Web Research Infrastructure Project
Final Report
Fall 2004

Mayank Gandhi
Karthik Jeyabalan
Jerrin Kallukalam
Ariel Rabkin
Patrick Reilly
Nurwati Widodo

CS490/790 project
Fall Semester 2004
Submitted 12/17/04

Table of Contents

Introduction

Doing large-scale research on the web is a difficult endeavor. Since the thing being examined is spread out across the globe, and the data content of the web varies over time, it is hard to do large experiments.

Thanks to the NSF, Cornell is acquiring a new computer system that will have enough attached storage to store essentially the complete crawl-able content of the web. Indeed, we hope to store several snapshots, taken at different times to facilitate studies of the evolution of the World Wide Web. We can have all the data sitting on local hard disk platters, available to researchers. But how to get the data to Cornell in the first place, and how to expose it to researchers? What follows is the report of a semester's examining these and related questions.

Research Objectives

We interviewed a number of researchers here at Cornell to find out what capabilities would be useful for their research, and to solicit advice about how to build the system.

Researchers Interviewed

- Professor Jayavel Shanmugasundaram
- Professor Johannes Gehrke
- Professor John Hopcroft
- Professor Rich Caruana
- Professor Thorsten Joachims
- Professor Claire Cardie

All of the researchers were keen on having a flexible interface which they would be able to modify and use as needed. Most researchers preferred the interface to be encapsulated in a C/C++ library which they could link against.

Researchers wanted methods to:

- Get a set of documents by URL prefix.
- Get a set of documents by keyword.
- Get the PageRank of a document
- Given a set of pages, find all pages one link away.
- Perform various matrix operations on large link adjacency matrices.
- Get a unique identifier from URL.
- Get the text of a document, given a unique identifier. There was desire for both

the raw data from the server, and also just the plaintext.

Only one researcher required images for their research. Most asked that at least the link-structure information about image files be preserved. Some thought it would be useful to have Internet Archive's usage logs, in some searchable format, keyed to the document being accessed

Some researchers also requested the data in XML format, with their timestamps (HTTP headers preserved). Other suggestions included having a method to extract sets of pages based on URL prefixes, or by root domain name. For querying handling, multi-level boolean query was preferred over a free-text query system. John Hopcroft urged us to set up a comprehensive system so that any additional metadata computed for a document by one research effort could be stored by the system, and used by other projects.

Most researchers refrained from commenting on the number of snapshots they'd use though most asked for at least one initial snapshot for testing purposes.

Capabilities of Theory Center

The machines at the Theory Center that we hope to use are multiprocessor machines, each with 16 64-bit processors running at 1.3GHz. Each computer has 64 GB of RAM. CIT has configured the machines to have an I/O rate 500 MB/s, but the specifications for the machines states that the maximum I/O rate will be 800 MB/s. CIT has the disks divided into 9 partitions, each of of 5.5 TB and the Maximum disk space of 22 TB shared with several projects. There are also robotic tapes available for massive data storage.

Capabilities of Internet Archive

The Internet Archive(IA), located in San Francisco, has been accumulating web-data (including web "snapshots") since 1996. Each web snapshot is between forty and fifty terabytes, and each is produced by a crawl performed and donated by Alexa Internet (www.alexa.com). IA's full collection of web snapshots is currently around four hundred terabytes, and grows at the rate of roughly twenty terabytes per month (a new snapshot is taken every two months). The data at IA is stored at their Data Center on slightly modified, low-end x86 machines, and each machine has a disk capacity of around one terabyte. There are legal limitations concerning exactly what web-data we will be able to access.

Overall Design

The overall design for Web Research Infrastructure project can be divided into several major subsystems:

1. The first subsystem is *Data Transfer*, which transfers data from the Internet Archive to Cornell.

2. The files will then be processed by the second subsystem, *Test Data Set Generation*. This is where the data will be filtered, extracted based on certain features, and saved to trial data set collection.

3. *Research* subsystems take the data from the test data sets and organize the data so that researchers can access it through an API. These subsystems interact with the researchers who are going to exploit the data. Initially, as shown in Figure 1, two Research subsystems are planned: The *Webgraph* subsystem and *Pseudo-web* subsystem.

We will now break down these 4 subsystems to show further details.

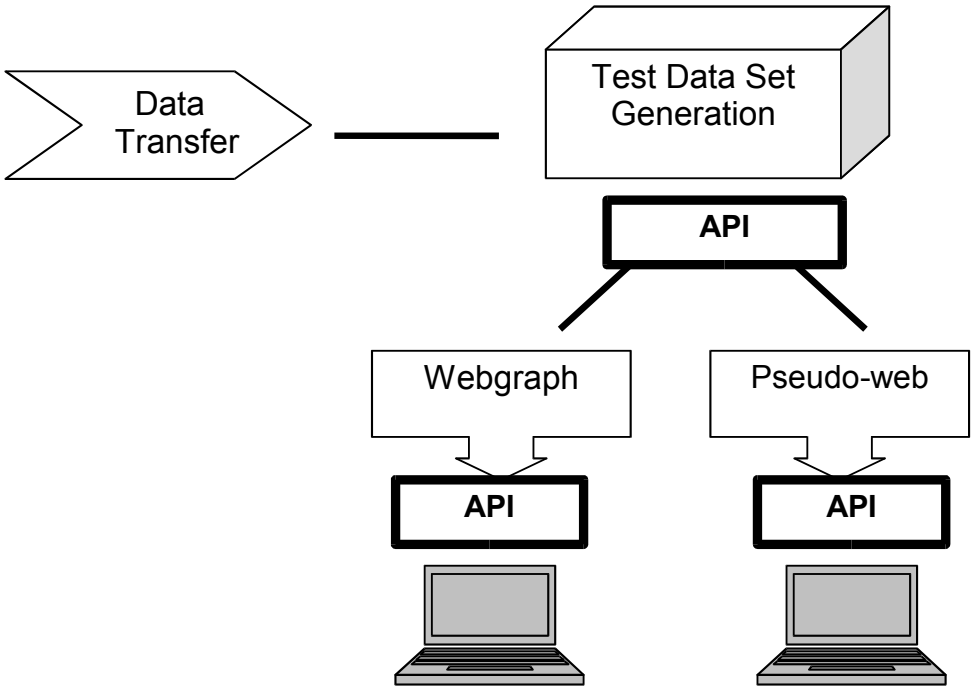


Figure 1. Overall Design with 4 subsystems

Data Transfer Subsystem

The main purpose of this subsystem is to transfer ARC and DAT files that were generated by the Internet Archive Company from their web-crawls. These files have to be transferred to Cornell using some networking strategy able to handle the vast quantity of data. Once the data arrives at Cornell, it is going to be saved to a storage location unmodified.

