Full Text Search of Web Archive Collections

MICHAEL STACK

Internet Archive

The Presidio of San Francisco

116 Sheridan Ave.

San Francisco, CA 94129

stack@archive.org

The Internet Archive, in cooperation with the International Internet Preservation Consortium, is developing an open source full text search of Web archive collections. Web archive collection search presents the usual set of technical difficulties searching large collections of documents. It also introduces new challenges often at odds with typical search engine usage. This paper outlines the challenges and describes adaptation of an open source search engine, Nutch, to Web archive collection search. Statistics and observations indexing and searching small to medium-sized collections are presented. We close with a sketch of how we intend to tackle the main limitation, scaling archive collection search above the current ceiling of approximately 100 million documents.

Web archives, full text search, Nutch

1. Introduction

The Internet Archive (IA) is a 501(c)(3) non-profit organization whose mission is to build a public Internet digital library [1]. Since 1996, the IA has been busy establishing the largest public web archive to date, hosting over 600 terabytes of data. Currently the only public access has been by way of the IA Wayback Machine (WM) [2] in which users enter an URL and the WM displays a list of all instances of the URL archived, distinguished by crawl date. Selecting any date begins browsing a site as it appeared then, and continued navigation pulls up nearest-matches for linked pages. The WM suffers one major shortcoming: unless you know beforehand the exact URL of the page you want to browse, you will not be able to directly access archived content. Current web URLs and published references to historic URLs may suggest starting points, but offer little help for thorough or serendipitous exploration of archived sites. URL-only retrieval also frustrates users who are accustomed to exhaustive Google-style full text search from a simple query box. What is missing is a full text search tool that works over archived content, to better guide users 'wayback' in time.

The International Internet Preservation Consortium (IIPC) is a consortium of twelve National Libraries and the Internet Archive [3]. The mission of the IIPC is to acquire, preserve and make accessible knowledge and information from the Internet for future generations everywhere, promoting global exchange and international relations.

In December of 2004, the IA and IIPC began collaboration on a search tool that could navigate the historic Web. After experiment and consultation, Nutch [4], "...an open-source Web search engine that can be used at global, local, and even personal scale" [5], was selected as the search engine platform on which to base WAC search. Technically, Nutch provides basic search engine capability, is extensible, aims to be cost-effective, and is demonstrated capable of indexing up to 100 million documents with a convincing development story for how to scale beyond to billions [6]. Just as importantly, policy-wise, core attributes of the Nutch project align with the mission of the IIPC. A founding goal of the Nutch project is to provide a transparent alternative to commercial web search engines. One of Nutch's signature features is the ability to explain its result rankings.

This paper outlines the current state of WAC search development. It begins with a listing of challenges searching WACs. This is followed by an overview of Nutch operation to aid understanding of the next section, a description of the initial Nutch adaptation and additions made to support WAC search. Statistics on indexing rates, index sizes, query times, and hardware are presented as well as observations on the general WAC indexing and search procedure. We close the paper with a sketch of future developments focusing on how we intend to tackle scaling up to index collections of one billion documents and beyond.

2. Challenges Searching Web Archive Collections

WACs tend to be large. A WAC usually is an aggregate of multiple, related focused Web crawls run over a distinct time period. For example, one WAC, made by the IA comprises 140 million URLs collected over 34 weekly crawls of sites pertaining to the United States 2004 Presidential election. Another WAC is the complete IA repository of more than 60 billion URLs. (Although this number includes exact or near duplicates, the largest live-Web search engine, Google, only claims to be "[s]earching 8,058,044,651 web pages" as of this writing.) WACs are also large because archives tend not to truncate Web downloads and to fetch all resources including images and streams, not just text-only resources.

A single URL may appear multiple times in a WAC. Each instance may differ radically, minimally or not at all across crawls, but all instances are referred to using the same URL. Multiple versions complicate search query and result display: Do we display all versions in search results? If not, how do we get at each instance in the collection? Do we suppress duplicates? Or do we display the latest with a count of known instances in a corner of the search result summary?

A WAC search engine gets no help from the Web-at-large serving search results. What we mean by this is that for WAC searching, after a user clicks on a search result hit, there is still work to be done. The search result must refer the user to a viewer or replay utility – a tool like the IA WM – that knows how to fetch the found page from the WAC repository and display it as faithfully as possible. (Since this redisplay is from a server other than the page's original home, on-the-fly content rewriting is often required.) While outside of the purview of collection search, WAC tools that can reassemble the pages of the past are a critical component in any WAC search system and will receive further attention below.

