An Approach of Web Crawling and Indexing of Nutch

N. KOWSALYA
ASSISTANT PROFESSOR
DEPARTMENT OF COMPUTER APPLICATIONS
VIVEKANANDHA COLLEGE OF ARTS AND SCIENCES FOR WOMEN
(AUTONOMOUS)
TIRUCHENGODE-05, TAMILNADU

Abstract-Nutch has a highly modular architecture. Crawling is the operation that navigates and retrieves the information in web pages, populating the set of documents that will be searched. Typically, at the centre of any IR system is the inverted index. For each term, the inverted index contains a posting list, which lists the documents represented as integer document-IDs (doc-IDs) containing the term. The nutch command eliminates duplicate documents from a set of Lucene indices for Nutch segments. Nutch also has to add support for HTML extraction to Lucene. Nutch includes a link analysis algorithm similar to PageRank. The WebDB containing the web graph of pages and links. These lists contain every URL we're interested in downloading. The query engine part consists of one or more front-ends, and one or more back-ends. Each back-end is associated with a segment of the complete data set. The driver represents external users and it is the point at which the performance of the query is measured, in terms of queries per second (qps).

Index Terms – Crawling; Retrieving; Indexing; Link analysis; PageRank; Web DB; Web Graph;

1 Introduction:

Apache Nutch is an open source web crawler that is used for crawling websites. It is extensible and scalable. It provides facilities for parsing, indexing, and scoring filters for custom implementations. Nutch is an effort to build an open source search engine based on *Lucene* Java for the search and index component. It has a highly modular architecture, allowing developers to create plug-ins for media-type parsing, data retrieval, querying and clustering. Nutch is a complete open-source Web search engine package that aims to index the World Wide Web as effectively as commercial search services. As a research platform it is also promising at smaller scales, since its flexible architecture enables communities to customize it; and can even scale down to a personal computer. Its founding goal was to increase the transparency of the Web search process as searching becomes an everyday task. The nonprofit Nutch Organization supports the open-source development effort as it addresses significant technical challenges of operating at the scale of the entire public Web. Nutch server installations have already indexed 100M-page collections while providing state-of-the-art search result quality. At the same time, smaller organizations have also adopted Nutch for intranet and campus networks. At this scale, all of its components can run on a single server.
Nutch is an open-source project hosted by the Apache Software Foundation. Nutch provides a complete, high-quality Web search system, as well as a flexible, scalable platform for the development of novel Web search engines. Nutch includes:

- a web crawler;
- parsers for web content;
- a link-graph builder;
- schemas for indexing and search;
- distributed operation, for high scalability;
- an extensible, plugin-based architecture.

Nutch is implemented in Java and its code is open source. Thus runs on many operating systems and a wide variety of hardware.

2 Nutch Architecture:

Nutch has a highly modular architecture that uses plug-in APIs for media-type parsing, HTML analysis, data retrieval protocols, and queries. The core has four major components:

Searcher: Given a query, it must quickly find a small relevant subset of a corpus of documents, and then present them. Finding a large relevant subset is normally done with an inverted index of the corpus; ranking within that set to produce the most relevant documents, which then must be summarized for display.

Indexer: Creates the inverted index from which the searcher extracts results. It uses Lucene storing indexes.

Database: Stores the document contents for indexing and later summarization by the searcher, along with information such as the link structure of the document space and the time each document was last fetched.

Fetcher: Requests web pages, parses them, and extracts links from them. Nutch's robot has been written entirely from scratch.

Figure-1 outlines the relationships between elements that refer on each other, placing them in the same box, and those they depend on in a lower layer. For example, protocol does not depend on net, because protocol is only an interface point for plugins that actually provide much of Nutch's functionality.

