Proceedings of

, , ,

OVERCOMING THE MEMORY BOUND OF BIG DATA ANALYTICS TO IMPROVE SERVER THROUGHPUT USING FAST STORAGE

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

ABSTRACT

Managing big data analytics i.e. Apache Spark poses challenges due to limited memory resources in data centers. The memory pressure that arises during data processing can result in low server throughput, causing delays and inefficiencies. Memory is wasted in long GC (Garbage Collection) cycles leaving no room for useful work. In this paper, we propose a novel approach to improve server throughput for managed big data analytics using smart heap offloading to fast storage devices and reducing memory pressure. Our approach involves offloading data from heap memory to fast storage devices in a smart and efficient manner, thereby freeing up heap memory and reducing memory pressure without suffering from storage latencies. We present a detailed methodology for running Apache Spark using our proposed mechanism of smart heap offloading, which significantly improves server throughput for managed big data analytics. We implement our proposed approach in Oracle's OpenJDK8 and evaluate its performance using various workloads of the Spark Bench suite on a real-world cluster. Our experimental results show that our approach significantly improves server throughput while reducing memory usage against native Spark, making it a promising solution for managed big data analytics in data centers. We also include results to show that our implementation can save money for someone if deployed in a world cluster like Amazon's EC2 that is accessible by everyone.

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 a fast storage device 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 under the colocation of multiple instances required to achive max throughput.

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.

TeraHeap 1) eliminates Serialiazation/Deserialization overheads posed by these kind of frameworks when moving data offheap to/from fast storage 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.

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 real world 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 discuss related work on Spark optimization techniques and offloading techniques. In section 3, we describe our experimental methodology in order for someone to achieve the desired performance using TeraHeap. In section 4, we present our experimental results and evaluate the performance and cost trade-offs of our approach. In section 5, we discuss future research directions. Finally we conclude the paper in section 6 with an outline of our work.

RELATED WORK

Several studies have been conducted to improve the performance of big data processing systems. One approach is to utilize memory-aware task co-location to improve Spark application throughput, which has been investigated by Marco et al. in [3]. Meanwhile, in [4], Kirisame et al. proposed optimal heap limits to reduce browser memory use. Another research direction is to leverage far memory to improve job throughput, as studied by Amaro et al. in [5]. To facilitate memory offloading in datacenters, Weiner et al. presented TMO, a transparent memory offloading system in [6]. In cloud computing platforms, Sharma et al. proposed per-VM page cache partitioning to improve performance in [7]. Chen and Wang introduced Spark on Entropy, a reliable and efficient scheduler for low-latency parallel jobs in heterogeneous clouds, in [8]. Thamsen et al. developed Mary, Hugo, and Hugo*, three learning-based schedulers for distributed data-parallel processing jobs on shared clusters in [9]. Additionally, Bhimani et al. proposed a lightweight virtualization framework for accelerating big data applications on enterprise cloud in [10], while Zhang et al. focused on understanding and improving disk-based intermediate data caching in Spark in [11]. Finally, Intasorn et al. investigated using compression tables to improve HiveQL performance with Spark in a case study on NVMe storage devices in [12].

These studies demonstrate a variety of approaches for optimizing big data processing systems, ranging from memory-aware task co-location and memory offloading to scheduler design and virtualization frameworks. The findings from these studies can provide insights and guidance for future research in the field of big data processing.

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 free computation-intensive tasks running on the Spark workers from memory pressure, 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 compare 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 and memory management.

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] Chen, H., and Wang, F. Z., 2015. "Spark on entropy: A reliable and efficient scheduler for low-latency parallel jobs in heterogeneous cloud". In LCN 2015, Clearwater Beach, Florida, USA, Association for Computing Machinery.

[9] Thamsen, L., Beilharz, J., Tran, V. T., Nedelkoski, S., and Kao, O., 2019. "Mary, hugo, and hugo*: Learning to schedule distributed data-parallel processing jobs on shared clusters". In Euro-Par 2019, Association for Computing Machinery.

[10] Bhimani, J., Yang, Z., Leeser, M., and Mi, N., 2017. "Accelerating big data applications using lightweight virtualization framework on enterprise cloud". In 2017 IEEE High Performance Extreme Computing Conference (HPEC), IEEE.

[11] Zhang, K., Tanimura, Y., Nakada, H., and Ogawa, H., 2017. "Understanding and improving disk-based intermediate data caching in spark". In 2017 IEEE International Conference on Big Data (Big Data), IEEE.

[12] Intasorn, Y., Rattanaopas, K., and Chuchuen, Y., 2022. "Using compression tables to improve hiveql performance with spark a case study on nvme storage devices". In 2022 26th International Computer Science and Engineering Conference (ICSEC), IEEE.

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

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

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

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

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

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