Finally, an interesting question arises when considering search engine link-text and link-structure influenced page ranking in WACs. Inlinks and outlinks can be time skewed with link connections made across crawls run days, weeks, months, or in some cases, even years apart. How should this time skew be discounted in any ranking calculation?

3. Overview Of Nutch Operation

A brief description of Nutch's normal operation will help contextualize the Nutch WAC search adaptations and additions described later.

The Nutch search engine indexing process runs in a stepped, batch mode. With notable exceptions discussed later, the intent is that each step in the process can be "segmented" and distributed across machines so no single operation overwhelms as the collection grows. Also, where a particular step fails (machine crash or operator misconfiguration), that step can be restarted. A custom database, the Nutch webdb, maintains state between processing steps and across segments. An assortment of parse-time, index-time, and query-time plugins allows amendment of each processing step.

After initial setup and configuration, an operator manually steps through the following cycle indexing:

1. Ask the Nutch webdb to generate a number of URLs to fetch. The generated list is written to a "segment" directory.
2. Run the built-in Nutch fetcher. During download, an md5 hash of the document content is calculated and parsers extract searchable text. All is saved to the segment directory.
3. Update the Nutch webdb with vitals on URLs fetched. An internal database analysis step computes all in-link anchor text per URL. When finished, the results of the database in-link anchor text analysis are fed back to the segment. Cycle steps 1-3 writing new segments per new URL list until sufficient content has been obtained.
4. Index each segment's extracted page text and in-link anchor text. Index is written into the segment directory.
5. Optionally remove duplicate pages from the index.
6. Optionally merge all segment indices (Unless the index is large and needs to be distributed).

Steps 2 and 4 may be distributed across multiple machines and run in parallel if multiple segments. Steps 1, 3, and 5 require single process exclusive access to the webdb. Steps 3 and 6 require that a single process have exclusive access to all segment data. A step must complete before the next can begin.

To query, start the Nutch search Web application. Run multiple instances of the search Web application to distribute query processing. The queried server distributes the query by remotely invoking queries against all query cluster participants. (Each query cluster participant is responsible for some subset of all segments.) Queries are run against Nutch indices and return ranked Google-like search results that include snippets of text from the pertinent page pulled from the segment-extracted text.

4. Nutch Adaptation

Upon consideration, WAC search needs to support two distinct modes of operations. First, WAC search should function as a Google-like search engine. In this mode, users are not interested in search results polluted by multiple duplicate versions of a single page. Phase one of the Nutch adaptation focused on this mode of operation.

A second mode becomes important when users want to study how pages change over time. Here support for queries of the form, "return all archive versions crawled in 1999 sorted by crawl date" is needed. (Satisfying queries of this specific type is what the IA WM does using a sorted flat file index to map URL and date to resource location.) Phase two added features that allow version- and date-aware querying. This phase also added features that would allow modified Nutch to be used as the engine to drive an IA WM-like WAC viewer application.

4.1 Phase one

Because the WAC content already exists, previously harvested by other means, the Nutch fetcher step had to be recast to pull content from a WAC repository rather than from the live Web. At IA and other institutions using the Heritrix web crawler [7], harvested content is stored in the ARC file format [8]; composite log files each with many collected URLs. For the IA, an ARC-to-segment tool was written to feed ARCs to Nutch parsers and segment content writers. (Adaptation for formats other than IA ARC should be trivial.) Upon completion of phase one, using indices purged of exact duplicates (i.e. the deduplicate indexing step was run), it was possible to deploy a basic WAC search that used the IA WM as the WAC viewer application.

4.2 Phase two

To support explicit date and date range querying using the IA 14-digit YYYYDDMMHHSS timestamp format, an alternate *date* query operator implementation replaced the native Nutch YYYYMMDD format. To support retrieval of WAC documents by IA WM-like viewer applications, location information -- *collection, arcname* and *arcoffset* -- was added to search results as well as an operator to support exact, as opposed to fuzzy, URL querying (*exacturl)*. Nutch was modified to support sorting on arbitrary fields (*sort, reverse*) and deduplication at query time *(hitsPerDup, dedupField*). (Deduplication at query time, rather than at index time, is needed so it is possible to write queries against a single non-deduplicated index that return all versions of a resource, or just the latest instance). Here is the complete list of new query operators:

- *sort*: Field to sort results on. Default is no sort.
- *reverse*: Set to true to reverse sort. Default is false.
- *dedupField*: Field to deduplicate on. Default is 'site'.
- *hitsPerDup*: Count of *dedupField* matches to show in results. Default 2.
- *date*: IA 14-digit timestamps. Ranges specified with '-' delimiter between upper and lower bounds.
- *arcname*: Name of ARC file that containing result found.

