied to big data include massively parallel-processing (MPP) databases, search-based applications, data-mining grids, distributed file systems, distributed databases, cloud computing platforms, the Internet, and scalable storage systems (WikiPedia: Big Data, 2013).

Even though there are many suitable technologies to solve Big Data issues as indicated in the list above given by the McKinsey report, this work will explore a handful of the more popular ones. In particular the Amazon Cloud, Hadoop's MapReduce, and three open-source parsers.

2.1. Amazon Cloud Technologies

Amazon Web Services (AWS) cloud platform, one of the most popular cloud platforms. It already has Hadoop's MapReduce Framework available for use. We store the large RDF files in Amazon Simple Storage Service (S3) (Amazon S3, 2013) and use
the Amazon Elastic MapReduce (EMR) (Amazon EMR, 2013) framework running on the Amazon Elastic Compute Cloud (EC2) (Amazon EC2, 2013) to run the parsers.

2.2. Hadoop's Map/Reduce

The Hadoop Map/Reduce architecture provides a parallel processing framework that might be a better solution than multithreading. Multithreading requires adept knowledge and skill for the programmer in that coordinating each thread and critical sections can cause many problems. Multithreading requires semaphores, locks, etc. which require tedious testing to ensure there is no race or dead lock conditions. Hadoop eliminates the shared state completely and reduces the issues mentioned above.
This is the fundamental concept of functional programming. Data is explicitly passed between functions as parameters or return values which can only be changed by the active function at that moment. In this case functions are connected to each other in the shape of a directed acyclic graph. Since there is no hidden dependency (via shared state), functions in the directed acyclic graph can run anywhere in parallel as long as one is not an ancestor of the other.

![Map/Reduce Directed Acyclic Graph](image)

**Figure 2.2 Map/Reduce Directed Acyclic Graph**

Map/reduce is a specialized directed acyclic graph which can be used for many purposes. It is organized as a “map” function which transforms a piece of data into some number of key/value pairs. Each of these elements will then be sorted by their key and reach to the same node, where a “reduce” function is used to merge the values (of the same key) into a single result. The code snippet below shows a typical map and reduce function. Several may be chained together to implement a parallel algorithm for different use cases as shown in Figure 2.3.
map(input_record) {
    ...
    emit(k1, v1)
    ...
    emit(k2, v2)
    ...
}
reduce (key, values) {
    aggregate = initialize()
    while (values.has_next) {
        aggregate = merge(values.next)
    }
    collect(key, aggregate)
}

Figure 2.3 Map/Reduce Chaining

HDFS

The distributed file system is another innovation essential to the performance of the Hadoop framework. HDFS can handle large files in the gigabytes and beyond with sequential read/write operation. These large files are broken into chunks and stored across multiple data nodes as local files.

There is a master “NameNode” to keep track of overall file directory structure and the placement of chunks. This NameNode is the central control point and may re-distributed replicas as needed. DataNode reports all its chunks to the NameNode at bootup.
To read a file, the client API will calculate the chunk index based on the offset of the file pointer and make a request to the NameNode. The NameNode will reply which DataNodes has a copy of that chunk. From this points, the client contacts the DataNode directly without going through the NameNode.

To write a file, client API will first contact the NameNode who will designate one of the replica as the primary (by granting it a lease). The response of the NameNode contains who is the primary and who are the secondary replicas. Then the client push its changes to all DataNodes in any order, but this change is stored in a buffer of each DataNode. After changes are buffered at all DataNodes, the client send a “commit” request to the primary, which determines an order to update and then push this order to all other secondaries. After all secondaries complete the commit, the primary will response to the client about the success.

Figure 2.4 HDFS Architecture (Apache Hadoop)
Job Management

The job execution starts when the client program, in this case the Amazon EMR GUI, submits a job configuration to the JobTracker as well as the Java jar file that specifies the map and reduce functions, as well as the input and output path of data. It must be noted here that the job configuration has many properties that can be configured.

The JobTracker will first determine the number of splits (each split is configurable, ~16-64MB) from the input path, and select some TaskTracker based on their network proximity to the data sources, then the JobTracker sends the task requests to those selected TaskTrackers.

Each TaskTracker will start the map phase processing by extracting the input data from the splits. For each record parsed by the “InputFormat”, it invokes the user provided “map” function, which emits a number of key/value pair in the memory buffer. A periodic wakeup process will sort the memory buffer into different reducer nodes by invoking the “combine” function. The key/value pairs are sorted into one of the local files created by reducer nodes.

All splits are complete when the map task completes; then the TaskTracker will notify the JobTracker. When all the TaskTrackers are done, the JobTracker will notify the selected TaskTrackers for the reduce phase.

Each TaskTracker will read the region files remotely. It sorts the key/value pairs and for each key, it invokes the “reduce” function, which collects the key/aggregatedValue into the output file. There is one key/aggregatedValue per reducer node.
The Hadoop Map/Reduce framework is resilient and attempts to prevent crashes of any components. The JobTracker keeps track of the progress of each phase and periodically checks the TaskTracker for their health status. When any of the map phase TaskTracker crashes, the JobTracker will reassign the map task to a different TaskTracker node, which will rerun all the assigned splits. If the reduce phase TaskTracker crashes, the JobTracker will rerun the reduce at a different TaskTracker. That way all map and reduce tasks will be completed.

After both phases complete, the JobTracker will unblock the client program. There may be several outputs correlating with how many reducers were created.

Figure 2.5 Hadoop Job Management

Note: Elements of this section excerpted from (Ho, 2008)
3. SEMANTIC WEB TECHNOLOGIES

The semantic web has become more realized in recent years. The evolution from keyword-based search on the Internet to semantic-based search is becoming more apparent. Microsoft BING uses semantic search to help users find information from ontological association. The semantic web is sometimes referred to as "Web 3.0".

The Semantic Web is a collaborative movement led by the international standards body, the World Wide Web Consortium (W3C). The standard promotes common data formats on the World Wide Web. By encouraging the inclusion of semantic content in web pages, the Semantic Web aims at converting the current web dominated by unstructured and semi-structured documents into a "web of data". The Semantic Web stack builds on the W3C's Resource Description Framework (RDF).

According to the W3C, "The Semantic Web provides a common framework that allows data to be shared and reused across application, enterprise, and community boundaries."

The term was coined by Tim Berners-Lee, the inventor of the World Wide Web and director of the World Wide Web Consortium, which oversees the development of proposed Semantic Web standards. He defines the Semantic Web as "a web of data that can be processed directly and indirectly by machines." (Semantic Web)
3.1. Ontologies

In computer science and information science, an ontology formally represents knowledge as a set of concepts within a domain, and the relationships between pairs of concepts. It can be used to model a domain and support reasoning about entities.

In theory, an ontology is a "formal, explicit specification of a shared conceptualisation". An ontology renders shared vocabulary and taxonomy which models a domain with the definition of objects/concepts, as well as their properties and relations.

Ontologies are the structural frameworks for organizing information and are used in artificial intelligence, the Semantic Web, systems engineering, software engineering.
biomedical informatics, library science, enterprise bookmarking, and information architecture as a form of knowledge representation about the world or some part of it.

The creation of domain ontologies is also fundamental to the definition and use of an enterprise architecture framework (Ontology (Information Science)).

Examples of current published ontologies include:

- IDEAS Group, a formal ontology for enterprise architecture being developed by the Australian, Canadian, UK and U.S. Defence Depts
- Plant Ontology for plant structures and growth/development stages, etc.
- Dublin Core, a simple ontology for documents and publishing
- BabelNet, a very large multilingual semantic network and ontology, lexicalized in many languages
- Basic Formal Ontology, a formal upper ontology designed to support scientific research
Figure 3.2 Main Entities and Relationships of the Innovation

Figure 3.3 Main Example Ontology (Innovation Ontology)
3.2. Rich Data Format (RDF)

RDF is a standard model for data interchange on the Web. RDF has features that facilitate data merging even if the underlying schemas differ, and it specifically supports the evolution of schemas over time without requiring all the data consumers to be changed.

RDF extends the linking structure of the Web to use URIs to name the relationship between things as well as the two ends of the link (this is usually referred to as a “triple”). Using this simple model, it allows structured and semi-structured data to be mixed, exposed, and shared across different applications.

This linking structure forms a directed, labeled graph, where the edges represent the named link between two resources, represented by the graph nodes. This graph view is the easiest possible mental model for RDF and is often used in easy-to-understand visual explanations.
Below is an RDF graph and the RDF represented in XML describing Eric Miller (RDF):

```xml
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:contact="http://www.w3.org/2000/10/swap/pim/contact#">
  <contact:Person rdf:about="http://www.w3.org/People/EM/contact#me">
    <contact:fullName>Eric Miller</contact:fullName>
    <contact:mailbox rdf:resource="mailto:em@w3.org"/>
    <contact:personalTitle>Dr.</contact:personalTitle>
  </contact:Person>
</rdf:RDF>
```

Figure 3.4 RDF Graph Representing Eric Miller

<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:contact="http://www.w3.org/2000/10/swap/pim/contact#">
  <contact:Person rdf:about="http://www.w3.org/People/EM/contact#me">
    <contact:fullName>Eric Miller</contact:fullName>
    <contact:mailbox rdf:resource="mailto:em@w3.org"/>
    <contact:personalTitle>Dr.</contact:personalTitle>
  </contact:Person>
</rdf:RDF>
3.3. Parsers

The Web Data Commons files are stored in RDF NQuads (RDF NQuads, 2012) and can only be parsed directly with a small handful of parsers. Triples, the more popular format is supported by almost all RDF parsers and tools.

