

The Web Laboratory: Preload System

Fall 2005 Final Report

Benzaquen, Samuel
Guo, Wei
{sb387, wg27}@cornell.edu

December, 2005

1 Introduction

The Web Laboratory is a joint project of Cornell University and the Internet Archive to provide data and create a computing platform for research about the Web and information on the Web. The data, currently around 600 TB of Web crawls conducted every two months starting from 1996, is provided by the Internet Archive, and the computing facilities are based at the Cornell Theory Center. The goal of the project is to support researchers in computer science, the social sciences and humanities, whose interests lie in the information on the Web, as well as computer scientists, who carry out research on the Web as an information structure.

The Web Laboratory must satisfy the following requirements:

- Provide a set of easy to use tools for manipulating with the content of web pages, and metadata about them. In particular, this includes the ability to extract and run algorithms on stratified subsets of one or multiple crawls and the ability to extract link sub graphs according to various criteria.
- Allow fast processing of the incoming data from Internet Archive, at a rate of at least 250GB/day. Possibility to run with different amounts of available memory and processing power.

- Ensure reliable storage of the received data, thus providing a mirror for the Internet Archive.
- Employ a flexible design, making it easy to extend the system to process multimedia formats in the future, as well as take data from sources other than Internet Archive.

To satisfy the above requirements, we decided to separate link information and metadata about pages from their content and to store the meta-information in a relational database on a single high-performance machine.

The resulting architecture of the Web Laboratory is shown in Figure 1. The main objectives of the groups are:

- Data transfer group: Transfer the ARC and DAT files from the Internet Archive to the Cornell facilities, backup them and get them ready for the preload system.
- Preload group: Process the transferred files into database loadable files extracting the web metadata.
- Database group: Load the output from the preload group into the database and maintain the main database structures.
- Data Tracking and Monitoring Group: Maintain a record of every file transferred and created with attributes like size and location.

To test the ability of the database to support the above configuration before the actual data was available in large amounts, a series of student projects were conducted to create a prototype schema, and generate a 1 billion node graph with properties similar to the Web graph [4], using RMat algorithm [2]. In addition to the graph structure, domain names were generated using real domain name statistics obtained from the ISC Domain name survey [1] [5].

At the same time, a prototype of the preload system was created [3]. This prototype was developed in Java to take advantage of the ARC reader libraries available at that time. Although the performance achieved by the prototype was far from the performance required, the results described the critical parts of the process and the bottlenecks, allowing a better design of the production system.

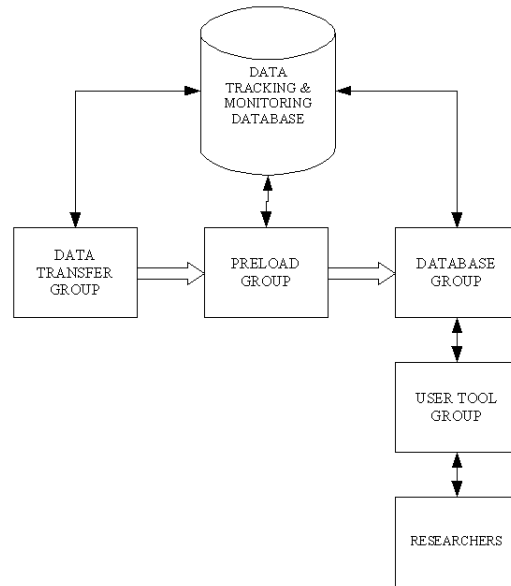


Figure 1: Overview of the complete system

Given the encouraging results with synthetic data, and taking into account the experiments with the preload prototype, a production preload system was created during Fall 2005. This report describes the architecture and experimental evaluation of the system.

2 System Architecture

2.1 Design

2.1.1 Design Goals

The data comes from Internet Archive in files in ARC and DAT formats. ARC files contain pages in the order received from the Web crawler. Every ARC file has a corresponding DAT file, which contains metadata for the pages, such as URL, IP address, crawl date and time, and links from the page. Both files are compressed with GZIP. Each compressed ARC file is about 100 MB, and a DAT file is about 15 MB on average.

The preload system takes incoming ARC and DAT files, decompresses

them, parses them to extract relevant information, and generates two types of output files: metadata for loading into the database and the actual content of the Web pages to be stored separately. The design of the subsystem does not require the corresponding ARC and DAT files to be processed together.

Our design goal is to process 250GB of data per day on 4 full time running Itanium 2 1.5GHz CPUs with light memory usage on Scidata1 machine [5].

