MATEC Web of Conferences **22**, 02029 (2015)
DOI: 10.1051/matecconf/20152202029

Optimization of Distributed Crawler under Hadoop

Xiaochen Zhang* & Ming Xian
College of Electronic Science & Engineering, National University of Defense Technology, Changsha, Hunan, China

ABSTRACT: Web crawler is an important link in the data acquisition of the World Wide Web. It is necessary to optimize traditional methods so as to meet the current needs in the face of the explosive growth of data. This paper introduces the process and the model of the current distributed crawler based on Hadoop, analyzes reasons for influencing the crawling efficiency, and points out defects of the parameter setting, the Urls distribution and the operating model of distributed crawler. The working efficiency is improved through the optimization of parameters configuration and the optimizing effect is further enhanced through the model modification. The experiment indicates that the working efficiency of the distributed crawler after the optimization is increased by 23%, which achieves the expected result.

Keywords: Hadoop, Nutch, distributed crawler

1 INTRODUCTION

Web crawler, also known as web spider, is a program that obtains the required information on the World Wide Web automatically in line with rules set by users. In fact, web crawler is applied in the crawling stage of the search engine. With the explosive growth of information, the traditional single-thread crawler is unable to meet our demand for information both in quantity and quality. Distributed crawler has more excellent performance in terms of crawling quantity, crawling mode, crawling flexibility, crawling efficiency, and so on.

Nutch [1] is a search engine realized by the open-source Java. It provides us with all the required tools for operating our own search engines, including the full-text search and the Web crawler. As an excellent open-source tool, Nutch can be integrated in the environment of Hadoop so as to accomplish tasks of distributed acquisition and distributed query. Hadoop is now a mainstream open-source distributed computing tool and has become a synonym for the current big data. Hadoop features high reliability, high scalability, high efficiency and high fault tolerance. Hengfei Zhan [4] optimized Nutch by limiting the latency time of crawling and monitoring invalid links. Although the crawling efficiency is improved in this way, the recall ratio of webs is influenced. Wei Yuan [5] improved the efficiency and accuracy of Nutch by optimizing relevant parameters and models of Nutch. But the optimization of parameters only reduces the task of each node and improves the efficiency by increasing the number of nodes linearly. Obvious effects cannot be reflected in practical applications through the modification of I/O model. Shilong Zhou, et al [6] increased the performance of job through the parameter optimization of Nutch crawling based on Starfish so as to optimize the performance of Nutch. But there are still interval values of monitoring and the optimal value can be only found by continuous attempts. The deployment of the Nutch after modification is complicated and the number of job configuration parameters of the Starsfish optimization is limited. For this reason, this paper optimizes relevant parameters and models of the distributed crawlier so as to improve the working efficiency of crawlers.

2 TECHNICAL BACKGROUND

2.1 *Introduction of MapReduce*

Formed by the combination of two classical functions, MapReduce is one of the core parts of Hadoop. Mapping is the process that applies data to be processed in the same operation. Reducing is the process that returns data of Mapping back to the comprehensive result after traversal.

The figure 1 is the operational process:

The front task is divided into several small tasks. Parts needed to be processed identically are respectively mapped through a large number of nodes and then integrated through reduce so as to obtain the final result. MapReduce will generate a huge number of temporary files, which require distributed management and access.

2.2 *Introduction of Nutch*

Nutch is an open-source web crawler project with two functions, that is, web crawling and query. The crawler can be directly used in web fetching and then index establishment. But the query feeds back required information of users in accordance with the index. Nutch1.7 is now a relatively mature version, the bottom of which stores information in the HDFS way so