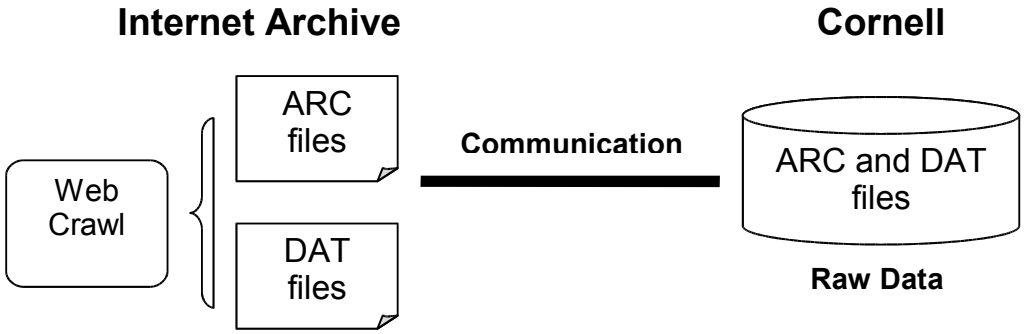


Figure 2. Data Transfer Subsystem

Test Data Set Generation Subsystem

Using the raw data retrieved from the previous subsystem, the ARC and DAT files are going to be filtered based on different characteristics, such as MIME type, domain name, data types, etc. The subset retrieved from this step will go through feature extraction phase where features, such as, URL, anchors, or specific tags is going to be extracted. The resulting files become a trial data set of ARC and DAT files. An API is provided for interactions between the trial data set and the next subsystems.

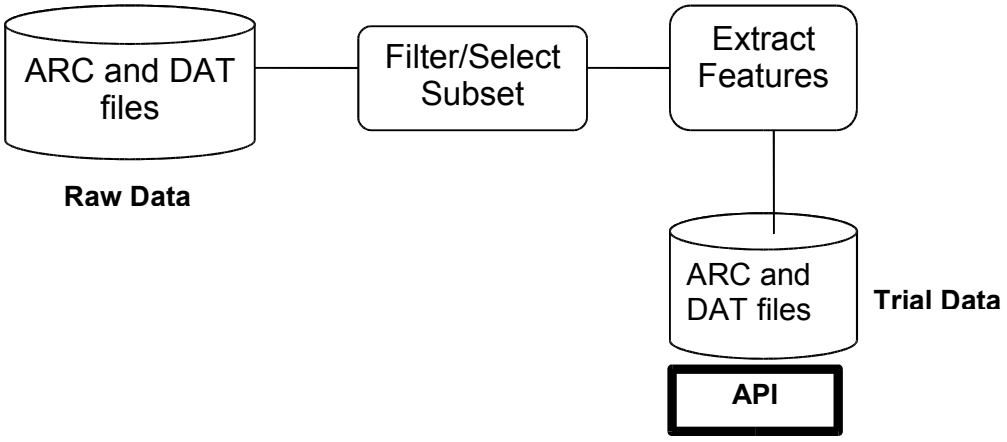


Figure 3. Test Data Set Generation Subsystem

Webgraph Subsystem

This subsystem aims to provide webgraphs to researchers. First, an adjacency matrix is generated from the trial data set. This matrix is then stored in query format. In order for the researchers to get useful information from this, tools and custom code with an API to link against will be provided.

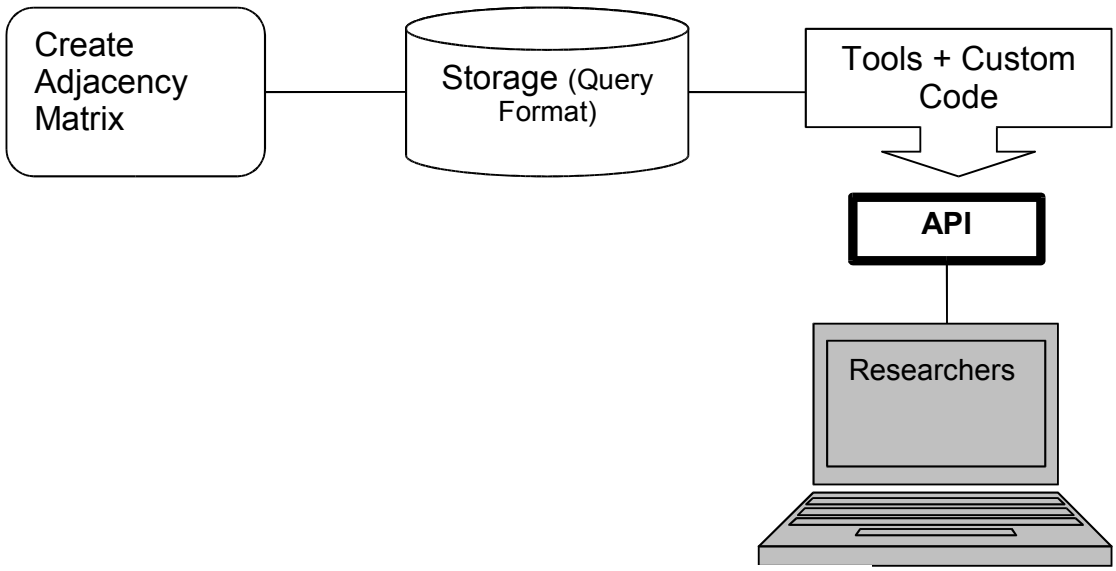


Figure 4. Webgraph Subsystem

Pseudo-web Subsytem

The trial data set can also be used to generate pseudo-web data, with a URL mapping to

the content. Here again, a set of tools and custom code library code will be provided to support researchers who want to use the available information.

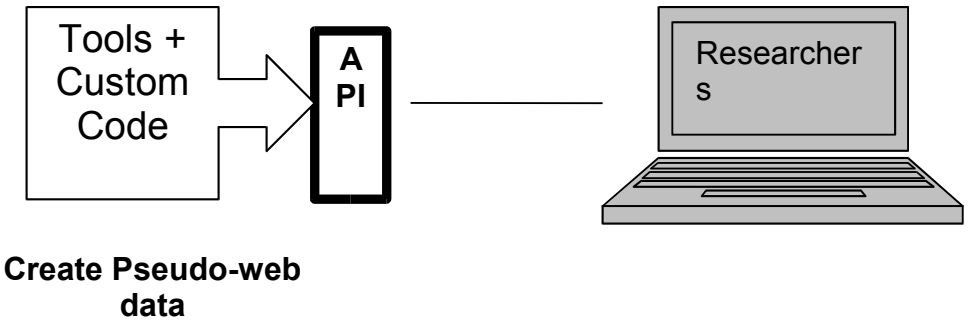


Figure 5. Pseudo-web Subsystem

Data sets and Test data sets

The tools and programs should be tested before the whole collection of data from the Internet Archive arrives. We should obtain a test data set as a representation of the whole set. To measure the performance of out tools and programs, we should get a few hundred GB of data.

During the semester, we used Heritrix to crawl various Cornell domains to generate test data. These crawls were run from the CFS cluster; due to space limitations, the generated ARC files have been deleted. We discovered that in order to generate DAT files using the Internet Archive's tools, Heritrix should be set to create compressed ARC files.

Programming environment

Programming will be done on the computer cluster running Microsoft's Windows Server 2003, Enterprise edition (64bit). It has been decided that the core components of the project will be written in C++ for speed, and that all the tools that will be developed will be made available to everyone (i.e. We will be practicing open source development from the very beginning).

Other Systems

We reviewed several existing large-scale systems that handle HTML-flavored data . Our purpose was to study the models with an eye to the structures and the mechanisms used. We looked closely at Internet Archive and at Google.

Internet Archive

Internet Archive is a valuable example, as their data storage scheme requires a scalable mechanism which can handle the ever-larger monthly crawls.

File Organization

Internet Archive organizes the crawl data in Archive (ARC) format files, each file being around 100 megabytes. (See Section 10 for Data Format). The ARC format is extensible, stream-able, and each file is independent of other files in the crawl. Along

with ARC files there are DAT files associated with them. These files store metadata for the objects in the ARC files. These files are relatively small compared to the ARC files. Each crawl or snapshot is somewhere between 40 to 50 terabytes. The full collection of currently crawled snapshots is about 300-400 terabytes.