Apache Jena ARQ RIOT

Apache Jena is a Java framework for building Semantic Web applications. Jena provides a collection of tools and Java libraries to help you to develop semantic web and linked-data apps, tools and servers.

The Jena Framework includes:

- an API for reading, processing and writing RDF data in XML, N-triples and Turtle formats - this work is using this module.
- an ontology API for handling OWL and RDFS ontologies;
- a rule-based inference engine for reasoning with RDF and OWL data sources;
- stores to allow large numbers of RDF triples to be efficiently stored on disk;
- a query engine compliant with the latest SPARQL specification
- servers to allow RDF data to be published to other applications using a variety of protocols, including SPARQL

In April 2012, Jena graduated from the Apache incubator process and was approved as a top-level Apache project (Apache Jena, 2013).

"RDF has an XML syntax and many who are familiar with XML will think of RDF in terms of that syntax. This is mistake. RDF should be understood in terms of its
data model. RDF data can be represented in XML, but understanding the syntax is secondary to understanding the data model." (Apache Jena Tutorial, 2013)

Anything To Triples (any23) is a library, a web service and a command line tool that extracts structured data in RDF format from a variety of Web documents.

- RDF/XML, Turtle, Notation 3
- RDFa with RDFa1.1 prefix mechanism
- Microformats: Adr, Geo, hCalendar, hCard, hListing, hResume, hReview, License, XFN and Species
- HTML5 Microdata: (such as Schema.org)
- CSV: Comma Separated Values with separator auto-detection.

Apache Any23 is used in major Web of Data applications such as sindice.com and sig.ma. It is written in Java and licensed under the Apache License. Apache Any23 can be used in various ways:

- As a library in Java applications that consume structured data from the Web.
- As a command-line tool for extracting and converting between the supported formats.
- As online service API available at any23.org.

Any23, which is based on Sesame Parser, appears to have ended its incubator status in August of 2012. However, at this time, it is not listed in the formal list of Apache Projects (Any23 Incubator, 2013).
NxParser is a Java open source, streaming, non-validating parser for the Nx format, where x = Triples, Quads, or any other number. For more details see the specification for the NQuads format, a extension for the N-Triples RDF format. Note that the parser handles any combination or number of N-Triples syntax terms on each line (the number of terms per line can also vary).

It ate 2 million quads (~4GB, (~240MB compressed)) on a T60p (Win7, 2.16 GHz) in approximately 1 minute and 35 seconds. Overall, it's more than twice as fast as the previous version when it comes to reading Nx.

It comes in two versions: lite and not-so-lite. The latter is provided "as-is" and comes with many features, most of which is not needed for this work. This work used the lite version. There is some code for batch sorting large-files and various other utilities, which some may find useful. If you just want to parse Nx into memory, use the lite version.

The NxParser is non-validating, meaning that it will happily eat non-conformant N-Triples. Also, the NxParser will not parse certain valid N-Triples files where (i) terms are delimited with tabs and not spaces; (ii) the final full-stop is not preceded by a space (NX Parser, 2012). The NxParser comes from YARS created by DERI Galway.

DERI Galway, the Digital Enterprise Research Institute is an institute of the National University of Ireland, Galway located in Galway, Ireland. The focus of the research in DERI Galway is on Semantic Web (NxParser Deri, 2012).

YARS is Yet another RDF store. YARS is a data store for RDF in Java and allows for querying RDF based on a declarative query language, which offers a
somewhat higher abstraction layer than the APIs of RDF toolkits such as Jena or Redland.

YARS uses Notation3 as a way of encoding facts and queries (Nxparsen Yars, 2012).
4. RELATED WORKS

This work is considerably unique. Many days of research were spent looking for exactly the same kind of work. The IEEE was searched as well as the Internet but nothing was found that deals with these three parsers in Amazon Hadoop. Furthermore, there is not many benchmarks yet done for BigData technologies. Let this work be one of the first. Even though this work seems to be unique, the following three theses have similarities to this paper. See Appendix A for a complete abstract for each of the other related works.

The **Data Intensive Query Processing for Large RDF Graphs Using Cloud Computing** (Mohammad Farhan Husain, 2010) paper is working with RDF, Hadoop in the Cloud like this paper but it is using a database and triples instead of no database and NQuads. This paper is also not concerned with the Web Data Commons content containing most of the Internets pages of RDF. This paper describes an algorithm that determines the performance cost of executing a SPARQL query on the RDF. The RDF query language SPARQL is used to query an RDF repository. In Mohammad's paper, RDF needs to be converted and it is handed over to the Hadoop cluster to process. In some ways Mohammad's paper is a performance analysis of an algorithm that uses Hadoop. On the other hand, this paper compares the performance of three parsers being testing on the Amazon application services.

The **RDF on Cloud Number Nine** (Raffael Stein, 2009) paper covers using SimpleDB on the Amazon Cloud. The team created an RDF store which acts as a back end for the Jena Semantic Web framework and stores its data within the SimpleDB. They stored RDF triples and queried them while checking performnace against the Berlin SPARQL Benchmark to evaluate their solution and compare it to other state of the art
triple stores. This paper is a great example of using Jena and the Amazon Cloud to query RDF. However, this paper, like the one above, use databases to store and process RDF. There is still challenges using databases when data sizes are in gigabytes, terabytes, and beyond.

The RDF Data Management in the Amazon Cloud (Bugiotti, 2012) paper also uses SimpleDB in the Amazon Cloud. This team attempts using various indexing techniques to try to guess the query path in order to optimize performance.

Finally, the Web Data Commons – Extracting Structured Data from Two Large Web Corpora (Hannes Mühleisen, 2012) paper reports about the content of the entire Internet represented by RDF Quads and shows the popularity of various format types found within the pages.
5. ANALYSIS AND DESIGN OF QUERY PROGRAM

This section will cover the design of the program that will compare performance of three open source parsers by querying large RDF data sets. Included will be an overview of the process and methods of performing such work on the Amazon Cloud using Java technology. This section will describe the components of the system used to compare the parsers.

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RDF Files</td>
<td>Web Data Commons' RDF NQuad files</td>
</tr>
<tr>
<td>Amazon EC2</td>
<td>Highly scalable, parallel computing infrastructure</td>
</tr>
<tr>
<td>Amazon S3</td>
<td>Scalable and distributed storage running on EC2</td>
</tr>
<tr>
<td>Amazon EMR</td>
<td>Implementation of Hadoop's MapReduce services running on EC2</td>
</tr>
<tr>
<td>Parsers</td>
<td>Jena, NXParser, and Any23 open source parser programs used to manipulate the RDF NQuad files</td>
</tr>
<tr>
<td>Java</td>
<td>Language used to write the query program with Hadoop libraries that compares parser performance</td>
</tr>
</tbody>
</table>

Table 5.1 Component Descriptions
This work will query the Web Data Common RDF extraction from the Common Crawl Corpus which comprises over five billion web pages on the Internet. The query will be performed by three open source parsers running separately on Hadoop in the Amazon Cloud. They are run separately to compare their performance extracting and parsing the RDF NQuad files. This will allow us to perform analytics on the entire Internet by scanning the RDF files and performing query functions on the NQuads. The high level process is described below:

Figure 5.1 Query Program Architecture
5.1. Process Steps

<table>
<thead>
<tr>
<th>Design Steps</th>
<th>Component</th>
</tr>
</thead>
<tbody>
<tr>
<td>Raw web pages containing RDFa</td>
<td>Common Crawl Corpus</td>
</tr>
<tr>
<td>Compressed RDF NQuads converted from RDFa</td>
<td>Web Data Commons (WDC)</td>
</tr>
<tr>
<td>Decompress WDC files and store them on Amazon S3</td>
<td>This Work: File Utilities</td>
</tr>
<tr>
<td>Run Amazon Elastic Map Reduce (Hadoop) and parsers on decompressed files</td>
<td>This Work: RDFParseParallelMain.java (see Appendix F)</td>
</tr>
<tr>
<td>Combine Hadoop's distributed analytics output files into one report file (see Appendix B)</td>
<td>This Work: File Utilities</td>
</tr>
</tbody>
</table>

Table 5.2 Basic Process Steps

The general way to use Hadoop:

1. Determine the number of jobs needed to answer a query
2. Minimize the size of intermediate files so that data copying and network data transfer is reduced
3. Determine number of reducers

Note: Usually we run one or more MapReduce jobs to answer one query. We use the map phase to select data and the reduce phase to group it.