*Corresponding author: x.c.zhang@outlook.com

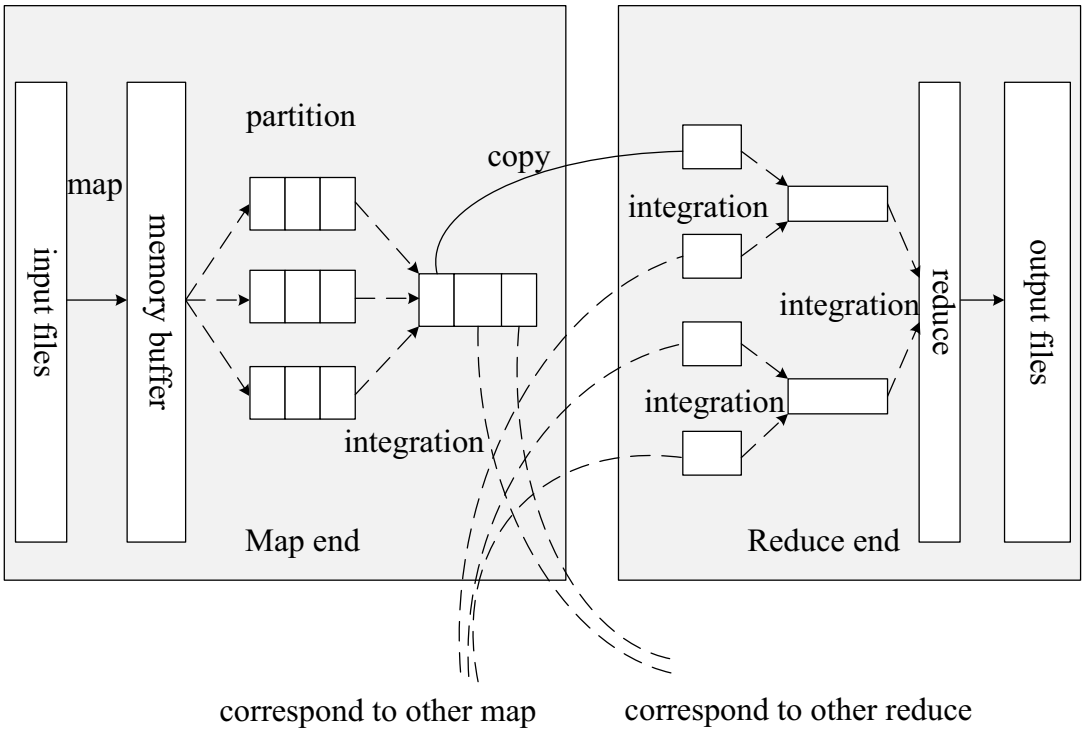

Figure 1. Operational process of MapReduce.

as to meet the conditions of Hadoop architecture. It is used in this experiment.

There are mainly 10 operating steps of Nutch:

1) Select and scrap the initial Urls;

2) Inject the initial Urls in the WebDB;

3) Inject the fetchlist in the segment of the generate part;

4) Fetch webs from the World Wide Web according to the fetchlist;

5) Analyze the content and generate data;

6) Update the database according to the fetched content;

7) Repeat the part from step 3 to step 6 until the preset fetch depth is satisfied;

8) Establish an Index;

9) Delete the repeated content and establish an index database through the integration;

10) Users check the final result.

The figure 2 is the operational process of Nutch:

The core part of the whole process can be summarized as three databases and four operational modules. Three databases are WebDB, segment and Index. Four operational modules are Inject, Generate, Fetch and Update.

WebDB: Store Urls information needed to be fetched, including the initial Urls, link relations among Urls and the update of Urls after fetching.

Segment: Each crawling generates segment and the temporary storage of segment fetches queues. Web pages fetched in the crawling process and generated indexing information.

Index: Obtain the index database of all web pages needed to be fetched through the combination and duplicate removal of all segments.

Inject: It includes three processes, that is, the process of adding, the given Urls list in WebDB, the process of eliminating duplicate Urls and the process of adding the updated Urls in WebDB.

Generate: Insert the list of WebDB into the fetching list. In this process, fetched Urls can be part screened, such as the identification of illegal Url and the limitation of the length of fetched Url. Sorted Urls are finally injected in segment.

Fetch: Fetch in line with the sequence of Urls in segment and store the obtained web page data and index in segment.

Update: Update WebDB in accordance with fetched Urls and newly generated data in the fetching process.