Storage

Currently all the data is stored on DLT Tapes and IDE hard drives. Internet Archive refers to the whole structure of PCs storing data as the "Wayback Machine". The storage is straightforward: data is written up to the current device's capacity and then the system moves onto the next device available. The total speed for moving is around 10 terabytes per month.

The Wayback machine is kept up to date on an hourly basis, and handles requests via a load balancing scheme that distribute queries among several machines. These machines hold the indexes (which are closely related to DAT files), which reflect a page's presence in the storage. Once the user selects the URL they wish to visit, the index is searched back on the storage device and the correct document is retrieved. The links within the document are replaced with links with in the archive. This process is fairly efficient and completes within a fraction of a second.

Google

File Organization

Google was another interesting system for study of their file system, as they too store large quantities of indexed web data. We reviewed Google File System, which uses a custom scalable cluster structure for storing the large database, which holds a cache of the web along with the indexing.

Each system (storing commodity system) stores around a million files which are each around 100 MB. There are many large sequential writes to these files which are very rarely updated. There are well defined semantics specifying how multiple clients concurrently append to files. These files are used as producer-consumer queues or many-way merging. Files are divided into chunks, where each chunk is identified by its 64 bit handle assigned during chunk creation.

The Interface of the system allows files to be organized hierarchically in directories identified by their paths. Google file system has "*snapshot*" and "*record append*" operations: *snapshot* creates copies of the files, and *record* allows atomic concurrent append.

Storage

Storage is on done on inexpensive commodity hardware that will often fail. The systems are equipped to detect and tolerate common failures. Read operations on these files are usually done in one of two ways, *large streaming reads* (individual or successive operations from the same client) and *small random reads* (small batches of reads usually within few KB).

The basic architecture employs clusters, single master and multiple chunkservers. These are usually Linux machines running user-level processes. Chunks are stored on the chunkservers on local disks as Linux files. The read/write operations on performed using the chunk handle. For reliability, these chunks are replicated on multiple similar chunkservers (by default, there are 3 copies). The master chunkserver maintains the metadata for the chunks, e.g. location, mapping data, etc. This is similar to what the index files for the Internet Archive do.

Google does indexing of terms within Office files (.doc, .ppt, etc) and PDF files. This currently is not done by Internet Archive, but might be worth considering for our pre-processing before we build an index.

Data Format

The Internet Archive generates two kinds of files that concern us. The first are ARC files, which store the indexed content, and the second are DAT files, which present information about items in a convenient format.

An ARC file consists of a version block, followed by ARC records. Each ARC record is a one line header, followed by the complete HTTP [or other] transaction. Thus, all the HTTP headers are included, unparsed, in the ARC record. Internet Archive's ARC files are around 100 MB each, and are compressed with Gzip in a somewhat peculiar way: each record is separately compressed. See the Heritrix source code for details.

A DAT file stores tagged metadata for a corresponding ARC file. Each DAT record consists of a URL, followed by a list of metadata elements, one per line, introduced with a single character tag specifying the content of that line. For instance, a DAT record lists all the links out from a given web page, the length, and so forth.

Internet Archive's documentation is online at
http://www.archive.org/web/researcher/data_available.php .

Transfer of Data

ARC and DAT files (as previously described) are stored in a compressed format at IA and the Web Research Infrastructure Project is concerned with three different sets of data for each web snapshot. Specifically, one subset of data will provide sufficient information to create a web-graph, while another larger subset will contain all the textual data. The third data set will include the entire snapshot. The size estimates for these sets, the web-graph, textual content of a snapshot, and the entire snapshot are 2.5 TB, 3-5TB, and 35 TB, respectively. The webgraph number represents uncompressed size, the other two figures are post compression.

The overall size of each data transfer can be reduced by removing specified unneeded MIME types, such as binary data, from each ARC file and rebuilding the DATs. After all feature extraction, if any, is completed, the compression rate will be governed by the compressibility of the remaining data formats.

Two general approaches have been investigated for transferring data from Internet Archive to Cornell: long-haul networking and copying the data to disks at the IA Data Center and then shipping the disks via commercial freight service. Although network transport is expected to be more convenient at both ends of the transfer, the time required to establish the desired network connection will likely cause the first transfer of data to be accomplished by shipping disks.

Disks

The costs associated with this option include the price of the disks (including the costs to have them shipped directly to Internet Archive from the vendor), the time required to copy the data to disks, and the shipping costs to have the disks delivered to Cornell.

The disks used for cost estimation are 160 GB disk drives (USB and FireWire compatible) at around $150 per drive. Therefore, the price of disks is estimated at roughly $1000/TB (actual calculations can be found in Appendix C). The time required to copy the data to disks will be governed by two main factors: the networking at IA (the data cannot be pulled directly from the machines of the Data Center, it must be pulled over their gigabit network), and the write speed of the disks used. While the product specification lists a maximum transfer rate of 41 MB/s, Jon Aizen at IA has indicated that we should realistically expect an estimated network transfer rate of 10 MB/s based on the performance observed on similar operations at IA. Consequently, at an average of 10MB/s, transferring 16 TB onto disks would take around twenty days (this calculation is given in more detail in appendix C).

The cost of shipping the disks will depend on the dimensions, weight, and number of the boxes used, as well as the insurance we choose to purchase. Based on the aforementioned 160 GB drives used for our estimates, shipping 16 TB from IA to Cornell

would require 103 disks, packed into five padded boxes totaling 220 pounds and the cost of shipping would be around $970 (shipped as "Computer Hardware" via FedEx freight). As a result, the total estimated cost to transfer 16 TB (including purchasing disks and shipping to Cornell) is roughly $17,000.

Network

Numerous networking alternatives have been considered and investigated. The two most promising options are Internet2 and the National Lambda Rail (NLR).

Internet2

Current university policies impose a network usage fee of one dollar per 500 MB, which works out to more than two thousand dollars per terabyte transferred. Based on this rate schedule, (over thirty thousand dollars for fifteen terabytes), top-quality service via Internet2 is prohibitively expensive for this project. However, our discussions with Cornell Information Technology (CIT) indicate that this project is eligible to apply for a fixed-rate arrangement that will allow us to use otherwise unused Internet2 bandwidth for a flat fee.

Cornell's current connection to Internet2 is a 155 Mbps (~19 MB/s) OC3 pipe to New York City, where larger-scale infrastructure is in place. Based on analysis of Cornell's Internet2 traffic, usage rarely reaches 4 MB/s, leaving more than 85% of the bandwidth unused on average. There are also plans to establish a larger-capacity fiber network link from Cornell to New York City, which is projected to increase the bandwidth capacity to 2500 Mbps (more than fifteen times the current capacity).

In order for data transfer to occur over Internet2, Cornell will need to provide a letter of support to Internet2 on behalf of IA, and IA must submit plans to detailing how exactly they will connect to the network. IA is finalizing the details of this plan.

Based on traffic analysis of IA's commodity gigabit fiber connection, in conjunction with comparative statistics from Cornell's Internet2 interface, it is expected that performance will primarily be governed by the limitations of bandwidth on Cornell's side of the transfer (at least until the connection to NYC is upgraded). Communications with members of the CIT Network Operations Center (NOC) regarding reasonable performance expectations for this alternative suggest that a transfer rate of 6 MB/s to 8 MB/s should be attainable. Jon Aizen has confirmed that other institutions have experienced similar performance connecting to IA via Internet2. Table 1 shows the time required to transfer one terabyte at various different transfer rates and further details are found in Appendix B.

