

Web Graph Generation: Fall 2007 Report

Anthony Jawad, Jie Teng
 Department of Computer Science
 Cornell University
 Ithaca, NY 14853

I. INTRODUCTION

Nowadays, the Web has become a social phenomenon (e.g., political campaigns and online retailing) and evidence for the development and evolution of the world (e.g. development of legal concepts across time). There are social science research conducted on the Web. The Web Laboratory [1] is a joint project of Cornell University and the Internet Archive to provide data and computing tools for research about the Web and the information on the Web. The Web Laboratory project comprises several small projects, such as Web Graph Generation and PageRank Calculation.

Our job is to generate the Web Graph, which is the linking structure of the Web and represented by its adjacency matrix using a compressed sparse row representation. With the web graphs, researchers can see the structure and evolution of the Web. They can also be used for PageRank calculation, social network research and etc. Our work builds upon the previous work done in Spring 2007 by Sangwoo Kim, Sanjay Rajan, and Sean Seguin[2]. They began to develop tools for generating a sparse matrix representation of an arbitrary subset of the web graph that is suitable for further computation, such as the calculation of the PageRank as an estimate of the importance of a particular web resource [3].

Our goals for the semester have been to continue development of the tools, ensure their correctness, and access and improve the scalability of the process to arbitrarily large data sets. In this report we shall summarize the process and our findings about its scalability.

The rest of the report is organized as follows. In Section II, we formulate the problem. Section III introduces the *Hadoop* software framework [4] and cluster environment that our work is based on. In Section IV, we describe each step to generate the web graph, discuss the issues/limitations and our solutions. The results of experimental evaluations are presented in Section V. Detailed running instructions are described in Section VI.

II. PROBLEM STATEMENT

The web graph is a directed graph in which each node is a single web page and each edge is a hyperlink from one document to another. Each node is identified by a *Uniform Resource Locator*, or URL. Each edge is represented by an ordered pair of URLs whose first component—the fromURL—identifies the page containing the link and whose second component—the toURL—identifies the page to which the link is targeted.

The input used to generate our graph is a set of hyperlinks. In practice, this is the set of hyperlinks extracted from the pages from a web crawl. This necessitates the following two considerations.

First, note that there is a many-to-one relationship between URLs and pages; that is, two syntactically distinct URLs may identify the same page. As such, in the process of generating our graph representation, we must *canonicalize* these URLs so that each resource is identified by at most a single URL.

Second, because crawling the entire web is practically impossible, it is invariably true that hyperlinks may be targeted at pages that are not included within that crawl. We desire that only pages that have been crawled represent nodes in our graph. Note that our only evidence of a page having been crawled is the appearance of a URL identifying it as a from-link within our input. Thus, we exclude from our graph any links whose toURL identifies a page that has not been crawled yet. This also means that any “dangling” pages—pages with no out-links—are also excluded from our graph. Refer to Page et al. §2.7 for a justification of the soundness of this practice[3].

With these issues and solutions in mind, we formulate the problem of generating the web graph below.

Notation

- (u', v') : an URL pair where u' and v' are two URLs in uncanonicalized form and the page identified by u' contains an outgoing link to v' . u' is called the fromURL, and v' is called the toURL.

- u, v : canonicalized forms of u' , v' respectively.
- i, j : integer indices assigned to nodes.
- (i, j) : an edge linking from node i to node j , where the nodes are represented by their indices.

Problem

Given a set of URL pairs in uncanonicalized form (u', v')

- 1) Canonicalize every u' and v' .
- 2) Create a list of all nodes of the graph, i.e., the set of all unique u . And create a list of all edges, i.e., discard all duplicates and all (u, v) pairs where v is not a node of the graph.
- 3) Create an index list for the nodes, (i, u) , where the i are consecutive integers.
- 4) For each node, with index i , create a list of the edges from that node, (i, j) , where j is the set of nodes that have edges from node i .
- 5) Sort the set of (i, j) by i and then by j , and output them as the final output.

III. COMPUTING ENVIRONMENT

Our work is based on an open-source software framework called *Hadoop*[4]. Since the needs of processing of a huge data set in a time-efficient way exceeds the ability of a single generic computer, we need a distributed environment with programming and runtime support that allows a network service to be programmed quickly for execution on a cluster with very large amounts of data and high-volume user traffic while shielding application programmers from the complexities of replication, service discovery, failure detection and recovery, load balancing, resource monitoring, and management. *Hadoop* is for running applications on large clusters built of commodity hardware and transparently provides applications both reliability and data motion. It combines a computational paradigm named Map/Reduce [5] and a distributed file system *HDFS*[6].

Map/Reduce divides an application into many small blocks of work, each of which may be executed or reexecuted on any node in the cluster [4]. Users specify a *map* function that processes a key/value pair to generate a set of intermediate key/value pairs, and a *reduce* function that merges all intermediate values associated with the same intermediate key. The number of map tasks and reduce tasks are also specified by users. Each map task will generate as many output files as there are reduce tasks. Each output file will be targeted at a specific reduce task and the map output pairs from all the map tasks will be routed so that all pairs for a given key end up in files targeted at a specific reduce task. *HDFS*

stores data on the compute nodes, providing very high aggregate bandwidth across the cluster. It creates multiple replicas of data blocks for reliability, placing them on compute nodes around the cluster for MapReduce to process later. Both Map/Reduce and *HDFS* are designed so that node failures are automatically handled by the framework.