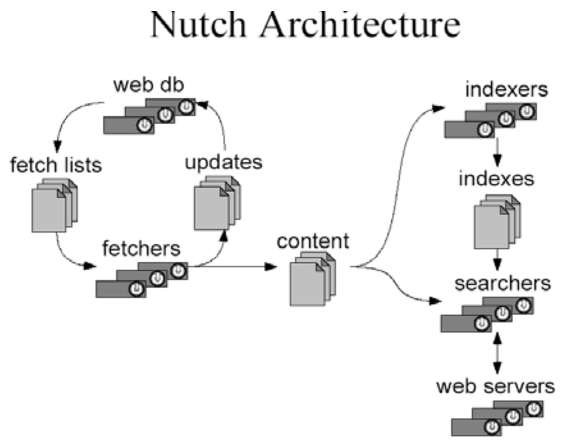

Figure-1 Nutch Architecture

2.1 Crawling:

Crawling is the operation that navigates and retrieves the information in web pages, populating the set of documents that will be searched. This set of documents is called the *corpus*, in search terminology. Crawling can be performed on internal networks (Intranet) as well as external networks (Internet). Crawling, particularly in the Internet, is a complex operation. Either intentionally or unintentionally, many web sites are difficult to crawl. Sophisticated technology is required to develop a good crawler.

The performance of crawling is usually limited by the bandwidth of the network between the system doing the crawling and the outside world. In the Commercial Scale Out project, system setup did not have a high-bandwidth connection to the outside world and it would take a long time to populate the system with enough documents to create an interesting corpus.

An intranet or niche search engine might only take a few hours to crawl, in a single-machine while a whole-web crawl might take many machines several weeks or longer. A single crawling cycle consists of generating a fetchlist from the webdb, fetching those pages, parsing those for links, then updating the webdb. In the terminology of Nutch's crawler supports both a *crawl-and-stop* and *crawl-and-stop-with-threshold* (which requires feedback from scoring and specifying a floor). It also uses a uniform refresh policy; all pages are refetched at the same interval (30 days, by default) regardless of how frequently they change**.** Java can set individual recrawl-deadlines on every page). The fetching process must also respect bandwidth and other limitations of the target website. However, any polite solution requires coordination before fetching, Nutch uses the most straightforward localization of references possible namely, making all fetches from a particular host run on one machine.

2.2 Indexing:

To allow efficient retrieval of documents from a corpus, suitable data structures must be created, collectively known as an index. Usually, a corpus covers many documents, and hence the index will be held on a large storage device – commonly one or more hard disks. Typically, at the centre of any IR system is the inverted index. For each term, the inverted index contains a posting list, which lists the documents represented as integer document-IDs (doc-IDs) containing the term. Each posting in the posting list also stores sufficient statistical information to score each document, such as the frequency of the term occurrences and, possibly, positional information (the position of the term within each document, which facilitates phrase or proximity search) or field information (the occurrence of the term in various semi-structured area of the document, such as title, enabling these to be higher-weighted during retrieved). The inverted index does not store the textual terms themselves, but instead uses an additional structure known as a lexicon to store these along with pointers to the corresponding posting lists within the inverted index. A document index may also be created which stores meta-information about each document within the inverted index, such as an external name for the document (e.g. URL), and the length of the document. The process of generating these structures is known as indexing.

2.2.1 Indexing Text:

Lucene meets the scalability requirements for text indexing in Nutch. Nutch also takes the advantage of Lucene's multi-field case-folding keyword and phrase search in URLs, anchor text, and document text. The typical definition of Web search does not require some index types which Lucene does not address, such as regular-expression matching (using, say, suffix trees) or document-similarity clustering (using, say, term-vectors).

2.2.1 Indexing Hypertext:

Lucene provides an inverted-file full-text index, which suffices for indexing the text but not the additional tasks required by a web search engine. In addition to this, Nutch implements a link database to provide efficient access to the Web's link graph, and a page database that stores crawled pages for indexing, summarizing, and serving to users, as well as supporting other functions such as crawling and link analysis. In the terminology of one web search engine survey, Nutch combines the text and utility databases into its page database. Nutch also has to add support for HTML extraction to Lucene. When a page is fetched, any embedded links are considered for addition to the fetchlist and that link's anchor text is also stored. What is eventually indexed by Lucene is only the text of a Web page, though, while a high-fidelity copy is stored in the page database, font, heading, and other structural information does not propagate to the Lucene indexing layer.

