The Web Lab
Cornell University, Ithaca, NY

Efficient extraction of individual pages from a complete web crawl

Oleg Krokhin
ok43@cornell.edu

Project advisor: William Y. Arms
Spring 2009

Table of contents

Abstract

This is an independent sub-project done as part of Cornell University's Web Lab research group. The goal of the project is to be able to efficiently extract large numbers of individual web pages from a complete web crawl. A web crawl consists of multiple large ARC files that store compressed web pages in a concatenated manner. An application was developed that, given metadata about the location of individual pages, extracts these pages and outputs them using the same ARC format as the input files. In the future, these output files will be fed to an indexing application such as NutchWAX in order to perform efficient full-text search on the pages' contents.

Due to the scale of web crawls, it was necessary to build a distributed application. The application was developed using Hadoop and it meets the efficiency requirement. Using Cornell's distributed computation facilities, it is possible to extract on the order of millions of pages in the range of an hour.

Introduction

The Internet Archive has “crawls” or archived copies of web pages and documents taken at a certain period of time. Because the size of a web crawl is very large relative to the size of a typical web page, the Internet Archive uses the ARC file format to store these pages. Currently, a complete web crawl can take anywhere in the 2-50 TB range [1]. Each ARC file is usually around 100 MB in size and contains approximately 10,000-20,000 pages [2]. Thus, a web crawl consists of thousands or hundreds of thousands of ARC files.

In the Web Lab project, an important objective is to be able to perform full-text searching on a subset of pages extracted from a particular web crawl. In order to do that, it is necessary to extract individual pages from an entire web crawl and create new ARC files that can then be used as input to an indexing tool such as NutchWAX. The objective of this project in particular is to be able to perform the first part of the task - given a subset of pages with corresponding metadata, to efficiently extract them from a web crawl and output them as new ARC files.

The application to solve this objective was developed using Hadoop, which is an open-source implementation of the parallel computing MapReduce framework, and uses Java as its underlying language. It fulfills the desired efficiency objective by extracting pages without uncompressing them and by avoiding the use of reduce tasks. In practice, when using Cornell’s Center for Advanced Computing Hadoop cluster, it is possible to extract 5 million pages in less than 50 minutes, using 40 nodes to perform the task.

This document assumes a good understanding of MapReduce programming and some experience using Hadoop.

Application design and implementation

ARC file format

The format of ARC files is discussed extensively in other publications [2][3], so only a brief description will be provided here. Figure 1 shows a sample ARC file.

Figure 1

```
filedesc://DP_crawl10.20030201081028.arc 0.0.0.0 20030201081028 text/plain 76
1 0 InternetArchive
URL IP-address Archive-date Content-type Archive-length

http://www.cerote.com:80/carrier.htm 207.21.211.180 20030201081028 text/html 606
HTTP/1.1 200 OK
Server: Microsoft-IIS/5.0
Date: Sat, 01 Feb 2003 08:10:27 GMT
Content-Type: text/html
Accept-Ranges: bytes
Last-Modified: Fri, 06 Sep 2002 05:49:18 GMT
ETag: "31c6ea236955c21:ae8"
Content-Length: 380
<html>
<head>
<meta http-equiv="Content-Language" content="en-us">
<meta name="GENERATOR" content="Microsoft FrontPage 5.0">
<meta name="ProgId" content="FrontPage.Editor.Document">
<meta http-equiv="Content-Type" content="text/html; charset=windows-1252">
<title>Aqui aparecera el mensaje del carrier</title>
</head>
<body bgcolor="#D00020">
</body></html>

http://phyweb.lbl.gov:80/robots.txt 131.243.48.204 20030206183409 text/plain 440
HTTP/1.1 200 OK
Date: Thu, 06 Feb 2003 18:34:08 GMT
Server: Apache/1.3.27 (Unix)  (Red-Hat/Linux) mod_perl/1.23
Last-Modified: Fri, 06 Feb 1998 06:07:33 GMT
ETag: "813-a4-34daa8a5"
Accept-Ranges: bytes
Content-Length: 164
Connection: close
Content-Type: text/plain
# robots.txt for http://www-physics.lbl.gov/
User-agent: *
Disallow: /physdiv/
Disallow: /cgi-bin/
Disallow: /div-office/
Disallow: /bin/
Disallow: /theorygroup

[Other web pages and documents]
```

The file consists of a three line header, followed by entries that contain both the metadata and the contents of web pages and documents. Each new entry begins with the URL of the page. The contents of the ARC file are compressed using the GZIP format. The GZIP format allows each entry to be compressed individually and then concatenated, so that individual pages can be uncompressed without uncompressing the entire ARC file first. In order to do this individual uncompressing, it is necessary to know the byte offset inside the ARC file where a page starts.

