Optimization and application of web crawler architecture

Kuncheng Li[a], Junqi Fei[b], Chunmei Fan[*cd]
[a]Data Research Center, China Academy of Information and Communications Technology, Beijing, China; [b]Data Research Center, China Academy of Information and Communications Technology, Beijing, China; [c]Network Education College, Beijing University of Posts and Telecommunications, Beijing, China; [d]Beijing Key Laboratory of Network System and Network Culture, Beijing, China

ABSTRACT

For monitoring of cutting-edge technologies by obtaining the massive data on the internet, the pyspider framework is used to regularly crawl the information of a large number of websites, and the crawled data is regularly imported into the business application system through python scripts. At the initial stage of the project, the distributed architecture is used as officially recommended by pyspider. Later, it is optimized as the clustered architecture in order to adapt to the actual network environment and the characteristics of crawler tasks. After testing, the work efficiency and stability of the improved architecture have been greatly improved and the expected results have been achieved.

Keywords: Web crawler, pyspider, the clustered architecture

1. INTRODUCTION

With the popularity of computers and mobile phones, the Internet has become an important way of information dissemination, but the massive amount of data has also brought great difficulties to the monitoring of Internet information. The traditional way of checking key websites one by one can no longer meet the demand. Instead, the content of these websites is crawled to the application system through web crawlers, and then centralized management and use.

1.1 Web crawler

Web crawler came into being with the emergence of the Internet, and was written as early as 1993[1]. A web crawler is a software which browses the Internet in a systematic, automated manner[2] and saves the information to the designated system through program script according to certain rules for centralized reading and analysis. Because its working mechanism is very similar to spider crawling, it is also called web spider[3].

The main work of web crawler includes three parts: page acquisition, page parsing and data storage[4]. Some crawler frameworks (such as pyspider) provide data processing interfaces (processors), which can add other customization functions while crawling data, such as downloading attachments and images, data cleaning, etc., improving the flexibility and applicability of web crawlers.

1.2 Pyspider

Pyspider is a distributed web crawler framework written in Python language, with powerful WebUI, supporting distributed architecture and script editor[5]. Compared with other crawler frameworks, pyspider not only supports mysql, mongodb, SQLite and other databases, but also has perfect tools in task monitoring, project management and result query. Pyspider is an open source crawler framework, which can use Python to build crawler scripts to crawl target data. It has harvested 15.5K stars and 3.7K forks on GitHub[6], and is one of the most popular crawler frameworks at present.

Figure 1[7] is the structure and task flow of pyspider. Pyspider consists of four components.

- WebUI: This is one of the features of pyspider. It can debug crawlers online, supervise tasks, manage projects, and view the completion of tasks.
- Scheduler: It is used to summarize the collection tasks, complete the task scheduling, and assign the grab tasks to the fetcher.

* chunmei.fan@bupt.edu.cn

Third International Conference on Computer Science and Communication Technology (ICCSCT 2022)
edited by Yingfa Lu, Changbo Cheng, Proc. of SPIE Vol. 12506, 125060N
 · 0277-786X · doi: 10.1117/12.2661783

- Fetcher: It is responsible for crawling the page content, and then sending the content to the processor.
- Processor: It is responsible for processing the crawled content, realizing data cleaning and warehousing, and handing over the new collection task to the scheduler.

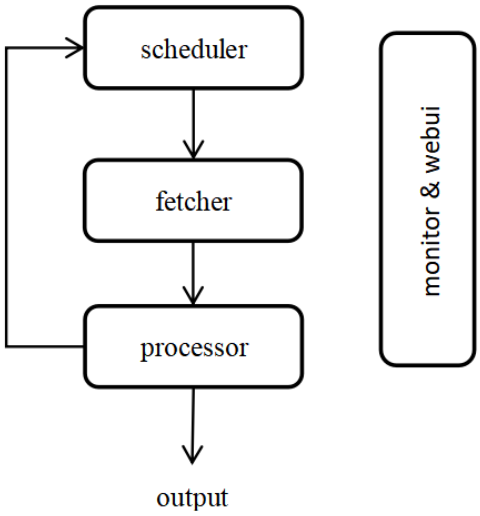

Figure 1. Pyspider architecture and task flow.

Each pyspider system can only have one scheduler, but multiple fetchers and processors can be deployed distributed, and they communicate through message_queue[8].

The task_queue of pyspider is executed in sequence. Only when a task is completed, the scheduler will assign new tasks in the queue to the fetcher. The maximum number of tasks in the task queue is limited by the queue maxsize parameter. When the length of the task queue reaches this parameter, the fetcher will crash[9].

1.3 Task background

There are more than 100 websites to be followed. The news, reports or logs (blogs) are obtained, and the attached attachments and pictures are downloaded from these websites at the first time. For this purpose, the application system is built with the web crawler based on pyspider. In order to ensure real-time performance as much as possible, it is very necessary to improve the working frequency of the web crawler, for example, trying to crawl each website at least once every hour. Therefore, it brings many performance and efficiency problems.

2. GENERAL WEB CRAWLER ARCHITECTURE

According to the architecture of pyspider, MySQL database is used as an example, and deploys web crawlers with a general distributed structure in the early stage of the system. The system structure is shown in Figure 2.

