Implementation of A Distributed Web Community Crawler

Seonyoung Park, Youngseok Lee
Chungnam National University
Daejeon, Republic of Korea
{siraman, lee}@cnu.ac.kr

***Abstract*—A web community is an important space for online users to exchange information, ideas and thoughts. Due to collective intelligence of the web communities, marketing and advertisement activities have been highly focused on these sites. While articles in the web communities are open to the public, they cannot be easily collected and analyzed, because they are written in natural languages and their formats are diverse. Though many web crawlers are avaialble, they are not good at gathering web documents. First, the URLs of web articles are frequently changed and redundant, which will make the crawling job difficult. Second, the amount of articles is significantly large that the crawler should be designed in a scalable manner. Therefore, we propose a distributed web crawler optimized for collecting articles from popular communities. From the experiemnts we showed that our implementation achieves high throughput compared with the open-source crawler, Nutch.**

Keywords*—*Distributed web crawler, community, web forum.

I. INTRODUCTION

Web or online communities are important cyber-space where people share information about various topics with others. For instance, Ubuntu forum [1] is a place where people can ask and get information of Ubuntu Linux. Apple forum [2], like Ubuntu forum, is a web site where people can exchange reviews, opinions, and tips regarding various apple products such as iPhone, iPad, or Macbook. Joonggonara [3] is one of the most huge online second-hand product community in Korea.

Due to the rich information and the large population of online communities, many studies have been performed to analyze the opinion trends of communities, Before the analysis process of the web communities, we need to gather documents or comments from the web sites. Though well-known web crawlers are being used for this purpose, universal web crawlers do not fit to collect articles from the web communities [4], [5] because of the following reasons. First, URL flipping often occurs in online community web sites. Second, universal web crawlers will collect plenty of duplicated or meaningless web pages. In order to address this problem, J. Jiang et al. [5] developed a crawling method called FOCUS which crawls only meaningful posts, discovers and generates the url patterns.

While the web crawler studies [4], [5], [6] encompass several methodologies for crawling documents from web communities, their web crawlers run on a single server and cannot address the scalability issue against the rapid growth of web communities, In order to solve this problem, a few distributed web crawlers have been developed and released. However, there are only few open source projects such as Apache Nutch [7] which is based on the Hadoop framework. Nevertheless, Apache Nutch is not suitable for collecting information from onlne communities because it gathers duplicated or meaningless web pages. Furthermore, it is not easy to extend or customize Nutch for the web community crawler, because Apache Nutch is not well documented.

In this paper, we present implementation and evaluation results of the distributed web crawler based on ZooKeeper to collect information from online communities. Among several candidates such as RabbitMQ [8], ZeroMQ [9], and OpenMPI [10] for the distributed applications, we have chosen ZooKeeper, because it provides a distributed coordination framework. From the experiments on the Linux clusters, we have shown that our implementation outperforms Nutch in the aspect of gathering performance as well as accuracy.

II. RELATED WORK

A. *Web crawler for online communities*

Web crawlers are programs that automatically visit and download web pages, following hyperlinks in web pages [11]. A Web crawler starts with a list of URLs to hit, called the seeds. As the crawler traverses these URLs, it identifies all the hyperlinks in the page and adds them to the list of URLs to visit, called the crawl frontier. The frontier could be implemented by FIFO queue or priority queue, and supports deduplication. There are many open source web crawlers including HTTrack [12], Heritrix [13], and Scrapy. HTTrack is a free and open source Web crawler and allows users to mirror web sites from the Internet. Heritrix is a web crawler designed for web archiving developed by Internet Archive and free software license written in Java. Scrapy is a open source web crawling framework for web scraping. and is written in Python. However, in general, universal web crawlers cannot gather the documents from web communities, because of URL flippings and dulications of the web community.

B. *Distributed web crawler*

While studies [4], [5], [6] on the web crawler have proposed several methodologies for crawling online communities, none of these studies have addressed the scalability that a crawler may face against the rapid growth of online communities.