The initial approach was to uncompress pages using only this offset. Given a valid offset, GZIP can detect where an individual compressed page ends. After uncompressing, pages were again recompressed and then combined into an output ARC file.

The above approach, while usable, was not efficient. Instead, the uncompressing step was avoided altogether. One of the Web Lab databases used to store metadata for web crawls also contains information about the length of each compressed page. Having the offset and the length, it is possible to perform a simple byte copy of the desired pages.

Application input

Figure 2 shows a sample input file to the application that consists of four fields.

Figure 2

```
www.cs.cornell.edu/johannes/  DP_crawl10.20030201183942.arc.gz  42256222  5306
cs230.cs.cornell.edu/         DP_crawl10.20030203012315.arc.gz  35803382  51
www.cs.cornell.edu/home/rdz/  DP_crawl10.20030201183942.arc.gz  61119657  40
```

The first field is the URL of the page to extract, in the same format as it is stored inside the input ARC file itself. This field is not necessary to perform the extraction, but it is useful to easily identify pages that failed to extract. It is also possible to compare this URL with the URL that is stored inside the compressed page to ensure that the right page was extracted. However, this defeats the purpose of performing a byte copy without uncompressing, and therefore the option is not currently provided as it would considerably affect performance.

The second field is the input ARC file name where the page is stored. The third field is the byte offset inside the ARC file where the compressed page starts. The fourth field contains the compressed length of the page. Note that the fields are separated by tabs and each page entry is on a separate line.

All of the input information is readily available in Web Lab databases, so no additional computation is required to generate the input.

In addition to the input file with the pages to extract, the relevant web crawl needs to reside on the Hadoop Distributed File System. The application takes a directory path as an argument and expects all of ARC files to be located in that directory.

Application output

The application takes a single input file in the abovementioned format and the Hadoop framework then splits the input entries and assigns each split to a map task. Each map task creates a corresponding output ARC file that is named using the task ID and a counter. For example, an output file could have the following name: task_200905131026_0001_m_000000_0.arc.gz.
Here, task_200905131026_0001_m_000000 is the task ID and the last 0 is the counter. In order to be consistent with the Internet Archive's 100 MB file size for ARC files, if a

single map task receives more than 100 MB worth of pages, then new output ARC files will be created as necessary and the counter value will be increased every time. Using this format, output ARC files should not have any naming conflicts.

Implementation discussion

The application logic, excluding configuration and setup, is contained entirely in a single map task. This includes extracting pages from input ARC files and placing them in new output ARC files. An IdentityReducer is used, which performs no reduction and writes all input values directly to the output.

Another alternative was considered where all pages from the same input ARC file were combined and processed in a single reduce task, with pages sorted by the byte offset within the input ARC file. A reduce task was created for each input ARC file, and page extraction would take place sequentially. This alternative was discarded for several reasons. The Hadoop Distributed File System, where the input ARC files are stored, can distribute parts of the same file on different servers, so processing a single input ARC file sequentially in the same task does not provide any obvious advantage. Furthermore, this approach produced a very large number of output ARC files that showed a high variation in size. Since pages are spread across many input ARC files, there was a very large number of reduce tasks. In addition, some input ARC files contained many more pages than others causing some reduce tasks to take much longer to finish. Finally, this approach was also discarded because of the general notion that due of the large scale of the web crawls, avoiding an extra pass is usually desirable.

The initial approach for building up the output ARC file was to use file appending. However, Hadoop Core v0.18.3, which is the current version installed on the Cornell's Hadoop cluster, does not support file appending even though it is part of the API. To work around this issue, the current solution uses a file stream that is kept open until completion of the map task.

