Optimizing Apache Nutch For Domain Specific Crawling at Large Scale

Luis A. Lopez[1], Ruth Duerr[2], Siri Jodha Singh Khalsa[3]
NSIDC[1], The Ronin Institute[2], University of Colorado Boulder [3]
Boulder, Colorado.
luis.lopez@nsidc.org

Abstract— **Focused crawls are key to acquiring data at large scale in order to implement systems like domain search engines and knowledge databases. Focused crawls introduce non trivial problems to the already difficult problem of web scale crawling; To address some of these issues, BCube - a building block of the National Science Foundation's EarthCube program - has developed a tailored version of Apache Nutch for data and web services discovery at scale. We describe how we started with a vanilla version of Apache Nutch and how we optimized and scaled it to reach gigabytes of discovered links and almost half a billion documents of interest crawled so far.**

Keywords: focused crawl, big data, Apache Nutch, data discovery

I. INTRODUCTION

Once a problem becomes too large for vertical scaling to work scaling horizontally is required. This is why Apache Nutch and Hadoop were designed[10]. Hadoop abstracts most of the perils of implementing a distributed, fault tolerant file system, uses MapReduce as its data processing algorithm and has become a defacto standard for big data processing. Hadoop has advantages and limitations, it scales well and has a solid user base, however a plethora of configuration properties need to be tuned in order to achieve a performant cluster[8].

However, having a performant cluster is not enough, optimizing the crawl and speeding up fetch times within the range of polite crawl delays given by the remote servers is also required. At that point, other problems appear - slow servers, malformed content, poor standard implementations, dark data, semantic equivalence issues and very sparse data distribution.

Each challenge requires a different mitigation technique and some can only be addressed by the site's owners. For example, if a slow server has relevant information all we ethically can do to fix is to contact the organization to let them know they have a performance issue. This is also the case for malformed content and poor standard implementations like wrong mime type and dates given by the servers. In BCube we ran into most of these problems and we attempted to improve the crawl by tackling those that are under our control.

The role of BCube as part of the NSF funded project *EarthCube* is to discover and integrate scientific data sets, web services and information to build a "google for scientific data", for this reason a simple web crawler was not enough and a focused crawler had to be developed. The crawler is a core component of BCube but not the only one, at the moment our stack is also using semantic technologies to extract valuable metadata from the discovered data.

Several academic papers have been written about Hadoop cluster optimization and advanced techniques for focused crawling however the purpose of this paper is to describe how Nutch can be used as a domain specific (topical crawler) discovery tool and our results so far in using it at scale. First, we describe the issues we ran into while doing focused crawls and what a user gets from a vanilla Nutch cluster. Second, we talk about customization made to maximize cluster utilization and lastly, we discuss the improved performance and future work for this project.

II. RELATED WORK

Previous work on this topic has focused on the use of supervised and semi-supervised machine learning techniques to improve relevant document retrieval [1][2][11], but there is little literature on how these techniques perform at scale and in most cases the core component of the research is a custom made crawler that is only tested in a few hundred pages [3][7][11]. This is not just inconclusive but also hides problems that focused crawlers will run into at large scale [6].

Some open source projects can and have been used to perform focused crawls as well, Scrapy Frontera, Heritrix and Bixo are among the most mature and they all support extensions. Sadly, none of these frameworks have been used extensively in domain specific crawls at scale with publicly available results. Future work would necessarily include porting the code used in BCube to these frameworks and benchmark the results using similar hardware.

Current efforts like JPL's MEMEX use Nutch and other Apache components to crawl the deep web and extract metadata on more specialized content[13]. As MEMEX and other projects evolve it will be interesting to compare their findings and performance once that the size of the search space grows to cover a significant portion of the Internet.

