Applying Hugepages to In-memory Distributed Processing Frameworks

Yuya Miyazato Hiroshi Yamada

Tokyo University of Agriculture and Technology

Abstract

in-memory distributed processing frameworks that utilize a large amount of memory, the address translation becomes a major performance bottleneck. The hugepage feature is promising to improve the performance of in-memory distributed processing frameworks. This thesis tries to answer the following question: How effectice is The hugepages for in-memory distributed frameworks? We used Apache Spark 3.2.0 as the representative in-memory distributed frameworks, and we implemented a per-object hugepage allocation mechanism to confirm the effect of hugepages on spark memory structure. In our experiments, we used the three hugepage allocation policy: no hugepage, allocation entire, and partial allocation(only the StorageMemory to Spark) WE conducted experiments with 11 benchmarks. The experimental results showed that entire allocation reduces execution time by 3%-13% for 8 workload, but causes memory bloat of 1GB-17GB for 11 workload. The partial allocation reduces execution time as much as or more than entire allocation in 4 benchmarks, and reduced memory bloat in 10 benchmarks.

1. Introduction

In-memory processing is a widely-accepted approach to efficiently perform large-scale calculation such as machine learning and graph analytics. In-memory distributed processing frameworks are a type of distributed processing frameworks [3]. Examples include Apache Spark [12, 14], Apache Flink [5], and Prestro [11]. Apache Spark is a distributed processing system that is designed to perform as much of the processing on each machine as possible in memory, making it faster than traditional distributed processing systems such as Apache Hadoop. Distributed processing frameworks are distribute processing by connetcting networks of multiple computers that , and have the advantages of processing very large data that cannot be handled by a single server, reducing the cost of preparing servers, and making it easy to expand resources, and can be used to analyze huge graphs [4], streaming processing [2, 13], machine learning processing [8], and many other applications. By utilizing large amounts of memory and completing processing in memory, in-memory distributed processing frameworks can reduce disk I/O and network I/O bottlenecks in addition to the benefits of traditional distributed processing. The reason why in-memory distributed processing frameworks require large amounts of memory is due to the difference in processing methods between in-memory distributed processing frameworks and traditional distributed processing frameworks. The distributed processing framework writes the intermediate results of processing to disk each time data is processed, resulting in I/O processing such as disk access and networking, which is time consuming [15]. On the other hand, the in-memory distributed processing framework speeds up processing by holding data in memory instead of writing it to disk each time after processing. A huge amount of memory is needed to hold this data. As an example of a machine with a large amount of memory, the High Memory instance provided by Amazon EC2 [1] has 24TB of memory. In-memory distributed processing frameworks use this huge memory size.

The hugepage feature is useful to mitigate the address translation overhead of such in-memory processing. hugepages can expand the range of addresses that can be converted at one time by increasing the page size, the unit of address conversion, to a larger than usual size. This can reduce the level of page tables, which would normally be multi-level, reduce page conversion time, and reduce the page table size. In addition to Converting many sizes at once also increases the size that can be stored in the TLB and reduces the TLB miss rate. For example, TLBs available for 4KB pages on an Intel Xeon E-2124 have a total of 1600 entries in L1 and L2, covering 6.4MB of pages. In contrast, 2MB pages have 1568 available TLB entries, which can cover approximately 3GB of pages. When dealing with several GB of data, hugepages significantly increases the size that can be stored in the TLB and improves the TLB hit ratio. In-memory distributed processing frameworks use a large amount of memory, so the time required for address translation is expected to be critical.

Although existing researches have studied ways to leverage the hugepage feature so far, it is still unclear how hugepages are effective to in-memory application written in Java. Using hugepages in Apache Spark uses THP, the Linux huge page allocation mechanism. Allocation is performed on a JVM basis, and does not take into account Apach Spark's unique memory structure. The validity of using THP in Spark is not clear, since THP has various problems such as memory bloat and CPU hogging for memory compaction, and it is sometimes recommended that the feature be disabled. There are some studies on in-memory distributed processing frameworks, such as HotTub [7]and Yak [9], but they mainly focus on the improvement of JVMs suitable for distributed processing.

Although Ingens [6] and Illminator [10] have conducted research on hugepages, they do not take into account the unique behavior of distributed processing frameworks, and their research is focused on optimal huge page allocation for the entire OS. Ingens uses Spark, but it is one of many benchmarks used to ascertain the effectiveness of Ingens, and the relationship between Spark and hugepages has not been studied in detail. The impact of hugepages on each of the applications with different characteristics, such as machine learning and graph processing running on Spark, is also unknown.

