Spark Memory Management

Wei Zhang(✉) and Jingmei Li

College of Computer Science and Technology, Harbin Engineering University, Harbin 150001, China
zhangwei72@hrbeu.edu.cn

Abstract. In order to obtain detailed information about Spark framework and realize fine grained monitoring of cluster operation information, a performance analysis system is designed. Therefore, the problems of Spark1.6 memory management scheme are researched in depth and improved. The experimental results show that the original memory management scheme is inconsistent with the requirements of Spark's official website. However, the improved memory management scheme not only meets the requirements of Spark's official website, but also makes the application run successfully under the condition of small memory capacity.

Keywords: Spark framework · Memory management · Memory overflow

1 Introduction

As a computing engine that excels in memory computing, memory management in Spark is a very important module [1, 2]. Because of its outstanding memory calculation, putting the application running data in memory as much as possible, the most of the errors in running applications are caused by spark memory overflow. Due to its outstanding memory calculation, the application's running data is stored in memory as much as possible, resulting in most of the errors in running applications from the spark memory overflow.

Through the research on the working mechanism of the distributed platform Spark, a performance analysis system based on Spark log is designed to realize fine grained monitoring of the cluster operation information. On the basis of fine grained monitoring, the memory management of Spark framework is studied deeply and the existing problems are optimized.

2 Performance Analysis System Design

In order to determine the memory efficiency of the system, it is necessary to accurately judge the application details of each stage of execution. So a performance analysis system is designed.

Large data performance analysis system is divided into three main layers and the overall framework is shown in Fig. 1. The first layer is Spark source plug in, which can generate Spark application running log by the tool of the slf4j log and calling

G. Sun and S. Liu (Eds.): ADHIP 2017, LNICST 219, pp. 65–69, 2018.
https://doi.org/10.1007/978-3-319-73317-3_9

high-precision timing tools of the operating system, methods of Java, etc. The second layer is the data collation layer. The running log generated by Spark is a readable natural language of human oriented. It needs regular expression to extract the data it needs. The original unstructured data is organized into structured data [3], which is convenient for reading on the third level. The third layer is data visualization, which can help users analyze the implementation of the application by drawing the appropriate chart with visualization tools.

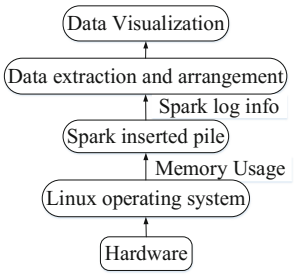

Fig. 1. Spark performance analysis system framework

3 Spark Memory Management

The memory management scheme in this article refers to the new memory management scheme in the Spark1.6 framework [4–6]. The memory management scheme is implemented using dynamic pre-emption, which means that Execution can borrow free Storage memory and vice versa. The borrowed memory is recycled when the amount of memory increases. In memory management, memory is divided into three separate blocks as shown in Fig. 2.

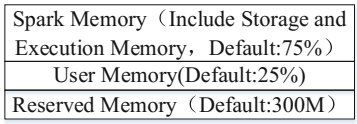

Fig. 2. Unified memory manager in Spark 1.6

By analyzing the memory management scheme, there is a problem that how much memory Execution can borrow from Storage. Storage Fraction is configured on the Spark official website to indicate the memory ratio that Storage occupies at least. To study the Spark memory management scheme, it finds that when storage remaining memory is larger than the difference between Storage's memory and the initial allocated memory, and when Execution needs more memory than it can borrow, then the final borrow memory is equal to Storage of the remaining memory. To update Storage and Execution memory, the now available memory of Storage is equal to the difference

between previously owned memory and Storage's remaining memory. Depending on the condition, the now available memory of Storage is smaller than the initial configuration memory, which is inconsistent with the configuration described on the Spark official website. This problem leads to the static configuration algorithm degenerate into an approximate first come first service algorithm. The improved approach is to set the borrowing memory size to the difference between the now available memory of Storage and the initial configuration memory.