All jobs utilizing Hadoop are run on the *v2linux* cluster at the Cornell Center For Advanced Computing (CAC). *v2linux* contains 100 nodes, each of which is configured with Dual 2.4GHz P4 Xeon Processor, 2 GB RAM, 72GB Hard Drive, 512KB Cache/Processor (SMP), and Force10 Gigabit Ethernet. We use the same architecture (shown in Fig.1) used in the previous work done by Sangwoo Kim, Sanjay Rajan, and Sean Seguin [2]. The version of Hadoop we use is 0.9.2, which runs in Java Runtime Environment (JRE) 1.5.0.11.

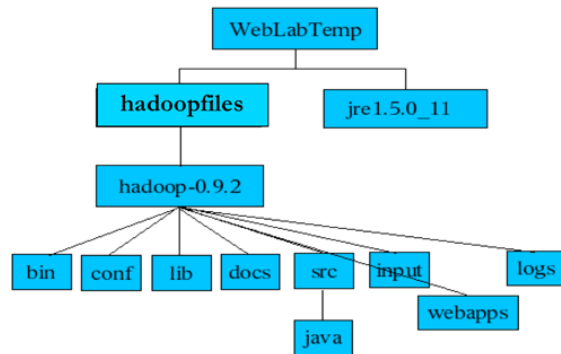


Fig. 1. System Architecture [2]

IV. PROCESS

In this section, we describe each step to generate the web graph. The whole problem is divided into 4 parts, so the whole process contains 4 steps accordingly. In each step, we also discuss the issues/limitations and propose our solutions. The overall flow of the process is shown in Fig.2.

Next, we will present each step in detail.

Step 1

The input for this step is a ASCII encoded tab-separated-value (TSV) file, where each line is a hyperlink tuple, expressed in the form below.

fromURL < TAB > toURL

Given this input, Step 1 performs the following two tasks.

- 1) Canonicalize each URL pair; that is, canonicalize each URL in the pair.

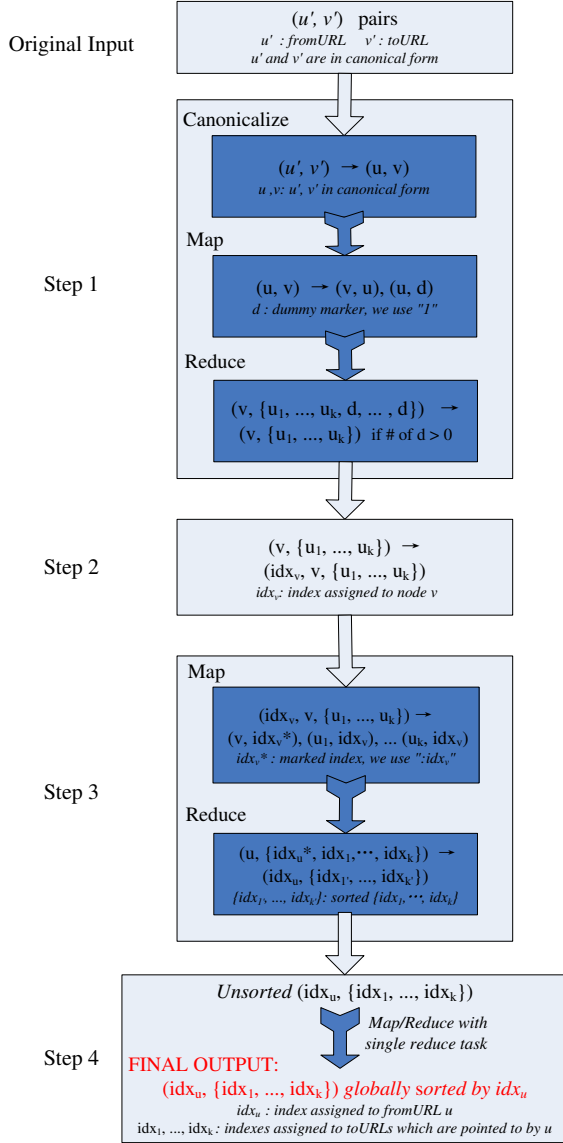


Fig. 2. Overview of Steps

- 2) Create a list of all nodes of the graph with the set all edges *to* each. As discussed earlier, all (u, v) pairs such that v does not identify a node of the graph are removed. Note, however, the duplicates links are retained, but are removed in Step 3.

We utilize Hadoop to construct a MapReduce for this step.

The mapper performs *map* tasks which first canonicalize each pair of URLs. Canonicalized *fromURL* and *toURL* are passed to the mapper output collector in the form of $(\text{toURL}, \text{fromURL})$ and $(\text{fromURL}, \text{Marker})$, where *Marker* is a dummy marker to indicate that a URL has appeared as a fromURL (that is, to indicate

that that page identified by the respective fromURL has been crawled and is in the data set of interest). We use the string ‘1’ as the marker. Conversely, if a page identified by a particular URL does *not* have a marker in the *valuelist* corresponding to its URL as the *key*, this page does not appear as a fromURL (i.e., it is not in the data set, it has not been crawled, or it is a “dangling” node) and thus should be excluded from the web graph.

The reducer takes the output from the mapper, and groups together the $(\text{key}, \text{value})$ pairs which have the same *key*. We examine whether a marker exists in the grouped *valuelist* (which comprises a list of URLs) for a *key* (which is a URL) and pass the $(\text{key}, \text{valuelist})$ to the reducer output collector if a marker exists, otherwise, this $(\text{key}, \text{valuelist})$ is not collected because the page represented by the key URL is not a valid node.

A point of complication is that the Hadoop library does not specify an order in which the items for a particular key are returned. As such, the dummy marker may appear at any time. The general solution to this problem that the team last semester decided upon was to store each URL seen into some data structure, and then emit each item in the list once the marker had been seen. If the data was not seen, then the data in the structure could be thrown away.

Last semester’s team presented three options for this data structure.