3 EXISTING PROBLEMS OF NUTCH CRAWLER

Through the analysis on the operating principle of the distributed crawler and the factors that influence the working efficiency of the distributed crawler, problems of crawling efficiency can be summarized as follows.

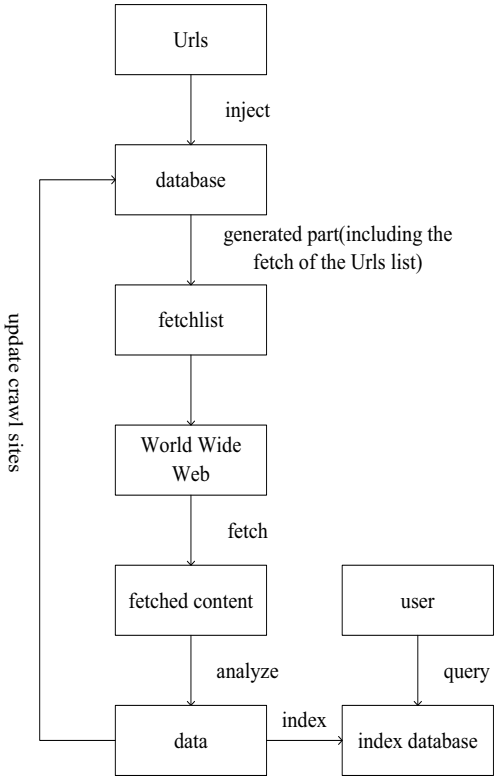

Figure 2. Operational process of Nutch.

3.1 *Few initial Urls*

Crawling of pages is conducted by distributed crawler, which features parallel and multi-thread operation. If the number of initial Urls is insufficient and the scale of nodes is huge, a large number of threads will be in the idle state and the crawling efficiency will be influenced as well. The distribution of demand information is wide on the Internet. The effect of crawling for a dozen layers when the initial WebDB content is simple can be achieved by crawling for a few layers when the initial WebDB content is rich. The proportion of useless and duplicate information will be increased and the used time will be increased largely as well when the crawling layer reaches a certain depth. Final crawling efficiency will be influenced thusly.

3.2 *Mal-distribution of Urls*

Distributed crawler adopts the parallel and multi-thread way. There are multiple threads completing the fetching task in one Map. In the process of one fetching, if the Url content fetched by thread A is complicated with rich pages while the Url content fetched by other threads is single in pages, other threads will accomplish the fetching task very soon. However, the next round of fetching will not start when other threads accomplish the task. It will start when thread A accomplishes the task. Crawling efficiency will be inevitably influenced if the case occurs frequently.

3.3 *Different delay time of Urls*

It can be found from the practical fetching process that most of the pages are able to respond quickly and only a few pages need long latency time. It is concluded from the analysis that some administrators of Urls set a high crawling delay and some malicious websites even set the crawling delay as a few days. The delay of a few Urls is relatively high due to the slow access of the server. Some Urls set the number of IP of accesses in the same time period. When the server IP connection is limited, it will queue up until the access is permitted. The three situations mentioned above will influence the page crawling time of a thread and lead to the problem of thread idleness.

3.4 *A great deal of time spent by index*

An index is generated after the web crawling. That is to say, the fetched data is converted into texts and then stored in the database after the analysis. Data will be exchanged constantly with the hard disk if the generation and the storage operation are conducted in each crawling. Thus, the I/O load of the crawler is large, which has an impact on the final crawling efficiency.

4 T HE OPTIMIZATION SCHEME OF NUTCH CRAWLER

The following optimization scheme is put forward in accordance with the problems mentioned above:

4.1 *Increase the number of initial Urls*

Increasing the number of initial Urls is the simplest and the most effective way of improving the crawling efficiency. The selected Urls should be preliminarily screened in the expansion of the initial WebDB so that effective addresses can be selected as many as possible. In the meantime, the index of Nutch is optimized so as to simplify the index range and eliminate unnecessary part of the index.