There are two methods to implement distributed web crawlers [15]: dynamic assignment and static assignment. In a dynamic assignement implementation of distributed web crawlers, a central server assigns new URLs to different crawlers dynamically. This allows the central server to balance the load of each crawler. There is a fixed rule to assign the new URLs to the crawlers. In static assignment, a hashing function can be used to transform URLs or complete web site names into a number that corresponds to the index of the corresponding crawling process.

Nutch is one of the most popular distributed web crawlers written in Java and released in open source. Nutch can run on multiple machines while running in a Hadoop cluster. In Fig. 1, we describe how Apache Nutch works. First, Apache Nutch creates a new crawlDB. and injects root URLs into the CrawlDB from seed urls. Then, Apache Nutch generates a fetchlist from the crawlDB in a new segment, fetches contents from URLs in the fetchlist, and updates the WebDB with links from fetched pages. Apache Nutch repeats this processes until the required depth is reached.

C. *ZooKeeper*

ZooKeeper [16] is the Apache open source project that provides distributed configuration service, synchronization service, and naming registry for large distributed systems. ZooKeeper includes a file system-like API comprised of a small set of calls that enables applications to implement their own primitives. ZooKeeper makes use of small nodes, called Znodes, to create and register master and workers, or to manage distributed lock. Futhermore, Znodes representing a master will maintain information on managing worker, and Znodes for workers will retain information on the job progress. In our work, we resort ZooKeeper for the distributed cooardination framework to implement the distributed web crawler.

III. Design and Implementation of Distributed Web Crawler

A. *Overview*

At first, we considered Nutch as the candidate by extending its functions on the distributed platform, but we excluded Nutch because it requires a lot of resources based on Hadoop and it is not easy to address diverse requirements. Then, we decided to develop our own distributed web crawler for web forums instead of extending Nutch.

The overall architecture of our system is illustrated in Fig. 2. A master and multiple workers run on a Linux cluster. A master node monitors and manages workers and their tasks. Workers consists of three modules LoginModule, HTTP Client Module and Main Module, and they run independently without any coordination among other crawlers. Crawled web pages are stored in the local storage.

Figure 3 illustrates how distributed web crawlers work. For the sake of simplicity of our implementation, we assign a single crawling task to one worker. For instance, we need four workers when collecting documents from four web boards of Apple forum. If the number of worker nodes is smaller than that of crawling tasks, then worker nodes that have completed the crawling task will be assigned to the next crawling task.

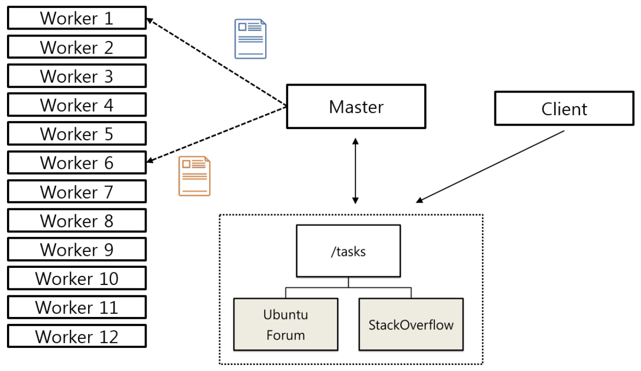

Fig. 3. Distributing crawling tasks to workers

B. *Internal view of workers*

We now explain the internal view of the workers how distributed web community crawler works. The implementation of the distributed community crawler comprises three modules: login, HTTP client and crawler core. The login module is responsible for logging into online communities when a valid account is needed before crawling starts. The HTTP client module sends HTTP request messages and processes HTTP response messages. The crawler core module manages whole process of the distributed community crawler.

We implemented the login module using Selenium Web-Driver [18] which allows users to write web interface tests to drive a browser. Selenium WebDriver is a tool for writing automated tests of web sites. It aims to mimic the behaviour of a real user, and interacts with the HTML of the application. However, web browsers are slower than other HTTP libraries since web browsers must render web pages. Consequently, since crawling web pages using the webdriver makes progress much slower than using a simple HTTP library, we have developed the login module and the HTTP client module, respectively.