- In-memory linked list
 The team argues that such an implementation will offer poor performance and affect scalability:
 - A linked list may be relatively CPU-intensive.
 - A linked list may be wasteful with regard to memory usage.
 - A linked list must be traversed again once a marker has been seen.

The team does not report any trials in support of their arguments against the use of a linked list, however.

- In-memory string
 The team proposed concatenating each URL seen into a single string. They found that performance became rather poor as the size of the string increased. However, the team was unclear about how, precisely, they carried out their experiment. In particular, there are two classes of approaches that could be used to accumulate each seen fromURL, with one being significantly more efficient than the other.
 - *String*
 Note that in Java, any *String* is immutable. As such, each intermediate result will require

the allocation of a new String to store it, and each concatenation will require that the value of the previous string be copied in memory. This process is arguably relatively slow and wasteful of resources, in a manner consistent with the performance that they describe having gotten.

– *StringBuffer*

A *StringBuffer* is a mutable data structure whose size may grow as needed. As such, appending a string to this data structure will not require additional copies of it to be made, so its use is much more suitable for scalability.

• On-disk file

The approach the team ultimately chose was to write each URL in the collection to a disk file.

There are, however, arguments against writing to disk that the team has not addressed. First, disk access is many orders of magnitude slower than normal memory access. As such, there is incentive to use an in-memory approach when possible. Second, using files for each reduce task potentially puts an upper-bound on the number of simultaneous reduce tasks that may be running on a given machine, due to the existence of a limit on the number of file handles for a particular instance of the Java Virtual Machine.

Nevertheless, the team argues that using disk is best the best option for scalability. Once the Web Lab cluster is operational, we recommend performing trials to determine which method performs best on the newer machines.

We shall now address a few of the challenges encountered with Step One.

1) *I/O collision*: One issue that the team anticipated with their approach to this temporary storage was I/O collision. They implemented this solution using a fixed, uniform file name for each reducer instance; thus, when running multiple reduce threads on a single machine, one reduce task could overwrite the data produced by another.

This finally turned out to be a concern for correctness, as it introduced a race condition that lead to indeterminate output. One thread could create a file and begin writing to it, but another thread could then overwrite that file with its own data. The only conditions upon which the program would produce the correct result would be when the threads were executed serially.

Our solution to the problem was to use a randomly generated filename using three random numbers between 1 and 10000 as part of the file name. We chose this solution for evaluation due to the simplicity of implementation. A concern with this approach, of course, is

that although the probability for I/O collision becomes negligible, it is not statistically zero. There are, of course, deterministic (though nontrivial) ways to generate unique file names such as by a certain encoding, which should be preferred as we move to larger data sets.

2) *Trailing tabs*: The team from also anticipated a potential issue resulting from their placing a tab after each output line in the temporary file. Trailing tabs are proven to affect the correctness of Step 3 (see IV-.2). As such, we have removed the trailing tabs to conform more rigidly to the output specification, possibly preventing other potential adverse effects as well.

Step 2

In the second step, we assign each page in our graph a unique integer identifier. The identifiers are the consecutive natural numbers. Because this process is not computationally intensive and the need for consecutive indices is necessary, this step is performed serially, on a single machine.

Recall that each line of output from Step 1 corresponds to a unique URL identifying a single page, followed by a nonempty set of URLs identifying pages that link to it. This program thus iterates over each output line, assigning a unique identifier to the line (and thus the toURL).

The result of Step 2 is the input with the unique index for each toURL prepended to its line.

Step 3

This step takes the output files from Step 2 as its input. Each line in the input contains a $(i, toURL, \{fromURL\})$ tuple, where the i is the index for $toURL$ and $\{fromURL\}$ is the nonempty set of fromURLs for the $toURL$. For each node i , Step 3 outputs a list of edge set (i, j) , which represents edges from node i to the set of nodes j where j is a node index. j is sorted in ascending order and has duplicates removed, however, the (i, j) tuples may be in random order of i .

Because it is not straightforward to find the index for a fromURL, we use an index marker function. The mapper takes $(i, toURL, fromURL)$ passed to the mapper output collector in the form of $(toURL, i^*)$ and $(fromURL, i)$, where i^* is the marker function that retains the value of i but marks it as special (we use "-i" as i^*). Since reducer will group all of the pairs with the same key value, we therefore can obtain a fromURL along with its index which is marked and the indices for the nodes it points. The reducer restores the marked indices to their original form, collect $(i, tab-separated string which concatenates all values in the j)$, and output them to the output files.

Our code is based on the previous code [2], which, however, is only able to take tens of $(i, toURL, fromURL)$ tuples as input. We find the cause of this limitation is because the tab-separated string passed to the reducer output collector has an extra the "trailing tab" at the end. After fixing this, our code scales well on the input data.

Note that the output of this step is unsorted because the output of the reducer is not re-sorted while the output of the mapper is sorted by the reducer. A reducer has 3 primary phases: *shuffle*, *sort* and *reduce*. The shuffle phase fetches the relevant partition of the output of all the mappers via HTTP and the sort phase groups inputs by keys, and the shuffle and sort phases occur simultaneously, i.e. while outputs are being fetched they are merged. The reduce phase eliminates the duplicates in the $(list\ of\ values)$ of each $(key, (list\ of\ values))$ pair in the grouped inputs and outputs the results, however, the output is not re-sorted [7]. Therefore, we need Step 4 to sort the output to a global order.

Step 4

This step utilizes the automatic sorting feature of the reducer to sort the $(index\ of\ toURL, a\ sorted\ list\ of\ indices\ of\ fromURLs)$ tuples, which are taken from the Step 3 output and are in random order, into a single final output file sorted by the *index of toURL*. The mapper reads in the tuples and passes them to the reducer. The reducer then sorts them, and writes them to the final output file. To obtain this global sorting order, we have to set the number of reduce tasks in Hadoop to 1.