First, enlarge the memory buffer and improve the reading performance of Hadoop. The modification of parameters is provided below:

blockdev --report

blockdev --setra 1024 /dev/sda

The default buffer is 256. The time of disk seek and the I/O wait time can be reduced and the reading performance can be improved greatly if the default buffer is modified as 1024. This is the level that can be achieved by existing hardware.

mapred.reduce.tasks: 4

mapred.reduce.parallel.copies: 10

dfs.replication: 2

mapreduce.map.output.compress: true

mapreduce.map.output.compress.codec: org.apache.hadoop.io.compress.DefaultCodec

mapreduce.output.fileoutputformat.compress: true

mapreduce.reduce.shuffle.parallelcopies: 10

mapred.child.java.opts: -Xmx512m

mapreduce.task.io.sort.factor: 15

mapred.tasktracker.tasks.map.maximum:12
mapreduce.tasktracker.tasks.reduce.maximum: 10

The system default is relatively conservative in order to meet the needs of all users. The parameters above are optimized in line with the performance parameters of our computers, mainly including the number of map and reduce, the module size and the number of backup, the memory capacity of each task, and so on. The optimization can effectively improve the reading and writing speed, reduce the memory and CPU idle probability, and read more content in the memory instead of the hard disk so as to achieve the goal of improving the fetching efficiency.

4.2 *Model modification*

Webpage content on the Internet is rich and diverse. Threads will be idle with lower efficiency if the same fetching strategy is adopted. However, blind abandonment of part of pages will influence the integrity of information fetching. So, it is necessary to modify the fetching model and establish a hard Url database (hardUrlDB). The fetching process is modified as two times of fetching and the maximum waiting time of fetching is modified as 3 seconds. Pages with the waiting time of more than 3 seconds are put into hardUrlDB. After the crawling of all pages, pages in hardUrlDB are fetched and crawled with modified parameters so as to achieve the goal of improving the efficiency.

Key parameters are set as follows:

The first fetching:
http.max.delays: 3
fetcher.server.delay: 3
fetcher.threads.fetch: 50
fetcher.threads.per.host:20

Where, the waiting latency is set as 3 seconds, which means that Urls with the waiting time of more than 3 seconds in the first fetching are put into hardUrlDB. Thus, the fetching efficiency of the first time will be significantly improved. Meanwhile, the latency between two fetches is reduced to 3 seconds, the number of fetching threads is enlarged to 50, and 20 threads have access to the same host simultaneously in order to improve the fetching efficiency.

The second fetching:
http.max.delays: 100
fetcher.server.delay: 100
fetcher.threads.fetch: 50
fetcher.threads.per.host: 50

The second fetching is the fetching of pages in hardUrlDB. Where, the waiting latency is set as 100 seconds so as to make sure that valid web pages can be successfully fetched and improve the integrity of information fetching. In the meantime, the latency between two fetches is also set as 100 seconds, the number of fetching threads is enlarged to 50, and 50 threads have access to the same host simultaneously. It is very necessary to keep the integrity of information fetching as far as possible in the case of sacrificing certain efficiency.

The modified flow chart is shown below:

