Efficient Resource Cluster Management for Distributed In-Memory Processing

Nguyen Duc Rohr
Technische Universität Berlin
Berlin, Germany
n.rohr@tu-berlin.de

***Abstract*—Big Data's exponential expansion has made the creation of effective distributed data processing systems necessary. This study looks at the difficulties and obstructions in resource management in frameworks such as Apache Spark and Apache Flink. Through the examination of an open trace dataset of Hadoop MapReduce and Spark executions, we demonstrate the advantages of in-memory processing over disk-based techniques in terms of performance.**

In this paper, we analyze how optimizing memory allocation and balancing resources can enhance productivity and reduce costs. The results show that achieving high performance and efficient resource use in large-scale data processing requires dynamic management strategies that can adapt to changing workloads.

***Index Terms*—Distributed Data Processing, Apache Spark, Apache Flink, Resource Management, In-Memory Processing**

I. Introduction

Building a data, growth of data, which is commonly known as Big Data, is a core problem in computer science, in particular in data processing. The quantity, frequency, and types of data in the present-day information environment are beyond the reach of classical data processing mechanisms. Due to this expansion, there is a need for durable and highly efficient software that would be able to manage and process big databases, which is necessary in order to extract useful information for decision making in sectors like finance, healthcare, and e-commerce.

Various data handling frameworks embraced by more than one entity, like Apache Hadoop MapReduce, Apache Spark, and Apache Flink, are dominating the software world. The very first turnkey solution, namely Apache Hadoop MapReduce, carries out data in a disk-based fashion. Though this method makes the data more reliable, it takes a long time due to lots I/O operations. The increase of response time is behind why the cutting-edge frameworks resort to in-memory processing methods. [6]. In-memory processing frameworks such as Apache Spark and Apache Flink are of utmost importance for the achievement of the needed performance in processing data that comes in very large scale. Instead of faster memory then, they take the memory speed granted to them to complete data operations which are more than just to retarget the data on the disk in a quicker mode. But, as is the case with every advance, these schemes also have some downsides. A well-organized resource aptitude especially the allocation and utilization of memory is an indispensable tool in speeding up the process and preventing system bottlenecks. The main issue is to use the limited memory resources effectively by carrying out better performance on different workloads [7], [8].

However, it is often unclear how to manage resource clusters efficiently due to the complexity and variability of workloads [9], [10]. Further research by Delimitrou and Kozyrakis (2014) on Quasar, a resource-efficient and QoS-aware cluster management system, shows that inefficient resource management not only reduces infrastructure utilization but also increases operational costs. This is because additional resources are needed to avoid performance bottlenecks caused by unpredictable workloads [11].

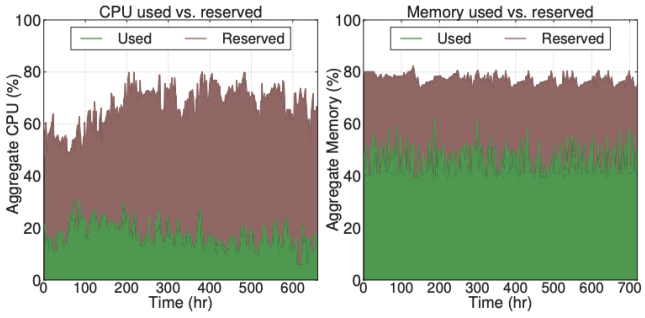


Fig. 1. Resource utilization over a 30-day period for a large production cluster at Twitter managed by Mesos is depicted. The charts (a) and (b) illustrate the relationship between utilization and reservation for the cluster's total CPU and memory capacity. [11]

Figure 1 shows the resource usage of a Twitter production cluster in a month and its management by Mesos. The cluster is used primarily for the user-facing applications, with CPU usage which is less than 20% of the total, while there are reservations that go up to 80% of total capacity. Memory usage, in general, is higher, at 40-50%, but still not as much as the reserved capacity. These variations tend to be highly overestimated in the case of a good number of workloads, which in the end will cause the utilization of resources to be inefficient [11].

