Proceedings of

, , ,

IMPROVING SERVER THROUGHPUT FOR BIG DATA ANALYTICS USING SMART HEAP OFFLOADING

Emmanouil Anagnostakis
Graduate Research Assistant
Computer Architecture and VLSI Systems Laboratory, ICS-FORTH
Heraklion, Greece
manosanag@ics.forth.gr

ABSTRACT

With the increasing size and complexity of big data, there is a growing need for efficient and scalable data processing frameworks such as Apache Spark. However, the high computational and memory requirements of big data analytics workloads pose significant challenges for cluster computing systems. In this paper, we propose a heap offloading technique to improve job throughput and increase DRAM utilization for Big data analytics workloads on Spark clusters.

Our offloading technique leverages the capabilities of servers that run instances of the Java Virtual Machine, the core of Big Data Analytics like Apache Spark, to move large parts of the Java Heap from the main memory to a memory mapped secondary heap over a fast storage device called TeraHeap. TeraHeap 1) eliminates Serialiazation/Deserialization overheads posed by these kind of frameworks when moving data offheap to/from fast storages devices 2) eliminates GC over the secondary heap, therefore significantly minimizing overall GC overhead. By offloading the managed Java Heap and relaxing computation-intensive tasks, we aim to reduce the workload on Spark workers, thereby improving their performance and job throughput. We also explore the trade-offs between the cost of offloading and the performance gains achieved.

We evaluate our offloading technique using various big data analytics workloads on a Spark cluster. Our results show that our offloading technique can significantly improve job throughput for big data analytics workloads, while maintaining a reasonable cost of offloading.

Overall, our offloading technique offers a promising approach to improving job throughput for big data analytics workloads on Spark clusters, particularly for computation-intensive tasks. The proposed technique can be easily integrated into existing Spark clusters and offers a scalable solution for processing increasingly large and complex big data workloads.

INTRODUCTION

With the exponential growth of data in various fields such as finance, healthcare, social media, and e-commerce, there is a significant need for scalable and efficient big data processing frameworks. Apache Spark [?] is one such framework that has gained popularity due to its ability to handle large-scale data processing and analytics. Spark provides a distributed computing platform that can process data in parallel across multiple nodes in a cluster. However, with the increasing size and complexity of big data workloads, Spark clusters are facing significant challenges in meeting the performance and throughput requirements.

One of the main challenges in Spark clusters is the high computational and memory requirements of big data analytics workloads. These requirements can result in excessive CPU and memory usage on Spark workers, leading to performance bottlenecks and slow job completion times. To address these challenges, researchers have proposed various techniques to optimize the performance of Spark clusters, including data partitioning, caching, and resource allocation.

In this paper, we focus on the memory limit problem of servers becoming an obstacle for further throughput increase and we propose a new technique for improving the performance and job throughput of Spark clusters by moving parts of the main managed Java Heap to fast storage devices such as NVMe, thereby saving memory for other more useful tasks. Our approach leverages the capabilities of the underlying machine to create less memory-hungry computation tasks, thereby reducing the workload on the Spark workers to improve job throughput, while maintaining effective single instance performance as the instances of Spark executors increase to provide max throughput until the server runs out of DRAM capacity.

Specifically, in order to achieve higher throughput and better performance for Spark, we use TeraHeap, a secondary managed memory-mapped heap over an NVMe storage device, which is used to hold the Resilient Distributed Datasets (Spark RDDs) instead of the main managed Java Heap and remove any Serialization/Deserialization and Garbage Collection (GC) cost over them.

The main contribution of this paper is a comprehensive evaluation of the performance and cost trade-offs of creating lightweight computation tasks in Spark clusters. We demonstrate the effectiveness of our approach using various big data analytics workloads on a Spark cluster. We also compare our approach with the native Spark distribution and show that our approach can be used instead of this distribution to improve performance and server throughput.

The rest of the paper is organized as follows. In section 2, we offer background knowledge about Apache Spark and TeraHeap. In section 3 we discuss related work on Spark optimization techniques and offloading techniques. In section 4, we describe our experimental methodology in order for someone to achieve the desired performance using TeraHeap. In section 5, we present our experimental results and evaluate the performance and cost trade-offs of our approach. In section 6, we discuss future research directions. Finally we conclude the paper in section 7 with an outline of our work.

BACKGROUND

TeraHeap

RELATED WORK

EXPERIMENTAL METHODOLOGY

EVALUATION

FUTURE WORK

While our proposed offloading technique shows promising results in improving job throughput for big data analytics workloads on Spark clusters, there are several avenues for future work to further improve the performance and scalability of Spark clusters.

Firstly, one potential direction for future work is to investigate the use of other types of storage mediums such as the hybrid NVM. This medium could improve the performance of Big data analytics further by combining the advantages of memory and storage.