V. EXPERIMENTAL RESULTS

Here, we shall discuss a series of tests that we performed to access how well the process can scale with more computational power. Among the 4 steps, Step 1 is the most computationally intensive step of the process, which needed hours to complete computation for our test set while the other steps only took less than 15 minutes. The huge difference is caused by the fact that the input data for step 3 and step 4 was significantly smaller than that of step 1 which we will discuss below, and step 2 is only a simple file read/write processing. Thus we will only focus on Step 1, which is the bottleneck of the whole process.

In particular, we were interested in estimating the correlation between how the time needed to complete the job is dependent upon the number of nodes available. We chose a fixed subset of links from the Amazon link database crawled in 2005. There are 1,486,741,847

links (approximately 200GB) in the entire corpus of links that we currently have available.

We first took the first 1.459GB partition of this file, which consists of 10 million links, as our test set. We ran Step 1 on this data set using 3, 7, 10, and 15 nodes. We varied the amount of map and reduce tasks in total according to figure 4, in accordance with the recommendations given by the Hadoop documentation[4]. The approximate runtime for the job is also shown in Fig.4, and is plotted against the number of nodes used in Fig.3.

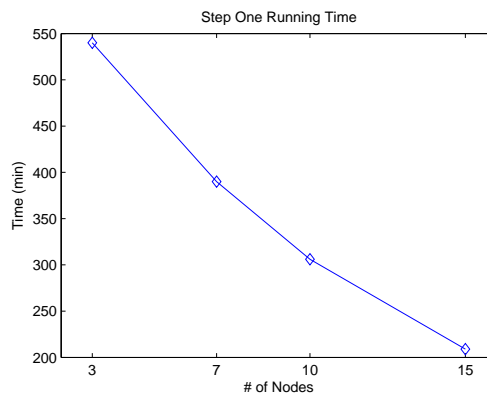


Fig. 3. Step One Running Time on the set of 10 million URL pairs

Machines used	Map tasks	Reduce tasks	Time (min)	Output (toURL,fromURL) tuples
3	7	3	540	12,762
7	19	7	390	12,687
10	31	11	306	12,809
15	31	13	209	12,735

Fig. 4. Configurations for Step One Experiments on the set of 10 million URL pairs

The results indicate that time decreases roughly linearly as the number of nodes increases, which suggests that Step 1 will be able to scale rather well.

We then took the first 14.59GB partition of this file, which consists of 100 million links, as our test set, and ran Step 1 on it using 10 nodes. The configuration was same as the 10 million set. From Fig.5, we can see the increase in number of output almost is proportional to the increase in the input data. However, with limited experiments, we can not see a clear relationship between the input data size and running time, which is our future work to explore.

All the 4 steps are chained; the output from each step is the input for the next step. Starting from the original data set, we successfully went through all the steps and obtained the desired final output. However, there are still

Data Set	Time (min)	Output (toURL,fromURL) tuples
10m	306	12,809
100m	2016	148,138

Fig. 5. Comparison between 10m and 100m link data sets

some concerns regarding our experiment that ought to be considered and addressed. We take the 10 million link set as an example and list these below.

Data dependence

Recall that our input link set was chosen as the first 1.459GB of link data from the database extracted hyperlink set. The important thing to note about this is that the links are not chosen at random, which may bring about particularly interesting phenomena. In particular, although the order in which links are extracted from the database is technically unspecified, observation indicates that the query’s result set is ordered by fromURL.

This could have a number a number of effects. First, this may bias the cache performance of the system when processing URLs. Because Strings in Java are immutable, the fromURL might hit in cache on the machine for each mapping task quite often compared to data sets in which the fromURL varied frequently. This might misleadingly improve performance.

Second, we should consider that because we are examining a relatively small set of links, the overwhelming majority of links will not be to pages in the closure. This is evident in the fact that from a starting set of 10 million links, there are only on the order of 10 thousand (toURL,fromURL) tuples (Interestingly, we later found that of these nodes, only a few were linked to by nodes other than themselves.). As a result, we can imagine that the irregularity of the data set might bias the performance of the system. In particular, consider that in most cases, the temporary file written by each reduce task will never be read, since the task will almost never see a dummy marker. Because file-system access is generally a relatively slow operation, the fact that the file does not need to be re-read in most cases may also misleadingly improve performance.

Limited number and variety of trials

The reader should take note that our claim is based only upon four data points, all using a very small number of machines. The reader should also note that we have been unable to provide an estimate of the correlation for the amount of time needed to run Step 1 on a varying number of input links for a given number of machines.

This is largely due to the difficulty in attaining computing nodes on the shared v2linux cluster. More thorough experiments should surely be carried out when the dedicated Web Lab cluster is operational, and we have greater control over the computing environment.

Indeterminate output

By far, our largest concern is that for a fixed input, the number of output nodes in our graph varies with the number of machines used, which is an unexpected and incorrect result. We suspect that this is the result of a race condition either within the reduce tasks or within the internals of Hadoop, but more thorough testing in a controlled environment is needed.

VI. CODE COMPILATION AND RUNNING

In this section, we will show step by step how to build and run each program. Programs for Step 1, 3 and 4 are written in Java and are called in shell scripts, and Step 2 is written in C++. Step 1, 3 and 4 have very similar scripts and running steps, so in the below, we only show the details for Step 1, which can be adjusted accordingly for Step 3 and 4. Steps 3 and 4 do not require the *Canonicalizer* class. Note that in Step 4, *mapred.reduce.tasks* must be set to 1 so that the output appears in only one sorted file.