5.2. Common Crawl Corpus

Common Crawl is an attempt to create an open and accessible crawl of the web. Common Crawl is a Web Scale crawl, and as such, each version of our crawl contains
billions of documents from the various sites that we are successfully able to crawl. This dataset can be tens of terabytes in size, making transfer of the crawl to interested third parties costly and impractical. In addition to this, performing data processing operations on a dataset this large requires parallel processing techniques, and a potentially large computer cluster. (Common Crawl, 2012)

5.3. Web Data Common

More and more websites embed structured data describing for instance products, people, organizations, places, events, resumes, and cooking recipes into their HTML pages using markup formats such as RDFa, Microdata and Microformats. The Web Data Commons project extracts all Microformat, Microdata and RDFa data from the Common Crawl web corpus, the largest and most up-to-date web corpus that is currently available to the public, and provide the extracted data for download in the form of RDF-quads and also in the form of CSV-tables for common entity types (e.g. product, organization, location, etc.). In addition, Web Data Commons calculates and publishes statistics about the deployment of the different formats as well as the vocabularies that are used together with each format.

Web Data Commons has extracted all RDFa, Microdata and Microformats data from the August 2012 and the 2009/2010 Common Crawl corpus. This work will use the August 2012 RDF files which is about 100 gigabytes. Webpages are included into the Common Crawl corpora based on their PageRank score, thereby making the crawls snapshots of the current popular part of the Web. For the future, Web Data Commons plans to rerun their extraction on a regular basis as new Common Crawl corpora are becoming available. (Web Data Commons, 2012)
5.4. Amazon Cloud

This work will use several services with the Amazon Cloud. The Hadoop implemented by Amazon Elastic Map Reduce (EMR) framework will be used to analyze the large set of RDF data. The EMR provides fine-grain metrics of many aspects of the running job. From these metrics, we produce the test results charts. Since WDC has done some of the difficult work for us, we start with fairly structured and validated RDF NQuad files. These files are stored in the Amazon S3. After "jarring" up the RDFParseParallelMain program with the parsers jar files, this jar file is saved to the S3 as well. We also put the input files on S3 and setup an output directory for the report files. To run RDFParseParallelMain we go into the EMR control panel and start a new job. The EMR will run the RDFParseParallelMain program on as many processors as we indicate and write output files to the S3.

RDFParseParallelMain

The capability of the program RDFParseParallelMain is to either count entries for each node type in the NQuad, i.e. subject, predicate, object, or context or count occurrences of a query string in a specified node type. The parser comparison testing was done using the former capability of counting occurrences of the node type. We have provided extensive argument passing so we could run the tests in various modes using each of the parsers separately.

The map and reduce functions are fairly simple. The map for the testing is key = node type; value = count. The map for the query is key = query string; value = count. The reduce function simply adds the reoccurring words and outputs the total to the report file.
The complexity in this program is in the parsers. Each parser must break down the NQuad into separate nodes. Once that is done, the program passes the node to the reduce function where it eventually gets counted.

The final Java jar file was a jar of jars along with the main Java program. In other words, the Hadoop, Jena, NXParser, and Any23 application jar files, several for each, were all "jarred" together with the RDFParseParallelMain Java program. This parent jar file is what is used by the Amazon EMR to run the tests.

5.5. Test Design

The testing of each parser is done separately. This ensures there is sufficient control to yield more accurate performance results. The RDFParseParallelMain program was written such that no processing would be done within it. Instead, all processing was handed over to the parsers. The RDFParseParallelMain only provided transport between the input and the Hadoop "reduce" method. More specifically, the Any23 and Jena parsers used callbacks to process whereas the NXParser parser used a tight loop.

The RDF NQuad files sizes are 1, 4, 16, and 24 gigabytes and the CPU count is 10, 24, 64, and 90 cpu's. These quantities are not entirely random. Earlier testing, not documented in this work, gave a basis to establish amounts for this testing. The file sizes were established based on a nice progression. The last file size of 24 was used to reduce the cost of processing uploading the files as well as processing. The CPU count was established with similar justifications but furthermore, Amazon has a significant cost increase over 100 CPU's. Unfortunately, cost became an issue in determining how much
of the Amazon would be reasonable for a college graduate project. Irrespective of the cost issue, the results of the testing are very sound as shown in the graphs.

In the next section, the paper will cover the test results and what we can conclude from them. Hopefully, the results and testing methodology will be useful to other Big Data and Semantic Web practitioners.
6. RESULTS AND CONCLUSION

We will analyze two aspects of the results, i.e. the performance comparison and CPU cost of the parsers. The performance comparison will include looking at the performance curve to see where each parser did their best. For example, parser X did its best when processing the 4.7 GB moderately sized file with 64 CPU's. In the same way, we can look at the CPU cost curve and see where the best cost occurs.

Before looking at the results, it is important to understand a few points. Statistical integrity was not achieved in this testing due to the cost to run the jobs. However, due to the stability and consistency of the Amazon Cloud infrastructure, there is value in the final results of the testing. The main question is, if we rerun the tests again will we get significantly different results. It is highly unlikely that running these simple parsing programs several times will show significant differences. Therefore, outcomes of the testing are taken with full face value. We have suggested in the future work section of this paper to run more tests for statistical validity.
This chart shows parser performance based on file size.

In general, the testing produced unexpected results. However, the results seem to have some consistency within the overall testing. In other words, the charts show unusual behavior of many of the tests but there is reasonable justification.

For the most part, Jena is the slowest overall. Any23 and NXParser out-performed each other in different permutations of file size and CPU count. For the important test where the largest size of file meets low CPU count and therefore CPU utilization, NXParser is the clear winner.

If we look at each file size and then the CPU count we can see that there is a different winner in different permutations. For the 0.7 GB file the NXParser parser was the fastest in the extreme cases of 10 CPU's and 90 CPU's. It was the slowest in the middle CPU counts. Then the NXParser parser was the fastest in the 24 and 90 runs of
the 4.7 GB file and slowest on 64 and the middle on 10. For the final 15.9 GB file, the
NXParser parser was the fastest at 10 and 24 CPU’s.

In the following, we explore each of the parsers and their individualized
performance curves. As mentioned before, each parser does better or worse at a particular
file size to CPU count permutation. We have taken the parser performance chart above
and created separate charts for each parser to ensure the individualization is apparent.

Any23

![Parser Performance by File Size](image)

Figure 6.2 Parser Performance - Any23

The Any23 parser was in the middle of almost all permutations except it was
faster in the 4.7 GB, 10 CPU and 15.9 GB, 64 and 90 CPU runs. Strangely, it was much
slower in the 4.7 GB, 90 CPU run. For the smallest file is slow at first, gains momentum
and then quickly slows down again. Apparently the overhead of CPU’s slows it down.
For the medium file size, it optimizes performance at the 24 CPU count. For the large file, it improves over the initial progression of CPU count but falters slightly at the 90 CPU count.

NXParser

![Graph showing parser performance by file size](image)

**Figure 6.3 Parser Performance - NXParser**

The NXParser Any23 parsers had very similar performance characteristics which makes it difficult to see who the ultimate winner would be. It appears the NXParser parser won more times than Any23. The NXParser also has the best CPU usage index as noted in the next section. The NXParser shows normal time increases for the small file but speeds up when using 90 CPU's. The optimal time on the medium size file is 24 CPU's and does not take advantage of more CPU's. The large file performance improves as expected with more CPU's until the use of 90 CPU's where there is a minor dip.
Jena

The Jena parser, for the most part, was the slowest of all and used the most CPU time, especially in the 15.9 GB file runs. Oddly, it was the best performer in the 0.7 GB, 64 CPU run. For the small file it progressed negatively as the CPU count increased. On the medium file size, performance increased as the number of CPU’s increased until the 90 CPU count. Finally, Jena took full advantage of throwing CPU’s at it.

CPU Cost

It is not always enough to see how fast a program runs but also the cost of running it. Especially when running on cloud environments where there is a fee for everything and costs can grow quickly. This section compares the three parsers with CPU usage and therefore cost on each of the runs.
This chart shows CPU utilization per parser based on file size. The Parser CPU Cost by File Size chart correlates with the Parser Performance chart. It shows the total time spent running the job by adding together the CPU utilization for each of the CPU's. The chart shows that the parsers CPU usage is normally distributed for the smaller file sizes but not for the 15.9 GB file. There seems to be a dip at the 90 CPU test for all the parsers. The dip correlates to the performance graph of each of the processors. This may indicate there is a diminishing return as the CPU's increase for each file size.
This chart shows CPU utilization per parser based on different number of processors.

This last chart shows CPU utilization from a slightly different perspective. The chart is similar to the CPU utilization by file size as it show the same dip in the 90 CPU column. This chart may help to strengthen the other perspective.

Reviewing bother charts it is very clear that Jena utilizes the most CPU than either of the other two. The NXParser seems to be the least greedy when utilizing compute resources. Therefore the NXParser would be the least expensive to run in most cases.
Conclusion

It is clear that the NX Parser is the most efficient and may be the best for applications needing no validation. The Any23 parser is comparable to the NX Parser in performance but not in efficiency.