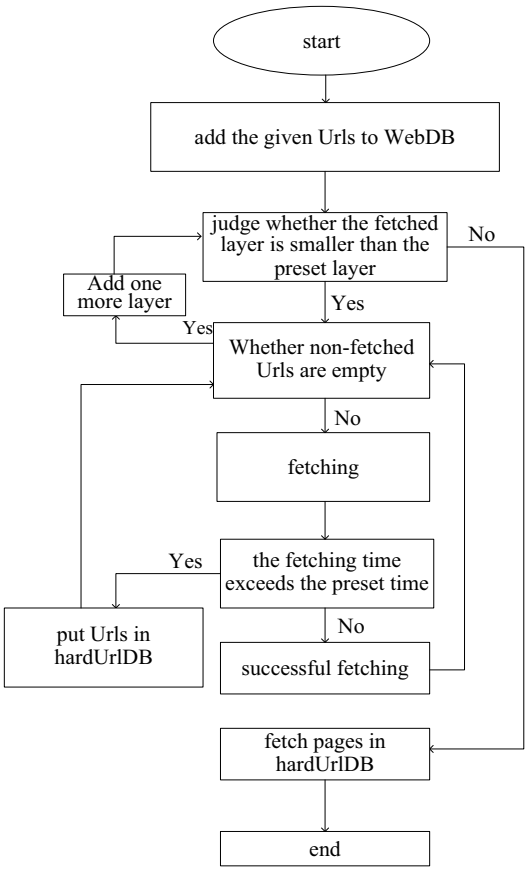

Figure 3. Part of the process after the model modification

4.3 *Setting of other parameters*

indexer.mergeFactor:100

This parameter indicates the number of indexes that are written in the database. The improvement of the parameter is able to reduce the time of disk seek and enhance the index efficiency. But an excessively large parameter will lead to the opening error of files. So, the parameter is properly increased here.

indexer.minMergeDocs: 1000

This parameter refers to the minimum number of merging files in the establishment of the index, which greatly influences the memory performance. Therefore, the parameter should be increased in a reasonable range so as to improve the efficiency.

5 EXPERIMENTAL RESULTS AND ANALYSIS

5.1 *Experimental environment*

The environment of this experiment is the campus network with a bandwidth of 4Mbps. Three hosts are applied in the construction of the distributed crawler. The configuration of the host node is: Intel(R)

Core(TM) i7-4770 CPU @ 3.40GHz, 32G internal storage, CentOS6 operating system. The configuration of two slave nodes is: Intel(R) Pentium(R) 3558U @ 1.70GHz, 8G internal storage, CentOS6 operating system. The initial Urls of crawling contain 3000 pages obtained from one time of crawling of a certain web portal.

5.2 *Experiment results*

Results of the monitoring on one crawling process are provided in the following table.

Table 1. Time of each part of a single crawling

Item	Time (s)	Percentage
inject	109	1%
generate	300	3%
fetch	4200	36%
parse	1283	11%
crawlDB	261	2%
link inversion	1000	8%
dedup	238	3%
index	4052	35%
cleanup index	129	1%

This crawling takes 11682 seconds altogether. It can be concluded from the above table that the fetching time is 4200 seconds and the indexing time is 4052 seconds, accounting for 36% and 35% of the time for one crawling. Therefore, the improvement of the fetch and index part of the distributed crawler is the most effective. In fact, the optimization of parameters of the distributed crawler and the improvement of the model are optimizations on the fetch part and the index part. The direction of our improvement can be proved to be correct thusly.

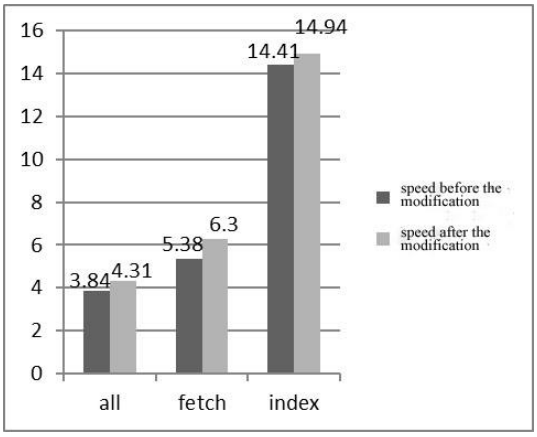

Figure 4. Comparison of the speed before and after the modification of parameters

The comparison of the crawling efficiency before and after the modification of parameters is shown in figure 4.

It can be seen from the data in the figure that the crawling speed of original parameters is 3.84 pages per second and the crawling speed of modified parameters is increased to 4.31 pages per second. The efficiency of Nutch and index is improved to a certain extent after the modification of parameters. The crawling speed of the fetch part is increased by 17% and the crawling speed of the index part is increased by 4%. The experiment indicates that the increase of the crawling efficiency of the fetch and index part influences the improvement of the final overall efficiency, which is increased by 12%.

The last figure is the comparison of crawling speeds before and after the modification of the model.