- *arcoffset*: Offset into *arcname* at which result begins.
- *collection*: The collection the search result belongs to.
- *exacturl*: Query for an explicit url.

Natively Nutch passes all content for which there is no explicit parser to the text/html parser. Indexing, logs are filled with skip messages from the text/html parser as it passes over audio/*, video/*, and image/* content. Skipped resources get no mention in the index and so are not searchable. An alternate parser-default plugin was created to add at least base metadata of crawl date, *arcname*, *arcoffset*, type, and URL. This allows viewer applications, which need to render archived pages that contain images, stylesheets, or audio to ask of the Nutch index the location of embedded resources. Finally, an option was added to return results as XML (RSS) [10]. Upon completion of phase two, both modes of operation were possible using a single non-deduplicated index.

All WAC plugin extensions, documentation, and scripts are open source hosted on Sourceforge under the *Nutchwax* -- Nutch with Web Archive eXtensions -- project [11].

5. WAC Search Operation

Indexing an IA WAC follows the general form of the Nutch indexing process outlined above with modified setup and fetcher steps. Setup consists of bringing ARCs local or NFS mounting collections onto the indexing machine and the fetcher step comprises the feeding of WAC ARCs to the ARC-to-segment tool.

To illustrate query operation, below we compare a query for *thomas jefferson* made September, 9 2005 on Google – confined to the .gov domain -- to a Nutch query run against a WAC made of .gov crawls from late 2004. (Near a year separates the Nutch and Google harvests). Only the first 8 hits from each are shown.