2.1.2 Parallelism

To satisfy the speed and flexibility requirements, the system is designed to run as a set of independent single-thread processes, which completely avoid inter-process communication and locking over input or output data. To determine each process input, input files are partitioned by the last k bits of the MD5 hash sum of the filename, where 2^k is the total number of processes in the system. Likewise, each process writes its own output file. This design allows for easy configuration of the system to run a required number of processes on a given number of processors, on one or more machines.

2.1.3 Main Modules

The system is divided in the following main modules:

- Main module: Responsible of the program initialization and the main loop, making use of the other modules.
- DAT Reader: Parses DAT files into DAT entries.
- ARC Reader: Parses ARC files into ARC entries.
- DB Writer: Writes the database output files. Maintains the state of all the files been written to allow checkpointing.
- Content Writer: Writes the content files from the ARC entries into a set of concatenated compressed files.

2.1.4 Other Modules

Other modules that also play a part in the system:

- Log Writer

- GZip Reader
- Configuration file reader

2.1.5 Output Structure

The content of the web files is stored in a set of compressed files, one file for each ARC file. Since many pages on the Web do not change between crawls, the preload subsystem checks for content duplicates using MD5 check sum of the content. Thus, a copy of the content is stored only once, however many pages have that content. In order to guarantee fast access to the stored content, each pages content is compressed individually, and the compressed contents are concatenated into a single file. This allows extracting the content for a particular page without uncompressing the whole file.

For the database loadable files, a text TSV file was selected for simplicity. This format is useful for bulk loading data instead of inserting record by record. Each TSV file has at most 40GB in size as a requirement from the database group.

2.2 Implementation

2.2.1 External Tools

The program includes the free zlib library to decompress the input files (see <http://www.zlib.net>). This library was selected because of its openness, stability and performance. The C version of the library was used and a C++ wrapper was created to decompress the data. The library was also used to compress the content output into zlib streams.

2.2.2 Technical Choices

The language chosen to develop this tool was C++ through the IDE Visual Studio C++ 2003. This satisfy the principles of performance and portability established on the project. In the same manner, object oriented programing was chosen to add flexibility and extensibility to the system.

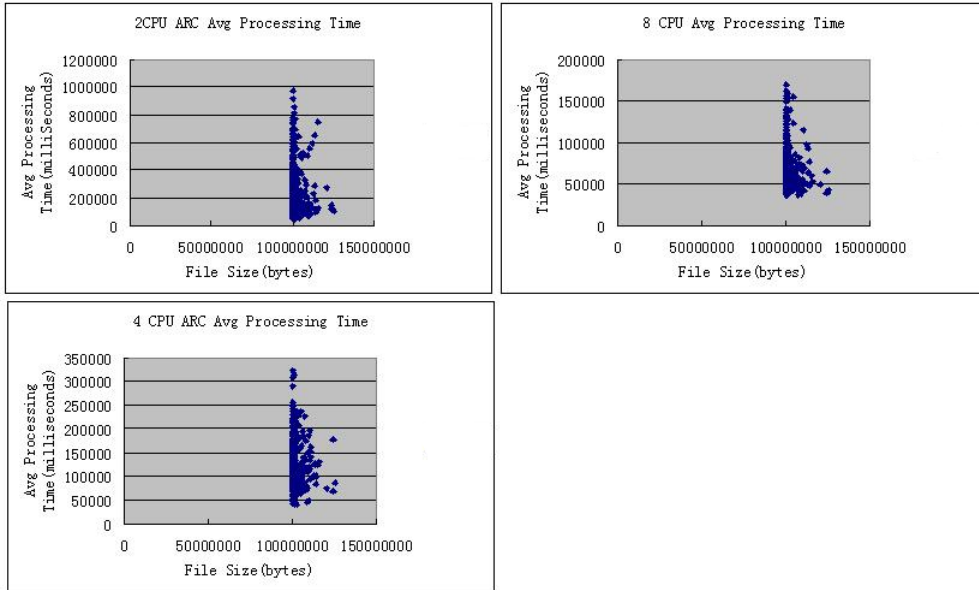


Figure 2: Average processing time against file size for ARC files

3 Experimental Results

A series of experiments were run to test the performance of the system on generating metadata for database load. The experiment results are categorized into two sets. The first set measures the average processing time, and the second set measures the average disk throughput rate. Each set of experiments used 1, 2, 4 and 8 processors, with the data partitioned into 16 parts, according to the first 4 bits of the hash sum. Separate experiments were made for ARC and DAT files.

In the set of experiments measuring processing time, sizes of most of the ARC files are around 100 Megabytes. Figures 2 and 3 shows plots of average processing times against file sizes for different number of CPUs. There was no experiment performed on just one CPU since it would have taken an unreasonably long time to run.

