Analysing SOJA Algorithm for Parallel Outer Join

This report is based on analysing the paper SOJA: A Memory-efficient Small–large Outer Join for MPI (Liang et al., 2021).

Tan Wei Chun
Monash University
Subang Jaya, Malaysia
wtan0095@student.monash.edu

I. Introduction

The selected research domain focuses on optimizing the performance of parallel join operations for MPI on HPC. With the exponential growth of data, many applications rely on efficient computation to perform analysis on large datasets in a timely manner (Zinieris, 2021). Joins between tables (one of the most expensive operations) presents a significant challenge for optimization due to the high communication costs between distributed processors. This article presents a novel algorithm for performing parallel outer joins that minimizes memory usage while maintaining performance that is equal to or better than industry implementations. I chose this article due to its potential impact on the performance and scalability of distributed joins.

II. Research Background

A. *Problem Statement*

Currently, a large number of studies have been conducted on parallel joins in popular distributed processing platforms such as Spark. However, little to no research had been performed on traditional HPC systems, which communicate using specialized MPI communication protocols. Additionally, existing parallel join approaches rely on either redistribution or broadcast (Blanas et al., 2010) to produce the final output. As a result, in a distributed environment where memory is critical and limited, the join operation will be slowed down due to the unavoidable data skewness or duplication as more data needs to be swapped with the disk causing more expensive I/O operations.

B. *Hypothesis*

The authors of this article present a hypothesis that SOJA algorithm is capable of (1) minimizing the communication costs between distributed processors and (2) reducing the local join cost. The article claims that this leads to a significant reduction in memory usage. To accomplish this goal, the article propose replacing the redistribution or broadcast approaches with a swap approach, in which the smaller table is swapped between processes in each iteration in a ring topology. This approach could help to limit the impact of data skew and duplication, by ensuring that each process encounters the same amount of unseen data.

C. *Key contributions*

Overall, the article introduces a novel method for optimizing small-large outer joins. The SOJA methodology aims to:-

- Reduce memory requirements significantly
- Maintain, in the worst case, comparable or better performance than existing implementations

In most cases, the swap-based join approach reduces memory usage by half and retains efficiency when scaling up to larger datasets indicating good scalability measures. The article also presents further optimization steps that reduces the number of additional joins, filters, and redistributions as seen in existing join implementations.

TABLE I. ANALYSIS BETWEEN ROJA, DOJA, DER, AND DDR

Methods	ROJA	DOJA	DER	DDR
Details	1. Redistribute both tables 2. Local outer join and output	1. Broadcast table R_i 2. Local inner join (T_i) and output 3. Redistribute R_i and T_i 4. Local outer join and output	1. Broadcast table R_i (with row ID) 2. Local outer join (T_i) 3. Redistribute IDs of non-matching tuples (K) in previous step 4. Filter IDs which appears N times (K_i^{filter}) 5. Local inner join of K_i^{filter} and R_i 6. Output union of results in step 2 and step 5	1. Broadcast table R_i (with row ID) 2. Local outer join (T_i) and output 3. Redistribute non-matching tuples (D_i) from previous step 4. Filter dangling tuples which appears N times and output.
Distribution Method	Redistribution	Broadcast	Broadcast	Broadcast
Communication Cost	$(R + S) \times t^{redis}$	$R \times (N-1) \times t^{trans} + (R + T) \times t^{redis}$	$R \times (N-1) \times t^{trans} + K \times t^{redis}$	$R \times (N-1) \times t^{trans} + D \times t^{redis}$
Local Execution Cost	$(R_i^{redis} + S_i^{redis}) \times t^{join}$	$(R + S_i + R_i^{redis} + T_i^{redis}) \times t^{join}$	$(2R + S_i + K_i^{filter}) \times t^{join} + K_i^{redis}$ x $\times t^{filter}$	$(R + S_i) \times t^{join} + D_i^{redis} + t^{filter}$

Fig. 1. Summary of analysis between ROJA, DOJA, DER, and DDR where $|R| << |S|$, t^{redis} = redistribution cost, t^{trans} = broadcast cost, t^{join} = join cost, t^{filter} = filter cost, and N = number of processors.