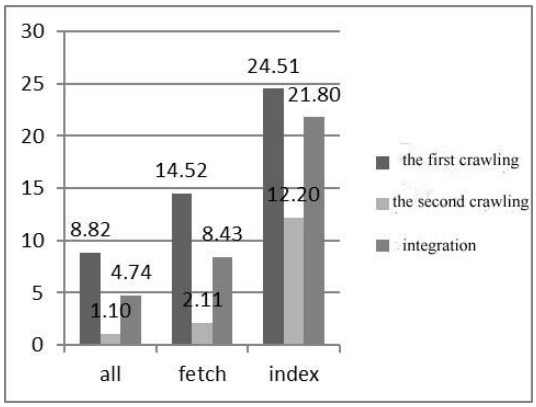

Figure 5. Crawling speed after the modification of the model

Table 2. Comparison of three fetching results

Item	all	fetch	index
The first time	3.84	5.38	14.41
The second time	4.31	6.30	14.94
The third time	4.74	8.43	21.80
Improvement	23.4%	56.7%	51.3%

The result obtained from the figure 5 is in consistent with the prediction. In the first crawling, pages with longer crawling latency are first put in hardUrlDB and pages with fast response speed are crawled so that the waiting time and the probability of thread idleness can be reduced and the crawling speed can be improved largely. It can be seen from the figure that the speed, 8.82 pages per second, is increased by 230%, which means that the latency of web pages has a great impact on crawling. In the second crawling, pages with longer response time are crawled again and the efficiency is reduced significantly. It further indicates that the major factor influencing the crawling efficiency is the website response time. But in order to keep the integrity of the crawling of web pages, this part is not abandoned. In conclusion, although the efficiency is lowered by the crawling of pages with longer latency, the overall efficiency is still improved.

It can be concluded from the table above that the crawling efficiency is improved after the modification of parameters and the page crawling efficiency is further increased by the model modification based on the

parameter modification. The overall crawling speed is increased by 23.4% compared with the original model, the crawling speed of the fetch part is increased by 56.7%, and the crawling speed of the index part is increased by 51.3%. It proves that the optimization of the distributed crawler is effective and both the parameter optimization and model modification can improve the crawling efficiency with expected results. However, in fact, the overall crawling efficiency is generally low and the analysis suggests that the low bandwidth of the network limits the working performance of the distributed crawler. The next job is to compare working efficiencies of distributed crawlers in different network environment through experiments.

6 CONCLUSION

This paper introduces the principle of distributed crawler based on Hadoop, analyzes existing problems of distributed crawler, optimizes the performance of distributed crawler through the modification of parameters and the model, and verifies through an experiment. The result suggests that the crawling efficiency of the improved distributed crawler is increased and the scheme is effective and feasible.

However, it can be found from the final result that the improvement of the overall efficiency fails to reach the improvement level of fetch and index. The next job is to carry out studies on this problem.

REFERENCES

[1] Nutch, Apache. 2010. Nutch. 2011-07-05. http://nutch. apache. org.

[2] Khare R, Cutting D. & Sitaker K, et al. 2004. Nutch: *A flexible and scalable open-source web search engine*, Oregon State University, 32.

[3] Crawler H. s. Nutch Crawler [EB].

[4] Zhan, H.F., Yang, Y.X. & Fang, H., et al. 2011. A study on distributed network crawler and its applications, *Journal of Frontiers of Computer Science and Technology*, 05(1): 68-74.

[5] Yuan, W., Xue, A.R. & Zhou, X.M., et al. 2014. A study on the optimization of distributed crawler based on Nutch, *Wireless Communication Technique*, 23(3): 44-47, 52.

[6] Zhou, S.L., Chen, X.S. & Luo, Y.G., et al. 2013. The optimization of Nutch crawling performance from the perspective of Hadoop, *Computer Application*, 33(10): 2792-2795.

[7] Li, X.Z., Cheng, G. & Zhao, Q.J., et al. 2014. Design and implementation of the distributed crawler system, *China Science and Technology Information*, (15): 116-117

[8] Zheng, B.W. 2011. *The Distributed Network Crawler Technology Based on Hadoop*, Harbin Institute of Technology.