Step 1

Compilation:

- 1) `javac -classpath hadoop-0.9.2.jar Canonicalizer.java StepOne.java`
- 2) `jar -cvf StepOne.jar Canonicalizer.class StepOne$MapClass.class StepOne.class StepOne$Reduce.class`
- 3) `mv StepOne.jar {PROJECT ROOT}/hadoopfiles/hadoop-0.9.2/`

Configuration:

- 1) Set up Hadoop environment variables by modifying *hadoop-env.sh* file in *hadoopfiles/hadoop-0.9.2/conf*. Make sure JRE home directory *JAVA_HOME*, Hadoop home directory *HADOOP_HOME* and Hadoop configuration directory *HADOOP_CONF_DIR* are set up correctly.
- 2) Check *JobStep1.sh* and *Hadoop_Step1.sh* (or *Hadoop_Step1_local.sh*), and make sure the environment variables used in them are consistent with those in *hadoop-env.sh*. Input and output directories, running jar file are also specified in the two scripts.

- 3) Set up number of cluster nodes, cluster affiliation and job running time by changing the *nodes*, *minutes*, and *affiliation* fields in *MyJob1.xml*
- 4) Modify *hadoop-site.xml* file in *hadoopfiles/hadoop-0.9.2/conf*, and *mapred.map.tasks* and *mapred.reduce.tasks* to proper values
- 5) Put input files into *hadoopfiles/hadoop-0.9.2/\$input*

Running: Use "vsched -s MyJob1.xml" to run on a distributed cluster, and "Hadoop_Step1_local.sh" to run locally.

Step 2

Compilation: Compilation of Step Two requires the GNU G++ compiler built upon GCC. The following was used to compile the source on any of the CTC Linux machines. Note that the value of the 'march' switch will depend on the target architecture.

```
g++ -O3 -march=pentium4 StepTwo.cpp -o StepTwo
```

Configuration: Step Two requires in the input directory a text file listing each input file (that is, each file outputted by Step 1). Such a file can be produced by inserting the output of 'ls {INPUT_DIR}' into a text file called 'InputFiles.txt'.

Running: The script accepts as input a directory in which to find its input files and a text file containing a list of files to process (InputFiles.txt, from above). Assuming that the StepTwo executable is executable for the user and on the user's PATH, the program can be run as follows:

```
StepTwo {INPUT_DIR} {LIST_FILENAME}
```

LIST_FILENAME (which would be InputFiles.txt, in our example) should be a file in INPUT_DIR.

An oddity we encountered was that Step Two would fail to run correctly unless it was run from the directory containing its input files.

VII. ACKNOWLEDGEMENT

The Cornell Web Lab is funded in part by National Science Foundation grants CNS-0403340 SES-0537606, IIS-0634677, and IIS-0705774

REFERENCES

- [1] "<http://www.infosci.cornell.edu/sin/weblab/>."
- [2] S. R. Sangwoo Kim and S. Seguin, "Web graph generation," in *Cornell University*, May 2007.
- [3] L. Page, S. Brin, R. Motwani, and T. Winograd, "The PageRank Citation Ranking: Bringing Order to the Web," 1998.
- [4] "<http://wiki.apache.org/lucene-hadoop/>."
- [5] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," in *Usenix SDI*, 2004.

- [6] H. G. Sanjay Ghemawat and S.-T. Leung, "The google file system," in *19th ACM Symposium on Operating Systems Principles*, Oct 2003.
- [7] "<http://lucene.apache.org/hadoop/docs/current/api/index.html>."

Appendix

■ StepOne.java

```
import java.io.BufferedWriter;
import java.io.ByteArrayOutputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStream;
import java.io.PrintWriter;
import java.util.Iterator;
import java.lang.Math;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.Writable;
import org.apache.hadoop.io.WritableComparable;
import org.apache.hadoop.mapred.JobClient;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reducer;
import org.apache.hadoop.mapred.Reporter;

public class StepOne {
    public static class MapClass extends MapReduceBase implements Mapper {
        private Text fromURL = new Text();
        private Text toURL = new Text();
        private final static Text one = new Text("1");

        public void map(WritableComparable key, Writable value, OutputCollector output, Reporter
reporter) throws IOException {
            String splitString[] = ((Text)value).toString().split("\t");

            Canonicalizer can = new Canonicalizer();
            String canFromURL = can.canonicalize(splitString[0].trim());
            String canToURL = can.canonicalize(splitString[1].trim());

            if (canFromURL.length()>0 && canToURL.length()>0)
            {
                fromURL.set(canFromURL);
                toURL.set(canToURL);
            }
        }
    }
}
```

```

        output.collect(toURL, fromURL);        // Output a (toURL, fromURL) pair
        output.collect(fromURL, one);        // Output a (fromURL, 1) pair
    }
}
}

public static class Reduce extends MapReduceBase implements Reducer {
    public void reduce(WritableComparable key, Iterator values, OutputCollector output,
Reporter reporter) throws IOException {
        try {
            String fname =
"fromURL"+(long)(Math.random()*10000)+(long)(Math.random()*10000)+
(long)(Math.random()*10000);

            PrintWriter tempOutput = new PrintWriter(new BufferedWriter(new
FileWriter(fname)));
            boolean oneSeen = false;
            boolean fromURLSeen = false;

            while(values.hasNext()) {
                String s = ((Text)values.next()).toString().trim();
                if(s.equals("1")){
                    oneSeen = true;
                }
                else{
                    if (fromURLSeen){
                        tempOutput.print('\t' + s);
                    }
                    else{
                        tempOutput.print(s);
                        fromURLSeen = true;
                    }
                }
            }
        }

        tempOutput.close();

        if(oneSeen && fromURLSeen){
            InputStream in = new FileInputStream(fname);
            ByteArrayOutputStream baos = new ByteArrayOutputStream();
            byte[] buffer = new byte[4096];
            for(int len = -1; (len = in.read(buffer)) != -1; ) {
                baos.write(buffer, 0, len);
            }
        }
    }
}