In the architecture shown in Figure 2, pyspider server 1 executes WebUI and scheduler tasks, maintains message queues, and runs the MySQL database required by the crawler system. Pyspider2 and pyspider3 servers communicate with pyspider1 server through the network, execute fetcher and processor tasks in parallel under the scheduling of the scheduler, download attachments and pictures by processor and save them on their respective servers. Finally, the crawled data is imported into the actual application system from pyspider1 server through ETL (Extract-Transform-Load) tools[10], and the downloaded attachments and pictures are synchronized to the application server through rsync service[11].

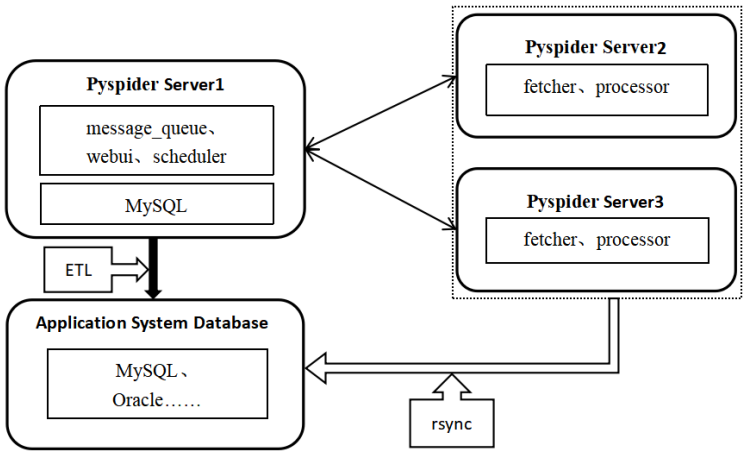

Figure 2. General pyspider distributed deployment.

During the operation of the system, the architecture gradually exposed the following problems:

1) The whole system is uniformly scheduled by the scheduler of pyspider server 1, and all servers share a task queue. In this way, when the crawling task of the details page corresponding to the website list is added to the queue, sometimes the task queue increases faster than the crawling speed of the server. When the queue maxsize of the queue is exceeded, the fetcher crashes.

2) Due to the distributed deployment, pypiper servers 2 and 3 need to communicate frequently with pypiper server 1. Once the network breaks down, delays or other failures, the servers cannot communicate normally, and the data acquisition task will stop and cannot work normally.

3) In the pyspider architecture, all crawler tasks are processed serially. For crawler tasks that need to download attachments, the processor will be under great pressure, especially when the attachments are too large (some of which are up to 200M in size) or the network condition is poor, the processor will process the download for a long time, and other tasks in the task queue will have to wait for the completion of the download task before they can be scheduled, resulting in a large number of monitored website information cannot be retrieved in time.

4) Pyspider's task management is carried out in project units. The completion of nearly 100 crawler tasks is saved in the same number of data Tables. It is difficult to count the overall data collection, and it is not easy to troubleshoot if there are problems.

5) The data collected by pyspider is saved in the project Table of Resultdb database in the form of JSON string, rather than structured data. Therefore, the collected data should be processed structurally for subsequent business.

3. OPTIMIZATION METHOD

In view of the above problems, the pyspider architecture has been upgraded and optimized. The new architecture is shown in Figure 3.

As shown in Figure 3, the three pyspider servers do not adopt a distributed architecture, but adopt a cluster approach[12]. The biggest difference between the two is that the three servers no longer communicate with each other, but work independently. Although the working pressure of each server increases in this way, the entire crawler queue will not be blocked due to a task or external factors of the network.

The crawler source management server in the architecture diagram is drawn with a dotted line because it only provides the management of crawler sources and does not generate a lot of load. Therefore, there is no need for a separate server, and it can share resources with one of the three crawler servers. Its main function is to uniformly record the crawler sources and the information allocated to the crawler server by each crawler source. It can realize the simplest load balancing by polling[13]. Because the three web crawler servers distribute all tasks equally, the number of queues of each server will not be too large, and the crawler system will not crash because of too many tasks in the queue.

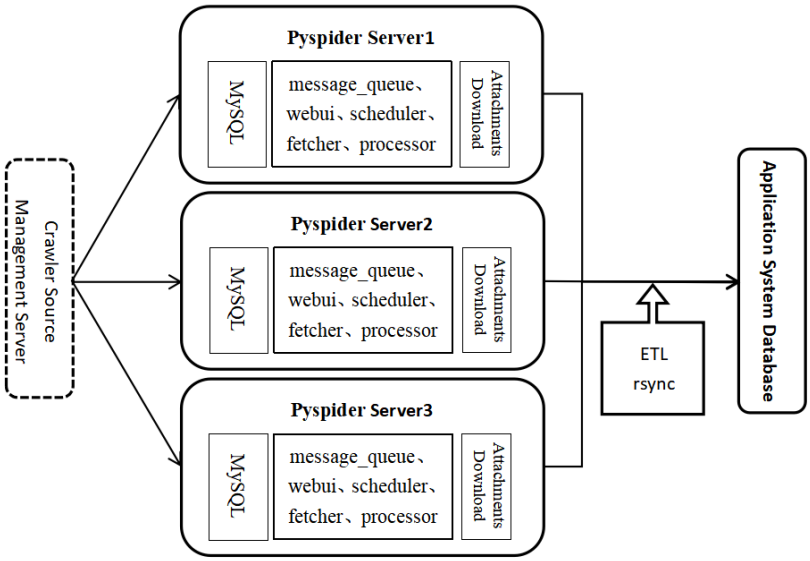

Figure 3. Pyspider cluster deployment.