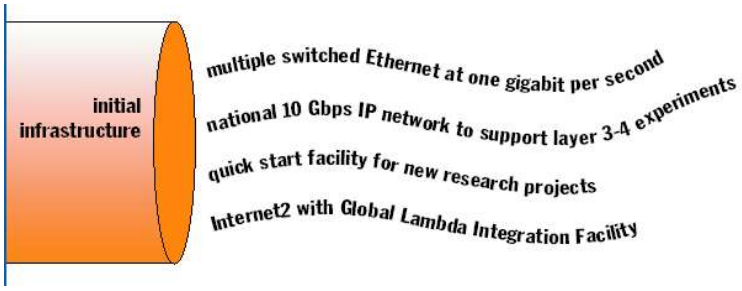
Table 1

TB	Transfer Rate	Hours for Xfer	Days for Xfer
1	6 MB/s	48.55	2.02
1	10 MB/s	29.13	1.21
1	14 MB/s	20.81	0.87

National LambdaRail (NLR)

The NLR is a nationwide fiber-optic infrastructure that is poised to greatly increase the networking power of research institutions throughout the country. It is the result of an initiative sponsored by major research universities and private companies such as Cisco Systems. Based on current schedule projections, Cornell expects to have access to the NLR by around March or April of 2005. During its initial stage of operation, NLR will support 4 "wavelengths," including a "quick start" wave that will serve as a "facility for new research projects in support of data- and computation-intensive science projects…" (as shown in Figure 1 and described in further detail in Appendix B). Consequently, if this project successfully attains a quick start wave via NLR, we would essentially have access to a gigabit Ethernet (layer 2) channel from Cornell to California.

Figure 1: NLR's 4 Initial Wavelengths



The complications in performing transfers over NLR are associated with finding a participating institution of NLR willing to sponsor IA's connection in California. The Corporation for Education Network Initiatives in California, CENIC, has been suggested as a potential sponsoring entity, and further administrative action is required to explore this opportunity.

The most attractive features of this solution include the expected transfer rates and the minimal estimated costs. According to CIT, Cornell has already paid the NLR participation fee, thus connection to a quick start wave would be free on Cornell's end (if we are successful in applying for one). Although the performance estimates for transfer rates using NLR are theoretical, as the connection does not yet exist, expectations indicate that the capacity of this solution would broaden the potential for a relationship between

Cornell and IA.

Features

Web Graph

The purpose of the web graph and related programs is to provide researchers with tools to analyze the structure of the WWW, and study its evolution over time. They would be able to run complex graph algorithms over the link-structure graph of the web, which would help them discover interesting patterns, collect statistics, make demographic studies and generally increase our knowledge about the Internet. Following paragraphs will explain what a web graph is, the data structure that will be use to store the web graph, and finally the algorithm that is used to construct the data structure.

What is web graph?

The web graph is a representation of the WWW at a given time. It stores the link structure that interconnects the HTML documents in the Internet. Each node in the graph corresponds to a unique webpage or other document, and an edge represents a HTML link from one page to another. The web graph is directed, asymmetric, and very large. It is quite sparse.

Web Graph Data Structure

The subset of the web available to this project, from which web graphs will be created, is estimated to contain about 1.5 billion unique pages, each of them a node in the web graph. The preferred storage data structure has been decided to be an adjacency matrix, in which the row index represents a page and the corresponding row contains the information specifying whether or not the page corresponding to the column index is an out-link of the page corresponding to the row index. This matrix, representing the web graph, will be extremely large and sparse; and we plan to keep the web graph entirely in RAM so it can be accessed rapidly. Hence the implementation has to be highly space efficient. The implementation should also provide tools for performing common matrix operations such as multiplication, inversion, and finding eigenvalues and eigenvectors.

All the URLs will be indexed using hash tables, and a unique number will be assigned to each URL. These unique numbers will be used in the graph representation. The adjacency matrix representation of the web will have a block structure because there tends to be much larger concentration of links between documents that share a common theme. There will be blocks of dense regions and large sparse areas in the graph. To exploit this feature of the web graph, we plan to separate the matrix into blocks and store them independently. The dense blocks can be stored using a Compressed Sparse Row representation, while the empty blocks require no storage. Professor Van Loan suggested

that this would be good representation for doing matrix operations.

The challenge is to order the adjacency matrix in a way that it will create dense blocks within the adjacency matrix. It's also necessary to have a clear definition of size and location for the different dense blocks within the matrix. Then individual blocks have to be compressed and indexed. Using a block structure will incur some computational overhead while accessing elements in the matrix. However, we believe that this method has a much better compression performance that more than compensates for this.

Reordering of the adjacency matrix is done by using the modified version of the Cuthill-McKee algorithm. The Cuthill-McKee algorithm also provides the size and location information of the blocks naturally; it makes some blocks dense by placing all the non-empty elements near the diagonal of the matrix. The modified Cuthill-McKee algorithm is as follows:

Algorithm:

```
Let A_Old be the adjacency matrix corresponding to the in-
links and out-links.
Let U be the set of all row or column number corresponding
to A_Old.
Let S be Vector;
Let S[0] = {a random element in U}.
U = U  - S[0];
i = 0
While U not empty, repeat
     S[i+1] =
     For each u in S[i],
          S[i+1] += {link | link  Outlinks(A_Old,u) s.t.
all y <i+1 link S[y]}
     End For
     If (S[i+1]==)
          S[i+1] = {a random link  U}
     End If
     U = U – S[i+1]
     i = i +1
End While
```

Now, the rows and columns of the adjacency matrix can be reordered in the order of elements in S[i] from i = 0 to |S|-1. The result will be a band matrix, with most of the entries concentrated at the diagonal.

We now construct a block matrix A from the band matrix such that the height of all the blocks in the given row i = |S[i]| and column j = |S[j]|. Block corresponding to matrix A element A_{ij} which is non-empty is stored by using Compressed Sparse Row Storage (CSR). Since A is a band matrix, the block A_{ij} is probably empty when |i – j| is

large. When |i – j| is close to zero, the block is near the diagonal, and there is a high probability that the block is relatively dense. Hence, matrix A is also stored in CSR.

Compressed Sparse Row Storage (CSR)

In this method a sparse matrix is represented by 3 arrays. All non-empty elements in each row are put together and stored in a single array called ValueIndex. Empty elements are not stored. The rows are stored in sequential order, for efficient indexing. Another array of the same size (called ColumnIndex) is used to store the position of each element in ValueIndex array. For example, ColumnIndex[i] stores the column number of the value stored in ValueIndex[i].

Another array called the RowPtr, stores the starting location of each row in the ValueIndex and ColumnIndex arrays. For example, if RowPtr[j] is k, and RowPtr[j+1] is k' then the j'th row in the adjacency matrix is stored in ValueIndex array starting at location k and ending at k'-1.

Elements of each row are kept sorted for faster retrieval. Using CSR, an element in the adjacency matrix can be retrieved in O(log n) where n is the number of elements in a row in ValueIndex (ie.. number of outlinks from the URL).