In general, average processing time of ARC files exhibits great variations as shown in Figure 2, but the range of variation shrinks and the overall average processing times exhibit a decreasing trend with increased number of CPUs. Plots generated from experiment results on DAT files shown in Fig-

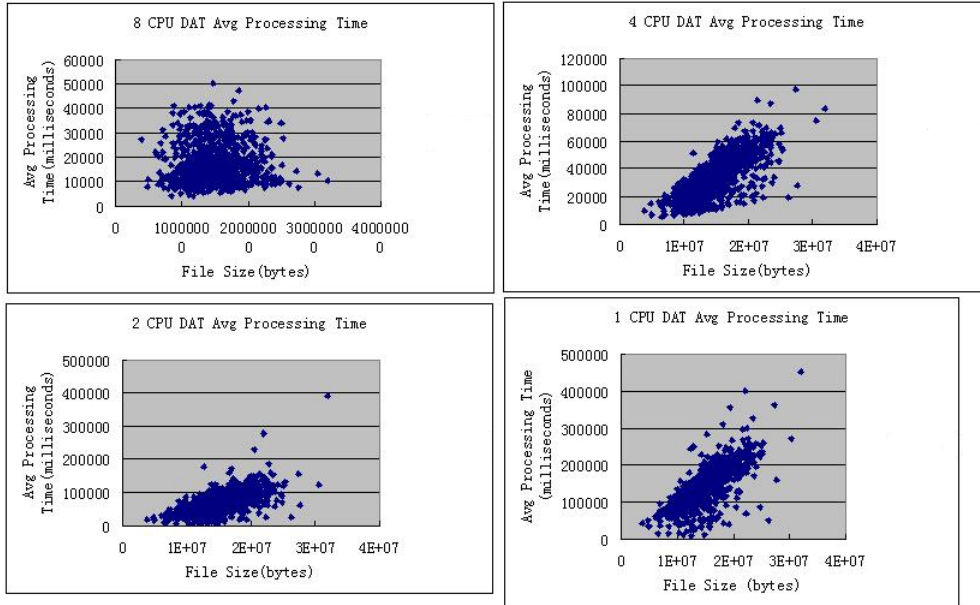


Figure 3: Average processing time against file size for DAT files

Figure 3 suggest a positive correlation between average processing times and file sizes for 1, 2, and 4 CPUs. However, there seems to be no clear correlation between average processing times and file sizes for 8 CPUs due to overwhelming number of random disk I/Os, which contributes greatly to the average processing time randomness of the plot. Again, the overall average processing time exhibits a decreasing trend with increased number of CPUs.

Figure 4 and 5 show the average disk throughput for the ARC and DAT files. X axis shows the number of CPUs used, and Y axis shows the throughput rate in KB/s. The blue bar shows throughput per processor, and the red bar shows total throughput. Adding more processors slightly decreases the throughput per processor due to increased number of random disk accesses. The total throughput increases steadily up to four processors. After that, disk contention becomes too high, and the throughput actually declines for ARC files. The results for DAT files are similar, with the total throughput flattening after four processors.

From the set experiments on measuring disk throughput rate, we conclude that four processors are optimal. The corresponding throughputs are 73