The file stream is opened in an initialization method that is called before any input is processed by the map task. Then, individual pages are extracted from the input ARC files and written to the stream. When the input is exhausted, a finalization method is executed and the file stream is closed. It is also worth noting that with very small inputs (for example, 100 pages), extra map tasks are created and then killed by Hadoop when they receive no input. Since the initialization method is still called on these map tasks, output ARC files containing only headers are generated. This is not an issue with reasonably large inputs.

The header of the output ARC file is also written in the initialization method and deserves some discussion because of its complexity. As shown in Figure 1, the header consists of three lines, where only the first line contains variable parameters. When creating the header, it is necessary to specify the ARC file name, the creation date and the length, in bytes, of the header. While this is straightforward, some complications arise because the header itself also needs to be compressed using GZIP. Initially, the header

was written using Java's internal GZIP implementation but this did not produce valid files. Instead, the application uses a technique employed by Heritrix, the open source web crawler used by the Internet Archive, to write headers. The issue is that after being compressed, the header requires some manual byte manipulation that identifies the file as an Internet Archive GZIP ARC file. This manipulation is not possible using Java's internal GZIP implementation and thus, the application uses Heritrix's GZIP implementation. The current application, in addition to adapting some of the code from ARCWriter class, uses the following Heritrix classes without any modification: GZIPHeader, GzippedInputStream, NoGzipMagicException. GzippedInputStream in turn requires a library for repositionable streams. This library can be obtained from Heritrix's lib folder and it is called fastutil-5.0.3-heritrix-subset-1.0.jar. The necessary Heritrix files are included as part of the source code of the application, but the library needs to be provided as an external parameter during launch. More detailed explanations about the header writing process can be found in Heritrix's ARCWriter class documentation and source code [4][5].

Detailed instructions for running the application on Cornell's cluster are included in the README file that is packaged together with the application source code and binaries.

Error handling

The application does a considerable amount of error checking. Many types of problematic inputs are handled and files are checked for existence before use. In case that an error is detected, it will be logged to the error stream along with the input line that caused the problem. A single input page that fails to be extracted does not stop the processing of the rest of the input, so it is important to check the logs after completion. In Hadoop, each server stores its own log files. Therefore, in order to determine if any errors happened, it is necessary to look in the log files of all the servers. Hadoop's web interface makes this task easier, although it is still time consuming. A way to generate a single log file is discussed in the Conclusions and future improvements section.

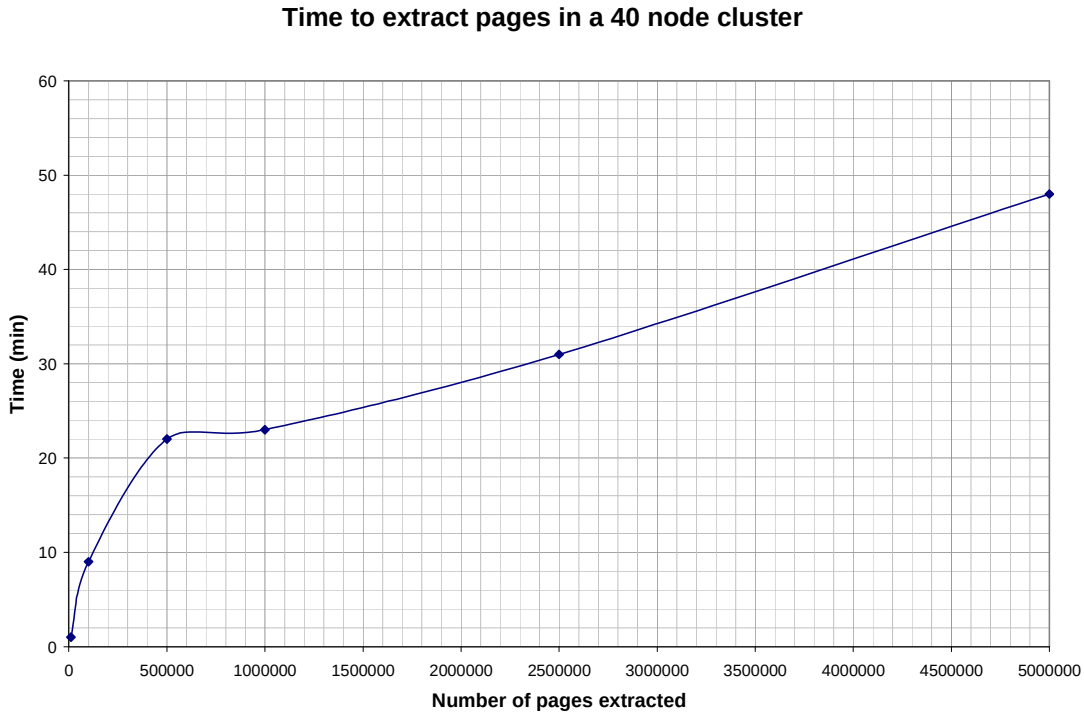
One particular type of error to be aware of is truncation of ARC files. Because downloading web crawls from the Internet Archive is a long and error-prone process, some ARC files in a web crawl may end up being truncated. The application checks for these problems by comparing the actual length of the input ARC file to the offset and length of a page within that ARC file specified in the input. If a page is located past the actual size of the input ARC file, this indicates that the file was truncated and it is logged in the same manner as other types of errors.