Experimental Results

The following graph shows the adjacency matrix obtained from a crawl of the www.csuglab.cornell.edu domain. The crawl contained about 3000 unique URLs and provided a sample of the web crawl that we expect to obtain from Internet Archives. The diagrams show the arrangements of the elements before and after modified Cuthill-McKee was applied. We can see that the algorithm reduces the envelope, identifies the empty areas and creates a blocks structure.

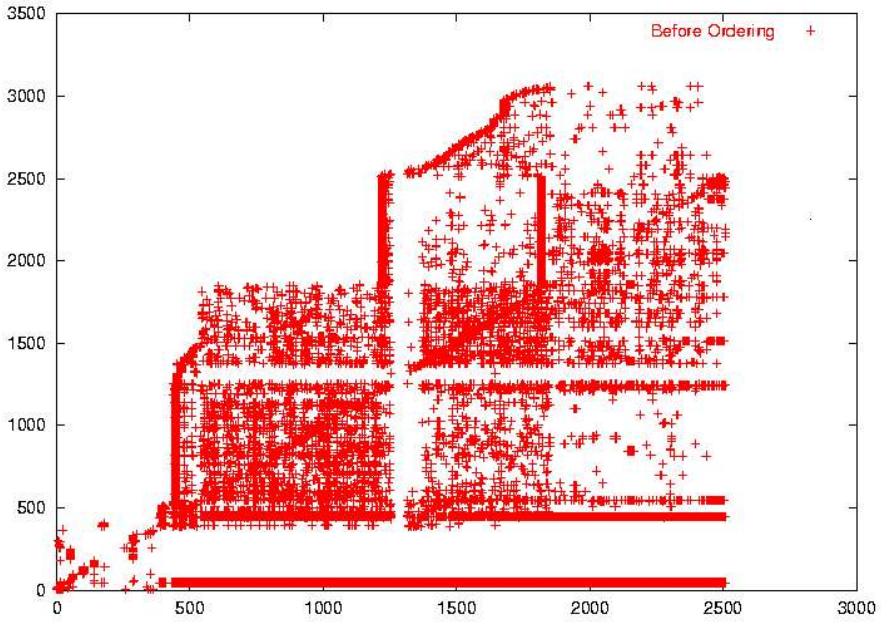


Figure 1: Before applying Cuthill-McKee

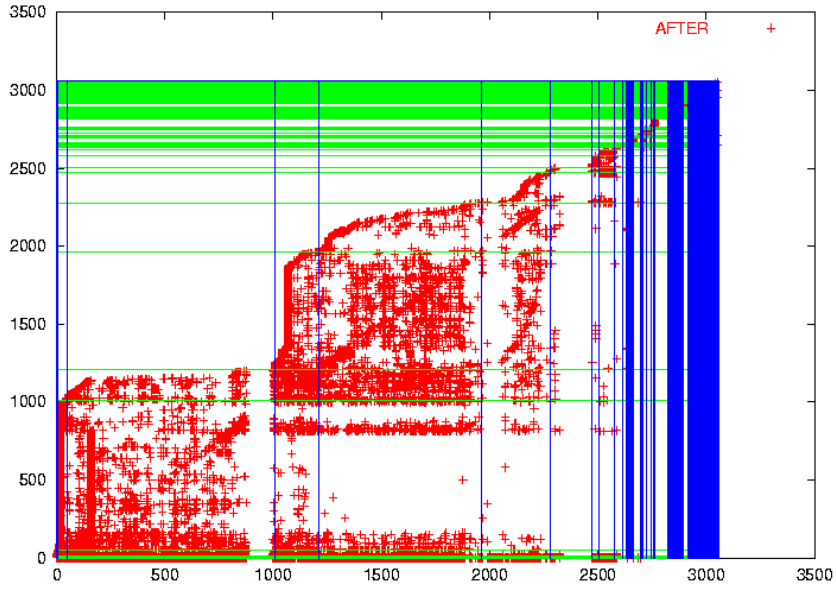


Figure2: After applying Cuthill-McKee, the graph is much more dense and there are well defined blocks, with a predetermined location and size

URL Index

The web graph requires an efficient way to index and store URLs. A hash table is probably the best way to store the URLs for fast access. This hash function maps a URL to a unique number.

There are a number issues involved in building a URL index. For example, there can be many duplicate URLs that points to the same document. Also, a URL may not be unique: we will be storing several different crawls obtained over a period of time. The page a URL points to may have changed in time, so we have to keep track of not merely the URL of a page, but also the date.

A URL contains a domain name and the document name. Since a large number of URLs have common domain names, a separate index can be used to store domain names to optimize storage requirements. URLs contain lowercase alphabetical characters and numbers in addition to a few special characters; they can be hashed using a good text hash function (Eg: MD5) to generate a checksum that is 8 bytes long. The hash table should be stored in the main memory of the machine.

It is common that several different URLs are names for the same page resulting in a large number of duplicated URLs stored in the index. There are also commonly several URLs that point to the different parts of the same HTML page. It might be wise to store a hash of each page to detect if two pages are same. This also allows the system to detect whether the page has changed over time. This will require a sophisticated hash function, which can help us compare two similar web pages and measure the similarity between them. This would be helpful in measuring the extent of the changes when a web page gets updated. This hash function needs to be more sensitive to text data contents.

There is other data that ought to be stored in the URL index. The type of the file a URL points to should be encoded and stored along with the URL. The index should also store the size of the document, and its location on disk if it is in the database.

Data Index by content (quark etc)

Indexing Content: Overview

The data in ARC files will be fetched on a per-URL basis and each URL will be assigned an ID (e.g. its hash). After which a sparse matrix will be implemented for assigning consequent URL's to a webgraph. As the content in the ARC files is not just Text/HTML, a solution is needed where content can be indexed with their URL's (root domain) as the identifiers.

Quark is a Cornell based effort to develop a DBMS based on an XML query language (XQuery). We examined the system's applicability to the type of data and requirements we have.

Filesystem level access to the data stored is possible, and envisioned in the current project road-map. One team member (Fang Yang) is currently writing a Linux wrapper. Though a similar version for Windows is not currently planned, the Quark Team didn't rule out its possibility. Researchers are interested in having direct access to the web data, without having to go through a higher-level API. Therefore such a feature would be a valuable addition to the DBMS.

Quark supports XML input which will be very useful for the type of data the Internet Archive holds. Support for a simple dump and store of arbitrary XML data exists, so Quark should be able to handle HTML without much modification. Though, as currently we have had limited test time an some pre-processing may be needed before the data can be imported to Quark. The data available to us contains binary files apart from HTML, such as PDF files and images. It may be useful to index this content; it would be possible to put the binary data in an XML wrapper, and then store it using Quark This is relatively easy task and an automated script for it can be written easily.

The data can be stored in a hierarchical tree-like structure, where the domain name of each page could be stored as an identifying node. This structure can have multiple levels. This would enable the users to search with queries restricted to domain names and filenames in the crawl. This specific structured data storage would be helpful for various research projects which will be aiming on targeted content.

Figure 2.1 Sample Tree structure

Quark has been tested with large datasets and should perform with similar efficiency on datasets in the several terabyte range. Our aim is to test this in the coming semester, using our current test sample set .

Quark accepts queries in TexQuery. TeXQuery is an extension to the XQuery language. It provides a rich set of fully compose-able full-text search primitives, such as Boolean connectives, phrase matching, proximity distance, stemming and thesauri. It enables users to seamlessly query over both structured and full-text data by embedding TeXQuery primitives in XQuery, and vice versa. It also supports a flexible scoring construct that can be used to score query results based on full-text predicates.