```

```

        baos.flush();
        in.close();
        baos.close();
    if (baos.size()>0){
        byte[] stuff = baos.toByteArray();
        output.collect(key, new Text(stuff));
    }
}

File fileInstance = new File(fname);
fileInstance.delete();
Runtime.getRuntime().gc();
} catch (Exception e) {
    System.out.println(e);
}
}
}
}

```

```

public static void main(String[] args) throws IOException {
    JobConf conf = new JobConf(StepOne.class);
    conf.setJobName("stepone");
    conf.setOutputKeyClass(Text.class);
    conf.setOutputValueClass(Text.class);
    conf.setMapperClass(MapClass.class);
    //conf.setCombinerClass(Reduce.class);
    conf.setReducerClass(Reduce.class);
    conf.setInputPath(new Path(args[0].trim()));
    conf.setOutputPath(new Path(args[1].trim()));
    JobClient.runJob(conf);
}
}

```

■ StepTwo.cpp

```

#include <cstdio>
#include <cstdlib>
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <sys/time.h>
#include <iomanip>

```

```
using namespace std;
```

```
struct timeval startTime;
```

```

struct timeval endTime;
struct timezone tz;

// Show standard out messages
const bool SHOW_MESSAGES = true;

// Specifies how often a visible update should occur
const int POST_INTERVAL = 200000;

int main(int argc, char *argv[]) {
    gettimeofday(&startTime, &tz);          // Start "stopwatch"
    std::cout << setiosflags(ios::fixed) << setprecision(6);
    std::cout << "-----" << endl;

    if(argc != 3) {
        std::cout << "Usage: ./WebLabStepTwo <remote directory> <name of list of files>" << endl;
        std::cout << "The file holding the list of files should be located in the remote directory." << endl;
        std::cout << "-----" << endl;
        return 1;
    }

    // Extract the path of the remote directory
    std::string remoteDirectory = argv[1];
    if(SHOW_MESSAGES) std::cout << "Remote directory: " << remoteDirectory << endl;

    // Create the path of the list of files
    std::string listOfFilesString = remoteDirectory + argv[2];

    if(SHOW_MESSAGES)
        std::cout << "List of files: " << listOfFilesString << endl;

    // Create a stream to read the list of files
    ifstream listOfFilesStream(listOfFilesString.c_str());

    // Holds the name of each file
    std::string lineString;

    // Only parse if the file has been opened
    if(listOfFilesStream.is_open()) {
        // Parse the file line by line until the end of the file (eof) has been reached
        while(!listOfFilesStream.eof()) {
            // Extract the file name
            getline(listOfFilesStream, lineString);
            // Ignore blank lines

```

```

        if(lineString.empty() != true) {
            string removeString = "rm -f " + remoteDirectory + lineString + "-output > /dev/null";
            // Execute the command
            system(removeString.c_str());
            if(SHOW_MESSAGES) std::cout << "Deleted: " << remoteDirectory + lineString + "-output"
<< endl;
        }
    }
} else {
    std::cout << "Unable to open " + listOfFilesString << endl;
    std::cout << "Exiting." << endl;
    return 1;
}

listOfFileStream.close();                // Close the file

string removeString = "rm -f " + remoteDirectory + "IndexFile.txt"; // Set up a command to delete the
existing index file
system(removeString.c_str());            // Execute the command
if(SHOW_MESSAGES) std::cout << "Deleted: " << remoteDirectory + "IndexFile.txt" << endl;
string indexFileString = remoteDirectory + "IndexFile.txt";
ofstream indexFile(indexFileString.c_str()); // Open a new index file
if(SHOW_MESSAGES) std::cout << "Created: " << remoteDirectory + "IndexFile.txt" << endl;

unsigned int numberOfLines = 0;          // Keeps track of number of lines encountered
over all files
unsigned int index = 0;                  // The number that the index should begin with

ifstream listOfFiles(listOfFilesString.c_str()); // Open the list of files again
std::string line;                        // Holds the name of each file
if(listOfFiles.is_open()) {
    while(!listOfFiles.eof()) {
        getline(listOfFiles, line);
        if(line.empty() != true) { // Ignore blank lines
            string currentInputFileName = line;
            if(SHOW_MESSAGES) std::cout << "Opening input file called " << currentInputFileName <<
"." << endl;
            ifstream inputFile(currentInputFileName.c_str()); // Open the specified file

            string currentOutputFileName = line + "-output";
            if(SHOW_MESSAGES) std::cout << "Opening output file called " << currentOutputFileName
<< "." << endl;
            ofstream outputFile(currentOutputFileName.c_str()); // Create a corresponding output file