2.2.3 Removing Duplicates:

The nutch command eliminates duplicate documents from a set of Lucene indices for Nutch segments, so it inherently requires access to all the segments at once. It's a batch-mode process that has to be run before running searches to prevent the search from returning duplicate documents. It uses temporary files containing the 5-tuple (MD5 hash, float score, int indexID, int docID, int urlLen) for each page. Presumably the documents with the same URLs were fetched at different times, so Nutch tries to sort the records so that the ones for the newest fetches come first. A second pass, using hash=MD5 (content) and slightly different sorting rules, eliminates multiple URLs for the same document from the index.

2.2.4 Link Analysis:

Nutch includes a link analysis algorithm similar to PageRank. It even uses 15 as the random-jump probability. It is performed by the Distributed Analysis Tool (DAT), even the single-machine Link Analysis Tool (LAT) merely calls into it. It uses an iterative method, for solving for a matrix value directly. Nutch has already demonstrated the ability to harness multiple servers to compute link ranking for 100Mpage subsets of the World Wide Web. Distributed link analysis is a bulk synchronous parallel process. At the beginning of each phase, the list of URLs whose scores must be updated is divided up into many chunks; in the middle, many processes produce score-edit files by finding all the links into pages in their particular chunk. At the end, an updating phase reads the score-edit files one at a time, merging their results into new scores for the pages in the web database. Distributed analysis doesn't use Nutch's homegrown IPC service; like fetching, work is coordinated through the appearance of files in a shared directory. There are better techniques for distribution (MapReduce) and accelerating link analysis.

Nutch includes a parallel indexing operation written using the MapReduce programming model. MapReduce provides a convenient way of addressing an important class of real-life commercial applications by hiding the parallelization and the fault tolerance from the programmers, letting them focus on the problem domain. MapReduce was published by Google in 2004 and quickly became a popular approach for parallelizing commercial workloads.

A parallel indexing operation in the MapReduce model works as follows. First, the data to be indexed is partitioned into segments of approximately equal size. Each segment is then processed by a mapper task that generates the (*key*, *value*) pairs for that segment, where *key* is an indexing term and *value* is the set of documents that contain that term (and the location of the term in the document). This corresponds to the map phase, in MapReduce. In the next phase, the reduce phase, each reducer task collects all the pairs for a given key, thus producing a single index table for that key. Once all the keys are processed, we have the complete index for the entire data set.

2.3 A Link Graph Builder:

The WebDB containing the web graph of pages and links. WebDB is a persistent custom database that tracks every known page and relevant link. It maintains a small set of facts about each, such as the last-crawled date. WebDB is meant to exist for a long time, across many months of operation. Since WebDB knows when each link was last fetched, it can easily generate a set of fetchlists. These lists contain every URL we're interested in downloading. WebDB splits the overall workload into several lists, one for each fetcher process. URLs are distributed almost randomly; all the links for a single domain are fetched by the same process, so it can obey politeness constraints. The fetchers consume the fetchlists and start downloading from the Internet. The fetchers don't overload a single site with requests, and they observe the Robots Exclusion Protocol. (This allows Web-site owners to mark parts of the site as off-limits to automated clients such as our fetcher.) Otherwise, the fetcher blindly marches down the fetchlist, writing down the resulting downloaded text. Fetchers output WebDB *updates* and *Web content*. The updates tell WebDB about pages that have appeared or disappeared since the last fetch attempt. The Web content is used to generate the searchable index that users will actually query.

2.4 Querying:

In most search applications, query represents the vast majority of the computation effort. When performing a query, a set of indexing terms is presented to a query engine, which then retrieves the documents that best match that set of terms. The overall architecture of the Nutch parallel query engine is shown in Figure-2. The query engine part consists of one or more front-ends, and one or more back-ends. Each back-end is associated with a segment of the complete data set. The driver represents external users and it is the point at which the performance of the query is measured, in terms of *queries per second* (qps).

