PROJECT 2: DATA-STRUCTURES AND ALGORITHMS ENABLING DATA-INTENSIVE COMPUTING

Purpose:

1. To understand the data-structures and algorithms for **data-intensive computing**
2. To design and implement the solutions for a data-intensive problem using MapReduce and Hadoop DFS
3. Understand **virtualization** and deploy a single data-node **HDFS** using **VMware player** or any other virtual machine technology such as Oracle's VirtualBox or Microsoft's Virtual PC.
4. Implement **MapReduce** algorithm for a sample application using single node HDFS as described above and also on the workflow provided
5. Study and compare the scalability of MapReduce programming model versus the traditional sequential and thread models.
6. **(optional)**To explore designing and implementing data-intensive computing solutions on a **cloud environment**: in this case on **Amazon Compute Cloud** (EC2)[1]. (In project 1 we explored Google App Engine (GAE)[2]).

Problem Statement:

Project 1 focused on content accumulation methods and the three tier web application (or regular application) infrastructure to enable data-intensive applications. This project will focus on parallel processing including by *mapreducing* ultra-scale data. MapReduce is a framework for processing large scale data sets. It exploits the write-once-read-many (WORM) characteristic typical of unstructured data to represent and process (i) as key-value pairs <key,value> and (ii) using a suite of operations such as map, partition, reduce, shuffle etc. A special data repository (like a file system used in a traditional operating system) is needed to efficiently and reliably store and deliver the large data set. Google file system (GFS)[4] is such a system, and Hadoop Distributed File System (HDFS)[5] is another GFS-like system available under Apache open source license.

Preparation before lab:

1. Read the chapters 1-3 in the Lin and Dryer text [6].
2. Review the foundations of MapReduce and Hadoop Distributed File System [3], [4], [5].

Assignment:

1. (15 points) Download and install a single-node (one master and one slave) on your laptop. Install the latest version of the MR framework and implement a solution for the *wordcount* problem. You may use any existing solutions. For data use any of the books from Project Gutenberg free ebook source (http://www.gutenberg.org/). Evaluate the performance of your wordcount program for scalability for various sizes of data and record the performance in time taken. Prepare a chart for size vs performance.

2. Design the solution for the word co-occurrence matrix problem described in Chapter 3 of Lin and Dryer's text [6]. Specifically study the problem described in section 3.2. **Implement the "pairs" approach as well as the "stripes" approach for any given corpora** of data from Project Gutenberg. Choose the data appropriately: for example, Shakespeare's works may **not** be a good selection since it contains very old usage of words and may not yield good co-occurrence.
   2.1. The context for co-occurrence can be a sentence or a paragraph. In the case of corpora with large number of document, even the document can be the context. We will choose "paragraph" as ***co-occurrence context***.
   2.2. Instead of counts of co-occurrence, you will store the ***relative frequencies*** of the co-occurrences as discussed in equation 3.1
   2.3. Compare the run-time for various sizes of data for the pairs and stripes implementations.
   2.4. Draw the graphs for the performance of the various approaches.

System Architecture

The system architecture for the assignment given is shown in Figure 1. In Figure 1, the development environment is shown on the left, that is a single-node (or multi-node) HDFS deployed in your local environment. For the second part of the assignment you will work with the same problem as above but in the ***production environment*** shown on the right. You will transfer the data for the two applications into 2 buckets of the amazon simple storage service (s3) and upload the map and reduce functions designed in the part1. You will then create a MR workflow to process the data. (working with amazon is optional)

