Re-evaluating the Performance Trade-offs for Hash-Based Multi-Join Queries

Shiva Jahangiri
shivaj@uci.edu
University of California, Irvine

Problem and Motivation

As one of the most common and expensive database management system operators, join plays an important role in the query response time and/or throughput of the system. Although the processing and performance evaluation of multi-join queries has been the topic of research for the past decades [8, 12, 13], the complexity and multi-dimensional nature of the problem makes it an unsolved problem for the database community. Our work studies the performance of different classes of query plans, memory distributions for join operators, intra-query concurrency under different assumptions of memory availability, and storage devices such as HDD and SSD. This provides the foundation for understanding basic "join physics", which is useful for designing a resource-based query scheduler for concurrent workloads. We use AsterixDB [1] utilizing both HDD and SSD, to re-evaluate the results of one of the early impactful studies from the 1990s [12] that was originally done using a simulator for the Gamma database system [4].

Background and Related Work

Each operator of a DBMS is made of one or more activities (e.g., build and probe activities in hybrid hash join), possibly with some blocking dependencies which can be used to transform a query to groups of co-schedulable activities (stages). Each query can be transformed into different shapes of query plans. Query plan shapes are typically classified as Right Deep Trees (RDT), Left Deep Trees (LDT), and Bushy Trees (BT). In an RDT, n-1 relations are used for build activities and one relation will drive the probe activity of all joins in a pipelined fashion. RDTs exploit the highest level of concurrency by running all of the build activities concurrently in one stage (Figure 1b). However, a LDT is a more sequential plan. In an LDT, each stage contains at most one build and one probe activity; the probe phase of join m is the input to the build phase of join m+1 (Figure 1a). Bushy Trees are a hybrid version and are not studied in this work due to their large search space. Schneider & Dewitt [12] studied the performance trade-offs of LDT and RDT using a simulator made for the Gamma database system. They showed that in the case of having sufficient memory, RDTs will outperform LDTs due to higher concurrency, while in the case of insufficient memory, LDTs have a lower response time due to lower intermediate result writes, as memory in each stage is only divided at most among two joins. Other types of query plans such as Segmented RDT [3] and ZigZag Tree [15] were proposed and showed improvements over the vanilla RDT by providing more flexibility for plan query generation, better response time, and lower amount of I/O.

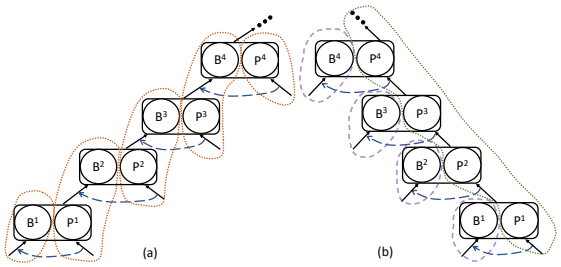

Figure 1-(a) Left Deep Tree, (b) Right Deep Tree. Each dashed area represents one stage.

Approach and Novelty

For a scientific approach, it is important to reproduce the results of prior studies before proceeding with more complex cases. Accordingly, we decided to first re-evaluate the results of a key study done by Schneider & Dewitt [12] in 1990 in which they studied the performance of multi-join queries in shared-nothing clusters. They used a simulator made for the Gamma database system on HDD and we re-evaluate their result using AsterixDB utilizing both SSD and modern HDD as the storage device. Similar to [12], we report the results of experiments on two classes of query plans, LDT and RDT, under two cases of memory availability: 1) Unlimited Memory 2) Limited Memory. We use a single node machine with one partition of 1GB uniformly distributed data. Accordingly, join selectivity and result size of each join is fixed over the query. In the limited memory case, we study two variations of memory distribution for RDTs: The original Right Deep Tree (RDT) and the Static Right Deep Tree (StatRDT). In the RDT case, the allocated memory will be divided equally among the join operators. In StatRDT, the memory is assigned to the join operators in a bottom-up fashion at compile time,

SIGMOD'20, June 14–19, 2020, Portland, OR, USA

ACM ISBN 978-1-4503-6735-6/20/06.
https://doi.org/10.1145/3318464.3384406

hence statically, until memory is not sufficient and one of the joins spills. In this case, StatRDT materializes the intermediate results of the query right before the spilling join operator, i.e. breaks the tree. We have also expanded the Gamma experiments by adding the notion of inter-stage dependency to control the concurrency of activities inside each stage. We introduced another variation of RDTs called Sequential RDT (SeqRDT) in which the build activities run one after another to utilize sequential I/O.