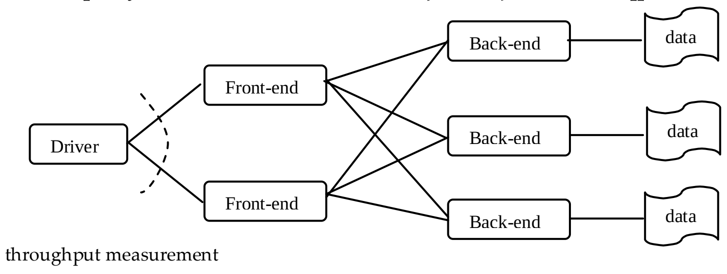

Figure 2: Overall architecture of Nutch/Lucene query.

A query operation works as follows. The driver submits a particular query (set of index terms) to one of the front-ends. The front-end then distributes the query to all the back-ends. Each back-end is responsible for performing the query against its data segment and returning a list with the top documents (typically 10) that best match the query. Each document returned is associated with a *score*, which quantifies how good that match is. The front-end collects the response from all the back-ends to produce a single list of the top documents (typically 10 overall best matches). Once the front-end has that list, it contacts the back-ends to retrieve snippets of text around the index terms. Only snippets for the overall top documents are retrieved, and the front-end contacts the back-ends one at a time, retrieving the snippet from the back-end that had the corresponding document in its data segment.

2.5 Parsing:

The PARSED column is always true when using the crawl tool in Nutch. This column is useful when running fetchers with parsing turned off, to be run later as a separate process. The STARTED and FINISHED columns indicate the times when fetching started and finished. This information is invaluable for bigger crawls, when tracking down why crawling is taking a long time.

The value of the property parser.character.encoding.default is used as a fallback character encoding (charset) when HTML parser cannot find the charset information in HTTP Content-Type header or in META HTTP-EQUIV tag. But the plain text parser behaves differently. It just uses the system encoding. To guarantee a consistent behavior, plain text parser should use the value of the same property.

Segment is basically a folder containing all the data related to one fetching batch. Fetch List is stored inside Segment. Besides the Fetch List, the fetched content itself will be stored there in addition to the extracted plain text version of the content, anchor texts and URLs of outlinks, protocol and document level metadata etc. The Link Database is a data structure (Sequence file, URL -> Inlinks) that contains all inverted links. In the parsing phase Nutch can extract outlinks from a document and store them in format source url -> target_url,anchor_text. In the process of inversion we invert the order and combine all instances making the data records in Link Database look like targetURL -> anchortext[] text so we can use that information later when individual documents are indexed.

2.6 Searching:

The Nutch Search system also has an external executable program. This program is called Update Context. Its function is to analyze the documents that are being visited by the user, and update the \navigation context" according to the documents visited by the user. That is, finding the nearest leaf cluster for each page that the user visits, and inserting this information into the database. The information about the \navigation context" will be used by the \Context Query Filter" plugin. This plugin will modify the queries submitted by the user according to the \navigation context", thus adding information about the navigation context of the user.

Note that the computation of the nearest cluster for a web page is done while the user is surfing that page, not while the user submits the query. Therefore, computing the nearest cluster for a page does not add time to the search execution. Thus, only the computation of the weight of context-related terms adds extra time to the query execution.

Nutch's search user interface runs as a Java Server Page (JSP) that parses the user's textual query and invokes the search method of a NutchBean. If Nutch is running on a single server, this translates the user's query into a Lucene query and gets a list of hits from Lucene, which the JSP then renders into HTML. If Nutch is instead distributed across several servers, the NutchBean's search method instead remotely invokes the search methods of other NutchBeans on other machines, which can be configured either to perform the search locally as described above or farm pieces of the work out to yet other servers. Distributed searching is built on top of a custom SIMD cluster parallelism toolkit in the package net.nutch.ipc, which provides a blocking 'call' method to perform the same operation in parallel across several servers, and then gather the results together afterwards. In the terminology of Nutch uses *partition-by-document*, where all the postings for a certain document are stored on the same node. Consequently every query must be broadcast to all nodes. In Nutch, the net.nutch.searcher.DistributedSearch.Client class provides this functionality; it implements the same net.nutch.searcher.Searcher interface that Nutch uses to invoke searches on locally-stored segments, represented by net.nutch.searcher.FetchedSegments objects.