The HTTP client module is responsible for accessing most web pages except that you need login. We developed the HTTP client module using Apache HttpClient [19] which is a simple HTTP library in java. It includes basic codes for handling errors such as timeout. In particular, we set the web browsing delay of the HTTP client module for the short time, because many web servers block web crawlers to avoide the overload by accessing web pages quickly.

Figure 4 illustrates how the main module works. Initially, the main module checks whether login is needed or not, and it requests the login module to login if necessary. Then, the login module returns a cookie to the main module. The main module forwards the cookie to the HTTP client to generate HTTP request messages. A seed set of users is inserted into the frontier. Then at each step, the first entry of the frontier is popped up and all URLs will be crawled. The main module uses a frontier which is a queue data structure to store the list of URLs that had been seen but not crawled. Once all of the URLs are crawled, the URL is marked as visited, and it is also stored in a separate queue in the frontier. Crawled web pages are stored in the local storage.

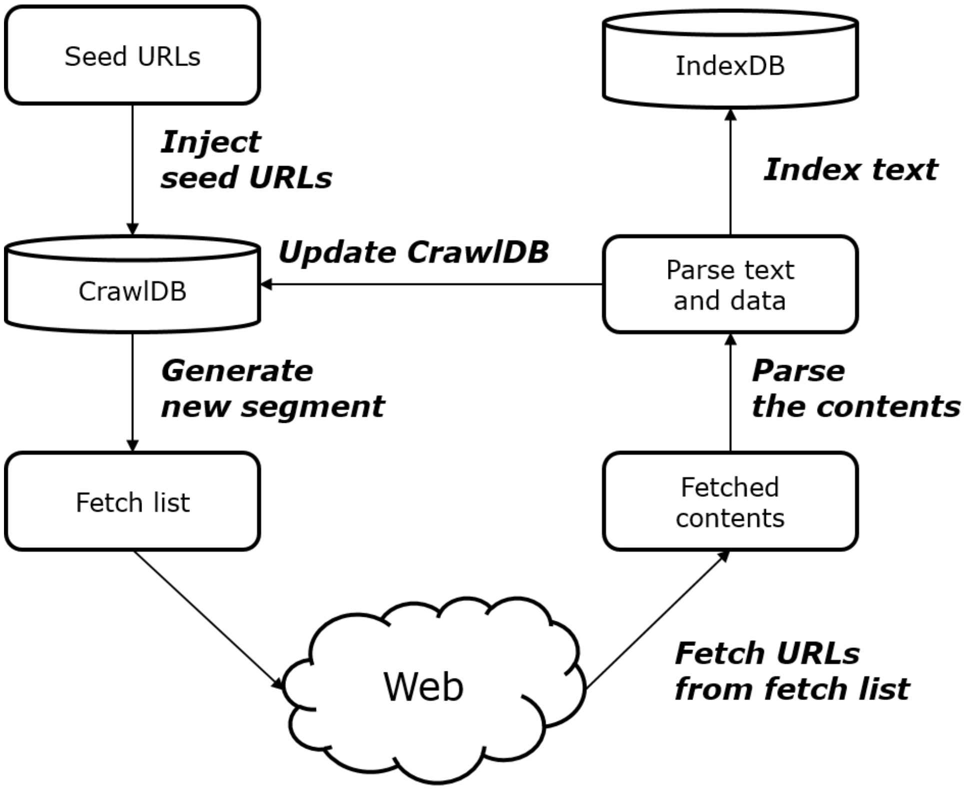

Fig. 1. Crawling process of Apache Nutch

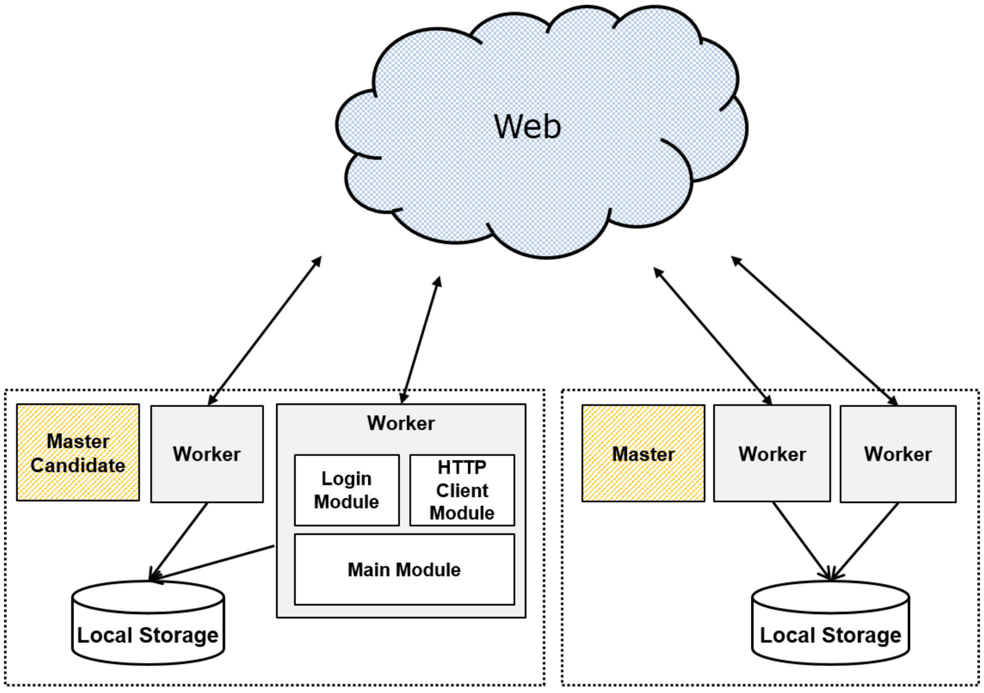

Fig. 2. Overview of distributed web crawler

C. Fault tolerance of web crawling

In Fig. 5, we describe the fault tolerance function in our implemenation that provides the process or node recovery process of the web crawler.

There are three cases that may experience failures in the distributed web community crawler. The first is when a process is stopped by accident. In this situation, the recovery process follows the flow of the recovery procedure as shown in Fig. 5 Intermediate progress status of each worker is updated to its

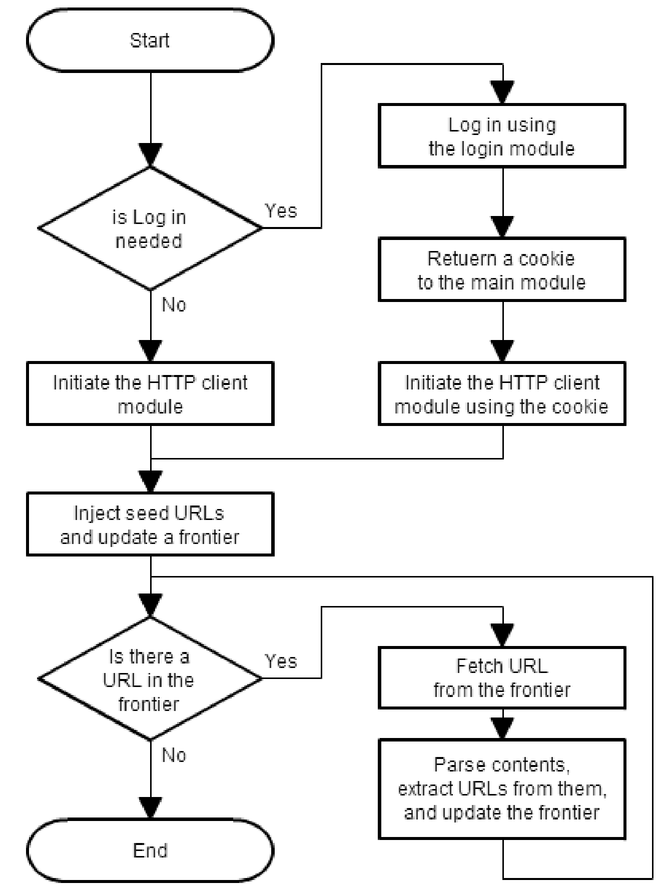
Start
is Log in needed
Yes
No
Log in using the login module
Retuern a cookie to the main module
Initiate the HTTP client module
Initiate the HTTP client module using the cookie
Inject seed URLs and update a frontier
Is there a URL in the frontier
Yes
No
Fetch URL from the frontier
Parse contents, extract URLs from them, and update the frontier
End