In order to understand why Jena is the slowest, the Jena parser has a larger codebase and is performing validations while parsing. Any23 and the NX Parser are doing simple format validations before writing to the destination output.

Computing speed, dataset size and quality of output are the critical parameters to be considered when determining an application's needs. Usually we want processing to be instantaneous regardless of the dataset size but this is not realistic. If the application has extremely large datasets and quality is critical, then we will have to give up speed and use the Jena processor. For some extremely large datasets, Jena may not be feasible. On the other hand, if the application with extremely large datasets is not subject to quality requirements or data validation has already been performed on the dataset, then the NX Parser or Any23 are recommended. A pre-validation step using a specialized format checker may be required for some applications that require both quality and performance when using the NX Parser or Any23.

It is also clear that there is acceptable speedup when using the Amazon EMR with the smaller file sizes, i.e. 0.7 and 4.7 GB. At the large 15.9 GB file size, the performance seemed to level off and the results were similar or worse than the smaller files. However, it is not clear where the best or optimal cost to performance curve might be. That is up to more testing of the RDFParseParallelMain program which could be done in some future
work. But it is interesting to witness the various performance curves within the permutations of file size and CPU count. The charts can be used for each parser as a starting point when the file size is known. This could prevent overspending of CPU usage for any given test.

Hopefully, this work has provided a rich set of information about tackling Big Data and Semantic Web challenges. And, more specifically, when your challenge is to query large RDF datasets, the determination of which parser to use may now be much easier. This work has been an exhausting challenge. But with any activity, there is always more to do. The next section highlights some of the areas where future investigation will be valuable.
7. FUTURE WORK

This report did not have the time or budget to perform extensive testing. Even though the testing program and methodology are very effective and produced reasonable results, more testing could solidify the observed trends. Larger dataset sizes would put higher stress on the parsers and CPU's. Using more and faster CPU’s could result in a better understanding of the speedup curve. CPU analysis could also indicate where there is a diminishing return. In other words, the use of different CPU models could give the optimal approach for running certain application types.

Even though this work was performed on pre-validated datasets, un-validated datasets could tested and, along with the above observations, a better heuristic could be developed for the determination of use for a variety of applications. In other words, there may be combinations of parsers and validators that could run in a sequence of steps optimized for a particular application. And, there are many CPU combinations that could be considered for the best result. For example, a very large dataset requiring an average level of validation, might optimize at 200 large standard, first-generation instances. A chart could be developed based on the application type.
8. REFERENCE


(n.d.). Retrieved February 2013, from Amazon EMR:

http://aws.amazon.com/elasticmapreduce/


(n.d.). Retrieved April 2013, from WikiPedia: Big Data:

http://en.wikipedia.org/wiki/Big_data


(n.d.). Retrieved November 2013, from Apache Jena Tutorial:

http://jena.apache.org/tutorials/rdf_api.html

(n.d.). Retrieved October 2013, from Any23 Incubator:

http://incubator.apache.org/projects/index.html#graduated


directed from http://sw.deri.org/2006/08/nxparser/

(n.d.). Retrieved October 2012, from NxParser Deri:

http://semanticweb.org/wiki/DERI_Galway


http://lists.w3.org/Archives/Public/www-rdf-logic/2001Aug/0007.html


http://stick.ischool.umd.edu/innovation_ontology.html


*Ontology (Information Science)*. (n.d.). Retrieved March 2013, from Wikipedia:

http://en.wikipedia.org/wiki/Ontology_(information_science)


*RDF*. (n.d.). Retrieved March 2013, from W3C: http://www.w3.org/RDF/

*Semantic Web*. (n.d.). Retrieved 2013, from Wikipedia:


http://www.computer.org/portal/web/membership/13-Top-Trends-for-2013
Appendix A: Related Works

Data Intensive Query Processing for Large RDF Graphs Using Cloud Computing Tools

Abstract—Cloud computing is the newest paradigm in the IT world and hence the focus of new research. Companies hosting cloud computing services face the challenge of handling data intensive applications. Semantic web technologies can be an ideal candidate to be used together with cloud computing tools to provide a solution. These technologies have been standardized by the World Wide Web Consortium (W3C). One such standard is the Resource Description Framework (RDF). With the explosion of semantic web technologies, large RDF graphs are common place. Current frameworks do not scale for large RDF graphs. In this paper, we describe a framework that we built using Hadoop, a popular open source framework for Cloud Computing, to store and retrieve large numbers of RDF triples. We describe a scheme to store RDF data in Hadoop Distributed File System. We present an algorithm to generate the best possible query plan to answer a SPARQL Protocol and RDF Query Language (SPARQL) query based on a cost model. We use Hadoop’s MapReduce framework to answer the queries. Our results show that we can store large RDF graphs in Hadoop clusters built with cheap commodity class hardware. Furthermore, we show that our framework is scalable and efficient and can easily handle billions of RDF triples, unlike traditional approaches.

RDF on Cloud Number Nine

Abstract—we examine whether the existing 'Database in the Cloud' service SimpleDB can be used as a back end to quickly and reliably store RDF data for massive parallel access. Towards this end we have implemented 'Stratustore', an RDF store which acts as a back end for the Jena Semantic Web framework and stores its data within the
SimpleDB. We used the Berlin SPARQL Benchmark to evaluate our solution and compare it to state of the art triple stores. Our results show that for certain simple queries and many parallel accesses such a solution can have a higher throughput than state of the art triple stores. However, due to the very limited expressiveness of SimpleDB's query language, more complex queries run multiple orders of magnitude slower than the state of the art and would require special indexes. Our results point to the need for more complex database services as well as the need for robust, possible query dependent index techniques for RDF.

**RDF Data Management in the Amazon Cloud**

Abstract—Cloud computing has been massively adopted recently in many applications for its elastic scaling and fault-tolerance. At the same time, given that the amount of available RDF data sources on the Web increases rapidly, there is a constant need for scalable RDF data management tools. In this paper we propose a novel architecture for the distributed management of RDF data, exploiting an existing commercial cloud infrastructure, namely Amazon Web Services (AWS). We study the problem of indexing RDF data stored within AWS, by using SimpleDB, a key-value store provided by AWS for small data items. The goal of the index is to efficiently identify the RDF datasets which may have answers for a given query, and route the query only to those. We devised and experimented with several indexing strategies; we discuss experimental results and avenues for future work.

**Web Data Commons – Extracting Structured Data from Two Large Web Corpora**

Abstract—More and more websites embed structured data describing for instance products, people, organizations, places, events, resumes, and cooking recipes into their
HTML pages using encoding standards such as Microformats, Microdata and RDFa. The Web Data Commons project extracts all Microformat, Microdata and RDFa data from the Common Crawl web corpus, the largest and most up-to-date web corpus that is currently available to the public, and provides the extracted data for download in the form of RDF-quads. In this paper, we give an overview of the project and present statistics about the popularity of the different encoding standards as well as the kinds of data that are published using each format.
### Appendix B: Testing Result Data

<table>
<thead>
<tr>
<th>Size (GB)</th>
<th>CPUs</th>
<th>Parser</th>
<th>Read Bytes</th>
<th>Written bytes</th>
<th>Date</th>
<th>End</th>
<th>Start</th>
<th>Run Dur</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.7 GB</td>
<td>10</td>
<td>Any23</td>
<td>38058409</td>
<td>38058409</td>
<td>3/2/2013</td>
<td>5:07:43</td>
<td>5:05:00</td>
<td>0:02:43</td>
</tr>
<tr>
<td>0.7 GB</td>
<td>24</td>
<td>Any23</td>
<td>38058409</td>
<td>38058409</td>
<td>3/2/2013</td>
<td>5:07:43</td>
<td>5:05:00</td>
<td>0:02:43</td>
</tr>
<tr>
<td>0.7 GB</td>
<td>64</td>
<td>Any23</td>
<td>38058409</td>
<td>38058409</td>
<td>3/2/2013</td>
<td>5:07:43</td>
<td>5:05:00</td>
<td>0:02:43</td>
</tr>
<tr>
<td>0.7 GB</td>
<td>90</td>
<td>Any23</td>
<td>38058409</td>
<td>38058409</td>
<td>3/2/2013</td>
<td>5:07:43</td>
<td>5:05:00</td>
<td>0:02:43</td>
</tr>
<tr>
<td>4.7 GB</td>
<td>64</td>
<td>Any23</td>
<td>220294173</td>
<td>220294173</td>
<td>3/2/2013</td>
<td>22:47:30</td>
<td>22:40:41</td>
<td>0:06:49</td>
</tr>
<tr>
<td>4.7 GB</td>
<td>90</td>
<td>Any23</td>
<td>220294173</td>
<td>220294173</td>
<td>3/2/2013</td>
<td>22:47:30</td>
<td>22:40:41</td>
<td>0:06:49</td>
</tr>
<tr>
<td>15.9 GB</td>
<td>10</td>
<td>Any23</td>
<td>735910914</td>
<td>735910914</td>
<td>3/2/2013</td>
<td>5:18:48</td>
<td>5:00:38</td>
<td>0:18:10</td>
</tr>
<tr>
<td>15.9 GB</td>
<td>24</td>
<td>Any23</td>
<td>735910914</td>
<td>735910914</td>
<td>3/2/2013</td>
<td>5:18:48</td>
<td>5:00:38</td>
<td>0:18:10</td>
</tr>
<tr>
<td>15.9 GB</td>
<td>64</td>
<td>Any23</td>
<td>735910914</td>
<td>735910914</td>
<td>3/2/2013</td>
<td>5:18:48</td>
<td>5:00:38</td>
<td>0:18:10</td>
</tr>
<tr>
<td>15.9 GB</td>
<td>90</td>
<td>Any23</td>
<td>735910914</td>
<td>735910914</td>
<td>3/2/2013</td>
<td>5:18:48</td>
<td>5:00:38</td>
<td>0:18:10</td>
</tr>
</tbody>
</table>