2.7 Distributed File System:

The Distributed Nutch File System, 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.

The NDFS fills an important hole for the Nutch project. Right now it is very difficult for a user with a handful of machines to run a Nutch installation. The files can easily grow to very large size, possibly larger than any single available disk. At best, the Nutch operator ends up spending a lot of time copying files back and forth, managing what storage is available. This software should solve the problem. Files are stored as a set of blocks scattered across machines in a NDFS installation. However, writers and readers just use a single traditional input/output stream interface. The details of finding the correct block and transferring data over the network is handled automatically by the NDFS.

Further, the NDFS gracefully handles changes in its machine set. Machines used for storage can be added or removed, and machines can crash or otherwise become unavailable. The NDFS will automatically preserve file availability and replication requirements, if possible. As long as active machines retain sufficient available storage, there's no reason to involve a human administrator at all. The NDFS uses only bare-bones commodity hardware, with no need for RAID controllers or any other specialized disk solution.

The end result is that Nutch users should be able to take advantage of very large amounts of disk storage with very little administrative overhead.

2.7.1 NDFS File System Semantics:

1. Files can only be written once. After the first write, they become read-only. (Although they can be deleted.)
2. Files are stream-oriented; you can only append bytes, and you can only read/seek forward.
3. There are no user permissions or quotas, although these could be added fairly easily.

So, all access to the NDFS system is through approved client code. There is no plan to create an OS-level library to allow any program to access the system. Nutch is the model user, although there are probably many applications that could take advantage of NDFS. (Pretty much anything that has very large stream-oriented files would find it useful, such as data mining, text mining, or media-oriented applications.)

2.7 Scalability:

Nutch hasn't scaled beyond 100 million pages so far, for both economic and technical reasons. Maintaining an up-to-date copy of the entire Web is inherently an expensive proposition, costing substantial amounts of bandwidth. (Perhaps 10 terabytes per month at minimum, which is about 30 megabits per second, which costs thousands of dollars per month)

Answering queries from the general public is also inherently an expensive proposition, requiring large amounts of computer equipment to house terabytes of RAM and large amounts of electricity to power it, as well as one to three orders of magnitude more bandwidth.

1. Nutch's link analysis stage requires the entire link database to be available on each machine participating in the calculation and includes a significant non-parallel final collation step.
2. As Nutch partitions its posting lists, across cluster nodes by document, each query must be propagated to all of the hundreds or thousands of machines serving a whole-web index. This means that hardware failures will happen every few hours, the likelihood of a single slow response causing a query to wait several seconds for a retransmission is high, and the CPU resources required to process any individual query become significant.
3. As Nutch crawls retrieve unpredictable amounts of data, load-balancing crawls with limited disk resources are difficult.

Much of Nutch's distribution across clusters must be done manually or by home-grown cluster management machinery; in particular, the distribution of data files for crawling and link analysis, and the maintenance of search-servers.txt files all must be done by hand. Large deployments will require fault-tolerant automation of these functions.

Further work is being done in this area to enhance Nutch's scalability. The Nutch Distributed File System (NDFS) is in current development versions of Nutch, to enhance performance along the lines proposed by the

Google File System. NDFS has been used recently to run a link analysis stage over 35 million pages on twelve machines.

In the last several years, much work has been focused on eliminating economic scalability limitations on services such as file downloading by distributing the work of providing the service among its users. This has appeared technically infeasible for a full-text search engine for the whole Web. However, recent and ongoing work suggests that this kind of peer-to-peer distribution may soon be possible.

2.9 Plugin Based Architecture:

The open source nature of Nutch makes it ideal for modification. This potential for customization is further aided by the modular nature of Nutch. The Nutch search engine is built upon a basic code backbone which is augmented heavily through the use of plugins. Plugins are program extensions which are be added to a host application. The release version of Nutch contains dozens of plugins which may be added or removed as desired by changing the Nutch configuration. These plugins are responsible for the parsing of different file types during the crawl, indexing of crawl results, protocols through which the crawl can operate, and querying of indexed crawl results, among other tasks. Essentially, the majority of the primary search engine functions are performed by plugins. Therefore, modifying the search engine may be accomplished by changing the configuration of the plugins, which may include adding new plugins. In order to add WordNet-related functionality to Nutch, a plugin should be created.