This research focuses on the main factors affecting the distribution allocation efficiency frameworks like Apache Spark and Apache Flink. Apache Spark and Apache Flink are used in the process of solving some of these factors for various workloads such as machine learning, graph processing, and stream processing. In addition, the study looks at how memory

management strategies contributed to performance and cost-efficiency, especially when we look at the different dataset sizes and workload complexities. By identifying optimal conditions for in-memory processing and addressing the challenges in resource management, this research aims to enhance the performance and efficiency of distributed data processing frameworks [6], [8].

II. Background

Big Data has impacted several industries in the present world, thus requiring the creation of reliable and efficient data processing frameworks. Proprietary data processing architectures have been demonstrated to be incapable of handling the volume, speed, and variety of data that is available today. Due to the aforementioned problems, distributed data processing frameworks such as Apache Spark, Apache Flink, and Apache Hadoop MapReduce were developed. These frameworks offer effective ways of handling large datasets since they allow map-reduce operations to be performed simultaneously over huge numbers of nodes of relatively low-end hardware.

The first framework in this field was Hadoop MapReduce from Apache. Hadoop also breaks the work into map and reduce stages and operates on a disk based model. While Hadoop utilizes disk-based architecture, which is rather reliable, the latency is often raised because of a large number of I/O operations. [14]

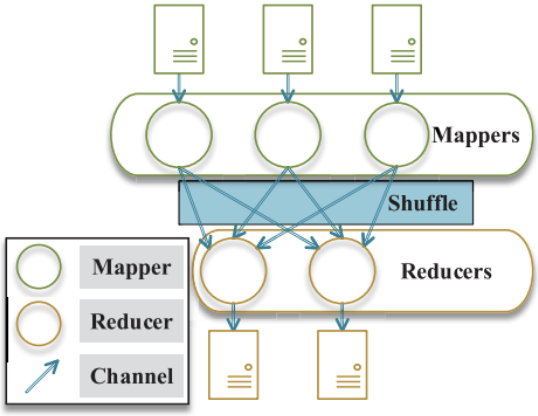


Fig. 2. MapReduce works with two reducers (yellow circles) and three mappers (green circle). Note that the shuffle stage is highlighted via blue arrows. [14]

Apache Spark represents the goals of an open-source, unified analytics engine for big data analysis. It is well known for its capacity to complete all calculations on data in memory, which lowers the iterative algorithms' time complexity.

Resilient Distributed Datasets (RDDs) are a notion offered by Spark that facilitate efficient interoperational data sharing across concurrent operations, allowing for fault tolerance and fast data processing. Spark's in-memory processing capability makes it especially useful for activities like machine learning algorithms that profit from a decrease in the frequency of disk I/O requests. [13]

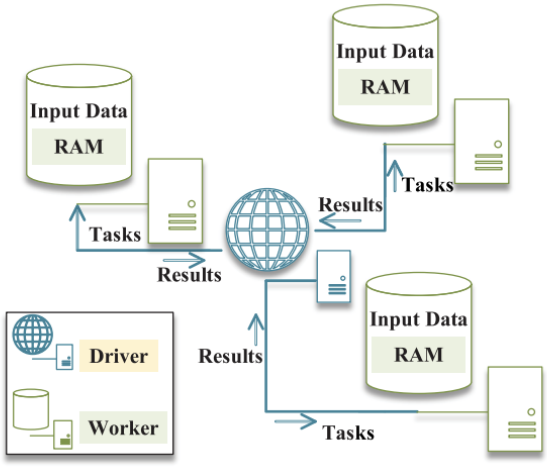


Fig. 3. Spark provides a shared memory to tackle applications in a master-worker scheme. To accelerate job executions, workers read data blocks from a distributed file system and store intermediate results in memory. [14]

With a focus on stream processing, Apache Flink extends the capabilities of in-memory processing to allow for real-time data analysis and continuous intake. With its extensive windowing and state management features, it is ideal for low-latency applications requiring real-time analytics. [12].

Faster data processing is made possible by both Spark and Flink's solutions to disk-based system constraints. On the other hand, there are drawbacks to in-memory processing, namely the need for effective memory allocation to maximize efficiency and avoid bottlenecks. Inadequate memory management can cause data shuffles, disk spills, and frequent trash collection, all of which have a detrimental effect on performance [7].