**Table B.1 Test Data - Run Duration**

49
<table>
<thead>
<tr>
<th>Size (GB)</th>
<th>CPUs</th>
<th>Parser</th>
<th>Maps</th>
<th>Reds</th>
<th>CPU Time Spent</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.7 GB (768480)</td>
<td>10</td>
<td>Any23</td>
<td>50</td>
<td>24</td>
<td>586460 ms 9.77 minutes 0.9774 min/CPU</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NX</td>
<td>49</td>
<td>24</td>
<td>263260 4.39 min/CPU</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Jena</td>
<td>48</td>
<td>22</td>
<td>910160 15.17 min/CPU</td>
</tr>
<tr>
<td>4.7 GB (4703019)</td>
<td>10</td>
<td>Any23</td>
<td>82</td>
<td>20</td>
<td>1820270 30.34 min/CPU</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NX</td>
<td>83</td>
<td>21</td>
<td>1180780 19.68 min/CPU</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Jena</td>
<td>83</td>
<td>20</td>
<td>4758400 79.31 min/CPU</td>
</tr>
<tr>
<td>15.9 GB (15880165)</td>
<td>10</td>
<td>Any23</td>
<td>256</td>
<td>19</td>
<td>5965360 99.42 min/CPU</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NX</td>
<td>256</td>
<td>20</td>
<td>4060650 67.68 min/CPU</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Jena</td>
<td>255</td>
<td>20</td>
<td>16305780 27.16 min/CPU</td>
</tr>
</tbody>
</table>

Table B.2 Test Data - CPU Usage
<table>
<thead>
<tr>
<th>Size (GB)</th>
<th>CPUs</th>
<th>Parser</th>
<th>Heap Usage</th>
<th>Vir Mem</th>
<th>Phys Mem</th>
<th>Map Out Recs</th>
<th>Red Out Recs</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.7 GB</td>
<td>10</td>
<td>Any23</td>
<td>8156758016</td>
<td>27679924224</td>
<td>9479708672</td>
<td>3604970</td>
<td>610352</td>
</tr>
<tr>
<td>(768480)</td>
<td></td>
<td>NX</td>
<td>8300728320</td>
<td>27678363648</td>
<td>9567363072</td>
<td>3605106</td>
<td>610352</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Jena</td>
<td>8458510336</td>
<td>27693436928</td>
<td>9879961600</td>
<td>3605106</td>
<td>610352</td>
</tr>
<tr>
<td>24</td>
<td>Any23</td>
<td>19680710656</td>
<td>67032420352</td>
<td>2298283264</td>
<td>3604970</td>
<td>610352</td>
<td>610352</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NX</td>
<td>19054542848</td>
<td>65101975552</td>
<td>22095044600</td>
<td>3605106</td>
<td>610352</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Jena</td>
<td>19369521152</td>
<td>65122095104</td>
<td>22883553280</td>
<td>3605106</td>
<td>610352</td>
</tr>
<tr>
<td>64</td>
<td>Any23</td>
<td>54352261120</td>
<td>178811256832</td>
<td>63157911552</td>
<td>3604970</td>
<td>610352</td>
<td>610352</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NX</td>
<td>52172595200</td>
<td>175045496832</td>
<td>60996018176</td>
<td>3605106</td>
<td>610352</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Jena</td>
<td>54052204544</td>
<td>178911289344</td>
<td>63759601664</td>
<td>3605106</td>
<td>610352</td>
</tr>
<tr>
<td>90</td>
<td>Any23</td>
<td>74637078528</td>
<td>247519703040</td>
<td>87260676096</td>
<td>3604970</td>
<td>610352</td>
<td>610352</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NX</td>
<td>72900059136</td>
<td>241716736000</td>
<td>85121056768</td>
<td>3605106</td>
<td>610352</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Jena</td>
<td>73963462656</td>
<td>245695819776</td>
<td>87905366016</td>
<td>3605106</td>
<td>610352</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>Any23</td>
<td>18524577792</td>
<td>44107022336</td>
<td>19254026240</td>
<td>21708560</td>
<td>3489507</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NX</td>
<td>17823592448</td>
<td>44108169216</td>
<td>18768084992</td>
<td>21711697</td>
<td>3489507</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Jena</td>
<td>18985467904</td>
<td>44120936448</td>
<td>19854336000</td>
<td>21711698</td>
<td>3489507</td>
</tr>
<tr>
<td>4.7 GB</td>
<td>10</td>
<td>Any23</td>
<td>20976795648</td>
<td>67987701760</td>
<td>24200663040</td>
<td>21708560</td>
<td>3489507</td>
</tr>
<tr>
<td>(470319)</td>
<td></td>
<td>NX</td>
<td>20894695424</td>
<td>67977109504</td>
<td>24014401536</td>
<td>21711693</td>
<td>3489507</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Jena</td>
<td>24289488896</td>
<td>68059758592</td>
<td>25861549894</td>
<td>21711698</td>
<td>3489507</td>
</tr>
<tr>
<td>64</td>
<td>Any23</td>
<td>54276911104</td>
<td>178459533312</td>
<td>63098895356</td>
<td>21708560</td>
<td>3489507</td>
<td>610352</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NX</td>
<td>54462210048</td>
<td>179824758784</td>
<td>63345123328</td>
<td>21711693</td>
<td>3489507</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Jena</td>
<td>55595166976</td>
<td>178466717696</td>
<td>65252413440</td>
<td>21711698</td>
<td>3489507</td>
</tr>
<tr>
<td>15.9 GB</td>
<td>10</td>
<td>Any23</td>
<td>25756259712</td>
<td>246615392256</td>
<td>88381861888</td>
<td>21708560</td>
<td>3489507</td>
</tr>
<tr>
<td>(15880165)</td>
<td></td>
<td>NX</td>
<td>27837414912</td>
<td>240707973120</td>
<td>85138845696</td>
<td>21711697</td>
<td>3489507</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Jena</td>
<td>75489120256</td>
<td>246667336064</td>
<td>89436454912</td>
<td>21711698</td>
<td>3489507</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>Any23</td>
<td>491172262912</td>
<td>127899938816</td>
<td>54412804096</td>
<td>73199143</td>
<td>11073880</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NX</td>
<td>49546445152</td>
<td>127861673984</td>
<td>54304096000</td>
<td>73208681</td>
<td>11073879</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Jena</td>
<td>53056798720</td>
<td>127985336320</td>
<td>57638412288</td>
<td>73209186</td>
<td>11073880</td>
</tr>
<tr>
<td>24</td>
<td>Any23</td>
<td>57510166528</td>
<td>140348903424</td>
<td>62076719104</td>
<td>73199143</td>
<td>11073880</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>NX</td>
<td>58101831248</td>
<td>140391297024</td>
<td>62031343616</td>
<td>73208681</td>
<td>11073879</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Jena</td>
<td>61345497088</td>
<td>140423589888</td>
<td>64983293952</td>
<td>73209186</td>
<td>11073880</td>
</tr>
<tr>
<td>64</td>
<td>Any23</td>
<td>66611113984</td>
<td>182585167872</td>
<td>71972209298</td>
<td>73199143</td>
<td>11073880</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>NX</td>
<td>668901709680</td>
<td>181576906008</td>
<td>71595692032</td>
<td>73208681</td>
<td>11073879</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Jena</td>
<td>68070436864</td>
<td>179750244352</td>
<td>7279317352</td>
<td>73209186</td>
<td>11073880</td>
</tr>
<tr>
<td>90</td>
<td>Any23</td>
<td>75611803648</td>
<td>245068771328</td>
<td>88245620736</td>
<td>73199143</td>
<td>11073880</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>NX</td>
<td>75309851520</td>
<td>245038436352</td>
<td>88108773376</td>
<td>73208681</td>
<td>11073879</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Jena</td>
<td>83700068352</td>
<td>242795454464</td>
<td>93066227712</td>
<td>73209186</td>
<td>11073880</td>
</tr>
</tbody>
</table>

Table B.3 Test Data - Memory Usage
Appendix C: RDF Quads From Web Data Commons File


<http://www.facebook.com/2008/fbmlapp_id> "120179524728527"
"Westminster: Boris has achieved little of note as Mayor"

"http://www.politicshome.com/uk/westminster_boris_has_achieved_little_of_note_as_mayor.html"


http://www.aukcje.fm/item-3280012_nowe_apple_iphone_3gs_najtaniej_32_gb_.html <http://opengraphprotocol.org/schema/app_id> "3003280012"


"http://www.aukcje.fm/uploaded/3280012/128x96/3280012_1.jpg"


_:node16oaq1dg6x9546 <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> 


_:node16oaq1dg6x9546 <http://rdf.data-vocabulary.org/#rating> _:node16oaq1dg6x9547 <http://www.shareware.de/3d-modeleditor/> .