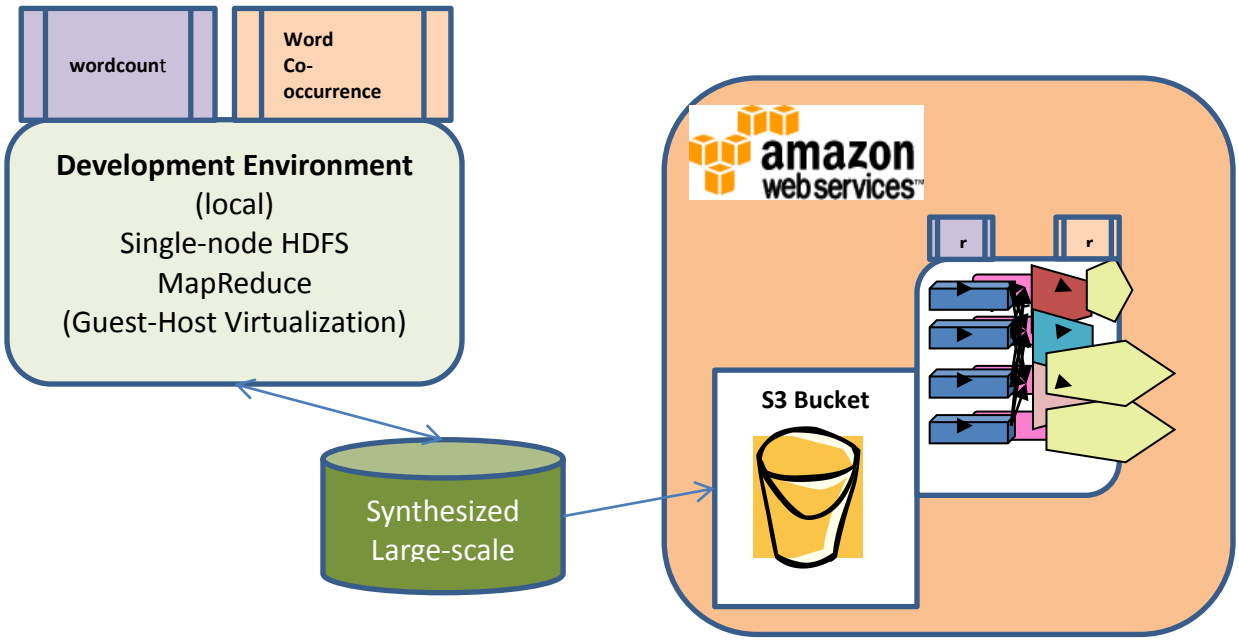

Figure 1: System Architecture for Data-intensive HDFS-MapReduce Application Development

Project Deliverables:

1. A tar file of the single node implementation of the MR-wordcount including the data; a tar file of the single node implementation of the co-occurrence for pairs approach and another for the stripes approach. (in all, three jar/tar files).
2. An experience report providing all the details of the project design, implementation and the performance evaluation report. This should have the user's manual, programmer's manual and any design diagrams.
3. You may have to demo the project to the TA.
4. Grading: 20% for the wordcount with performance evaluation report, 40% each for the pairs and stripes with performance evaluation and report.

Submission Details:
submit_cse487 files separated by space
submit_cse587 files separated by space

References:

1. Amazon Elastic Compute Cloud. http://aws.amazon.com/ec2/, last viewed October 31, 2011.
2. Google App Engine (GAE). http://code.google.com/appengine/, last viewed October 31, 2011.
3. MapReduce on Amazon. Advanced MapReduce Features. http://developer.yahoo.com/hadoop/tutorial/module5.html, last viewed October 31,2011.
4. Dean, J. and Ghemawat, S. 2008. **MapReduce: simplified data processing on large clusters**. *Communication of ACM* 51, 1 (Jan. 2008), 107-113.
5. Hadoop Distributed File System (HDFS). Apache Hadoop: http://hadoop.apache.org, last viewed September, 2010.
6. Lin, J and Dryer, C. Data-Intensive Text Processing with MapReduce 2010, Vol. 3, No. 1, Pages 1-177, (doi:10.2200/S00274ED1V01Y201006HLT007). **An online version of this text** is also available through UB Libraries since UB subscribes to Morgan and Claypool Publishers.