In distributed data processing frameworks, effective memory management becomes increasingly important as data quantities expand. Maintaining high performance requires effective resource allocation algorithms, particularly for workloads with variable resource demands or repeated data passes. In order to ensure that frameworks like Apache Spark and Apache Flink fulfill the expectations of contemporary Big Data applications, this research attempts to optimize these methodologies [3].

III. Problem Analysis

This section aims to identify the challenges and bottlenecks in resource management within distributed in-memory data processing systems such as Apache Spark. The specific objectives are to examine the impact of inefficient memory allocation on system performance and to explore methods for optimizing resource usage.

We used a well-known public trace dataset that was first made available by Hsu et al. in Arrow[1] for our problem study. This dataset includes eight distinct algorithms that were run on two different dataset sizes and consists of 1031 distinct Spark and Hadoop MapReduce executions that were benchmarked using Intel's HiBench tool. The workloads were executed on

69 different AWS cluster configurations, in sizes ranging from large to 2xlarge, with different memory and core allocations, and scale-outs spanning from 4 to 48 virtual machines (VMs) across instance types c, m, and r [17] [3].

[1]https://github.com/oxhead/scout/tree/master

A. Performance Comparison: In-Memory vs. On-Disk Processing

It is essential to assess the effectiveness of distributed data processing frameworks, especially when working with big datasets and intricate algorithms like K-means clustering. Several studies have demonstrated Apache Spark's benefits over Hadoop's MapReduce architecture. A performance analysis using the HiBench benchmark suite shows Spark often outperforms Hadoop, especially in iterative machine learning tasks like K-means

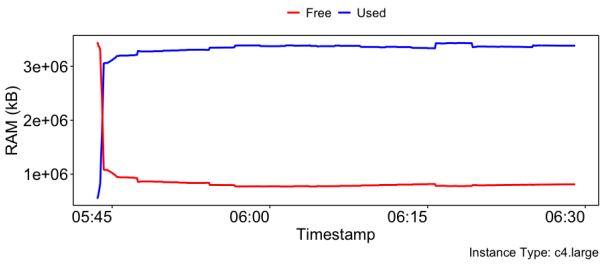
One significant bottleneck in Hadoop's MapReduce framework is eliminated by Spark's in-memory processing capabilities, which decrease read/write cycles to disk [1]. Spark executes K-means clustering more effectively thanks to its capacity to store datasets in memory between rounds; this allows it to speed up TeraSort workloads by up to 14 times [1].

These findings are backed by another study, showing that Spark's in-memory processing is faster than traditional disk-based systems for various data tasks. This highlights Spark's design advantages, especially for applications needing multiple runs over the same data [2].

Picking the right framework based on what your data tasks need is important. For jobs that involve a lot of data passes and iterative machine learning methods, Spark is much better than Hadoop. That's why Spark is the go-to choice for high-speed processing and low latency, especially when it comes to in-memory operations.

B. Workload-Dependent Memory Access

Abstraction of the workload is the act of carrying out the same workload for a variety of memory options, be it instance types or memory sizes. The information provided in the chart plays a crucial role in observing the dependency of memory on the workload. The differences in memory usage by c4.large and c4.2xlarge instances are represented by the RAM usage diagrams. On one side, the c4.large instance reveals a remarkable and immediate increase in the used RAM. On the other hand, it is chosen in terms of the sizeof the c4.2xlarge instance's RAM, which represents very similar behavior, the major discrepancy being the levels of RAM usage. The information herein that the additional memory has a sector to the workload freeing the processor to spare more ram for the operating system. The findings reaffirm the necessity of the correct choice of instance based on the type of memory the workload requires. Large instances with more RAM, such as the c4.2xlarge, are much better at memory-intensive tasks, resulting in shorter times of processing and a substantial improvement in performance in case of large-scale data processing tasks.