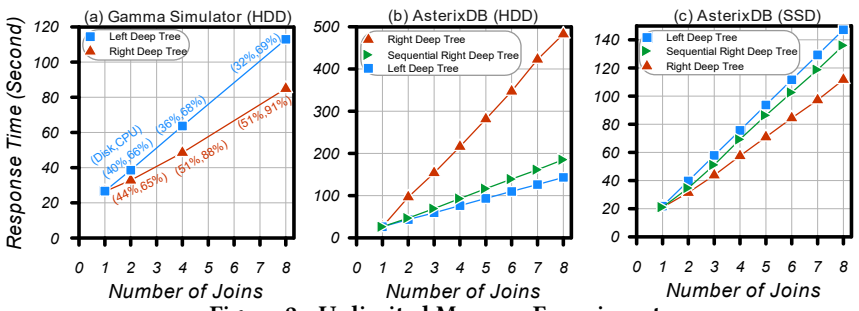

Figure 2 - Unlimited Memory Experiment

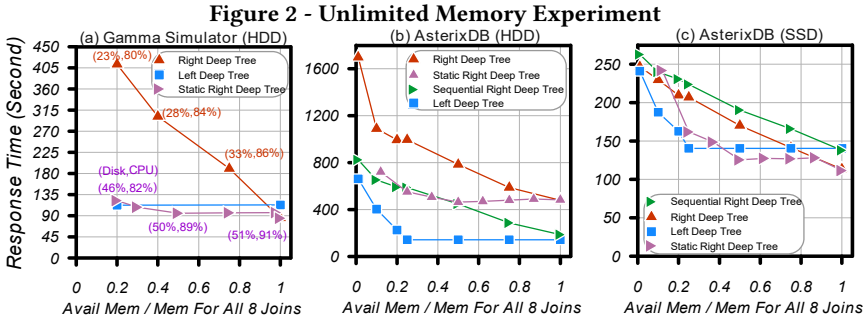

Figure 3 - Limited Memory Experiment

Results and Contributions

Unlimited Memory Experiment. In this experiment, enough memory is available to keep all joins in memory with no spilling to disk. Following [12], we increase the query complexity by increasing the number of joins and we compare the response times of LDT, RDT, and SeqRDT query plans. The (a) figures show simulation results from [12] and (b)/(c) show our AsterixDB HDD/SDD results. As shown in Figure 2b, RDT has a higher response time than LDT. This is due to high number of random I/Os and arm-related disk contention in HDD. SeqRDT, which is a sequential version of RDT, shows more than 2.5 times better response time in comparison to RDT due to more sequential I/O and lower disk contention. In Figure 2c, RDT uses the higher intra-query concurrency at the absence of arm-related disk contention in SSD and is faster than any sequential query plan. A comparison of Figures 2b and 2c clearly shows the importance of the underlying storage type on query performance. We also compared the results for the Gamma database system simulation [12] (Figure 2a) with our results from AsterixDB. Figure 2c matches with 2a in terms of RDT being faster than LDT, but 2a was done on the Gamma simulator for HDD and 2c was done on AsterixDB utilizing SSD. The observed behavior shows that Gamma simulator was not simulating the disk contention behavior accordingly. In Gamma (Figure 2a), CPU becomes more heavily utilized than disk, which confirms the lack of proper disk contention simulation. Figure 2a and 2b report different results for HDD. In Figure 2b, the arm-related disk contention is detrimental to the performance of RDT but for Gamma (Figure 2a) a better response times for RDT is reported due to higher concurrency. This shows that Gamma simulator was unable to simulate HDDs in full detail.