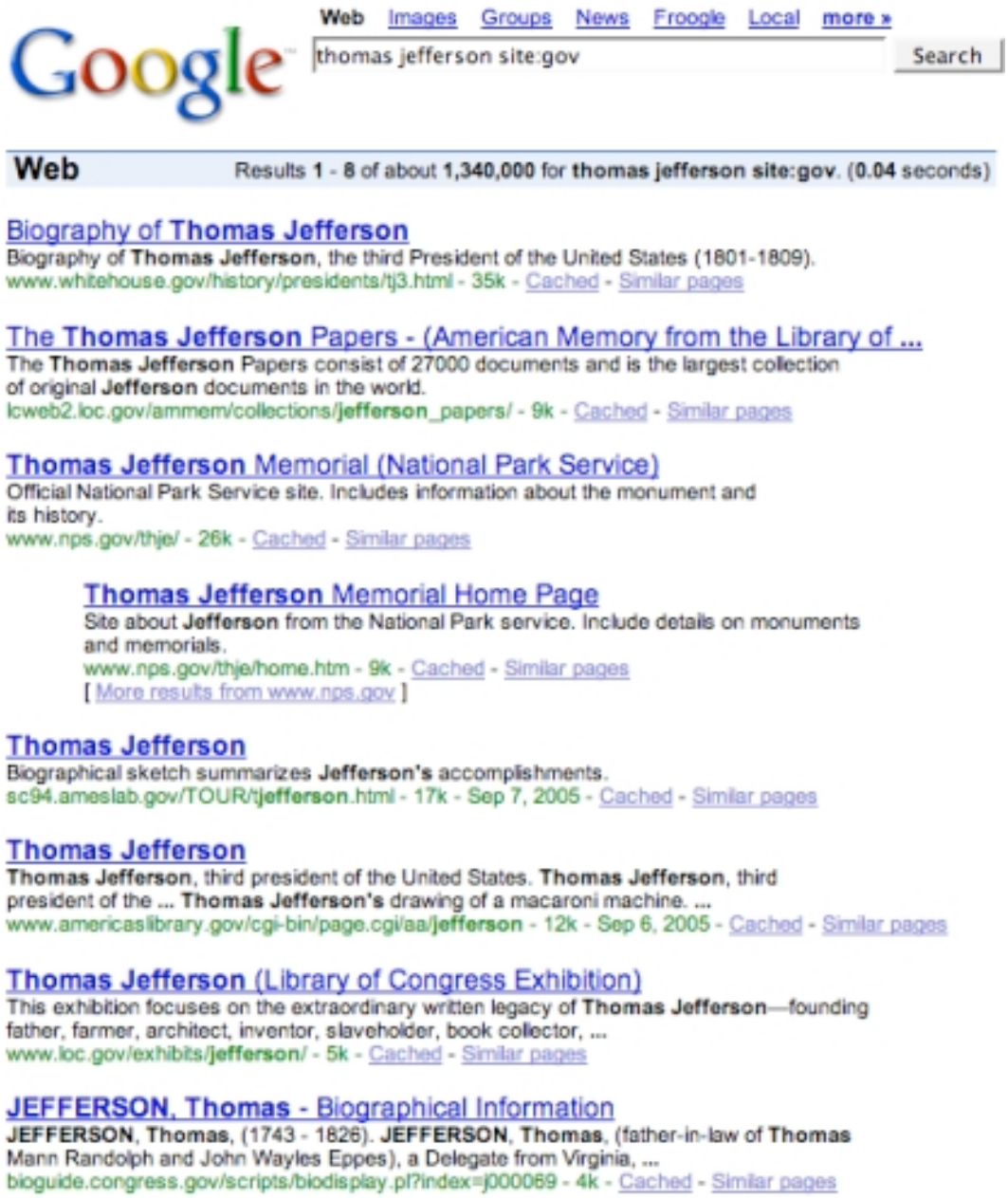

Figure 1: Google query

Figure 2: Nutch query

The two search engines have 4 of the top hits in common though differently ordered. Goggle's top hit is hit number 8 in Nutch. Nutch favors the Library of Congress Thomas – after Thomas Jefferson – congressional information site.

Clicking on the 'RSS' image in Figure 2 returns an RSS representation of the search result [11]; 'other version' does a look up into IM WM using the hit URL as query to return all versions in this collection. The 'explain' button, a feature unique to Nutch, fetches a page that lists the value for indexed fields for this hit and then details how this hits' ranking score was derived.

6. Indexing Stats

Discussed below are details indexing two WACs: One small, the other medium-sized. All processing was done on machines of the following profile: single processor 2.80GHz Pentium 4s with 1GB of RAM and 4x400GB IDE disks running Debian GNU/Linux. Indexing, this hardware was CPU-bound with light I/O loading. RAM seemed sufficient (no swapping). All source ARC data was NFS mounted. Only documents of type text/* or application/* and HTTP status code 200 were indexed.

6.1. Small-Sized Collection

This small collection was comprised of three crawls. Indexing steps were run in series on one machine using a single disk. The collection was made of 206 ARC files, 37.2GB of uncompressed data. 1.07 million of the collection total of 1.27 million documents was indexed.

Table 1: MIME Types

MIME Type	Size (MB)	% Size	Incidence
text	25767.67	79.32%	1052103
text/html	22003.55	67.73%	1044250
application	6719.92	20.68%	20969
application/pdf	4837.89	14.89%	16201
application/msword	487.89	1.50%	3306

Table 2: Indexing Timings

Segment	Database	Index	Dedup	Merge
16h32m	2h26m	18h44m	0h01m	02h35m

Indexing took 40.3 hours to complete. The merged index size was 1.1GB, about 3% the size of the source collection. The index plus the cleaned-up segment data -- cleaning involved removal of the (recalcuable) segment-level indices made redundant by the index merge -- occupied 1.1GB + 4.9GB, or about 16% the size of the source collection. Uncleaned segments plus index made up about 40% the size of the source collection.

6.2. Medium-Sized Collection

The collection was made up of 1054 ARCs, 147.2GB of uncompressed data. 4.1 million documents were indexed. Two machines were used to do the segmenting step. Subsequent steps were all run in series on a single machine using a single disk.

Table 3: MIME Types

MIME Type	Size (MB)	% Size	Incidence
Text	96882.84	65.32%	3974008
text/html	90319.81	60.81%	3929737
Application	50338.40	34.68%	122174
application/pdf	21320.83	14.40%	45427
application/msword	1000.70	0.67%	5468

Table 4: Timings

Segment	Database	Index	Dedup	Merge
12h32m 19h18m	7h23m	55h07m	0h06m	0h31m

Indexing took 99 hours of processing time (or 86.4 hours of elapsed time because segmenting was split and run concurrently on two machines). The merged index size was 5.2GB, about 4% the size of source collection. Index plus the cleaned-up segment data occupied 5.2GB + 14.5GB, or about 13.5% the size of the source collection. (Uncleaned segments plus index occupied about 22% the size of the source collection.)