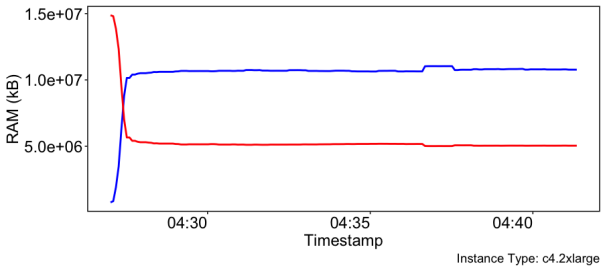


Fig. 4. RAM Usage for c4.large and c4.2xlarge

C. Comparison of Workloads

Different workloads, such as iterative machine learning tasks, batch processing, and stream processing, significantly impact the performance of distributed in-memory data processing frameworks. To manage resources effectively, it's crucial to understand these distinctions.

Research comparing Apache Spark and Apache Flink highlights some interesting points. As an example, a study [12] demonstrates that Flink is a great choice for real-time analytics because of its ability to handle states and process events at the moment, which makes it effective in stream processing. In contrast, Spark shines at in-memory data sharing, which makes it especially useful for workloads involving batch processing and iterative machine learning [13].

According to a research [15], Spark routinely beat Flink in iterative machine learning tasks including K-means clustering and logistic regression. Spark's Resilient Distributed Datasets (RDDs) are responsible for this benefit since they reduce the need for recomputation and data shuffles. Furthermore, Marcu et al. (2016) performed a benchmarking examination with the HiBench suite and found that Flink was superior in stream and graph processing, whereas Spark was superior in machine learning and SQL query processing. [16]

These results emphasize the importance of choosing the right framework based on specific workload requirements. While complex, iterative data processing tasks perform better with Spark, real-time stream processing applications benefit more from Flink. By leveraging the unique strengths of each framework for different types of workloads, it is possible to achieve optimal performance and resource utilization. Future research should continue to explore these differences to gain deeper insights into workload-specific optimizations for distributed data processing frameworks.

D. Impact of Dataset Size

Processing datasets of varying sizes, such as huge and big data, makes performance differences between different instance types apparent. The next table shows these variations. The amount of RAM and CPU power used greatly affects processing speed. Systems such as c4.2xlarge, m4.2xlarge, and r4.2xlarge, which have more RAM and faster CPUs, function consistently better, especially when dealing with larger datasets. This underscores the importance of choosing the appropriate RAM and CPU capacity, highlighting the need for efficient resource cluster management to achieve significant processing time reductions in distributed in-memory computing.

Processing smaller datasets takes a lot less time than processing bigger ones, illustrating the difficulties with big data scalability.

Strategic resource deployment is necessary to enhance distributed data processing systems. Optimizing resource allocation and management ensures cost-effectiveness and high performance, making it crucial for achieving efficient remote data processing.

Instance Type	Dataset Size	Time Span (in minutes)
c4.large	Huge	45
c4.large	Big Data	94
c4.xlarge	Huge	24
c4.xlarge	Big Data	46
c4.2xlarge	Huge	14
c4.2xlarge	Big Data	31
m4.large	Huge	36
m4.large	Big Data	117
m4.xlarge	Huge	16
m4.xlarge	Big Data	30
m4.2xlarge	Huge	3
m4.2xlarge	Big Data	16
r4.large	Huge	35
r4.large	Big Data	116
r4.xlarge	Huge	4
r4.xlarge	Big Data	31
r4.2xlarge	Huge	3
r4.2xlarge	Big Data	5

TABLE I
PERFORMANCE COMPARISON OF INSTANCE TYPES FOR DIFFERENT DATASET SIZES

E. Impact of Inadequate Memory Allocation

In distributed in-memory processing systems like Apache Spark, inadequate memory allocation can severely affect performance. When there isn't enough memory, the system resorts to frequent garbage collection and more data shuffling and spilling to disk. Since disk I/O operations are significantly slower than in-memory operations, this leads to higher latency and lower throughput. For example, iterative algorithms like K-Means clustering can execute much more quickly if the entire dataset fits into memory, thereby avoiding slow disk operations. This consideration becomes increasingly crucial as datasets expand, challenging the limits of cost-effective processing [3].