III. RELATED WORK

This section explores existing implementation of parallel outer joins namely, ROJA, DOJA, DER (Xu & Kostamaa, 2010) and DDR (Cheng et al., 2017), focusing on the communication and local execution cost. The analysis summary is shown in Figure 1.

From communication cost's perspective, ROJA claims the highest cost due to redistribution of entire tables while other methods share similar broadcast cost (minor difference from size of intermediate results). For other methods, the redistribution cost is impacted by selectivity ratio (DOJA) and imbalance workload due to redistribution (DOJA, DER, DDR). SOJA aims to reduce the additional memory needed from distribution and lower the performance impact due to skewness (Contribution 1).

From local execution cost's perspective, ROJA is highly impacted due to data skew from redistribution while DOJA's second join operation relies heavily on the join ratio and skewness degree. For DER and DDR, additional filter step is needed after the local join. DER has advantage on filtering only IDs instead of tuples, however DDR has advantage of not needing an additional join. SOJA aims to further optimize current implementations by reducing redundant operations such as additional joins, filters, and redistributions (Contribution 2).

IV. METHODOLOGY

SOJA utilizes two main concepts to achieve parallel outer join, which are based upon the ring topology (Figure 2) and local hash joins (hash then lookup). Unlike traditional joins, in SOJA, each processor is linked together in a ring structure within a distributed environment, as illustrated below, enabling processors to share data within a complete loop.

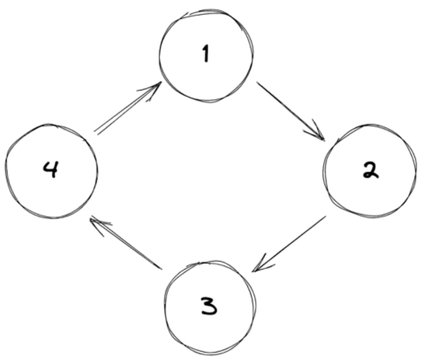

Fig. 2. Ring topology of 4 distributed processors interconnected with each other.

SOJA works using the following steps (see illustration in Figure 3):

1. During the initial data placement (assumed round-robin), create a hash table (H) on the larger table (S_i) on each processor.
2. Execute a local hash join by looping over table R_i and performing a lookup on the hash table H. Mark dangling tuples (tuples with no join matches) in R_i. Return the intermediate join results as direct outputs.
3. Each process sends the marked table Ri to the next (ith + 1) processor according to the ring topology. Note that the last processor will wrap around and send it to the first processor.
4. Repeat steps 2 and 3 N - 1 times where N is the number of processors.
5. Output marked tuples in table R_i as dangling tuples in the final iteration.