Results and performance

The application has been tested with various input sizes. In the Web Lab project, it is often necessary to extract and analyze a particular subset of pages from a specific web crawl. For example, it may be necessary to analyze all the pages from the government domain (URL hostnames that end in .gov) or all the pages from the Cornell University Computer Science domain (URL hostnames that end in cs.cornell.edu) from a crawl of the web that was completed in April 2003 (this crawl is the DP crawl that was used in testing) [1]. This means that input sizes can vary drastically. Taking the above example, there are approximately 8 million government pages and 14000 Cornell University Computer Science pages. Thus, in testing the performance of the application, different input sizes were used.

Figure 3 shows the application performance when using Cornell's Hadoop cluster, with 40 nodes allocated to perform the computation. For consistency, the tests were executed during times when no other Hadoop jobs were running. The test with 10000 pages was done using Cornell University Computer Science pages, while all other tests were done using pages from the government domain. The Cornell University Computer Science pages had an average size of 23 KB, while the government pages had an average size of 28 KB.

Figure 3

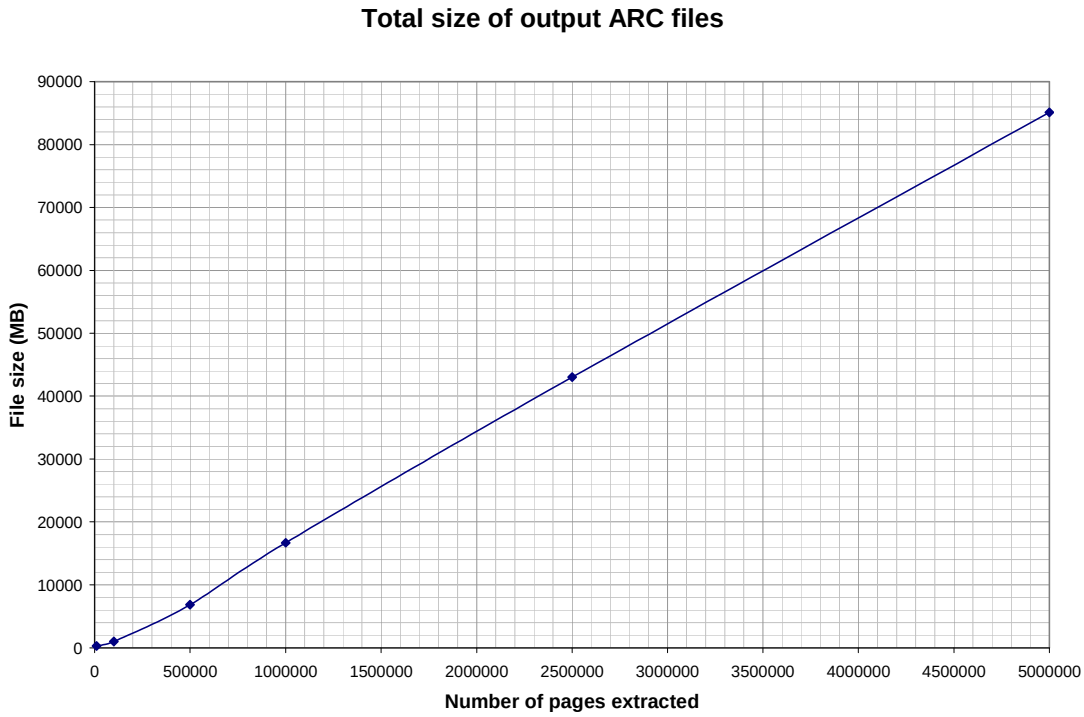


The application is efficient enough for use within Web Lab, since it takes less than an hour to extract 5 million pages, which is close to the number of pages in the government