_:node16oaq1dg6x9547 <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> 


"http://phpcamp.net/article/codeigniter-handling-errors"
<http://phpcamp.net/article/codeigniter-handling-errors/> .
"Leading cancer researchers meet at Turku BioCity Symposium at the end of August"

"http://www.goodnewsfinland.com/archive/news/leading-cancer-researchers-meet-at-
turku-biocity-symposium-at-the-end-of-august/"

"http://www.goodnewsfinland.com/archive/news/leading-cancer-researchers-meet-at-
turku-biocity-symposium-at-the-end-of-august/

"http://www.goodnewsfinland.com/goodnews2011/goodnewsfromfinland-logo.png"

"http://www.goodnewsfinland.com/archive/news/leading-cancer-researchers-meet-at-
turku-biocity-symposium-at-the-end-of-august/

"http://www.goodnewsfinland.com/archive/news/leading-cancer-researchers-meet-at-
turku-biocity-symposium-at-the-end-of-august/"

"http://www.goodnewsfinland.com/"
BioCity Symposium, which is held in Turku at the end of August, brings together the top of the world's cancer research now for the 19th time. This year's Symposium's theme is Tumor Microenvironment in Cancer Progression. The event is organized by the...
Appendix D: Sample Output

This output is the number of occurrences of unique context nodes found in the
Web Data Common uncompressed RDF NQuad files.

http://www.sanalika.com/destek 1
http://www.sanalika.com/destek/genel/soru/sanalika-nedir 1
http://www.sanalika.com/destek/oyunlar 1
http://www.sanalika.com/destek/uyelik 1
http://www.sanalika.com/destek/vip/soru/anemon-sitesinden-nasil-villa-satin-alabilirim 1
http://www.sanalika.com/destek/vip/soru/arkadaslik-ozelligini-nasil-aktiflestirebilirim 1
http://www.sanalika.com/destek/vip/soru/fisildama-ozelligini-nasil-kullanabilirim 1
http://www.sanalika.com/destek/vip/soru/monitte-loungea-nasil-giris-yapabilirim 1
http://0foreverth.buzznet.com/user/ 2
http://0o0graciex0o0.buzznet.com/user/ 2
http://0wishfulthinkr0.buzznet.com/user/ 2
http://10.buzznet.com/user/ 2
http://1057thehawk.com/bruce-brunch-12411/ 8
http://1057thehawk.com/hawk-2-0-playlist-11611/ 8
http://1057thehawk.com/show/jonesey/ 7
http://1057thexrocks.com/ 7

60
http://10starmovies.com/Watch-Movies-Online/No_Direction_Home_Bob_Dylan_2005/ 1

http://10starmovies.com/Watch-Movies-Online/Rooster_Cogburn_1975/ 1

http://10starmovies.com/Watch-Movies-Online/Two_Mules_For_Sister_Sara_1970/ 1

http://10x10.buzznet.com/user/ 2


http://1102grand.com/1102-grand-history/ 6

http://1102grand.com/1102-grand-reviews-green-announces-practices-choosing-data-center/ 14

http://1102grand.com/2009/08/ 2

http://1102grand.com/2009/11/ 2

http://1102grand.com/2010/01/ 2

http://1102grand.com/2010/04/ 2

http://1102grand.com/2010/07/ 2

http://1102grand.com/5-data-center-temperature/ 14

http://1102grand.com/5-grand-questions-greg/ 15

http://1102grand.com/category/1102-grand/ 2

http://1102grand.com/category/green-it/ 2

http://1102grand.com/category/raised-floors/2

http://1102grand.com/collocation-commodity/ 14

61
Appendix E: Parser and Amazon Configurations

Amazon Cloud

Amazon EMR has made enhancements to Hadoop and other open-source applications to work seamlessly with AWS. For example, Hadoop clusters running on Amazon EMR use Amazon EC2 instances as virtual Linux servers for the master and slave nodes, Amazon S3 for bulk storage of input and output data, and Amazon CloudWatch to monitor cluster performance and raise alarms. You can also move data into and out of Amazon DynamoDB using Amazon EMR and Hive. All of this is orchestrated by Amazon EMR control software that launches and manages the Hadoop cluster. This process is called an Amazon EMR job flow. (Amazon EC2, 2013)

The Hadoop version used was 1.0.3 and was built and distributed by Amazon. No special parameters or bootstrap actions were required to run the jobs. The debugger was turned on to capture performance metrics and potential issues. The Java program required the following packages to run Hadoop:

```java
import org.apache.hadoop.fs.Path;

import org.apache.hadoop.conf.*;

import org.apache.hadoop.io.*;

import org.apache.hadoop.mapred.*;

import org.apache.hadoop.util.*;
```

64
Apache Any23 Parser

The Any23 parser version used was 0.70 and the following Java packages were required to build the test program:

```java
import java.util.ArrayList;
import org.apache.any23.io.nquads.NQuadsParser;
import org.openrdf.model.Statement;
import org.openrdf.rio.ParserConfig;
import org.openrdf.rio.RDFHandlerException;
import org.openrdf.rio.RDFParseException;
import org.openrdf.rio.RDFParser;
import org.openrdf.rio.binary.BinaryRDFWriter;
import org.openrdf.rio.helpers.RDFHandlerBase;
import org.openrdf.rio.helpers.StatementCollector;
```

The NXParser parser version used was 1.2.3 and the following Java packages were required to build the test program:

```java
import org.semanticweb.yars.nx.Node; (see Note)
import org.semanticweb.yars.nx.*;
import org.semanticweb.yars.nx.Node;
```
The Apache Jena parser version used was 2.7.4 and Jena Riot ARQ 2.9.4 and the following Java packages were required to build the test program:

```java
import java.util.HashMap;
import java.util.Map;
import org.openjena.atlas.lib.Sink;
import org.openjena.atlas.lib.SinkNull;
import org.openjena.atlas.lib.SinkWrapper;
import org.openjena.riot.ErrorHandlerFactory;
import org.openjena.riot.Lang;
import org.openjena.riot.RiotParseException;
import org.openjena.riot.RiotReader;
import sun.org.mozilla.javascript.internal.Context;
import com.hp.hpl.jena.graph.Node; (see Note)
import com.hp.hpl.jena.graph.*;
import com.hp.hpl.jena.sparql.core.Quad;
import com.hp.hpl.jena.vocabulary.RDF;
```

Note: There was a conflict between org.semanticweb.yars.nx.Node and com.hp.hpl.jena.graph.Node which required implicit declarations.
Appendix F: Computing Configurations

There are several cost models that can be chosen from on the Amazon EC2 platform. Without naming all of them, we used the On-Demand instance cost model. On-Demand Instances let you pay for compute capacity by the hour with no long-term commitments. This frees you from the costs and complexities of planning, purchasing, and maintaining hardware and transforms what are commonly large fixed costs into much smaller variable costs. On-Demand Instances also remove the need to buy “safety net” capacity to handle periodic traffic spikes.

As mentioned before between 10 and 90 CPU’s where used in the testing. The type of CPU instance used was the first generation (M1) Standard instances. According to Amazon these instances provide customers with a balanced set of resources and a low cost platform that is well suited for a wide variety of applications. The exact system configuration follows:

M1 Small Instance (Default) 1.7 GB of memory, 1 EC2 Compute Unit (1 virtual core with 1 EC2 Compute Unit), 160 GB of local instance storage, 32-bit or 64-bit platform. (Amazon EC2, 2013)
Appendix G: Computing Cost

Figure G.1 Amazon Computing Cost
Appendix H: Amazon Pricing (Amazon EC2, 2013)

On-Demand Instances

On-Demand Instances let you pay for compute capacity by the hour with no long-term commitments. This frees you from the costs and complexities of planning, purchasing, and maintaining hardware and transforms what are commonly large fixed costs into much smaller variable costs.

The pricing below includes the cost to run private and public AMIs on the specified operating system (“Windows Usage” prices apply to Windows Server® 2003 R2, 2008, 2008 R2 and 2012). Amazon also provides you with additional instances for Amazon EC2 running Microsoft Windows with SQL Server, Amazon EC2 running SUSE Linux Enterprise Server, Amazon EC2 running Red Hat Enterprise Linux and Amazon EC2 running IBM that are priced differently.