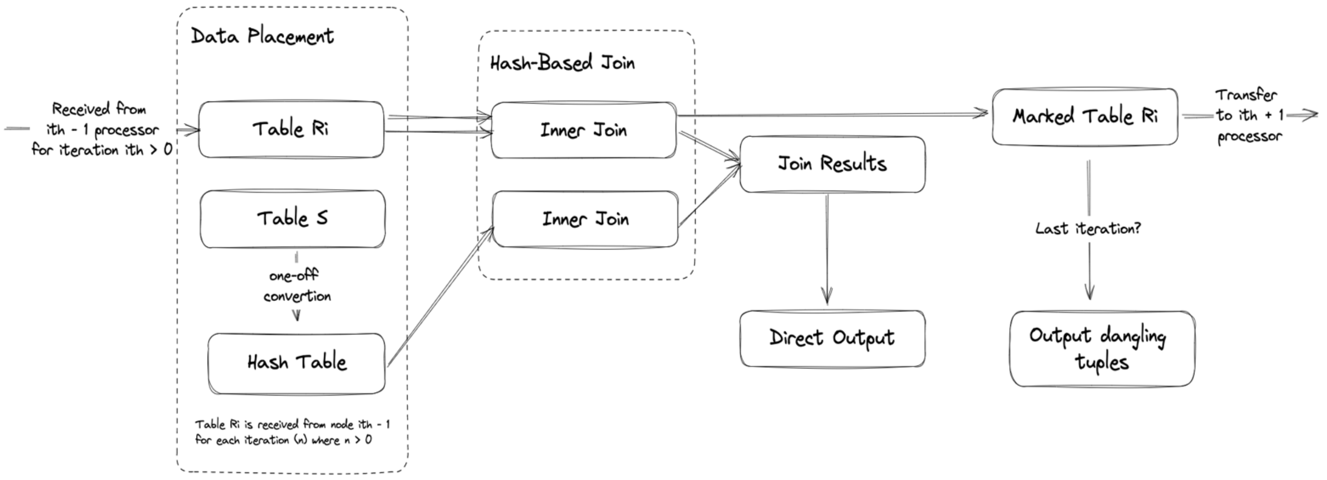

Fig. 3. Architecture overview of SOJA algorithm.

As a result, the communication cost of the SOJA algorithm is R x (N - 1) x t^{trans} and the local execution cost is $(R + S_i) \times t^{join} + R \times t^{update}$ respectively. By limiting network communication to only small table broadcasts, and eliminating unnecessary redistribution, filtering and joining during local execution, SOJA is able to outperform existing implementation and produce a minimal cost model as described above. Additionally, SOJA reduces the amount of temporary memory needed by outputting results readily in each iteration as seen in step 2, and eliminating table duplication and dangling data storage by using the table marking approach.

V. Analysis of Results

This section examines the experimental evaluation based on 3 main factors for performance evaluation in outer join, namely size of left table, selectivity ratio of left table, and data skewness. Metrics from execution time and memory usage are measured for all 5 algorithms in an HPC environment with same underlying hardware. Each evaluation is done with other factors remaining constant.

A. *Size of Left Table*

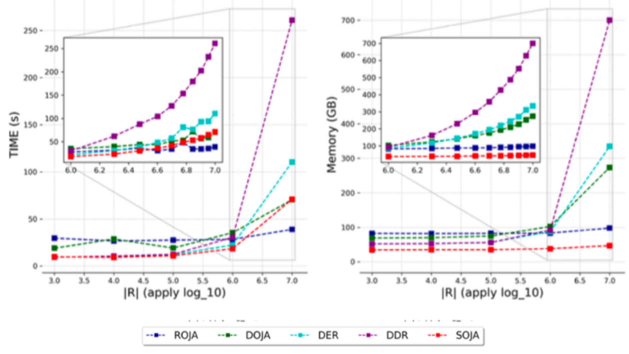

Fig. 4. Execution time versus memory usage of respective algorithm as table R's size increases from 10^3 to 10^7 (Liang et al., 2021).

In terms of performance, SOJA's performance decreases with an increase in table size as the swap step becomes more expensive. However, the performance decrease is not as significant as other algorithms. ROJA stands as a special case in this experiment as its cost model is instead dominated by the larger right table. In terms of memory, SOJA's memory usage remains steady throughout and has a significant advantage compared to the others. Overall, SOJA outperforms other algorithms when |R| is significantly smaller than |S|.

B. *Selectivity Ratio of Left Table*

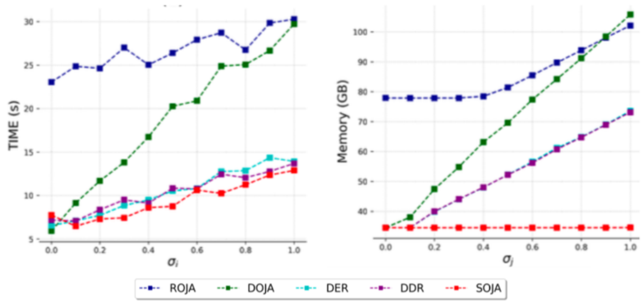

Fig. 5. Execution time versus memory usage of respective algorithm as table R's selectivity ratio increases from 0 to 1 (Liang et al., 2021).

In terms of performance, each algorithm displayed an upward trend as increase in selectivity ratio leads to increase in result size which in turn leads to more I/O cost. DER, DDR and SOJA show similar patterns, with SOJA performing the best followed by DDR then DER. In terms of memory usage, SOJA's usage remains consistently low as results are not kept in memory throughout. Overall, SOJA outperforms other algorithms as selectivity ratio increases.