```

```

    if(inputFile.is_open() && outputFile.is_open() && indexFile.is_open()) { // Make sure all the
files have been opened
    std::string previousURL("");

    while(! inputFile.eof() ) {          // Run through the entire file
        std::string line;                // Storage for the line currently being parsed
        getline(inputFile, line);        // Grab the next line

        if(! line.empty() ) {           // Only parse the lines that have text
            numberOfLines++;             // Update line number counter

            string token = line.substr(0, line.find("\t")); // Only take the first URL

            if( previousURL != token ) { // If the current URL is not the same as the last URL
                previousURL = token;    // Set the current URL as a "new" previousURL
                indexFile << index << "\t" + token + "\n"; // Store the registered index in a file
                index++;                 // Give the next unique URL a new index number
            }

            outputFile << (index - 1) << "\t" + line + "\n"; // Construct a new line by including the
index of the first URL

            if(numberOfLines % POST_INTERVAL == 0) { // Update the user every so often
                gettimeofday(&endTime, &tz);
                double diffInSeconds = endTime.tv_sec - startTime.tv_sec;
                double diffInMicroseconds = endTime.tv_usec - startTime.tv_usec;
                double timeElapsedInSeconds = (diffInSeconds + diffInMicroseconds*0.000001);
                std::cout << "Completed line " << numberOfLines << ". Total time elapsed: " <<
timeElapsedInSeconds << " (sec.)" << endl;
            }
        }
    }
} else {
    std::cout << "Unable to open one of the following: " + currentInputFileName + ", " +
currentOutputFileName + ", or " + indexFileString << endl;
    std::cout << "Exiting." << endl;
    return 1;
}

indexFile.flush(); // Flush all data to disk
outputFile.flush();

inputFile.close(); // Close the input file
if(SHOW_MESSAGES) std::cout << "Input file called " << currentInputFileName << " is

```

```

closed." << endl;
    outputFile.close();                // Close the output file
    if(SHOW_MESSAGES) std::cout << "Output file called " << currentOutputFileName << " is
closed." << endl;
    if(SHOW_MESSAGES)                   std::cout                               <<
"*****" << endl;
    }
    }
} else {
    std::cout << "Unable to open " + listOfFilesString << endl;
    std::cout << "Exiting." << endl;
    return 1;
}

listOfFiles.close();                  // Close the file holding the file list
indexFile.close();                   // Close the index file
if(SHOW_MESSAGES) std::cout << "Index file closed." << endl;

gettimeofday(&endTime, &tz);          // Show time statistics
double diffInSeconds = endTime.tv_sec - startTime.tv_sec;
double diffInMicroseconds = endTime.tv_usec - startTime.tv_usec;
double timeElapsedInSeconds = (diffInSeconds + diffInMicroseconds*0.000001);
std::cout << endl;
std::cout << "Index range: 0 to " << (index - 1) << endl;
std::cout << "Total number of lines: " << numberOfLines << endl;
std::cout << "Total time elapsed: " << timeElapsedInSeconds << " (sec.)" << endl;
std::cout << setiosflags(ios::fixed) << setprecision(10);
std::cout << "Average time per record: " << timeElapsedInSeconds / numberOfLines << " (sec.)" <<
endl;

std::cout << "-----" << endl;
return 0;
}

```

■ StepThree.java

```

import java.io.*;
import java.util.*;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.Writable;
import org.apache.hadoop.io.WritableComparable;
import org.apache.hadoop.mapred.JobClient;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.mapred.Mapper;

```

```

import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reducer;
import org.apache.hadoop.mapred.Reporter;
import org.apache.hadoop.mapred.MapReduceBase;

public class StepThree {
    public static class MapClass extends MapReduceBase implements Mapper {
        private Text index = new Text();
        private Text toUrl = new Text();
        private Text fromUrl = new Text();

        public void map(WritableComparable key, Writable value, OutputCollector output, Reporter
reporter) throws IOException {
            String indexedUrlToFrom = value.toString();
            String[] splitString = indexedUrlToFrom.split("\\t");

            index.set("-" + splitString[0].trim());
            toUrl.set(splitString[1].trim());

            output.collect(toUrl, index);

            index.set(splitString[0].trim());

            for(int i = 2; i < splitString.length; i++) {
                fromUrl.set(splitString[i].trim());
                output.collect(fromUrl, index);
            }
        }
    }

    public static class Reduce extends MapReduceBase implements Reducer {
        public void reduce(WritableComparable key, Iterator values, OutputCollector output,
Reporter reporter) throws IOException {
            TreeSet<Integer> treeSet = new TreeSet<Integer>();
            Text marker = null;
            boolean foundMarker = false;
            String currentPIN = null;

            try{
                while (values.hasNext()) {
                    currentPIN = values.next().toString();

                    if(currentPIN.charAt(0) == '-') {
                        marker = new Text(currentPIN.substring(1)); //remove the leading

```



```

import java.util.*;
import java.lang.*;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapred.JobClient;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reducer;
import org.apache.hadoop.mapred.Reporter;
import org.apache.hadoop.mapred.MapReduceBase;

public class StepFour {
    public static class MapClass
        extends MapReduceBase implements Mapper{
        public void map(WritableComparable key, Writable value, OutputCollector output, Reporter
reporter) throws IOException {
            try {
                String indexsToFrom = value.toString().trim();
                String[] indexs = indexsToFrom.split("\\s+"); //split on whitespaces
                LongWritable keyIndex = new LongWritable(Long.parseLong(indexs[0]));
                Text values = new Text(indexsToFrom.substring(indexs[0].length()).trim());
                output.collect(keyIndex, values);
            }
            catch (Exception e) {
                System.out.println(e);
            }
        }
    }

    public static class Reduce extends MapReduceBase implements Reducer {
        public void reduce(WritableComparable key, Iterator values, OutputCollector output,
Reporter reporter) throws IOException {
            try{
                if (values.hasNext())
                    output.collect(key, new Text (values.next().toString()));
            }
            catch (Exception e) {
                System.out.println(e);
            }
        }
    }

    public static void main(String[] args) throws IOException {

```

```

        JobConf conf = new JobConf(StepFour.class);
        conf.setJobName("stepfour");
        conf.setOutputKeyClass(LongWritable.class);
        conf.setOutputValueClass(Text.class);
        conf.setMapperClass(MapClass.class);
        conf.setReducerClass(Reduce.class);
        conf.setInputPath(new Path(args[0].trim()));
        conf.setOutputPath(new Path(args[1].trim()));
        JobClient.runJob(conf);
    }
}