Pricing is per instance-hour consumed for each instance, from the time an instance is launched until it is terminated or stopped. Each partial instance-hour consumed will be billed as a full hour.

The pricing below is based on data transferred "in" to and "out" of Amazon EC2.

Region: US West - Northern California

<table>
<thead>
<tr>
<th>Standard On-Demand Demand Instances</th>
<th>Linux/UNIX Usage</th>
<th>Windows Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small (Default)</td>
<td>$0.065 per Hour</td>
<td>$0.096 per Hour</td>
</tr>
<tr>
<td>Medium</td>
<td>$0.130 per Hour</td>
<td>$0.192 per Hour</td>
</tr>
<tr>
<td>Size</td>
<td>IN Price per Hour</td>
<td>Out Price per Hour</td>
</tr>
<tr>
<td>-----------------</td>
<td>-------------------</td>
<td>--------------------</td>
</tr>
<tr>
<td>Large</td>
<td>$0.260</td>
<td>$0.384</td>
</tr>
<tr>
<td>Extra Large</td>
<td>$0.520</td>
<td>$0.768</td>
</tr>
</tbody>
</table>

**Data Transfer**

<table>
<thead>
<tr>
<th>IN To Amazon EC2 From</th>
<th>Pricing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Internet</td>
<td>$0.00 per GB</td>
</tr>
<tr>
<td>Another AWS Region (from any AWS Service)</td>
<td>$0.00 per GB</td>
</tr>
<tr>
<td>Amazon S3, Amazon Glacier, Amazon DynamoDB, Amazon SQS, or Amazon SimpleDB in the same AWS Region</td>
<td>$0.00 per GB</td>
</tr>
<tr>
<td>Amazon EC2, Amazon RDS and Amazon ElastiCache instances or Elastic Network Interfaces in the same Availability Zone</td>
<td></td>
</tr>
<tr>
<td>Using a private IP address</td>
<td>$0.00 per GB</td>
</tr>
<tr>
<td>Using a public or Elastic IP address</td>
<td>$0.01 per GB</td>
</tr>
</tbody>
</table>
Amazon EC2, Amazon RDS and Amazon ElastiCache instances or Elastic Network Interfaces in another Availability Zone in the same AWS Region $0.01 per GB

<table>
<thead>
<tr>
<th>Data Transfer OUT From Amazon EC2 To</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amazon S3, Amazon Glacier, Amazon DynamoDB, Amazon SQS, Amazon SimpleDB in the same AWS Region</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Amazon EC2, Amazon RDS, or Amazon ElastiCache instances, Amazon Elastic Load Balancing, or Elastic Network Interfaces in the same Availability Zone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Using a private IP address</td>
</tr>
<tr>
<td>Using a public or Elastic IP address</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Amazon EC2, Amazon RDS or Amazon ElastiCache instances, Amazon Elastic Load Balancing, or Elastic Network Interfaces in another Availability Zone in the same AWS Region</th>
</tr>
</thead>
<tbody>
<tr>
<td>Another AWS Region or Amazon CloudFront</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data Transfer OUT From Amazon EC2 To Internet</th>
</tr>
</thead>
<tbody>
<tr>
<td>First 1 GB / month</td>
</tr>
<tr>
<td>Rate Tiers</td>
</tr>
<tr>
<td>----------------------------------</td>
</tr>
<tr>
<td>Up to 10 TB / month</td>
</tr>
<tr>
<td>Next 40 TB / month</td>
</tr>
<tr>
<td>Next 100 TB / month</td>
</tr>
<tr>
<td>Next 350 TB / month</td>
</tr>
<tr>
<td>Next 524 TB / month</td>
</tr>
<tr>
<td>Greater than 5 PB / month</td>
</tr>
</tbody>
</table>

Table H.1 Amazon Pricing

Rate tiers take into account your aggregate usage for Data Transfer Out to the Internet across Amazon EC2, Amazon S3, Amazon Glacier, Amazon RDS, Amazon
SimpleDB, Amazon SQS, Amazon SNS, Amazon DynamoDB, and AWS Storage Gateway.

Data transferred "in" to and "out" of Amazon Elastic Load Balancing is priced equivalent to Amazon EC2, except that Data Transfer OUT from Amazon Elastic Load Balancing in EC2 to another AWS Region or Amazon CloudFront is priced at "Internet" rates. Data transferred OUT from Amazon Elastic Load Balancing in Amazon VPC to another AWS Region or Amazon CloudFront is priced at Inter-region rates.
Appendix I: Source Code

/*
* Ted Garcia
* Computer Science Graduate Program Spring 2013
* Thesis titled:
* ANALYSIS OF BIG DATA TECHNOLOGIES AND METHODS:
* QUERY LARGE WEB PUBLIC RDF DATASETS ON AMAZON CLOUD USING HADOOP AND
OPEN SOURCE PARSERS
* California State University, Northridge (CSUN)
* Completed: February 5, 2013
* Copyright owned by Ted Garcia and CSUN, any use requires citation or permission
* No warranties or guarantees are given
*
* Hadoop program that runs Any23, NX Parser, and Jena Riot on raw NQuad RDF files.
* Tested on Amazon's EC2, S3, and Elastic MapReduce
* Two primary features are:
* 1. Count occurrences of words in the subject, predicate, object, or context
* 2. Count occurrences of a query string
*/

/******** Common Imports ********/
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Iterator;
import java.util.Date;
import java.io.ByteArrayInputStream;
import java.io.InputStream;
import java.io.StringReader;

/******** Hadoop *********/
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapred.*;
import org.apache.hadoop.util.*;

/******** START Any23 Imports *********/
import java.util.ArrayList;
import org.apache.any23.io.nquads.NQuadsParser;
import org.openrdf.model.Statement;
import org.openrdf.rio.ParserConfig;
import org.openrdf.rio.RDFHandlerException;
import org.openrdf.rio.RDFParseException;
import org.openrdf.rio.RDFParser;
import org.openrdf.rio.binary.BinaryRDFWriter;
import org.openrdf.rio.helpers.RDFHandlerBase;
import org.openrdf.rio.helpers.StatementCollector;

/******** END Any23 Imports *********/

/******** START NxParser Imports *********/
public class RDFParseParallelMain {

    private static InputStream is = null;
    private static FileInputStream in = null;
    private static PrintWriter out = null;
    private static FileOutputStream outs = null;
    private static String sInFileName = "";
    private static String sOutFileName = "";
    private static Parser ParserSelected = Parser.NONE;
    private static String sParserSelectedString = null;
    private static NodeItem NodeItemSelected = NodeItem.ALL;

    public static void main(String[] args) {
        // 1. Get arguments:
        // (required) input filename
        // (required) output filename
        // (required) parser to run -p - A=Any23, N=NxParser, R=RiotARQ
        // (optional) node item, which part of the RDF -n - S=subject, P=predicate, O=object, C=context
        // (optional) query string -q
        // 2. Open files - not Hadoop
        // 3. Setup parser environment
        // 4. Process query output
        //
        // Design Notes - Key Features
        // 1. Count entries for each node type (subject, predicate, object, context)
        // key = node type; value = count
        // 2. Count occurrences of query string in node type
        // key = query string; value = count
        // */

    }

}
private static int NodeItemSelectedNo = 3; // default to context
private static String sQueryString = "";
private static int sQueryCount = 0;
private static int iLineCount = 0;
private static String[] sNodes; // 0-subject, 1-predicate, 2-object, 3-context
private static boolean DEBUG = false;

private enum Parser {
    ANY23, NXP, RIOT, NONE
}

private enum NodeItem {
    // ALL is the default
    ALL, SUBJECT, PREDICATE, OBJECT, CONTEXT
}

public static class Map extends MapReduceBase implements Mapper<LongWritable, Text, Text, IntWritable> {
    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();
    private static OutputCollector o = null;

    private static String sParserSelectedString = "N"; // default to NX Parser
    private static int NodeItemSelectedNo = 3; // default to context

    public void map(LongWritable key, Text value, OutputCollector<Text, IntWritable> output, Reporter reporter) throws IOException {
        String line = value.toString();
        sNodes = new String[4];
        o = output;
        is = new ByteArrayInputStream(line.getBytes()); // convert String into InputStream

        // Determine which parser to use and then run it
        // **************** Any23 Parser ****************/
        if (sParserSelectedString.equalsIgnoreCase("A")) { // Any23 Parser
            RDFParser rdfParser = new NQuadsParser();
            StatementCounter sc = new StatementCounter();
            rdfParser.setRDFHandler(sc);
            try {
                rdfParser.parse(is, "rdfParser.parse-filenameStringIsEmpty");
            } catch (IOException e) {
                // handle IO problems (e.g. the file could not be read)
                System.err.println("Map: Any23: IO Problems");
                e.printStackTrace();
            } catch (RDFParseException e) {
                // handle unrecoverable parse error
                System.err.println("Map: Any23: Unrecoverable Parse Error");
                e.printStackTrace();
            } catch (RDFHandlerException e) {
                // handle a problem encountered by the RDFHandler
                System.err.println("Map: Any23: RDFHandler Problem");
                e.printStackTrace();
            }
        } // **************** Nx Parser ****************/
    }
}