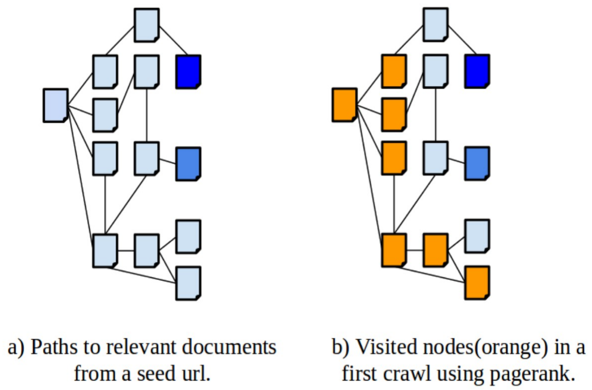
978-1-4799-9926-2/15/$31.00 ©2015 IEEE

III. FOCUSED CRAWLS.

3 *.1. Challenges*

A focused crawl could be oversimplified as a Depth-First Search (DFS) algorithm with the added complexity that we operate in an undirected graph with billions of edges. Our work then is to figure out how to get the most documents of interest possible visiting the minimum number of nodes(web pages). This is critical because the Internet is large and computational resources are usually limited by time and costs.

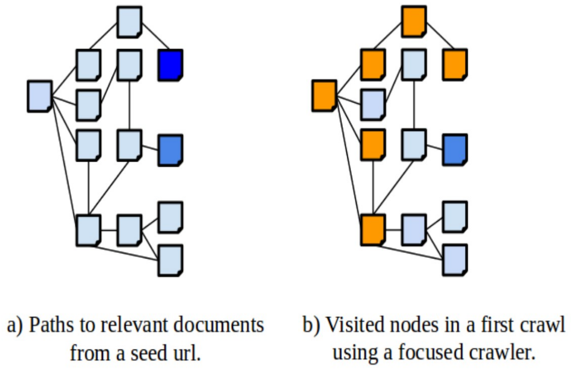
Precisely because of the intractable Internet size, Google came up with Page Rank[4] so links can be crawled in an ordered manner based on their relevance. The relevance algorithm used in PageRank is very similar to the one used by Nutch's Scoring Opic Filter. This type of algorithm prioritizes popular documents and sites that are well connected. This approach is good for the average end user but not really suitable for domain specific searches. This can be seen in figures 3.1 and 3.2.



a) Paths to relevant documents from a seed url.

b) Visited nodes(orange) in a first crawl using pagerank.

Fig. 3.1 PageRank



a) Paths to relevant documents from a seed url.

b) Visited nodes in a first crawl using a focused crawler.

Fig. 3.2 Focused Crawl

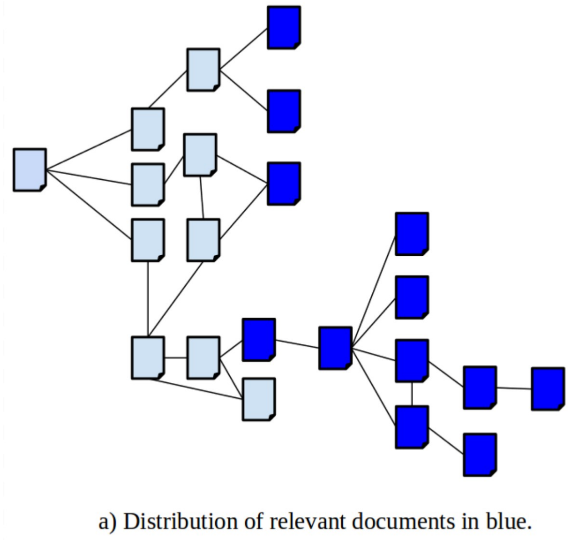
Note that in PageRank-like algorithms the nodes with more inlinks are visited first and this is not optimal for potentially diffuse, sparse, not very popular but relevant information. BCube's approach to scoring is not complicated, we used a semi supervised relevance filter where relevant documents were manually tagged and then tokenized so we can compare 2 things, content and urls path components.

BCube's scoring filter currently uses only content and in the future we plan to use the context path on which a relevant document is located and semantic techniques to improve and adapt the relevance based on user interaction[14].

As we stated before, the focus will be on how a focused crawler performs independently from the scoring algorithm. Let's assume that we have a good scoring algorithm in place and we are about to crawl the .edu search space. Our intuition says that we'll index more relevant documents as time passes, we'll find a gaussian distribution of these documents in our search space and that the performance will be predictable based on the input size. We will probably be wrong. As we scaled in BCube to cover millions of URLs we ran into 3 issues with our focused crawls.