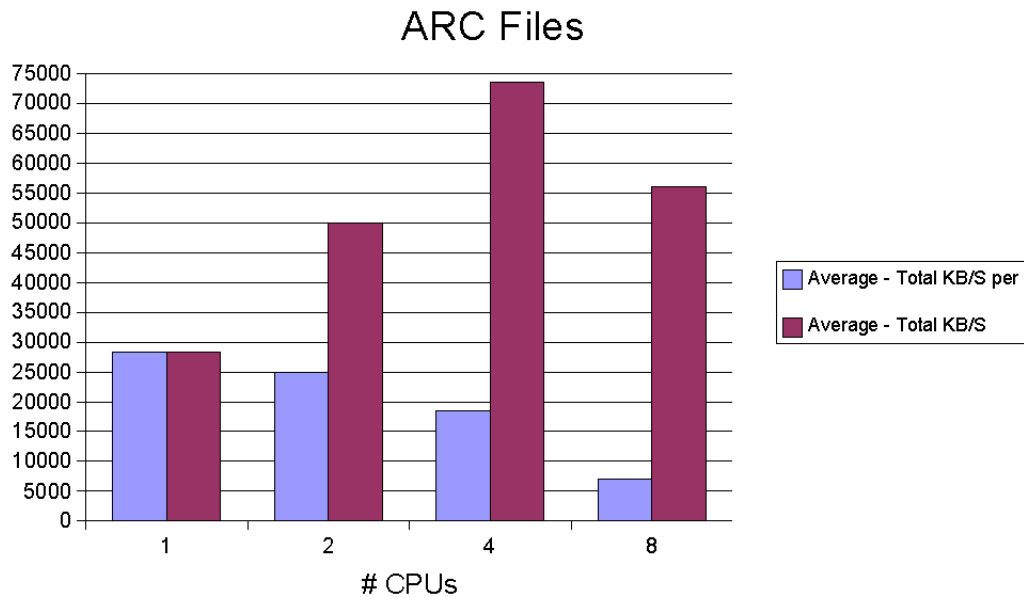


Figure 4: Average disk throughput for ARC files

MB/s (about 6 TB/day) for ARC files, and 12 MB/s (about 1 TB/day) for DAT files. For approximately 88GB of test data (ARC+DAT files), 38GB of metadata was generated to load into the database.

Note that during the above experiments no other processes were run on the system. Thus, the overall performance might be lower when the system is run in the production mode in conjunction with other subsystems.

4 Conclusion

In this report we described the architecture and results of the experiments with the preload system of the Web Laboratory. The results showed that the system satisfies the efficiency requirements, and can be configured to work with different amounts of available resources due to its flexible architectural design. We conclude that it is ready for integration with other systems of the Web laboratory in the production mode.

In order to further improve the efficiency we need to experiment with different buffer sizes for the decompression utility. Using different disk for

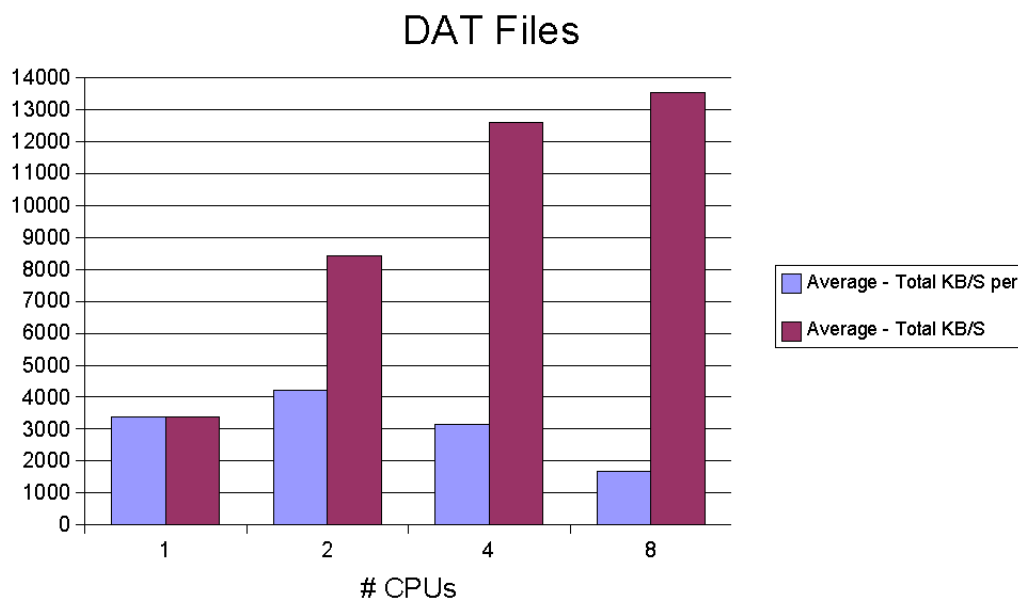


Figure 5: Average disk throughput for DAT files

the input and the output data should improve performance as well.

Other improvement possible is the use of the database, the tables maintained by the Data Tracking and Monitoring group more specifically, to retrieve the list of files pending for processing and to update this table accordingly of the results. Right now the list is created reading directly the directory in which the files are supposed to be and the results are passed as log files that have to be parsed later.

5 Acknowledgements

This work is part of the Web Laboratory, which is a joint project of Cornell University and the Internet Archive. This work is funded in part by National Science Foundation grants 0403340, 0127308, and 0537606.

We would also like to thank the Cornell Theory Center for their support on setting up the development environment and guiding us in its use.

References

- [1] Isc internet domain survey. January 2005.
- [2] Deepayan Chakrabarti, Yiping Zhan, and Christos Faloutsos. R-mat: A recursive model for graph mining. In Michael W. Berry, Umeshwar Dayal, Chandrika Kamath, and David B. Skillicorn, editors, *SDM*. SIAM, 2004.
- [3] Mayank Gandhi and Jimmy Yanbo Sun. Arc data extraction and summarization. May 2005.
- [4] Karthik Jeyabalan and Jerrin Kallukalam. Representation of web graph for in memory computation. May 2005.
- [5] Shantanu Shah. Generating a web graph. May 2005.