Equally important is balancing memory with other resources, such as CPU cores, to optimize overall efficiency. Although in-memory processing typically enhances performance, reallocating resources can sometimes yield even better results at a lower cost. Therefore, resource management strategies must be adaptable, responding to the specific demands of various workloads and dataset sizes to sustain high performance and cost efficiency in distributed data processing frameworks. [3], [4].

IV. Discussion

A. Optimal RAM Allocation in Distributed Data Processing

Efficient memory allocation is essential in distributed data processing to keep performance from dropping. When memory runs low, data spills over to the disk, which can slow things down and cause delays. To optimize performance and make sure memory is used effectively, we need dynamic resource allocation that adapts to the workload's needs.

Adaptive memory management has been found to significantly boost in-memory processing framework performance. The Ruya framework, for instance, assists with memory allocation and forecasts memory needs using job profiling. This approach provides for improved RAM allocation by having smaller activities completed in order to predict the memory requirements for bigger ones. [8]

For distributed data processing to be as resource-efficient as possible, memory utilization must be balanced with other resources such as CPU core usage. Research demonstrates that the use of dynamic memory management strategies, which adjust in real-time to workload requirements, may greatly reduce performance bottlenecks caused by inadequate memory allocation. Techniques that minimize memory use and improve overall speed include aggressive garbage collection tuning and adaptive caching [5].

To optimize resource allocation in large-scale distributed systems, load balancing throughout the network, efficient task and data flow scheduling, and lowering overall job execution time are critical. Adaptive memory management approaches can enhance distributed data processing framework performance and efficiency [4]. Systems are guaranteed to be able to effectively handle the demands of big datasets and complicated workloads because to this all-encompassing approach to resource management.

B. Resource Management and Optimization Strategies

Effective resource management is paramount in distributed data processing. The performance disparities observed across different instance types emphasize the need for strategic deployment of resources based on workload requirements. Instances with higher RAM and more powerful CPUs, such as c4.2xlarge, m4.2xlarge, and r4.2xlarge, consistently exhibit better performance for large-scale data processing tasks. This finding highlights the importance of selecting instances with appropriate RAM and CPU capacities to achieve optimal processing times and resource utilization.

Allocating too much RAM can lead to unnecessary costs without corresponding performance benefits. For instance, while large instances like c4.2xlarge or m4.2xlarge are highly effective for memory-intensive tasks, they come at a higher price. If the workload does not fully utilize the available memory, these additional costs do not translate into improved performance, resulting in inefficient resource usage. Therefore, it is crucial to match the instance type and size to the specific demands of the workload to balance cost and performance effectively. [11] [3]

V. Related Work

A. Advanced Cluster Management Systems: Quasar

Strong cluster management systems and the integrated capabilities of distributed in-memory data processing frameworks like Apache Spark can help optimize resource use. Consider Quasar. Its main design concerns are with large-scale cluster resource allocation and management.Quasar dynamically modifies resource allocations to achieve cost and efficiency savings based on machine learning predictions about the needs of individual apps. By allocating resources in accordance with genuine demands, Quasar helps to reduce inefficiencies and avoid performance bottlenecks.

This method enhances not just the performance of individual Spark processes but also the overall use of cluster resources. Such advanced cluster management systems serve as an example of the need of implementing adaptive and intelligent resource management techniques to preserve the efficacy of distributed data processing environments. [11]

VI. Conclusion

Effective resource management is crucial for distributed in-memory data processing frameworks like Apache Spark and Apache Flink, especially in light of big data's exponential development. Our investigation highlights the need for resource balance and memory optimization while also proving the clear performance advantages of in-memory processing over traditional disk-based methods.

We discovered, through the analysis of a public trace dataset, that computers with faster CPUs and more RAM constantly outperform one another, especially when dealing with bigger datasets. However, processing smaller datasets takes less time, which highlights the scalability issues that Big Data presents.

Using dynamic resource management techniques that can adjust to shifting workloads is essential to achieving high performance and cost effectiveness. Various workloads, including iterative machine learning, batch processing, and real-time stream processing—require specific approaches to make the best use of resources and maximize performance. Tackling memory allocation issues and focusing on

workload-specific optimizations will improve the efficiency of distributed data processing frameworks, ensuring they can keep up with the demands of modern Big Data applications.