Limited Memory Experiment. In this experiment, we run a query with 8 joins and vary the amount of the available memory as a fraction of the total memory needed for RDT to perform all 8 joins without spilling. By doing this experiment on AsterixDB with HDD (Figure 3b), the less-concurrent query plans have lower response times due to lower arm-related disk contention. Figure 3c shows that for SSD with very limited memory, variations of RDT have higher response times since memory is divided between all join operators (Figure 1b), which causes more spilling to disk and more random I/O. However, in LDT the memory is divided between at most two join operators (Figure 1a), which leads to less disk I/O. In the case of more available memory, RDT takes advantage of higher concurrency and with the absence of arm-related disk contention can run faster than LDT. In Figure 3a (Gamma), RDT shows a better response time in the case of high available memory due to high concurrency; however, Figure 2b shows that the impact of disk contention in HDD is so significant that the less-concurrent plans with lower amount of I/O will be better options for HDD. The results of AsterixDB with SSD (Figure 3c) are again more matching with the Gamma (3a). In conclusion, with HDD, sequential plans have better response times due to lower disk contention while with SSD the higher concurrency leads to better response times in the absence of arm-related disk contention. These results show the importance of the underlying storage device in choosing the query plan. They show that re-evaluation of the previous studies is necessary so often due to improvements in the underlying hardware. We saw also that simulators, while are helpful and important tools for understanding systems' behavior, they can produce incorrect results if not verified against real systems carefully.

Acknowledgements

I would like to thank my PhD advisors Michael J. Carey and Johann-Christoph Freytag for their continued help and support of this work.

REFERENCES

[1] Sattam Alsubaiee et al. (2014). AsterixDB: a scalable, open source BDMS. Proceedings of the VLDB Endowment. 7, 14, 1905-1916.

[2] Shivnath Babu and Herodotos Herodotou. (2013). Massively Parallel Databases and MapReduce Systems. Foundations and Trends in Databases archive. 5, 1. 1-104.

[3] Ming-Syan Chen, Ming-Ling Lo, Philip S. Yu, and Honesty C. Young. (1992). Using Segmented Right-Deep Trees for the Execution of Pipelined Hash Joins. In Proceedings of the 18th International Conference on Very Large Data Bases (VLDB '92), Li-Yan Yuan (Ed.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 15-26.

[4] David J. DeWitt et al. (1990). The Gamma Database Machine Project. IEEE Transactions on Knowledge and Data Engineering. 2, 1. Taiwan. 44-62.

[5] David J. DeWitt and Jim Gray. (1992). Parallel database systems: the future of high performance database systems. Communications of the ACM. 35, 6. New York, NY, USA, 85-98.

[6] Minos N. Garofalakis and Yannis E. Ioannidis. (1997). Parallel Query Scheduling and Optimization with Time- and Space-Shared Resources. In Proceedings of the 23rd International Conference on Very Large Data Bases (VLDB '97), San Francisco, CA, USA, 296-305.

[7] Goetz Graefe. (1993). Query evaluation techniques for large databases. ACM Computing Surveys. 25, 2. New York, NY, USA, 73-169.

[8] Laura M. Haas, Michael J. Carey, Miron Livny, and Amit Shukla. (1997). Seeking the truth about ad hoc join costs. The International Journal on Very Large Data Bases Journal. 6, 3. 241-256.

[9] Feilong Liu and Spyros Blanas. (2015). Forecasting the cost of processing multi-join queries via hashing for main-memory databases. In Proceedings of the Sixth ACM Symposium on Cloud Computing (SoCC '15). 153-166.

[10] Manish Mehta and David J. DeWitt. (1995). Managing Intra-operator Parallelism in Parallel Database Systems. In Proceedings of the 21th International Conference on Very Large Data Bases (VLDB '95). San Francisco, CA, USA, 382-394.

[11] Donovan A. Schneider and David J. DeWitt. (1990). Tradeoffs in Processing Complex Join Queries via Hashing in Multiprocessor Database Machines. In Proceedings of the 16th International Conference on Very Large Data Bases (VLDB '90). San Francisco, CA, USA, 469-480.

[12] Donovan A. Schneider (1990). Complex Query Processing in Multiprocessor Database Machines. Thesis submitted to University of Wisconsin-Madison.

[13] Leonard D. Shapiro. (1986). Join processing in database systems with large main memories. ACM Trans. Database Syst. 11, 3 (August 1986), 239-264. DOI=http://dx.doi.org/10.1145/6314.6315

[14] Annita N. Wilschut, Jan Flokstra, and Peter M. G. Apers. (1995). Parallel evaluation of multi-join queries. In proceeding of the 1995 ACM SIGMOD international conference on Management of data., San Jose, CA, USA,115-126

[15] Mikal Ziane, Mohamed Zaït, and Pascale Borla-Salamet. (1993). Parallel query processing with zigzag trees. The International Journal on Very Large Data Bases Journal. 2, 3. 277-302.