Fig. 4. How the main module works

Znode every 10 seconds so that other worker can replace its job under accidental errors. The second one is that the master node fails because of errors such as connection loss. Creating "/master" Znode is only one way to become a new master because a master does not maintain any information about workers and tasks. The last case is the failure that occurs in ZooKeeper ensemble itself. If a ZooKeeper node fails, workers connected to the ZooKeeper node does not play its role. In this case, ZooKeeper tries to re-establish a new connection to other ZooKeeper nodes.

IV. Performance Evaluation

A. Environment

In order to carry out performance evaluation of our distributed web crawler, we selected three Korean web communities and three international web communities. The basic information of the candidate web communities are explained in Table I. We compared the performance of distributed web crawlers for online communities with that of single-process web crawler HTTrack which is one of the most popular web crawler and provides high performance [20].

We built a Linux cluster consisting of five machines. Each node is equipped with Intel(R) Celeron(R) CPU 743 @ 1.30GHz with 4GB of main memory. The host machine runs Ubuntu Linux 12.04.2 with kernel 3.2.0-4 in 64-bit mode. The source code is compiled and launched using the Oracle JDK 1.8.0-5.

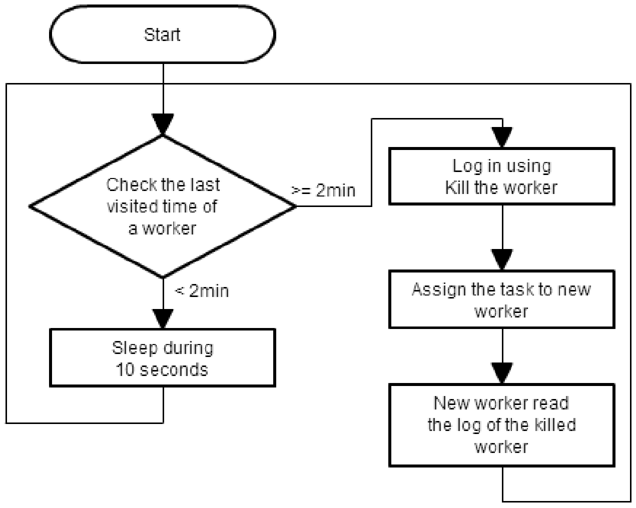
Start
Check the last visited time of a worker
>= 2min
< 2min
Sleep during 10 seconds
Log in using Kill the worker
Assign the task to new worker
New worker read the log of the killed worker

Fig. 5. Recovery of a crawling task

TABLE I. Target web communities. Alexa rank is from the site http://www.alexa.com

Name	URL	Alexa rank
Ppomppu	http://www.ppomppu.co.kr	4455
Ilbe	http://www.ilbe.com	3306
Todayhumor	http://www.todayhumor.co.kr	4685
Gaia online	http://www.gaiaonline.com/	7392
4chan	http://www.4chan.org/	1046
Offtopic	http://www.offtopic.com/	64605

Our implementation and Nutch launch 18 crawling processes concurrently. The maximum number of threads per each task for Nutch is set to 20. For instance, the number of threads in one node is 80, which executes four crawling processes. On the other hand, in our implementation, one crawling process has one thread. The number of threads for HTTrack is set to be 10. Other setting is as follows: the number of Mappers per each Hadoop node is 8, the number of Reduces is two, I/O buffer size is 1024 KB, and HDFS block size is 128 MB.