TeXQuery provides a number of features which might be useful for this project. It supports expressions like:

```
FTContainsExpr, FTScoreExpr, FTSelection,
FTContextModifiers, etc
```

***Sample queries**:*

Basic boolean keyword search:

```
getDocument("inex/tk/2002/volume-m.xml")//article/fm//atl
     [. ftcontains (("database" || "digital library") &&
"design") ordered]
```

Same sentence matching:

```
getDocument("fttf/sample.xml")/books/book//chapter
[.  ftcontains  ("user" && "would" && "know" && "step")
same sentence]
```

To advanced iterative scored results:

```
for $elem in getDocument("inex/tk/2002/volume-m.xml")//
article/fm//kwd
let $score := $elem ftscore "database" weight 0.4 ||
"databases" weight 0.3
where $score > 0
order by $score descending
return <result score="{$score}">
```

```
    {$elem}
  </result>
```

The system can support quite a number of optional features. A detailed paper exists, whose link is available in the references section. The Quark team was not very confident about the scoring system currently in place and suggested more work would be needed.

XRANK system is designed to handle the novel features of XML keyword search. The experimental results show that XRANK offers both space and performance benefits when compared with existing approaches. An interesting feature of XRANK is that it naturally generalizes a hyperlink based HTML search engine such as Google. XRANK can thus be used to query a mix of HTML and XML documents.

Quark is not a packaged product as yet, but core components of the system are in place and are being constantly upgraded and refined. The basic query system has been implemented, though some features, such as a good scoring system, are still being worked out. Quark would be an excellent tool to employ, unfortunately it is not yet mature enough for our purposes.

The Quark team seemed very excited on the interest from us (Web Research Infrastructure), as it would provide relative testing grounds for the system. The quality of the system is quite high; its immediate availability is in doubt.

Tools

The Internet Archive has developed a number of tools that can be used to processing data. These tools are freely available and are specifically designed to manipulate text in ARC files. We will need to use these tools if we implement a XML based data storage mechanism such as Quark. A number of tools need to be implemented, however.

Tools in the ARC handling pipeline:

- A tool to extract a document from ARC files, given a unique ID.
- A tool to iterate over all documents in an ARC file.
- A tool to filter content in ARC files by MIME type
- We may need to write our own program to generate DAT files from ARC files. This would be useful if we alter IA's ARCs, for instance by removing images. We have a linux binary of an IA tool to do this, but source code is unavailable.

Tools for the Web Graph

- A tool to extract link structure from DAT files. (A working prototype of this exists)
- A tool to put the link structure in to an adjacency matrix format. (A working prototype

of this exists)
- Tools to manipulate large matrices in memory.
- Tools for creating and querying the URL index.

Search and Retrieval
- A program to generate pagerank information for each document.
- A tool to find all documents with a given term, using some ranking method.
- Tools to generate various statistics from the ARC files.

Program Status

Things Completed:
- Procedure for reading data from the file and putting into a CSR Matrix.
- Procedure for converting CSR Matrix into Block Matrix with Cuthill-McKee applied.
- Implementation of a modified Cuthill-McKee algorithm that works for number of URLs below 2^32 – 2 (this may be 2^64-2 when compiled for a 64bit machine).

Tasks Remaining:
- The software has to be modified to take advantage of the memory and parallelism available in the multiprocessor system.
- Currently the code depends on the 'new' operator allocating the needed amount of memory, this might not be the case when the data has more than a billion URLs.
- Implement the functions to access values form the matrix when the matrix is Block matrix, currently it returns the block corresponding to the index.

Future Plans

In order to implement the whole system, we should start by establishing a proficient team; we will need to follow the steps below.

- Recruit a team of student programmers
- Train the team
- Create a collection of documents for user documentations, system documentations, etc.
- Set up a source control system to ensure transferability of the written program for the other teams who is going to continue the project.
- Agree on technical framework, including programming language and database system.

To organize the tasks for this project, we can break it down into several parts based on the following guideline:

- Break the subsystems (Data Transfer Subsystem, Test Data Set Generation Subsystem, Webgraph Subsystem, and Pseudo-web Subsystem) into more specific elements.
- Decide on file formats and how to store them.
- Estimate the size of data (determine the storage size).
- Write various program/tools to do feature extraction, filtering, adjacency matrix creation, and pseudo-web creation.

To give a rough idea of the task breakdown, the whole can be divided into 30-40 smaller tasks. A suggestion on how to divide tasks for each subsystem:

1. Approximately 5-7 tasks for Data Transfer Subsystem
2. Approximately 10 or more tasks for Test Data Set Generation Subsystem
3. Approximately 5-7 tasks for Webgraph Subsystem
4. Approximately 5 tasks for Pseudo-web Subsystem

Conclusion

We believe that it is eminently practical to build an infrastructure for web-related research at Cornell using the new capabilities of the Theory Center; we believe that a useful infrastructure could be established within the next 12 months. The basic parameters of the research infrastructure has been established: what its capabilities ought to be, and what the design trade-offs are. There are a number of outstanding questions left to be resolved, such as whether to transfer the data via disk or via network, and whether to trim out binary content from the data. Once these questions are resolved, the work can be pressed ahead quite rapidly.

References

On Internet Archive

- Internet Archive: www.archive.org
- Alexa Internet: www.alexa.com
- Mike Burner, Brewster Kahle, WWW Archive File Format Specification, Version 1.0, 1996, Internet Archive
- Richard Koman, How the Wayback Machine Works, 2002, webservices.xml.com

On Quark:

- XQuery 1.0 An XML Query Language: http://www.w3.org/TR/xquery
- Quark Project Homepage: http://www.cs.cornell.edu/database/quark
- Demonstration: Mr. Chavar Botev, http://www.cs.cornell.edu/~cbotev
- L. Guo, F. Shao, C. Botev, J. Shanmugasundaram, "XRANK: Ranked Keyword Search over XML Documents", SIGMOD Conference, 2003.
- L. Guo, J. Shanmugasundaram, K. Beyer, E. Shekita, "Efficient Inverted Lists and Query Algorithms for Structured Value Ranking in Update-Intensive Relational Databases", ICDE Conference, 2005.
- F. Shao, A. Novak, J. Shanmugasundaram, "Triggers over XML Views of Relational Data", ICDE Conference (poster), April 2005.
- S. Amer-Yahia, C. Botev, J. Shanmugasundaram, "TeXQuery: A Full-Text Search Extension to XQuery", WWW Conference, May 2004.
- J. Qiu, F. Shao, M. Zatsman, J. Shanmugasundaram, "Index Structures for Querying the Deep Web", WebDB Workshop, June 2003.
- TeXQuery Project Homepage:www.cs.cornell.edu/database/TeXQuery

Other:

- Sanjay Ghemawat, Howard Gobioff, Shun-Tak Leung, *The Google File System*, p.29-43, 2003, Bolton Landing, NY, United States
- John Bent, Douglas Thain, Andrea C. Arpaci-Dusseau, Remzi H. Arpaci-Dusseau, and Miron Livny. *Explicit Control in a Batch-Aware Distributed File System*. Computer Sciences Department, University of Wisconsin, Madison.
- National Lambda Rail. http://www.nlr.net/files/200311-brochure.pdf
- The Heritrix Crawler: http://crawler.archive.org

Appendix A: Researcher Interviews

Professor Jayavel Shanmugasundaram

API: Nothing unusual needed; way to get page based on URL, way to get properties of a given page, and keyword search.