For the MySQL database service included in each crawler server in the architecture, in addition to providing the Taskdb, Projectdb and Resultdb databases required for the management of the native pyspider, the other database name article is also created according to the uniformly planned crawl information, in which the data Table named article is used as the data crawl summary table (as shown in Figure 4) to save the crawled data and the corresponding crawler source information. Through this data table, you can use the query tool of structured database to query the crawling results of crawler server uniformly, instead of querying the crawling results of each item separately.

Field	Type	Null	Key	Default
id	int(11)	NO	PRI	NULL
guid	varchar(512)	YES		NULL
href	varchar(1024)	YES		NULL
title	varchar(1024)	YES		NULL
author	varchar(512)	YES		NULL
pub_date	datetime	YES		NULL
info	longtext	YES		NULL
source	varchar(255)	YES		NULL
lang	varchar(255)	YES		NULL
path	varchar(1024)	YES		NULL
category	varchar(1024)	YES		NULL
industry	varchar(255)	YES		NULL
crawl_date	datetime	YES		NULL
upload_state	int(5)	YES		0
upload_date	datetime	YES		NULL
upload_log	text	YES		NULL
server_ip	varchar(100)	YES		10.5.65.88

Figure 4. Summary of data crawling information.

The attachment download data table is also set in the article database, as shown in Figure 5. By saving the attachments and picture URL that need to be downloaded in each crawling task in this Table, allowing a separate program to download the

attachments regularly outside the data crawling task, it can avoid the blocking of the entire data crawling task due to large attachments or slow network speed.

After the data crawling task is completed, the content in the article table of each crawler server is regularly extracted into the database of the application system through ETL tool, and then the downloaded attachments are synchronized to the file system of the application system through rsync service. In this way, the crawled data can be supplied to the system for convenient use.

Field	Type	Null	Key	Default
id	int(11)	YES		NULL
article_id	int(11)	YES		NULL
local_link	varchar(255)	YES		NULL
original_link	varchar(255)	YES		NULL
file_name	varchar(255)	YES		NULL
file_type	varchar(255)	YES		NULL
create_date	datetime	YES		NULL
download_status	varchar(255)	YES		NULL

Figure 5. Attachment download Table.

4. CONCLUSION

The optimized web crawler framework runs stably, which solves the problem of data collection task stopping due to the network communication between servers and too large attachments. In the processor of each crawler task, the storage of structured data and the consolidation of various project data are added to facilitate the statistics of data collection and problem troubleshooting, which has achieved good results in the application. In the later stage, load balancing among multiple data crawling servers and the efficiency of attachment downloading and data synchronization need to be further optimized.

REFERENCES

[1] Mirtaheri, S. M., Dinçktürk, M. E., Hooshmand, S., et al., "A brief history of web crawlers," arXiv preprint arXiv:1405.0749, (2014).
[2] Kausar, M. A., Dhaka, V. S. and Singh, S. K., "Web crawler: A review," International Journal of Computer Applications 63(2), (2013).
[3] Zhai, P., "Comparative analysis of Python web crawler crawling strategies," Computer Knowledge and Technology (01), 29-30+34 (2020).
[4] Lu, H. and Feng, X., "Application of web crawler in batch acquisition of teaching resources," Fujian Computer (06), 59-61 (2022).
[5] Cao, J., Lin, J., Wu, S., et al., "Lucene and deep learning based commodity information analysis system," 2016 IEEE Inter. Conf. on Consumer Electronics-China (ICCE-China), 1-4 (2016).
[6] binux/pyspider[R/OL], (2022). https://github.com/binux/pyspider
[7] Pyspider. Architecture[R/OL], (2022). http://docs.pyspider.org/en/latest/Architecture/
[8] Cao, Y., [Distributed Incremental Acquisition Method for Dynamic Network Data], (Beijing: Beijing University of Posts and telecommunications) Master's Thesis, (2017).
[9] Pyspider. Architecture[R/OL], (2022). http://docs.pyspider.org/en/latest/Architecture/
[10] Patel, M. and Patel, D. B., "Progressive growth of ETL tools: A literature review of past to equip future," Rising Threats in Expert Applications and Solutions, 389-398 (2021).
[11] Fang, C., Cottrell, R. and Kissel, E., "When to use rsync", (2021). doi:10.2172/1772472
[12] Takada, M., [Distributed Systems for Fun and Profit], (2013).
[13] Qu, Q. and Wang, J., "Distributed digital cluster dynamic load balancing algorithm based on load feedback," Computer Application Research 39(02), 526-530+542 (2022).