B. Results

TABLE II. Results of web crawling experiments

	Apache Nutch	HTTrack	Our Implementation
Total	4626	11213	10342
Valid	3157	7893	10342
Duplicate	1344	1023	0
Uninformative	125	2297	0

Table. II summarizes the experimental results. "Total" means the number of all the web pages that have been collected during the experiment. "Duplicate" is the number of redundant web pages. "Valid" is the number of web pages that have been valid. "Uninformative" is the number of web pages that are not meaningful. From the results, we confirm that our implementation achieves high accuracy as well as high throughput. Our implementation did not collect the duplicate or redundant web pages, whereas Nutch gathered 1,344 pages which is 30 % of 4,626 total collected documents. Similarly, HTTrack has collected 1,023 duplicate web pages out of 11,213 total documents, which is 9 % of duplication. Thus, Nutch and HTTrack have wasted the network bandwidth by

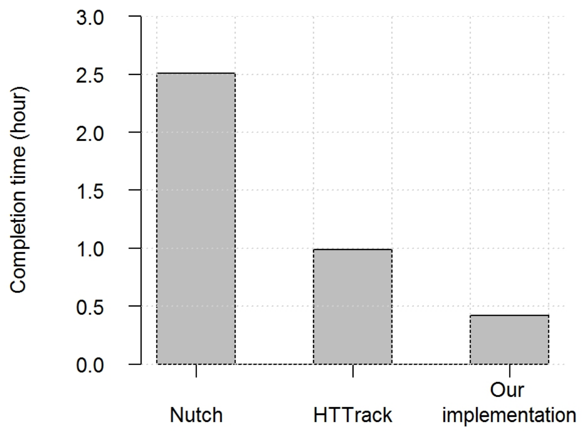

Fig. 6. Completion time of crawling articles of web communities

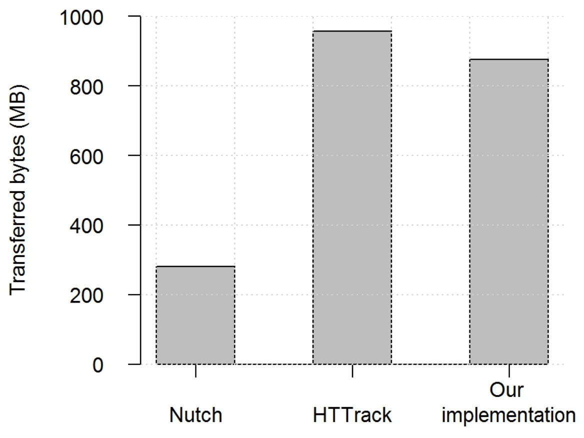

Fig. 7. The amount of traffic of crawling articles of online communities

gathering duplicate web pages. Uninformative web documents have been collected by Nutch and HTTrack. We can limit the URLs by using CSS selector, which is important to avoid the redundant or uninformative web page collection.

From Fig. 6, we can find that our implementation completes the collection job fast. Nutch takes 2 hour and 30 minutes, HTTrack takes 59 minutes, and our implementation takes 25 minutes. As Nutch is based on the MapReduce framework, it is slower than others. HTTrack is slower than our implementation, because it does not provide the fast processing capability with the distributed computing resources.

In Fig. 7, the traffic amount caused by web crawling is compared. For the experiments Nutch collects 280 MB of web pages, HTTrack collects 958 MB, and our implementation collects 876 MB. Since Nutch has crawled a small number of articles, its byte count is small. However, HTTrack and our implementation show the large traffic volume. Still, our implementation gets the smaller traffic volume than HTTrack while maintaining the high accuracy.

From Fig. 8, our implementation also achieves high throughput of 594 KB/s, which corresponds to 18.6 times of Nutch and 2.1 times of HTTrack. Nutch gathers 32 KB of web pages per second and HTTrack gathers 275 KB of web pages per second. In the aspect of the web page throughput, our implmentation collected 6.8 pages per second, while Nutch get only 0.5 page per second. The reason of low throuhput of Nutch is the MapReduce architecture, because a MapReduce job requires a Java virtual machine and the long initialization job process in the cluster. In the experiment, we observed that it usually takes at least 10 seconds to run a simple MapReduce job. In addition, Nutch does not distribute the load of the web page collection process to multiple workers, and it often utilizes the partial computing resources of the cluster. HTTrack uses a single-process web crawling framework, which is not enough to accommodate the overload of large web communities.