3.1.1 Falling into the Tar Pit

We use the term 'tar pit' to describe a host that contains many relevant documents, in our case these were data granules and web services (or data repositories). For these sites our scoring filter acts as expected and boosts the urls where there relevant documents were found. If the size of this 'tar pit' was big enough our crawler behaved not like a finder but like a harvester. We were indexing thousands of relevant documents from the same hosts and this left other interesting sites unexplored as we spent more time 'harvesting' the tar pits. This is better shown in figure 3.3, as our crawling frontier expands all the relevant documents in blue will have a higher priority if we discover them earlier.



a) Distribution of relevant documents in blue.

Fig. 3.3 A tar pit in blue. Other relevant documents on top.

To mitigate this we configured Nutch to only explore a limited number of documents from the same origin even if all of them are relevant. This allowed us to discover more diverse data and improve cluster performance as we were not crawling a limited number of high scored hosts. Crawling a limited number of hosts is however, an almost unavoidable scenario in focused crawls, this behavior led to our second problem, performance degradation.

3.1.2 Performance degradation

In focused crawls the number of hosts over time should decrease as we only explore paths that are relevant, causing our cluster performance to also decrease over time. This degradation follows a exponential decay pattern[18]. Figure 3.4 depicts the crawl performance we observed.

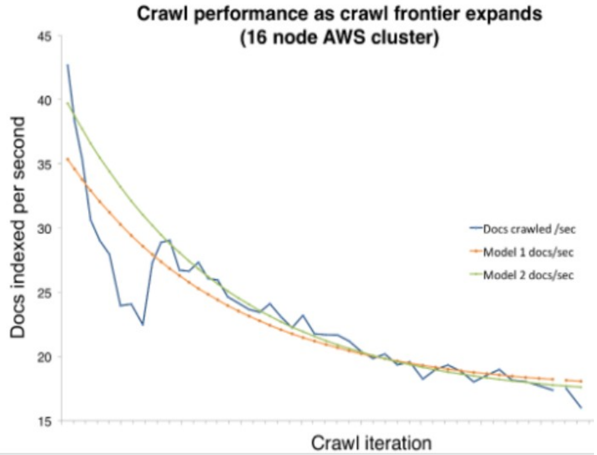


Fig. 3.4 Documents Indexed per second as the crawl frontier expands. The 2 models represent slightly different decay constants[18].

The reason for this exponential decay is driven by 3 factors, seed quality, crawling politeness and scoring effectiveness. If we use good URL seeds [9] we'll reduce our search space early and if we crawl politely we'll have to obey Robots.txt causing the underutilization of the cluster. This decay has been also found by other research [17][5]. In BCube we tried to overcome this issue by spawning separate clusters to handle specific use cases, scaling out or reducing the resources as needed. Thanks to Amazon's EMR this task was simplified, the only extra effort made was scripting the glue code to keep track of all the clusters running. In our case most of the servers had crawler-friendly Robots.txt values but some had Crawl-Delay values of up to 1 day for each request. Having multiple crawls working at different speeds allowed us to maximize our resources and avoid this problem.

3.1.3 Duplication

Every search engine has to deal with duplication problems, how to identify and delete duplicated content is a process that normally happens when we have all the documents indexed as a post processing step. At web scale however, we cannot afford to index completely similar repositories from different hosts and deduplicate later. Also there are different kinds of equivalence, clones, versions, copies and semantically similar documents.

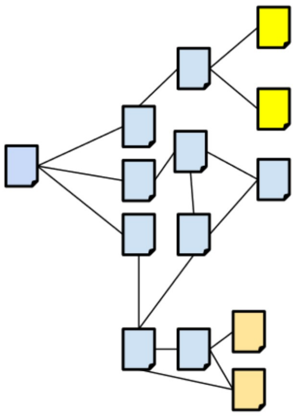
A curious case happened when our filter boosted similar relevant documents. We were crawling a repeated number of times indexing the same documents over and over. This was because there are sites that distribute content in different languages but the actual content doesn't change that much, given that we are looking for scientific data.

For example, after finding the domain cimate-data.org our crawler proceeded to boost the score for their documents and after a little while we noticed the following numbers in our crawl database.