76
else if (sParserSelectedString.equalsIgnoreCase("N")) { // Nx Parser
    NxParser nxp = new NxParser(is);
    Text mapWord = new Text();
    while (nxp.hasNext()) {
        Node[] ns = nxp.next();
        if (ns.length == 4) {
            // sNodes = 0-subject, 1-predicate, 2-object, 3-context
            sNodes[0] = ns[0].toString();
            sNodes[1] = ns[1].toString();
            sNodes[2] = ns[2].toString();
            sNodes[3] = ns[3].toString();
            // Only do one node type
            mapWord.set(sNodes[NodeItemSelectedNo]);
            try {
                output.collect(mapWord, one);
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
} // **************** RIOT Parser ****************
else if (sParserSelectedString.equalsIgnoreCase("R")) { // RIOT Parser
    SinkQuadStats sink = new SinkQuadStats(new SinkNull<Quad>());
    RiotReader.parseQuads(is, Lang.NQUADS, null, sink);
} // end else if
} // end of map method

@Override
public void configure(JobConf job) {
    sParserSelectedString = job.get("sParserSelectedString");
    NodeItemSelectedNo = job.getInt("NodeItemSelectedNo", 3);
}

public static void SendToReduce(String[] psNodes) {
    Text mapWord = new Text();
    mapWord.set(psNodes[NodeItemSelectedNo]);
    try {
        o.collect(mapWord, one);
    } catch (IOException e) {
        e.printStackTrace();
    }
} // end Map class

public static class Reduce extends MapReduceBase implements Reducer<Text, IntWritable, Text, IntWritable> {
    public void reduce(Text key, Iterator<IntWritable> values, OutputCollector<Text, IntWritable> output, Reporter reporter) throws IOException {
        int sum = 0;
        while (values.hasNext()) {
            sum += values.next().get();
        }
        output.collect(key, new IntWritable(sum));
    }
} // end Reduce class

77
public static void main(String[] args) throws Exception {
    int i = 0, j;
    String arg;
    char flag;
    sNodes = new String[4];

    //*********** PROCESS ARGUMENTS ************
    //Usage: RDFParseParallel infilename outfilename -p A|N|R [-n S|P|O|C] [-q querystring]
    //Example: RDFParseParallel Myinfilename Myoutfilename -p A -n S -q Myquerystring
    if (i < args.length)
        sInFileName = args[i++];
    if (i <= args.length)
        sOutFileName = args[i++];
    else if (i > args.length) printUsage();

    while (i < args.length && args[i].startsWith("-")) {
        arg = args[i++];
        flag = arg.charAt(1);
        if (DEBUG) System.out.println("arg: "+arg);
        switch (flag) {
            case 'p':
                if (i < args.length) {
                    sParserSelectedString = args[i];
                    char pArg = args[i++].charAt(0);
                    switch (pArg) {
                        case 'A':
                            ParserSelected = Parser.ANY23;
                            break;
                        case 'N':
                            ParserSelected = Parser.NXP;
                            break;
                        case 'R':
                            ParserSelected = Parser.RIOT;
                            break;
                        default:
                            System.err.println("RDFParseParallel: illegal -p value " + args[i-1] + ";");
                            printUsageWithExit();
                            break;
                    }
                } else {
                    System.err.println("RDFParseParallel: -p needs value");
                    printUsageWithExit();
                }
                break;
            case 'n':
                if (i < args.length) {
                    char pArg = args[i++].charAt(0);
                    // sNodes = 0-subject, 1-predicate, 2-object, 3-context
                    // sNodes = 0-subject, 1-predicate, 2-object, 3-context
                } else {
                    System.err.println("RDFParseParallel: -n needs value");
                    printUsageWithExit();
                }
                break;
        }
    }
}
switch (pArg) {
    case 'S':
        NodeItemSelected = NodeItem.SUBJECT;
        NodeItemSelectedNo = 0;
        break;
    case 'P':
        NodeItemSelected = NodeItem.PREDICATE;
        NodeItemSelectedNo = 1;
        break;
    case 'O':
        NodeItemSelected = NodeItem.OBJECT;
        NodeItemSelectedNo = 2;
        break;
    case 'C':
        NodeItemSelected = NodeItem.CONTEXT;
        NodeItemSelectedNo = 3;
        break;
    default:
        default:
            System.err.println("RDFParseParallel: illegal -n value " + args[i-1] +""");
            printUsageWithExit();
            break;
        }
        else {
            System.err.println("RDFParseParallel: -n needs value");
            printUsageWithExit();
        }
        break;
    case 'q':
        if (i < args.length)
            sQueryString = args[i++];
        else {
            System.err.println("RDFParseParallel: illegal -n value " + sQueryString+""");
            printUsageWithExit();
        }
        break;
    default:
        default:
            System.err.println("ParseCmdLine: illegal option " + flag);
            printUsageWithExit();
            break;
}
} // end while

if (DEBUG) printUsage();

/**************************** HADOOP CODE ****************************/

JobConf conf = new JobConf(RDFParseParallelMain.class);
conf.setJobName("RDFParseParallel: ParserSelected: " +ParserSelected.toString()+"NodeItemSelected: " +NodeItemSelected.toString());
conf.setOutputKeyClass(Text.class);
conf.setOutputValueClass(IntWritable.class);
conf.setMapperClass(Map.class);
conf.setCombinerClass(Reduce.class);
conf.setReducerClass(Reduce.class);

conf.setInputFormat(TextInputFormat.class);
conf.setOutputFormat(TextOutputFormat.class);

FileInputFormat.setInputPaths(conf, new Path(sInFileName));
FileOutputFormat.setOutputPath(conf, new Path(sOutFileName));

conf.set("sParserSelectedString", sParserSelectedString);
conf.setInt("NodeItemSelectedNo", NodeItemSelected);

JobClient.runJob(conf);
} // End of Main

private static void printUsage() {
    System.err.println("RDFParseParallelMain Hadoop Program");
    System.err.println("Usage: RDFParseParallel -p A|N|R [-n S|P|O|C] -q querystring infilename outfilename");
    System.err.println("input filename (required)");
    System.err.println("output filename (required)");
    System.err.println("parser to run (required) A=Any23, N=NxParser, R=RiotARQ");
    System.err.println("query string (optional)");
    System.err.println("node item, which part of the RDF (optional) - S=subject, P=predicate, O=object, C=context");
    System.err.println("sInFileName: "+sInFileName);
    System.err.println("sOutFileName: "+sOutFileName);
    System.err.println("ParserSelected: "+ParserSelected);
    System.err.println("NodeItemSelected: "+NodeItemSelected);
    System.err.println("NodeItemSelectedNo: "+NodeItemSelected);
    System.err.println("sQueryString: "+sQueryString);
}

private static void printUsageWithExit() {
    printUsage();
    System.exit(-1);
}

static void printNodes(String[] psNodes) {
    iLineCount++;
}

static void countQuery(String[] psNodes) {
    String s = psNodes[0]+" "+psNodes[1]+" "+psNodes[2]+" "+psNodes[3];
    if(s.toLowerCase().contains(sQueryString.toLowerCase())) sQueryCount++;
}

private static void printCurrentDate() {
    Date date = new Date();
    System.out.println(date.toString());
}
System.out.println("LineCount: "+iLineCount);
}
} // End of RDFParseParallelMain class

class StatementCounter extends RDFHandlerBase {
    private int countedStatements = 0;

    @Override
    public void handleStatement(Statement stm) {
        // localNodes = 0-subject, 1-predicate, 2-object, 3-context
        String[] localsNodes = new String[4];
        localsNodes[0] = stm.getSubject().toString();
        localsNodes[1] = stm.getPredicate().toString();
        localsNodes[2] = stm.getObject().toString();
        localsNodes[3] = stm.getContext().toString();
        RDFParseParallelMain.Map.SendToReduce(localsNodes);
        countedStatements++;
    }

    public int getCountedStatements() {
        return countedStatements;
    }
}

class SinkQuadStats extends SinkWrapper<Quad> {
    private long count = 0;

    public SinkQuadStats(Sink<Quad> output) {
        super(output);
    }

    public SinkQuadStats() {
        super(new SinkNull<Quad>());
    }

    @Override
    public void send(Quad quad) {
        // localNodes = 0-subject, 1-predicate, 2-object, 3-context
        com.hp.hpl.jena.graph.Node p = quad.getPredicate(); // Predicate
        String[] localsNodes = new String[4];
        localsNodes[0] = com.hp.hpl.jena.graph.Node.createURI(p.getNameSpace()).toString();
        localsNodes[1] = quad.getPredicate().toString();
        localsNodes[2] = quad.getObject().toString();
        localsNodes[3] = quad.getGraph().toString(); // Context
        RDFParseParallelMain.Map.SendToReduce(localsNodes);
        super.send(quad);
    }
}
} // end of SinkQuadStats class