V. Conclusion

In this paper, we have presented our own implementation of the distributed web community crawler and its performance evaluation results. We have designed the web crawler that can run in the distributed manner, and we have added the login and load balancing functions to the web crawler. Based on ZooKeeper, we have built a distributed web crawler suitable for the web community. The experiments show that our implementation achieves the high througput and the high accurarcy.

Acknowledgement

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology grant number (NRF-2013R1A1A2007326) and the corresponding author is Youngseok Lee.

References

[1] Ubuntu forum, http://ubuntuforums.org/

[2] Apple forum, https://discussions.apple.com/index.jspa

[3] Joonggonara, http://cafe.naver.com/joonggonara.cafe

[4] R. Cai, J.-M. Yang, W. Lai, Y. Wang, and L. Zhang, *iRobot: an intelligent crawler for web forums*, in the Proceedings of the 17th international conference on World Wide Web, Beijing, China, 2008.

[5] J. Jiang, N. Yu, and C.-Y. Lin, *FoCUS: learning to crawl web forums*, in the Proceedings of the 21st international conference companion on World Wide Web, Lyon, France, 2012.

[6] J.-M. Yang, R. Cai, C. Wang, H. Huang, L. Zhang, and W.-Y. Ma, *Incorporating site-level knowledge for incremental crawling of web forums: a list-wise strategy*, in the Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining, Paris, France, 2009.

[7] Apache Nutch, https://nutch.apache.org/

[8] RabbitMQ, https://www.rabbitmq.com/

[9] ZeroMQ, http://zeromq.org/

[10] Open-MPI, http://www.open-mpi.org/

[11] B. Liu, *Web Data Mining: Exploring Hyperlinks, Contents, and Usage Data*, Springer, 2007.

[12] HTTrack, http://www.httrack.com/

[13] Heritrix, https://webarchive.jira.com/wiki/display/Heritrix/Heritrix

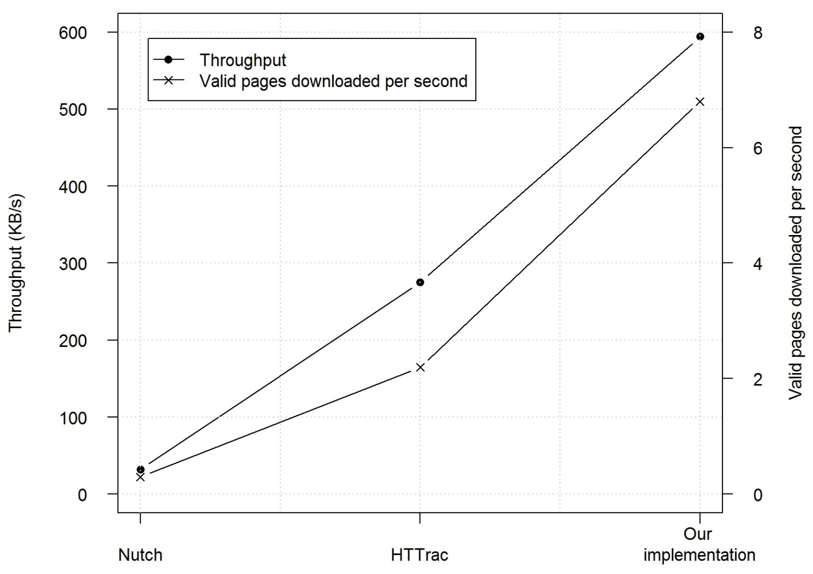

Fig. 8. Throughput and valid page count per second

[14] Scrapy, http://scrapy.org/

[15] J. Cho and H. Garcia-Molina, *Parallel crawlers*, in the Proceedings of the 11th international conference on World Wide Web, Honolulu, Hawaii, USA, 2002.

[16] Apache ZooKeeper, http://zookeeper.apache.org/

[17] F. Junqueira and B. Reed, ZooKeeper: Distributed Process Coordination: O'Reilly Media, 2013.

[18] Selenium WebDriver, http://docs.seleniumhq.org/docs/03_webdriver.jsp

[19] Apache HttpClient, http://hc.apache.org/

[20] HTTrack Evaluation, http://e-records.chrisprom.com/httrack-evaluation/