Secondly, another area for future work is to develop techniques for dynamically adjusting the heap offloading decisions based on workload characteristics and resource availability. For example, the offloading decision can be based on the size of the input data or the availability of DRAM capacity in the cluster. Such techniques can help maximize the performance gains achieved by offloading while minimizing the cost of offloading.

Thirdly, an interesting direction for future work is to explore the use of heap offloading in environments where Spark clusters are deployed across multiple machines using RDMA to achieve communication between the different machines. This can help utilize the DRAM, CPU and storage availability in more than one machine and provide a more cost-effective solution for big data processing.

Finally, another potential area for future work is to investigate the use of heap offloading for other big data processing frameworks beyond Spark. Many other big data processing frameworks such as Apache Giraph can potentially benefit from offloading techniques to improve their performance and scalability.

Overall, there are many exciting avenues for future work in improving the performance and scalability of big data processing frameworks such as Spark. Our proposed offloading technique provides a solid foundation for future work and offers a promising approach for addressing the challenges of big data processing.

CONCLUSION

In this paper, we proposed a new technique for improving the performance and job throughput of Spark clusters by moving parts of the managed Java Heap to a secondary memory-mapped heap over a fast storage devices such as NVMe. Our approach leverages the capabilities of the underlying running machine to relax computation-intensive tasks running on the Spark workers, thereby reducing the workload on the workers and improving their performance and job throughput.

Our experimental results demonstrate the effectiveness of our approach using various big data analytics workloads on a Spark cluster. We also compared our approach with the native Spark distribution and showed that our approach can be used instead of this distribution to further improve performance.

Our work contributes to the growing body of research on improving the performance and scalability of Spark clusters for big data analytics workloads. Our approach offers a scalable solution for processing increasingly large and complex big data workloads and can be easily integrated into existing Spark clusters.

Overall, our offloading technique offers a promising approach to improving job throughput for big data analytics workloads on Spark clusters, particularly for computation-intensive tasks. With the increasing demand for efficient and scalable big data processing frameworks, our approach provides a valuable contribution to the field of big data analytics.

ACKNOWLEDGMENT

REFERENCES

[1] Zaharia, M., Xin, R. S., Wendell, P., Das, T., Armbrust, M., Dave, A., Meng, X., Rosen, J., Venkataraman, S., Franklin, M. J., Ghodsi, A., Gonzalez, J., Shenker, S., and Stoica, I., 2016. "Apache spark: A unified engine for big data processing". In Communications of the ACM, Association for Computing Machinery.

[2] Kolokasis, I. G., Papagiannis, A., Pratikakis, P., Bilas, A., Zakkak, F., Evdorou, G., Akram, S., and Kozanitis, C., 2023. "Teraheap: Reducing memory pressure in managed big data frameworks". In ASPLOS '23, March 25–29, 2023, Vancouver, BC, Canada, Association for Computing Machinery.

[3] Marco, V. S., Taylor, B., Porter, B., and Wang, Z., 2017. "Improving spark application throughput via memory aware task co-location: A mixture of experts approach". In Proceedings of Middleware '17, Las Vegas, NV, USA, Association for Computing Machinery.

[4] Kirisame, M., Shenoy, P., and Panchekha, P., 2022. "Optimal heap limits for reducing browser memory use". In OOPSLA, Association for Computing Machinery.

[5] Amaro, E., Branner-Augmon, C., Luo, Z., Ousterhout, A., Aguilera, M. K., Panda, A., Ratnasamy, S., and Shenker, S., 2020. "Can far memory improve job throughput?". In EuroSys '20, April 27–30, 2020, Heraklion, Greece, Association for Computing Machinery.

[6] Weiner, J., Agarwal, N., Schatzberg, D., Yang, L., Wang, H., Sanouillet, B., Sharma, B., Heo, T., Jain, M., Tang, C., and Skarlatos, D., 2022. "Tmo: Transparent memory offloading in datacenters". In ASPLOS '22, February 28 – March 4, 2022, Lausanne, Switzerland, Association for Computing Machinery.

[7] Sharma, P., Kulkarni, P., and Shenoy, P. "Per-vm page cache partitioning for cloud computing platforms". Association for Computing Machinery.

[8] Apache, 2023. "Rdd programming guide (spark 3.4.0 - 2023 update) - https://spark.apache.org/docs/latest/rdd-programming-guide.html".

[9] Apache, 2023. "Building spark (spark 3.4.0 - 2023 update) - https://spark.apache.org/docs/latest/building-spark.html".

[10] Apache, 2023. "Tuning spark (spark 3.4.0 - 2023 update) - https://spark.apache.org/docs/latest/tuning.html".

[11] Apache, 2023. "Spark configuration (spark 3.4.0 - 2023 update) - https://spark.apache.org/docs/latest/configuration.html".

[12] Apache, 2023. "Monitoring and instrumentation (spark 3.4.0 - 2023 update) - https://spark.apache.org/docs/latest/monitoring.html".

[13] chriswhocodes, 2023. "Vm options explorer - openjdk8 hotspot - https://chriswhocodes.com".