7. Observations

Adding to a page its inlink anchor-text [21] when indexing improves search result quality. Early indexing experiments were made without the benefit of the Nutch link database — our custom fetcher step failed to properly provide link text for Nutch to exploit. Results were rich in query terms but were not what was 'expected'. A subsequent fix made link-text begin to count. Thereafter, search result quality improved dramatically.

Parsing content takes up the bulk of the time segmenting. Only the HTML parser was enabled at first. Enabling the native Nutch PDF parser slowed down extraction significantly, out of proportion with their percentage of total collection, partly because PDF documents are on average much larger than HTML documents, but also because the Nutch PDFBox [12] Java-based PDF parser was comparatively slower and actually hung on certain content. A change to call the external program pdftotext [13] made PDF parsing robust and run faster. Limiting the text parsed from PDFs to the first ten megabytes brought the rate of PDF text-extraction back within reasonable bounds. Our expectation is that as we add each new kind of parser to the text extraction step, e.g. an MS Word text-extractor, a new round of experiments figuring speedups and workarounds for crashes will be required. Parse times will expand. (Apparently speed and hangs have both been recently addressed in PDFBox warranting another look at the Java-based parser.)

Indexing big collections is a long-running manual process that currently requires intervention at each step moving the process along. Attention required compounds the more distributed the indexing is made. An early indexing of a collection of approximately 85 million documents took more than a week to complete with segmenting and indexing spread across 4 machines. Steps had to be restarted as disks overfilled and segments had to be redistributed. Little

science was applied so the load was suboptimally distributed with synchronizations waiting on laggard processes. (Others close in to the Nutch project have reported similar experiences [17].) An automated means of efficiently distributing the parsing, update, and indexing work across a cluster needs to be developed. In the way of any such development are at least the following obstacles:

- Some indexing steps are currently single process.
- As the collection grows, with it grows the central webdb of page and link content. Eventually it will grow larger than any available single disk.

We estimate that with the toolset as is, given a vigilant operator and a week of time plus 4 to 5 machines with lots of disk, indexing WACs of about 100 million documents is at the limit of what is currently practical.

The distributed Nutch query clustering works well in our experience, at least for low rates of access: ~1 query per second. (Search access-rates are expected to be lower for WACs than live-Web search engines.) But caches kept in the search frontend to speed querying will turn problematic with regular usage. The base Nutch (Lucene [9]) query implementation uses one byte per document per field indexed. Additions made to support query-time deduplication and sorting share a cache that stores each search result's document URL. Such a cache of (Java) UTF-16 Java strings gets large fast. An alternate smaller memory-footprint implementation needs to be developed.

Collections that are larger than a few million documents, when hosted on a single machine with all segments on a single disk, show a distinct lag when drawing search results. The pause happens after the search servlet has finished with its query of the index and after its written out the banner displaying the count of hits found and how long the search took. The search servlet is waiting on threads, one per search result item, to complete fetching text snippets from disk. If all of a collection's segments reside on a single disk, all threads are waiting in line on serial access to the single disk. Distributing a collection across multiple disks or across machines using Nutch's native distributed search removes the lag. (Queries take less than 1 second.)

8. Future Developments

From inception, the Nutch project has set its sights on operating at the scale of the public web and has been making steady progress addressing the difficult technical issues scaling up indexing and search. The Nutch Distributed File System (NDFS) is modeled on a subset of the Google File System (GFS) [16] and is "...a set of software for storing very large stream-oriented files over a set of commodity computers. Files are replicated across machines for safety, and load is balanced fairly across the machine set" [15]. The intent is to use NDFS as underpinnings for a distributed webdb. (It could also be used storing very large segments.) While NDFS addresses the problem of how to manage large files in a fault-tolerant way, it does not help with the even distribution of processing tasks across a search cluster. To this end, the Nutch project is working on a version of another Google innovation, MapReduce [17], "a platform on which to build scalable computing" [6]. In synopsis, if you can cast the task you wish to run on a cluster into the MapReduce mold -- think of the Python *map* function followed by *reduce*

function -- then the MapReduce platform will manage the distribution of your task across the cluster in a fault-tolerant way. Late 2005, core developers of the Nutch project are writing a Java version of the MapReduce platform to use in a reimplementation of Nutch as MapReduce tasks [6]. MapReduce and NDFS will make Nutch capable of scaling its indexing step to billions of documents.