Pre-Processing: Images required in the first phase. Text and Link structure both very necessary for initial research. Data in XML is preferred. No special pre-processing required.

Usage Logs: It would be nice to have indexed access logs, so you can quickly reconstruct when a specific page was accessed.

Search Capabilities: Simple queries suffice

One snapshot would be enough.

Professor Johannes Gehrke

API: His major emphasis was on developing a powerful API (preferably in C++). The API should serve as an adjunct solution to direct handling of the data. Allow some provision to create Sub-graphs, e.g. providing a copy of a website if its URL prefix is provided.

Pre-Processing: Images not required in the first phase though would be preferred at a later stage. Text and Link structure both very necessary for initial research.

Usage Logs: They are a must for his research.

Search Capabilities:

- Free Text Search
- Implementation of a Google (or similar) PageRank algorithm.
- Capability to provide searches on basis of URL prefix
- Indexing: desired

Caching Doesn't suggest during the initial stage though it may be implemented if need arises.

Professor Gehrke will be requiring a constant flow of snap shots. Remote access to the server is not a problem (need not be localized). Apart from an API interface, it is desirable to store data in a format that can be accessed directly, too.

Professor John Hopcroft

API: Professor Hopcroft commented that it would be good to have a way to store information about pages computed by other researchers: e.g., summaries, etc. Have this keyed to page ID, easily recovered.

He thought it would be wise to have various configurable ways to condense documents; this would involve a whole range of filters, from trimming out HTML tags to just returning a list of 20 keywords per documents.

C preferred to C++.

Pre-Processing: Images not required; Text and link structure both needed.

Usage Logs: Logs of the original documents would be nice.

Search Capabilities

- Free Text Search
- Implementation of a Google (or similar) PageRank algorithm.

Professor Hopcroft thought snapshots spaced one per year, would be most useful. Also, it would be good to have a central listing of what researchers are generating what data, or have written what tools.

Professor Rich Caruana

API:

- A way to extract bags of words [with frequencies] from a given document
- A way to get link structure matrix for an arbitrary subgraph
- Way to get indegree, outdegree, length, and HTML heading text for a given document.

Pre-Processing: Images not required. Timestamps would be useful, perhaps even for text within document, if possible.

Usage Logs: Not needed.

Search Capabilities

- Free Text Search, no need for ranking
- Would be nice to have PageRank
- Capability to provide searches on basis of URL prefix

Caching: Doesn't suggest during the initial stage though it may be implemented if need arises.

Professor Caruana has no immediate intention of using the web infrastructure, and hence no definite sense what would be useful.

Professor Thorsten Joachims

API: Be nice to be able to find the PageRank for a given page. It would be good to have a method which, given a subgraph, returns the set of pages in that subgraph, or one step away in either direction.

Perl a first choice, followed by C++

Pre-Processing No need for images. Text needed, also graph information.

Usage Logs It would be useful to have logs of search queries entered at IA.

Search Capabilities

- Simple keyword search, no need for ranking
- Implementation of a Google (or similar) PageRank algorithm.
- Would be good to have a robust index, with frequencies. Perhaps detailed postings information for each word, or way to generate same on the fly.

Professor Claire Cardie

API: She would prefer APIs in Java though she is adjustable with any other equally capable system also.

Pre-Processing Images not required. Text and Link structure required but might be limited to a small subset of the data only.

Usage Logs Not required.

Search Capabilities

- Standard NLP based Search
- Capability to provide searches on basis of URL prefix
- Indexing: no special indexing required

Caching

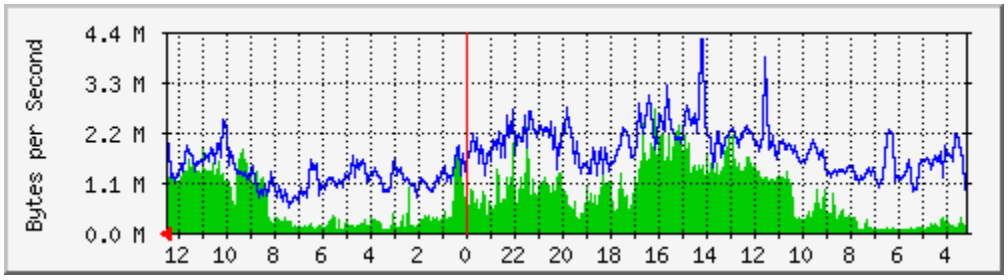
Doesn't suggest during the initial stage though it may be implemented if need arises.

Requirement for several snapshots as her research requires comparison of datasets. Professor Cardie currently doesn't have all the details of all the requirements from us for her research project, but emphasized that it would be on similar lines as mentioned above.

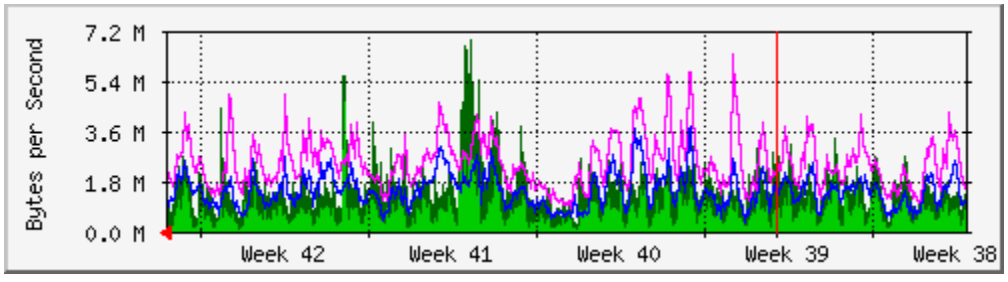
Appendix B: Calculation of Time and Costs for Network Transfer

We are hoping that the long-term networking solution for this project will be either Internet2 or NLR, with the former being more likely. While we are going through the process of trying to get things worked out with connecting Internet Archive to Internet2, we should be concerned with the size of our current connection to Internet2. Cornell's OC3 connection (155Mbps) to Internet2 travels to NYC, where larger-scale infrastructure is in place. There is a project "underway" to run a larger fiber network pipe down to NYC, but this will likely not be completed until April or May of next year. While this upgrade is likely to be considerable, (~OC48: 2500Mbps, [+]15x current) the real completion date is unknown and we would like data before then.

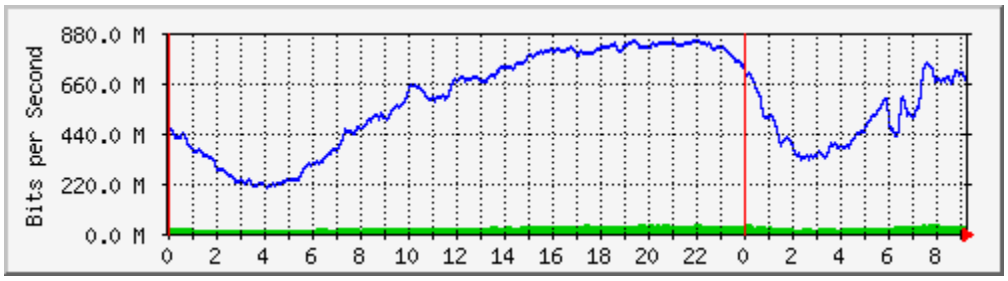
Therefore we are limited by the current OC3, which is around 19 MB/s. The statistics from the traffic analysis shows that traffic seldom runs over 4MB/s, with average use of less than 15% of total bandwidth. Figure 1 shows a **daily** graph from Tuesday October 26. Figure 2 shows a similar graph for the most recent **month**.