```

■ JobStep1.sh

```

#!/bin/sh

userDir="/tmp/$USER"
weblabDir="$HOME/WebLabTemp"

outputDir="/mnt/weblab/Step1_Output"
logName="logStep1"
hadoopSh="Hadoop_Step1.sh"

rm -rf $outputDir
mkdir -p $outputDir

rm -rf $userDir
mkdir -p $userDir

cp $weblabDir/$hadoopSh $userDir/

cd $userDir

./$hadoopSh $userDir $weblabDir $outputDir $logName >& $logName

```

■ Hadoop_Step1.sh

```

#!/bin/sh

ARGS=4
if [ $# -ne $ARGS ] then
    echo "It needs 4 arguments"
    exit 1
fi

userDir=$1 weblabDir=$2 outputDir=$3 logName=$4

```

```
hadoopHome="/tmp/hadoop-$USER" hadoopVer="hadoop-0.9.2"
jarClass="StepOne" input="input1" output="output1"

# create and set hadoop home directory for each compute node rm -rf
$shadoopHome mkdir $shadoopHome

# copy hadoop to hadoopHome directory cp -r
$weblabDir/hadoopfiles/$shadoopVer/ $shadoopHome/

# create a list of working nodes vsched -m

# get the host name which is just the first node in the list to
modify hadoop config file hostName=$(head -n 1 machines)

hadoopDir="${hadoopHome}/${hadoopVer}"

# put and replace the hostname in hadoop-site.xml configuration file
cat $shadoopDir/conf/hadoop-site.xml | sed s/localhost/$hostName/ >
$shadoopDir/conf/hadoop-site1.xml

# remove the old config file rm $shadoopDir/conf/hadoop-site.xml

# rename the modified config file mv
$shadoopDir/conf/hadoop-site1.xml $shadoopDir/conf/hadoop-site.xml

# getting the number of compute nodes numNodes=$(wc -l machines |
cut -f1 -d " ") numNodes=$(expr $numNodes - 1)

# get names of slaves tail -n $numNodes machines >
$shadoopDir/conf/slavesTemp

# copy hadoop to slave nodes while read LINE; do rsync -az -e ssh
$shadoopHome/ $USER@$LINE.tc.cornell.edu:$shadoopHome

done < $shadoopDir/conf/slavesTemp

# change slaves files on namenode rm $shadoopDir/conf/slaves mv
$shadoopDir/conf/slavesTemp $shadoopDir/conf/slaves

#rm backup files from the input directory rm -rf
$shadoopDir/$input/*~

#remove the output directory that may be left over from the last
```

```
session rm -rf $shadoopDir/$output

#change permissions of some files chmod 755 $shadoopDir/bin/* chmod
755 $weblabDir/jre1.5.0_11/bin/*

# go to the hadoop home dir cd $shadoopDir

# format namenode $shadoopDir/bin/hadoop namenode -format

# start all $shadoopDir/bin/start-all.sh

# sync virtual input directory $shadoopDir/bin/hadoop dfs -put $input
$input

# run actual program, for the UI team $shadoopDir/bin/hadoop jar
$jarClass.jar $jarClass $input $output

# sync output directory $shadoopDir/bin/hadoop dfs -get $output .

# finalize the process $shadoopDir/bin/stop-all.sh

# copy output file and logs scp -r $shadoopDir/$output
$USER@linuxlogin2.tc.cornell.edu:$outputDir scp $userDir/$logName
$USER@linuxlogin2.tc.cornell.edu:$outputDir
```

■ Hadoop_Step1_local.sh

```
#!/bin/sh
userDir="/tmp/$USER"
weblabDir="$HOME/WebLabTemp"
outputDir="/mnt/weblab/Step1_Output"
logName="logStep1"

hadoopHome="/tmp/hadoop-$USER"
hadoopVer="hadoop-0.9.2"
jarClass="StepOne"
input="input1"
output="output1"

# create and set hadoop home directory for each compute node
rm -rf $hadoopHome
mkdir $hadoopHome

# copy hadoop to hadoopHome directory
cp -r $weblabDir/hadoopfiles/$hadoopVer/ $hadoopHome/
```

```

hadoopDir="${hadoopHome}/${hadoopVer}"

#rm backup files from the input directory
rm -rf $hadoopDir/$input/*~

#remove the output directory that may be left over from the last session
rm -rf $hadoopDir/$output

#change permissions of some files
chmod 755 $hadoopDir/bin/*
chmod 755 $weblabDir/jre1.5.0_11/bin/*

cd $hadoopDir

# format namenode
$hadoopDir/bin/hadoop namenode -format

# start all
$hadoopDir/bin/start-all.sh

# sync virtual input directory
$hadoopDir/bin/hadoop dfs -put $input $input

# run actual program, for the UI team
$hadoopDir/bin/hadoop jar $jarClass.jar $jarClass $input $output

# sync output directory
$hadoopDir/bin/hadoop dfs -get $output .

# finalize the process
$hadoopDir/bin/stop-all.sh >&stop-all_log

# copy output file and logs
scp -r $hadoopDir/$output $USER@linuxlogin2.tc.cornell.edu:$outputDir

```

■ **MyJob1.xml**

```

<?xml version="1.0" ?>
<job>
<nodes>30</nodes>
<minutes>3000</minutes>
<type>batch</type>
<affiliation>v2linux</affiliation>
<run>/bin/sh $HOME/WebLabTemp/JobStep.sh 1</run>
</job>

```