Table. 3.1 Documents fetched from climate-data.org

Domain	Documents fetched
fr.climate-data.org	212142
bn.climate-data.org	209257
de.climate-data.org	203279
en.climate-data.org	197716

As one can guess the documents were very similar climate reports simply expressed in sites geared for French, German, Bengali and English users. This is also shown in figure 3.5.



a) Semantically similar content in different hosts(in yellow).

Fig. 3.5 Similar documents in slightly different yellow colors..

Currently BCube is exploring [12][14] semantic technologies to further characterize similar relevant documents to avoid crawling multiple times and index what's basically the same information.

IV. NUTCH CUSTOMIZATIONS

4.1 Out of the box configuration

Nutch comes with generic plugins enabled and its default configuration is good for general crawls but not so much for focused crawls. Nutch uses 2 main configuration files, nutch-default and nutch-site, the later used to override default configurations. We can adjust a number of things in the configuration files i.e. the number of documents we want to crawl on each iteration or the number of threads to use on each mapper. There are also Hadoop specific configuration properties that we probably need to tune.

Our crawling strategy required us to index the raw content of a relevant document and there was no official plugin developed for this. We also had to index documents based on the mime type(once that they pass all the filters) and Nutch didn't have this plugin until recent weeks.

4.1 BCube tuning and performance

After replacing the scoring filter and putting our extra plugins in place we had to maximize fetching speeds. The properties to do this are grouped in the “fetcher” section of the nutch-default configuration file. Something to take into account beforehand is the number of CPUs available to each Hadoop mapper, and this was used for our baseline calculation. Since document fetching is not a CPU bounded task we can play with the values to manage the workload distribution. i.e. If we have a 16 node cluster and we want to crawl 160,000 urls at the time then each mapper can get up to 10,000 urls. Given that we generate only 1,000 documents per host and all our hosts have more than 1,000 documents then we’ll have 10 queues.

There is another important property to configure, the number of threads that can access the same queue. This value can improve the speeds dramatically depending on how we group our URLs. Going back to our previous case, with 10 queues per host and configuring 2 threads per host then having more than 20 threads per mapping will be wasteful as the maximum number of working threads would be given by (queues * fetcher.threads.per.queue).

Nutch properties can be used to dynamically target a specific bandwidth. In our experiments these properties were less effective than doing the calculation of current domains and active queues. Once that we had the maximum number of working threads, we capped the fetching times to 45 minutes. This was because we noticed that the cluster crawled more than 90% of their queues in 30 minutes or less(see fig. 4.1) but lasted up to 2 hours to finish in some cases. The actual cause was that some queues had high Crawl-Delay values and the cluster waited until all the queues were empty.

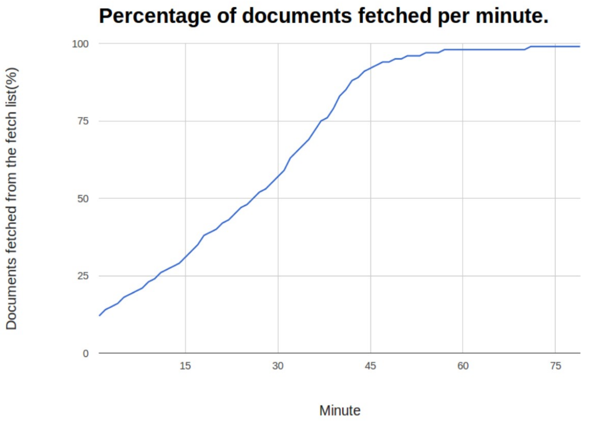


Fig. 4.1 Fetching time on 250k documents and 4223 queues.

A key difference between our focused crawls and a normal crawl lies on what we keep from them. Normal crawls index all the documents found but we don’t. This saves HDFS space (we only keep the crawldb and linkdb) and deletes the content of all the documents after the ones we consider relevant are indexed. The result is savings above 99% of the total HDFS file system.

Some operations in Nutch are performed on the current segment on which we are operating; a segment is the list of urls being fetched, parsed and integrated to our crawl database. Other operations are carried out on the entire crawl database. We improved our performance by only applying some filters in the parsing stage and pruning the URLs that we considered totally off topic or had very low scores. This was important because even if we only explore the high scoring nodes MapReduce will process the total number of URLs in our DB in operations like scoring propagation. This is similar to what happens with Search algorithms like A* with Alpha/Beta pruning[19].