REFERENCES

[1] N. Ahmed, A.L.C. Barczak, T. Susnjak, and M.A. Rashid, A comprehensive performance analysis of Apache Hadoop and Apache Spark for large scale data sets using HiBench," Journal of Big Data, vol. 7, no. 110, 2020. [Online]. Available: https://doi.org/10.1186/s40537-020-00388-5

[2] H. Al-Sayeh, M. A. Jibril, B. Memishi, and K. U. Sattler, Blink: Lightweight Sample Runs for Cost Optimization of Big Data Applications," in *New Trends in Database and Information Systems*, 2022. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-319-68385-0_6

[3] J. Will, Efficient Resource Allocation in Distributed Systems," arXiv preprint arXiv:2306.03672, 2023. [Online]. Available: https://arxiv.org/pdf/2306.03672

[4] M. Shafiee, Resource Allocation in Large-Scale Distributed Systems," Ph.D. dissertation, Columbia University, 2021. [Online]. Available: https://academiccommons.columbia.edu/doi/10.7916/d8-zyfb-xy42

[5] S. Jananee and M. Suresh, "Dynamic Resource Allocation in Cloud Computing," International Journal of Advanced Research in Computer Science, vol. 14, no. 3, pp. 231-239, 2023.

[6] S. Shahrivari and S. Jalili, "Beyond Batch Processing: Towards Real-Time and Streaming Big Data," arXiv preprint arXiv:1403.3375, 2014. [Online]. Available: https://arxiv.org/abs/1403.3375

[7] J. Will et al., "C3O: Collaborative Cluster Configuration Optimization for Distributed Data Processing in Public Clouds," arXiv preprint arXiv:2107.13317, 2021. [Online]. Available: https://arxiv.org/abs/2107.13317

[8] J. Will et al., "Ruya: Memory-Aware Iterative Optimization of Cluster Configurations for Big Data Processing," arXiv preprint arXiv:2211.04240, 2022. [Online]. Available: https://arxiv.org/abs/2211.04240

[9] P. Lama and X. Zhou, "AROMA: Automated Resource Allocation and Configuration of Mapreduce Environment in the Cloud," in Proceedings of the 9th International Conference on Autonomic Computing (ICAC). ACM, 2012.

[10] K. Rajan, D. Kakadia, C. Curino, and S. Krishnan, "PerfOrator: Eloquent Performance Models for Resource Optimization," in Proceedings of the Seventh ACM Symposium on Cloud Computing (SoCC). ACM, 2016.

[11] C. Delimitrou and C. Kozyrakis, "Quasar: Resource-efficient and QoS-aware Cluster Management," ACM Special Interest Group on Programming Languages (SIGPLAN) Notices, 2014.

[12] P. Carbone et al., "Apache Flink: Stream and Batch Processing in a Single Engine," Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, 2015.

[13] M. Zaharia et al., "Spark: Cluster Computing with Working Sets," in Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing, 2010.

[14] K. Wang, Q. Zhou, S. Guo, and J. Luo, "Cluster Frameworks for Efficient Scheduling and Resource Allocation in Data Center Networks: A Survey," IEEE Communications Surveys and Tutorials, vol. 20, no. 4, pp. 3560-3583, 2018. [Online]. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8416689

[15] A. Verma, L. Cherkasova, and R. Campbell, "Orchestrating an Ensemble of Iterative Tasks for MapReduce," ACM Transactions on Autonomous and Adaptive Systems (TAAS), vol. 8, no. 2, pp. 1-17, 2018.

[16] L. Marcu, C. Craciun, and D. Dobre, "Performance Comparison of Big Data Systems on Batch Processing Using High-Bench Benchmark," in 2016 IEEE 32nd International Conference on Data Engineering Workshops (ICDEW), pp. 108-115.

[17] Chin-Jung Hsu, Vivek Nair, Vincent W Freeh, and Tim Menzies. 2018. Arrow: Low-level Augmented Bayesian Optimization for Finding the Best Cloud VM. In 2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS). IEEE.