The IA is moving its collections to the Petabox platform; racks of low power, high storage density, inexpensive rack-mounted computers [20]. The future of WAC search development will be harnessing Nutch MapReduce/NDFS development on Petabox.

Finally, Heritrix, the IA open source, web-scale, archival-quality web crawler allows users to create their own IA-like WACs. But because of licensing restrictions, the IA WM cannot be distributed. Heritrix users currently are without an IA WM-like means of viewing harvested collections. While we've described how functionality was added to Nutch so it could serve as the index engine for a WAC viewer, as of this writing no open source WAC viewer application using this technique is available. While the IA intends to deliver an open source IA WM [18], the Nordic Web Archive (NWA) has been adapting the NWA Toolset [19], an open source "...software package for accessing archived web documents", to use Nutch with WAC extensions at its core. The first release of a NWA Toolset Nutch-based viewer application is slated for the end of 2005.

9. Conclusions

With vigilance, it is possible to use Nutch with extensions to inexpensively index and effectively search Web Archive Collections of up to 100 million documents (More CPU will make the indexing and querying run faster. More disks per machine make search results draw more promptly). Optional is how the particular character of web collections are handled. For example, indices can be built that gracefully suppress the fact that a single URL may appear multiple times in an archive collection. Alternatively 'rich indices' with a reference for every instance can be built, with query-time operators that allow showing the most recent version only, the most recent ten versions sorted by date, or all versions in the index. (Such rich indices could be used as the base for an open source version of the Internet Archive's Wayback Machine.) Scaling the current implementation beyond its current limits will be done by leveraging MapReduce/NDFS work underway in Nutch.

Acknowledgements

Doug Cutting developed the original ARC-to-segments tool and added to core Nutch on-the-fly sort, deduplication, and search results as XML. All members of the Internet Archive Web Team -- Michele Kimpton, Gordon Mohr, Igor Ranitovic, Brad Tofel, Dan Avery, and Karl Thiessen -- contributed from original conception through development and testing. Thanks to Sverre Bang and Jan-Arne Trulsen for persevering adapting NWA to use WAC search.

References

[1] Internet Archive http://www.archive.org

[2] Internet Archive Wayback Machine http://www.archive.org/web/web.php

[3] International Internet Preservation Consortium http://netpreserve.org

[4] Nutch http://lucene.apache.org/Nutch/

[5] Nutch: A Flexible and Scalable Open-Source Web Search Engine
http://labs.commerce.net/wiki/images/0/06/CN-TR-04-04.pdf

[6] MapReduce in Nutch, 20 June 2005, Yahoo!, Sunnyvale, CA, USA
http://wiki.apache.org/nutch-data/attachments/Presentations/attachments/mapred.pdf

[7] Heritrix http://crawler.archive.org

[8] ARC File Format http://www.archive.org/web/researcher/ArcFileFormat.php

[9] Lucene http://lucene.apache.org/

[10] A9 Open Search http://opensearch.a9.com/

[11] Archive-access Nutch extensions http:// archive-access.archive.org/projects/nutch

[12] PDFBox http://www.pdfbox.org/

[13] XPDF http://www.foolabs.com/xpdf/

[14] The Nutch Distributed File System by Michael Cafarella
http://wiki.apache.org/Nutch/NutchDistributedFileSystem

[15] [Nutch-dev] Experience with a big index by Michael Cafarella
http://www.mail-archive.com/nutch-developers@lists.sourceforge.net/msg02602.html

[16] Google File System http://www.google.com/url?sa=U&start=1&q=http://labs.google.com/papers/gfs-sosp2003.pdf&e=747

[17] MapReduce: Simplifed Data Processing on Large Clusters by Jeffrey Dean and Sanjay Ghemawat http://labs.google.com/papers/mapreduce-osdi04.pdf

[18] FutureWayback http://crawler.archive.org/cgi-bin/wiki.pl?FutureWayback

[19] NWA Toolset http://nwatoolset.sourceforge.net/

[20] Petabox http://www.archive.org/web/petabox.php

[21] Anchor text http://en.wikipedia.org/wiki/Anchor_text