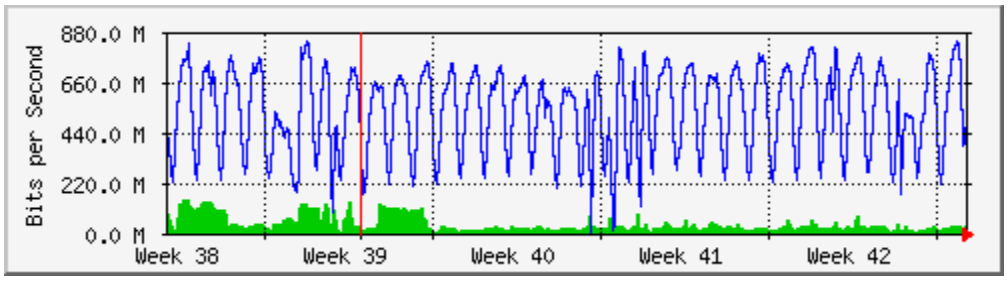


The red line represents midnight the previous night, and the red arrow at bottom left indicates that current time is going left (ie: the data on the right is older than on left).



On the other side of the coin, the Internet Archive's traffic analysis is shown in Figure 3 (daily) and Figure 4 (monthly).





The time representation for IA's traffic analysis is slightly different, as the red arrow at bottom right indicates current time extends to the right (ie: older data is to the left).

The Internet Archive's connection is more than six times as large as CU's, with the following conversions to megabytes per second: IA ~120 MB/s, CU ~18.5 MB/s. Based on the traffic analysis, IA's "off-peak" hours are from around 11pm – 10am PDT (2am – 1pm here).

Depending upon the agreement we will theoretically arrange with CIT to schedule "off-peak" hours for our use of I2, the amount of bandwidth we are able to secure on this end will likely be the primary influence on the transfer rate. Table 1 shows time calculations for different transfer sizes and rates.

Filesize	

TB's	MB's	Xfer Rate	Transfer Rate per Hour (MB)	Hours for Transfer	Total days *straight*
40	41943040	6 MB/s	21600	1941.81	80.91
30	31457280	6 MB/s		1456.36	60.68
8	8388608	6 MB/s		388.36	16.18
2.5	2621440	6 MB/s		121.36	5.06
40	41943040	10 MB/s	36000	1165.08	48.55
30	31457280	10 MB/s		873.81	36.41
8	8388608	10 MB/s		233.02	9.71
2.5	2621440	10 MB/s		72.82	3.03
40	41943040	14 MB/s	50400	832.20	34.68
30	31457280	14 MB/s		624.15	26.01
8	8388608	14 MB/s		166.44	6.94
2.5	2621440	14 MB/s		52.01	2.17
40	41943040	25 MB/s	90000	466.03	19.42
30	31457280	25 MB/s		349.53	14.56
8	8388608	25 MB/s		93.21	3.88
2.5	2621440	25 MB/s		29.13	1.21

Appendix C: Calculation of Time and Costs of Disk Transport

In order to estimate the costs associated with this transfer, we have identified a disk drive made by LaCie which has the following specifications:

Capacity:	160 GB
Cost:	$153.00 (includes shipping)
Max transfer rate:	41 MB/s
Compatibility:	FireWire / USB 2.0
Size:	1.4 x 4.4 x 7.4 inches
Weight:	900g
Shipping:	$3 per disk

Quantity and Costs of Disks:

Data Set TB's	Number of disks	Cost ($)
1	7	1,071
5	32	4,896
14	90	13,770
16	103	15,759
30	192	29,376
32	205	31,365

II. The Transfer:

When pulling the data from IA's machines onto our disks, the speed of the transfer will be governed by two primary factors: the network at IA (cannot pull directly from machines, must pull over their gigabit network), and the write speed of the disks.

From discussions with Jon Aizen, the requirement of transferring data from the Data Center over their intranet is unavoidable, and is likely to be at a rate of around 10 MB/s (max is around 15 MB/s). (On a side note, Jon has also explained that he has found the successful write speed of his FireWire drive is around 2 MB/s). We expect that, since the transfer rate advertised by LaCie is over 40 MB/s, that the drives will realistically write at least 10 MB/s (in which case the time of the transfer will be governed by how fast the data can be pulled over the network).

Estimated time costs for the transfer are shown in the following table.

Size of Data Set (TB)	MB's	MB/hour from -->	Actual transfer rate (from Data Center to disk) in MB/s	Time (Hrs)	Days
5	5242880	36000	10	145.6	6.1
16	16777216	36000	10	466.0	19.4
30	31457280	36000	10	873.8	36.4
32	33554432	36000	10	932.1	38.8

Transport to Cornell:

The shipping costs will be dictated by the size and weights of the disks when they are packaged for shipment, plus any insurance we purchase.

The sizes and weights of different shipment sizes are outlined in the following table.

TB's	Disks	Weight of shipment (Kg)	Pounds
1	7	7	15.4
5	32	29	63.8
14	90	81	178.2
16	103	93	204.6
30	192	173	380.6
32	205	185	407

Shipping:

The pertinent shipping information for shipping 16 TB via FedEx follows:

Box Size: 36 x 30 x 12 in (each disk is 1.5 by 4.5 by 7.5 inches)
Disks per box: 21
Layers: 3
Padding: Almost half the dimension of the box
Weight of Disks/box: 18 Kgs or 39.6 pounds
Estimated Padding weight and Box weight: 2 Kgs or 9.68 pounds
Class 250 of FedEX Freight, its the Computer Hardware category

Therefore 16 TB we require 103 disks in 5 boxes totaling:
Weight: 100 Kgs or 220 pounds
Inside University Pick charge: 61 USD

Send Zip Code: 94129
Receive Zip Code: 14850

Cost of Shipment: 912 USD

Total Cost: 973 USD
Transfer time: 4 Transit Days

Shipping cost for 16 TB: $ 973.00

So the total cost for 16 TB (disks & shipping) is roughly **$17,000** (16000 + 980)

Appendix D: Heritrix

The ARC data at Internet Archive is the product of web crawls conducted by Alexa Internet, with Alexa's proprietary crawler. Internet Archive developed their own crawler, Heritrix, which is written entirely in Java, and which is entirely open source. Heritrix is of interest to this project in three ways: First, as a source of test data, second, as a source of code, and third, for documenting the ARC format.

Heritrix has very high performance on small and medium-size websites: in test crawls of *.cornell.edu run from machines on the campus network, performance was limited only by the available network connection. Heritrix can be configured to be as polite or rude as desired, via a straightforward web interface; the interface is well documented and simple to use. It is possible that Heritrix would consume excessive memory if asked to crawl all of a large domain such as cornell.edu; it should work well on smaller domains. This means that Heritrix would be ideal for generating test data. For compatibility, it should be set to output compressed ARC files.

Heritrix is open source, which means that we can crib useful blocks of code from the source; in particular, the sections handling ARC files may be of use to us. The algorithms for handling web-graph data may also be of use.

Heritrix outputs data in ARC format, and since it was developed at IA, we can use the output component of Heritrix as a reference implementation of the ARC format.