This paper quantitatively shows the hugepage effectiveness for Apache Spark applications. Specifically, we investigate benchmarks running on Apache Spark by measuring how execution time, memory consumption, and TLB hit ratio change with and without hugepages. We implemented a mechanism for allocating hugepages per object in Apache Spark to investigate the effect of hugepages on Spark memory structure. Through these, we consider hugepages allocation and memory managemtn, suitable for in-memory distributed processing frameworks.

The contributions of this paper is as follows:

- Experiments were conducted on 10 real-world applications and microbenchmarks on Apache Spark to measure execution time, memory usage, etc., with three hugepage policies: allocation entire, partial allocation, and no hugepage.
- For investigation, we implemented a mechanism to allocate hugepages only in a portion of Apache Spark's memory. By modifying the JVM to allocate memory with hugepages before creating objects, it is possible to allocate hugepages only in StorageMemory in Spark.
- Experimental results showed that in 8 out of 11 benchmarks, the execution time could be reduced by up to 13% by allocating hugepages. However, the entire allocation increased memory usage in all benchmarks, with large memory bloat of up to 17GB. The partial allocation reduced execution time by as much as the entire allocation for the four benchmarks and reduced memory bloat more than the entire allocation, confirming the effectiveness of partial hugepage allocation.

2. Bakcgrounds

2.1. Apache Spark

2.2. Hugepages

2.3. Previous Approaches

3. Methodology

3.1. Hugepage Allocation

3.2. Target Applications

3.3. Allocation of Hugepages to Storage Memory

4. Experiments

5. Discussion

6. Conclusion

References

[1] Amazon ec2 high memory instance. (visited on 2023-1-30). [Online]. Available: https://aws.amazon.com/jp/ec2/instance-types/high-memory/

[2] M. Armbrust, T. Das, J. Torres, B. Yavuz, S. Zhu, R. Xin, A. Ghodsi, I. Stoica, and M. Zaharia, "Structured streaming: A declarative api for real-time applications in apache spark," in Proceedings of the 2018 International Conference on Management of Data, 2018, pp. 601–613.

[3] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," Communications of the ACM, vol. 51, no. 1, pp. 107–113, 2008.

[4] J. E. Gonzalez, R. S. Xin, A. Dave, D. Crankshaw, M. J. Franklin, and I. Stoica, "Graphx: Graph processing in a distributed dataflow framework," in 11th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 14), 2014, pp. 599–613.

[5] A. Katsifodimos and S. Schelter, "Apache flink: Stream analytics at scale," in 2016 IEEE international conference on cloud engineering workshop (IC2EW). IEEE, 2016, pp. 193–193.

[6] Y. Kwon, H. Yu, S. Peter, C. J. Rossbach, and E. Witchel, "Coordinated and efficient huge page management with ingens," in 12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16), 2016, pp. 705–721.

[7] D. Lion, A. Chiu, H. Sun, X. Zhuang, N. Grcevski, and D. Yuan, "Dont get caught in the cold, warm-up your {JVM}: Understand and eliminate {JVM} warm-up overhead in data-parallel systems," in 12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16), 2016, pp. 383–400.

[8] X. Meng, J. Bradley, B. Yavuz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. Tsai, M. Amde, S. Owen et al., "Mllib: Machine learning in apache spark," The Journal of Machine Learning Research, vol. 17, no. 1, pp. 1235–1241, 2016.

[9] K. Nguyen, L. Fang, G. Xu, B. Demsky, S. Lu, S. Alamian, and O. Mutlu, "Yak: A high-performance big-data-friendly garbage collector," in 12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16), 2016, pp. 349–365.

[10] A. Panwar, A. Prasad, and K. Gopinath, "Making huge pages actually useful," in Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems, 2018, pp. 679–692.

[11] R. Sethi, M. Traverso, D. Sundstrom, D. Phillips, W. Xie, Y. Sun, N. Yegitbasi, H. Jin, E. Hwang, N. Shingte et al., "Presto: Sql on everything," in 2019 IEEE 35th International Conference on Data Engineering (ICDE). IEEE, 2019, pp. 1802–1813.

[12] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, I. Stoica et al., "Spark: Cluster computing with working sets." HotCloud, vol. 10, no. 10-10, p. 95, 2010.

[13] M. Zaharia, T. Das, H. Li, T. Hunter, S. Shenker, and I. Stoica, "Discretized streams: Fault-tolerant streaming computation at scale," in Proceedings of the twenty-fourth ACM symposium on operating systems principles, 2013, pp. 423–438.

[14] M. Zaharia, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M. J. Franklin et al., "Apache spark: a unified engine for big data processing," Communications of the ACM, vol. 59, no. 11, pp. 56–65, 2016.

[15] P. Zhou, Z. Ruan, Z. Fang, M. Shand, D. Roazen, and J. Cong, "Doppio: I/o-aware performance analysis, modeling and optimization for in-memory computing framework," in 2018 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS). IEEE, 2018, pp. 22–32.