The following properties were the ones that had the biggest impact on the performance of individual clusters. We applied filters once, we reduced the minimum waiting time for each request, we only generated 1000 urls using the highest scores for each host and finally we set a limit for the fetching stage to 45 minutes.

Table 4.1 Configuration changes for a 16-nodes cluster.

Property	**Default Value**	**BCube Value**
crawldb.url.filters	True	False
db.update.max.inlinks	1000	100
db.injector.overwrite	False	True
generate.max.count	-1	1000
fetcher.server.delay	10	2
fetcher.threads.fetch	10	128
fetcher.threads.per.queue	1	2
fetcher.timelimit.mins	-1	45

Finally, after several tests we show the typical behavior of one of our crawls against what we got from a vanilla version of Nutch. This test was carried out using 16-node clusters on Amazon’s EMR.

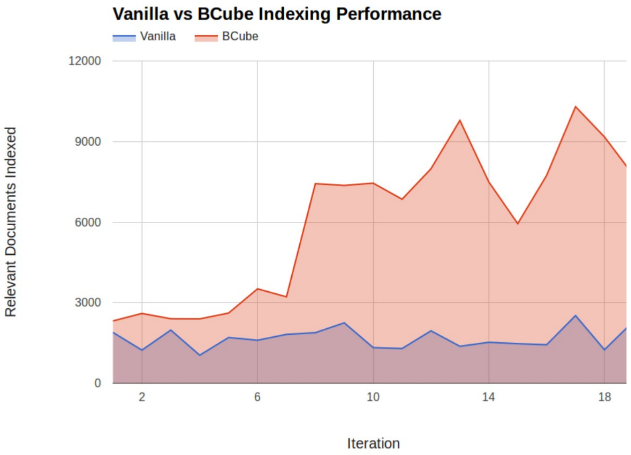


Fig. 4.2 Relevant documents indexed per iteration in a vanilla Nutch cluster and a BCube cluster.

As we can see the gains are noticeable, we could theoretically match our performance by doing a bigger cluster and indexing more documents using vanilla Nutch versions but this will be expensive and not efficient.

Figure 4.2 summarizes the content of our crawl database after 2 weeks of continuous crawling.

Table 4.2 Configuration changes for a medium size cluster.

Status	Number
redir_temp	670,937
redir_perm	826,173
notmodified	603,808
duplicate	958,431
unfetched	565,990,788
gone	65,527
fetched	408,368,666

From the total number of fetched documents only 614,724 were relevant enough to be indexed. That is only 0.0015% of the total documents retrieved.

A note on costs. The total cost over this two week crawl was less than $500 USD on the Amazon cloud. Obviously, this is considerably less expensive than buying hardware and attempting on an in-house cluster. We also used spot instances to decrease the costs.

V. CONCLUSIONS

Based on the empirical results we had after optimizing Nutch for focused crawling we have learned that there are major issues that can only be reproduced at large scale and they only get worse as the search space gets bigger. Among the two most difficult problems we ran into we mention performance degradation due sparse data distribution and duplicated content in the Web.

We also learned that some issues cannot be addressed by improving the focused crawl alone and there are problems out of our control that probably won't be solved by crawl processes themselves. Problems like prohibitive values for crawlers in robots.txt, poor performance in remote hosts, incorrect implementation of web standards, bad metadata and semantically similar content.

Despite the complexity of the overall problem we introduced different mitigation techniques to tackle some of the issues with good results. Search space pruning using manually tagged relevant documents, optimized configuration values for improving fetching times and the "discover and move on" approach for relevant data were the ones that proved to be the most effective.

Future work will be focused on optimizing the scoring algorithms and introducing graph context[1][11] and semantic information to make the crawl more adaptable and responsive to sparse data distribution. We also want to develop data type specific responses so that we both maximize the data set and data service information captured without crawling large amounts of very similar information.

BCube is committed to sharing all findings with the community and all the work presented here is the result of open sourced code available at http://github.com/b-cube. Our Nutch crawldb set with +100 Million URLs that might be useful to other research is publicly available in AWS S3 (200.62GB with more than 6761 unique domains)[15][16].