4 Experimental Verification

4.1 Memory Management Deficiencies and Improvements

In order to find problems of Spark memory management, it needs to obtain Execution and Storage memory change information. Therefore, the associated log information is added in the Spark framework and memory change information is recorded in the log file. This experiment uses Spark1.6 memory management scheme. The configuration parameters of Spark Memory Fraction and Spark Memory Storage Fraction are 0.75 and 0.5 respectively. Based on the configuration parameters and the 8G size of the committed memory, the Storage memory is calculated to be no less than 2887.5 M.

The Storage Memory change information is obtained by submitting the same PageRank application in the memory management scheme before and after the improvement, as shown in Fig. 3. From the diagram, the memory of the before improved memory management scheme varies with time and the minimum of Storage memory is only 0.297044 M, which obviously does not meet its minimum requirement of 2887.5 M. So there are problems about Execution and Storage memory borrowing from each other. However, memory of the improved memory management program changes in the range of 500 M at different times and memory size is more than 2887.5 M, in line with requirements.

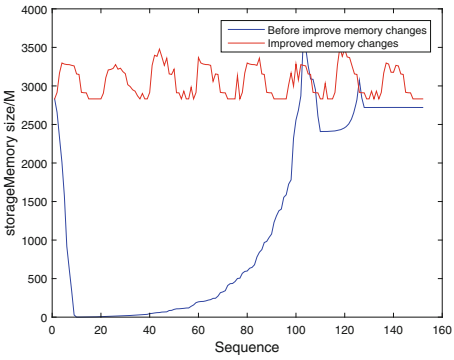

Fig. 3. Memory data changes

To verify the performance of the memory management scheme, the same application is presented and the change is only the size of the executor memory. The running time of PageRank applications before and after the improvement of the Spark memory management scheme is shown in Fig. 4. The performance of the before improved memory management scheme is basically the same as the improved when the memory is relatively large. However, the application in the before improved memory management scheme cannot run properly when the memory size is 7G and 6G, while the improved memory management solution application can still run successfully. Therefore, the overall performance of the improved memory management scheme is superior to that of the before improved memory management scheme.

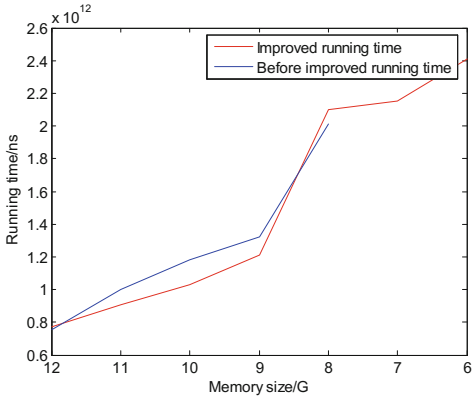

Fig. 4. Application run time

5 Conclusions

The source code and implementation principle of Spark framework are analyzed in this paper. The insufficiency of memory management scheme is proved through experiments and the shortcomings of memory management are improved. The improved memory management scheme not only meets the requirements of Spark's official website for Storage memory configuration, but also has a better performance than the before improved memory management scheme.

References

1. Bosagh, Z.R., Meng, X., Ulanov, A.: Matrix computations and optimization in apache Spark. pp. 31–38 (2015)
2. Sankar, K., Karau, H.: Fast Data Processing with Spark. Packt Publishing, Birmingham (2015)
3. Li, Y., Zhang, X., Tan, R.J., et al.: Establishment of traceability and supervision system for import and export products and its application on import food supervision. J. Food Saf. Qual. **6**, 4312–4317 (2015)

4. Tang, M., Yu, Y., Malluhi, Q.M., et al.: LocationSpark: a distributed in-memory data management system for big spatial data. Proc. VLDB Endow. **9**, 1565–1568 (2016)
5. Park, K., Baek, C., Peng, L.: A development of streaming big data analysis system using in-memory cluster computing framework: Spark. In: Park, J., Jin, H., Jeong, Y.S., Khan, M. (eds.) Advanced Multimedia and Ubiquitous Engineering. LNEE, vol. 393, pp. 157–163. Springer, Singapore (2016). https://doi.org/10.1007/978-981-10-1536-6_21
6. Duan, M., Li, K., Tang, Z., et al.: Selection and replacement algorithms for memory performance improvement in Spark. Concurr. Comput. Prac. Exp. **28**, 2473–2486 (2016)