domain. With a small number of pages (up to 500000 pages approximately), the time to completion grows more rapidly, likely because of the initial overhead of setting up the map tasks. Also, with smaller inputs, the output ARC files are smaller and writing smaller files means that, relatively, more time is spent initializing output files. With inputs of at least 1 million pages, computation time grows linearly, although 5 million pages are extracted in only twice (not five times) as much time as 1 million pages.

Figure 4 shows the total size of the output ARC files for different numbers of pages.

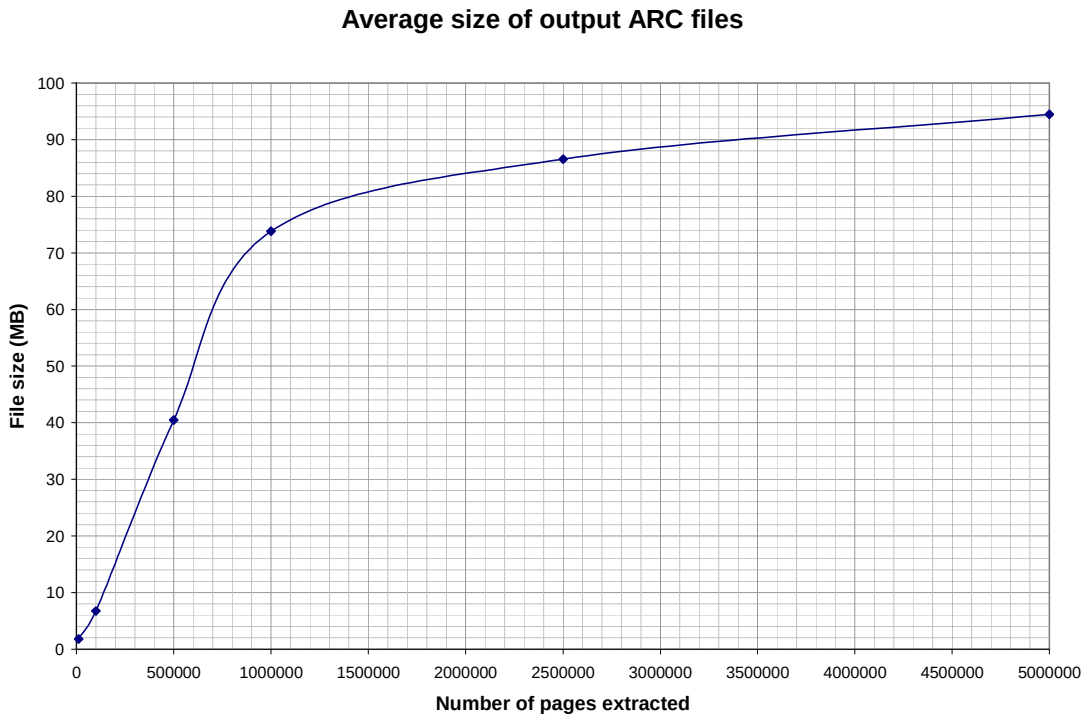
Figure 4



With 5 million pages, over 80GB of output is produced. This suggests that other approaches to extracting pages that use non-identity reduce tasks or that uncompress pages could have considerably slower performance due to increased network and file I/O.

Figure 5 shows the average size of output ARC files.

Figure 5



With input of 500000 pages of less, output ARC files average to approximately 40 MB, which is well below the usual ARC file size of 100 MB used by the Internet Archive. Though this is not a significant problem, it may be desirable to produce larger files instead, perhaps at the expense of some computation time. The application takes an optional argument where the user can specify the number of map tasks to be used. By reducing the number of map tasks it is possible to generate larger output ARC files since each map task receives a larger number of input pages. Another approach, using a custom input splitter, is discussed in the Conclusions and future improvements section.