C. *Data Skewness*

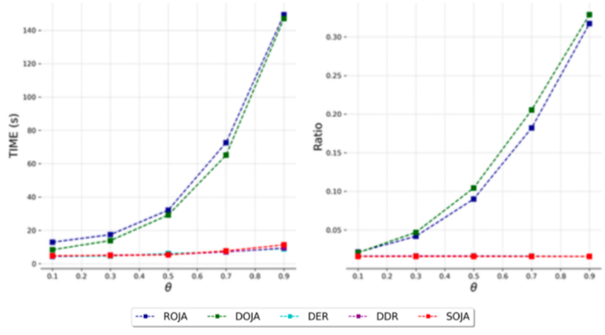

Fig. 6. Execution time versus memory usage ratio of respective algorithm as data skewness degree (in terms of join key) increases (Liang et al., 2021).

In terms of performance and memory usage ratio, both ROJA and DOJA are significantly impacted due to the redistribution step causing an imbalance workload. While DER and DDR have a redistribution step, the dangling tuples/ID size is insignificant to cause impact, which also results in relatively balanced memory ratio. SOJA is not impacted largely by the increase in data skewness since it's defined on the join key which ring topology helps to cater for.

All in all, SOJA achieved the hypothesis set prior to the experiment by leveraging swaps. It outperforms existing algorithms significantly in terms of memory usage while keeping an equal or better performance in terms of execution time.

VI. Conclusion

Based on the outcomes of the experimental evaluation done by the authors, it can be concluded that the SOJA algorithm provides a significant improvement in performing outer joins on large datasets in a distributed environment. The factors used in the experiment to evaluate the hypothesis are justified as they represent the common aspects that significantly impacts runtime and memory performance in outer joins. The experimental results also tally with the reasoning behind the improvement as described within the hypothesis. With SOJA, memory usage decreases significantly because it does not keep temporary results in memory and it reduced the amount of duplication and communication of tables between processors while performance remains equal or better as it eliminates unnecessary steps to redistribute and calculate dangling tuples.

On the other hand, since these experiments were performed using a specific set of benchmarks and constants, it is also possible that the performance of the approach is biased towards the authors' configurations and may vary with different dataset/hardware configuration. Similar experiments should be performed on larger real-world datasets and on different HPC setups to provide a holistic and accurate representation of the algorithm's performance.

Overall, the SOJA algorithm presents a promising approach to optimize parallel outer join by leveraging on data swaps between processors. This approach presents excellent optimization in memory usage with equal or better execution performance. Beyond the current contribution, efforts can also be made to explore the use of similar techniques for other expensive operations, such as inner joins or group-by operations in the future. Additionally, research can also be done to parallelize the swap approach further by scaling up the number of processors within each node.

References

[1] Blanas, S., Patel, J. M., Ercegovac, V., Rao, J., Shekita, E. J., & Tian, Y. (2010). A comparison of join algorithms for log processing in mapreduce. *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*. https://doi.org/10.1145/1807167.1807273

[2] Cheng, L., Tachmazidis, I., Kotoulas, S., & Antoniou, G. (2017). Design and evaluation of small–large outer joins in cloud computing environments. *Journal of Parallel and Distributed Computing*, *110*, 2–15. https://doi.org/10.1016/j.jpdc.2017.02.007

[3] Liang, L., Yang, G., Heinis, T., & Taniar, D. (2021). SOJA: a memory-efficent small-large outer join for MPI. In Y. Velegrakis, D. Zeinalipour, P. K. Chrysanthis, & F. Guerra (Eds.), *Advances in Database Technology - EDBT 2021: 24th International Conference on Extending Database Technology, Proceedings* (pp. 523-528). (Advances in Database Technology - EDBT; Vol. 2021-March). OpenProceedings.org, University of Konstanz, University Library. https://doi.org/10.5441/002/edbt.2021.62

[4] Xu, Y., & Kostamaa, P. (2010). A new algorithm for small-large table outer joins in parallel DBMS. *2010 IEEE 26th International Conference on Data Engineering (ICDE 2010)*. https://doi.org/10.1109/icde.2010.5447835

[5] Zinieris, M. (2021). *Data: A small four-letter word which has grown exponentially to such a big value*. Deloitte Cyprus. https://www2.deloitte.com/cy/en/pages/technology/articles/data-grown-big-value.html