3 Who Should Run Nutch-Based Web Search Engine:

Nutch.org is dedicated to making the Nutch software better for everyone. That might mean running a small demo site or making a search service available for academic research, but we do not intend to run a destination search site. Running such a service would put Nutch in competition with its users. Instead, we hope that primarily other institutions will run the Nutch software. Governments, universities, and nonprofits are terrific candidates for Nutch. These organizations often have special obligations that for-profit companies don't (e.g., a seniors' organization might want to offer search with a special usability focus), so having the source code to Nutch is a huge advantage. Further, these groups often don't have lots of cash to spend on solutions. We don't have great data yet on who is running Nutch. As far as we can tell, the most active Nutch users are universities and academic research groups. Some are using Nutch as part of a class, and some are using it because their research depends on access to indexed pages that they can control. Others are pulling apart the system, taking elements that seem useful. It's too early to expect any updates back from researchers, but we hope this is coming soon. One type of nonprofit in particular that we hope to see is a PSE (public search engine), a search site that is as usable as any commercial one, but that operates without advertising or commercial engagement. These engines will help make good on Nutch's promise to make search results more transparent to users. Conversely, they will make for-profit engines easier to spot if they adjust rankings for commercial gain. A PSE might get its funds through donations from users, corporations, or foundations, just as public broadcasting channels do. It's worth noting that PSEs do not need to process a huge percentage of search queries to be successful. Their existence will ensure that search users always have a good alternative (one that doesn't exist today). Profit corporations will want to run small search engines for in house use or on their public Web sites. For most of these companies, search will be just another item they have to take care of, not their main focus. Nutch should also enable small search-technology companies to be more creative, just as other open source projects have enlarged what small teams can accomplish. We hope that Nutch, by providing free, open source Web search software, will help both to promote transparency in Web search and to advance public knowledge of Web-search algorithms.

4 Conclusion:

In this paper, I have emphasized the various phases of Nutch Architecture. I have detailed the strategies of Web Crawling and indexing the documents in the Intranet and Internet. Particularly, I have shown that the web crawling which navigates and retrieves the information in the web and that the indexing must be created to allow efficient retrieval of documents from a corpus. I have presented that Nutch includes a parallel indexing operation using the MapReduce programming model and Query represents the majority of the computation effort. In the parsing phase Nutch can extract outlinks from a document and store them in the formatted source.

Finally, I have revealed that the Nutch Search system is an external executable program. Its function is to analyze the documents and finding the nearest cluster and insert the information into the database and the plugins are responsible for the parsing of different file types during the crawl, indexing of crawl results, protocols through which the crawl can operate, and querying of indexed crawl results, among other tasks. Much of the work was focused on eliminating economic scalability limitations on services such as file downloading by providing the service among its users. Overall, I have concluded that Nutch.org is dedicated to making the Nutch software is better for everyone. Hence, the Government, various Universities, and Nonprofit organizations are promising candidates for Nutch.

5 References:

1. White, Tom. 2006. "Introduction to Nutch, Part 1: Crawling". Retrieved from http://today.java.net/pub/a/today/2006/01/10/introduction-to-nutch-1.html.

2. Smart, John Ferguson. 2006. "Integrate Advanced Search Functionalities Into Your Apps". Retrieved from http://www.javaworld.com/javaworld/jw-09-2006/jw-0925-lucene.html .

3. Coar, Ken. 2009. "The Open Source Definition". Retrieved from http://www.opensource.org/docs/osd.

4. **http://en.wikipedia.org/wiki/Open_Source**

5. http://lucene.apache.org/nutch/

6. http://www.nutch.org/
M. Cafarella and D. Cutting. *Building Nutch: open source search.* 2004.