Conclusions and future improvements

The main objective of the project was achieved successfully. The application is able to extract the number of pages typically required for Web Lab research in a very reasonable timeframe. With the application in its current state, it should be possible to perform the next step in the process: indexing the output ARC files with a full-text indexer such as NutchWAX. Due to the fact that this project is still at an early stage, there are several areas that could use some improvement and they are discussed below.

Hadoop v0.19.0 introduced file appending functionality. Unfortunately, it was immediately removed in v0.19.1 because of bugs and inconsistencies. While the current approach of maintaining a file stream open throughout an entire map task is feasible, it would be interesting to study the performance of the application is the file appending API is used instead. A future version of Hadoop could provide an efficient implementation of this operation that would improve performance even further.

Currently, the application uses Hadoop's internal input splitting routines, where an arbitrary number of lines from input could be sent to a particular map task. Hadoop also provides the option of writing custom input splitters, which would solve the problem of small output ARC files being generated. A custom input splitter could group pages amounting to 100 MB (based on the compressed page length) and assign them to a single input split. This would ensure that most of the output ARC files are 100 MB in size. However, performance could suffer because it would result in a smaller number of map tasks being executed, so performance comparisons would be important if this option is implemented.

As discussed earlier, if the application receives a small number of input pages, it generates empty files containing only a header. One way to avoid this problem would be to delete the file in the map task's finalization method, but an even better approach would be to avoid writing the header at all if no input is received. A quick workaround is to write a bash script which deletes files that are too small to have any content in them.

The application produces output ARC files using version 1.0 of the ARC file format. This format has been replaced by newer versions and a new format, called WARC, is currently being considered by the Internet Archive. The application would be more useful if it provided options for writing output files in different formats. Fortunately, the Heritrix web crawler source code contains many useful classes (for example, Arc2Warc) that perform conversions between different ARC formats. However, before implementing new output formats it would be necessary to research which formats are supported by the NutchWAX indexing application.

As mentioned in the Error Handling sub-section, error logs are distributed across the Hadoop nodes used to perform the map tasks. One way to generate a single log file is to write the error contents to the output collector using a constant key, and also set the number of reduce tasks to 1. This would send all the caught error messages to a single reduce task, where it could be written to a single log file. However, this approach would

still not be a perfect solution as there may be uncaught errors that are worth inspecting. More importantly, creating a reduce step would decrease performance of the application and this is the main reason why this feature was not implemented.

References

1. The Web Laboratory: Crawl Sizes Statistics.
   http://weblab.infosci.cornell.edu/crawl_sizes_statistics.html

2. Mayank Gandhi, Jimmy Yanbo Sun. ARC Data Extraction and summarization. Section 5: Appendix. 2005.
   http://weblab.infosci.cornell.edu/publications.html

3. ARC file format.
   http://www.archive.org/web/researcher/ArcFileFormat.php

4. Heritrix ARCWriter class API.
   http://crawler.archive.org/apidocs/org/archive/io/arc/ARCWriter.html

5. Heritrix – Downloads.
   http://crawler.archive.org/downloads.html

6. Jeffrey Dean, Sanjay Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. OSDI. 2004.

7. Hadoop 0.18.3 API.
   http://hadoop.apache.org/core/docs/r0.18.3/api/index.html

8. Hadoop 0.19.0 API.
   http://hadoop.apache.org/core/docs/r0.19.0/api/index.html

9. NutchWAX Home Page.
   http://archive-access.sourceforge.net/projects/nutch/

10. Append to files in HDFS.
    http://issues.apache.org/jira/browse/HADOOP-1700

11. Disable append.
    http://issues.apache.org/jira/browse/HADOOP-5224

12. Hadoop FAQ.
    http://wiki.apache.org/hadoop/FAQ

Acknowledgements

I would like to thank Prof. William Arms for his guidance and organization of this sub-project. I would also like to thank Manuel Calimlim, who explained the contents of the Web Lab databases, helped with data management and provided test input for the application. I also appreciate the help of Yan Tang, who suggested different approaches to solving the problem during the initial phase of the project. Finally, I would like to thank Lucia Walle from Cornell's Center for Advanced Computing for promptly resolving issues that surfaced while using the cluster.

The Cornell Web Lab is funded in part by National Science Foundation grants CNS-0403340, SES-0537606, IIS-0634677, IIS-0705774 and IIS 0917666.