REFERENCES

[1] Michelangelo Diligenti, Frans Coetzee, Steve Lawrence, Lee C. Giles, Marco Gori. Focused Crawling using Context Graphs
26th International Conference on Very Large Databases, VLDB 2000 pp. 527-534, September 2000.

[2] O. Etzioni, M. Cafarella, D. Downey, et al. Web-scale information extraction in KnowItAll**.** 2004.

[3] John Garofalakis, Yannis Panagis, Evangelos Sakkopoulos, Athanasios Tsakalidis. Web Service Discovery Mechanisms: Looking for a Needle in a Haystack? Journal of Web Engineering, Rinton Press, 5(3):265-290, 2006.

[4] Page, L., Brin, S., Motwani, R., Winograd, T. The Pagerank Citation Algorithm: Bringing Order to the Web. Technical Report, Stanford Digital Library Technologies, 1998.

[5] Yang Sun, Ziming Zhuang, and C. Lee Giles. A Large-Scale Study of Robots.txt. The Pennsylvania State University, ACM 2007.

[6] Wei Xu, Ling Huang, Armando Fox, David A. Patterson and Michael Jordan. Detecting Large-Scale System Problems by Mining Console Logs. University of California, Berkeley Technical Report No. UCB/EECS-2009-103 July 21, 2009.

[7] Li, Wenwen , Yang, Chaowei and Yang, Chongjun(2010) 'An active crawler for discovering geospatial. Web services and their distribution pattern - A case study of OGC Web Map Service', International Geographical Information Science, 24: 8, 1127 — 1147

[8] Huang, Shengsheng, Jie Huang, Jinquan Dai, Tao Xie, and Bo Huang. "The HiBench Benchmark Suite: Characterization of the MapReduce-based Data Analysis." 2010 IEEE 26th International Conference on Data Engineering Workshops ICDEW 2010.

[9] Hati, Debashis, and Amritesh Kumar. "An Approach for Identifying URLs Based on Division Score and Link Score in Focused Crawler." International Journal of Computer Applications IJCA 2.3 (2010): 48-53.

[10] Cafarella, Mike, and Doug Cutting. "Building Nutch." Queue 2.2 (2004): 54. Web.

[11] Liu, Lu, and Tao Peng. "Clustering-based Topical Web Crawling Using CFu-tree Guided by Link-context." Frontiers of Computer Science Front. Comput. Sci. 8.4 (2014): 581-95.

[12] Isaacson, Joe. The Science of Crawl (Part 1): Deduplication of Web Content.2014 http://blog.urx.com/urx-blog/2014/9/4/the-science-of-crawl-part-1-deduplication-of-web-content web.

[13] White, Chris, Mattmann, Chris. "Memex (Domain-Specific Search)." Memex. DARPA, Feb. 2014. Web.
http://www.darpa.mil/program/memex

[14] Lopez, L. A.; Khalsa, S. J. S.; Duerr, R.; Tayachow, A.; Mingo, E. The BCube Crawler: Web Scale Data and Service Discovery for EarthCube. American Geophysical Union, Fall Meeting 2014, abstract #IN51C-06

[15] Nutch Logs are stored in: s3://bcube-nutch-test/logs and crawl data:
https://s3-us-west-2.amazonaws.com/bcube-crawl/big-crawl-clean-100m

[16] Nutch JIRA Tickets:
https://issues.apache.org/jira/browse/NUTCH-2044
https://issues.apache.org/jira/browse/NUTCH-2032
https://issues.apache.org/jira/browse/NUTCH-2033
https://issues.apache.org/jira/browse/NUTCH-2034
https://issues.apache.org/jira/browse/NUTCH-2035
https://issues.apache.org/jira/browse/NUTCH-2046

[17] Krugler, Ken. Performance problems with vertical/focused web crawling 2009 Web. http://ken-blog.krugler.org/2009/05/19

[18] Truslove, I.; Duerr, R. E.; Wilcox, H.; Savoie, M.; Lopez, L.; Brandt, M. Crawling The Web for Libre. American Geophysical Union, Fall Meeting 2012, abstract #IN11D-1482

[19] Alpha-Beta Pruning Algorithm, Wikipedia.
https://en.wikipedia.org/wiki/Alpha%E2%80%93beta_pruning