Towards a Memory-Adaptive Hybrid Hash Join Design

Giulliano Silva Zanotti Siviero
Department of Computer Science and Engineering
Santa Clara University
Santa Clara, United States
gsilvazanottisiviero@scu.edu

Shiva Jahangiri
Department of Computer Science and Engineering
Santa Clara University
Santa Clara, United States
sjahangiri@scu.edu

2023 IEEE International Conference on Big Data (BigData) | 979-8-3503-2445-7/23/$31.00 ©2023 IEEE | DOI: 10.1109/BigData59044.2023.10386098

Abstract—In database management systems (DBMSs) that handle multiple concurrent queries, adapting to fluctuating workloads is crucial. This flexibility allows the DBMS to revise decisions based on current workload and available resources. As memory availability changes with the arrival or completion of queries, having memory-intensive operators like the Hybrid Hash Join that dynamically adapt is vital. This paper introduces a new memory-adaptive Hash-Based join algorithm design implemented in Apache AsterixDB and evaluates its responsiveness to memory variability.

Index Terms—Hybrid Hash Join, Memory Adaptiveness, Resource Management

Introduction

Despite years of research, managing memory for concurrent queries remains a complex issue for most DBMSs. Due to limited memory capacity, allocating memory judiciously among memory-intensive operators of concurrent queries is essential. Many DBMSs use fixed memory allocations for these operators, but this static approach limits flexibility with unpredictable workloads, often leading to resource underutilization and suboptimal system performance.

The Hybrid Hash Join (HHJ), a commonly used and memory-intensive join operator, can greatly influence system performance. While studies in the 1990s focused on memory-adaptive HHJ designs using HDD simulators [6, 13], many current DBMSs still employ memory-static (non-adaptive) operators, retaining allocated memory throughout the query execution [1, 2, 3, 4]. We argue that static memory allocation remains problematic even with modern databases using rapid SSDs or disaggregated storage. Such static memory allocation can lead to unnecessary delays for queries, resource underutilization, and unnecessary scaling in cloud environments.

In this work, we design and implement a memory-adaptive HHJ (MA-HHJ) algorithm for big data management systems and evaluate its responsiveness to the memory change requests of resource manager. We use Apache AsterixDB [2, 5], an open-source big data management system, for implementation and evaluation of our approach on a real system.

The remainder of the paper is organized as follows: Section I discusses previous work related to this study. Section II describes the design of our MA-HHJ algorithm and provides details of its implementations. Section III describes the experiments conducted and their preliminary results, while Section IV concludes the paper.

I. Related Work

Hybrid Hash Join was proposed initially in the 1990s, and several studies were conducted analyzing its performance theoretically and empirically [8, 9, 14]. The initial design of HHJ relied heavily on precise data statistics to determine the optimal number of partitions and to choose the in-memory resident partition accurately. This approach aimed to maximize memory utilization and minimize disk spilling. Nevertheless, maintaining these statistics is not always practical.

To overcome this challenge, the authors of [11] developed a dynamic destaging approach for HHJ, enabling all partitions to expand within the available memory limits. When memory becomes limited, the algorithm dynamically selects partitions to spill during runtime. This approach termed Dynamic Hybrid Hash Join (DHHJ), has been incorporated into several contemporary DBMSs, including Apache AsterixDB [10], which is utilized in this study. While DHHJ enhances the flexibility of the original HHJ design, it still manages memory statically and cannot adjust to memory changes. To address this issue, the authors of [15] introduced a Hash-Based algorithm capable of adapting to memory fluctuations during the Build Phase through a bucket tuning method. Subsequently, two additional memory-adaptive algorithms, Partially Preemptible Hash Join [12] and Memory-Contention Responsive Hash Join [7], were proposed. These algorithms adjust to new memory allocations during the Probe Phase. Notably, the Memory-Contention Responsive Hash Join is straightforward to implement and, as indicated in [7], offers superior performance compared to Partially Preemptible Hash Join. Both algorithms were assessed and contrasted in Davison [7] using a simulator due to the limited computing resources available at that time.

To the best of our knowledge, these foundational papers set the stage for memory-adaptive database operations. In contemporary DBMSs, however, there is a noticeable preference for simpler, memory-static operators despite the efficiency gains offered by memory-adaptive alternatives [1, 2, 3, 4]. For instance, in Amazon Redshift, the memory of queued queries can be modified; however, the already-executing queries will retain their initial memory allocations [1].

II. Memory-Adpative Hybrid Hash Join: Design

In this section, we outline the structure and implementation specifics of MA-HHJ. This variant, which is an adaptation of AsterixDB's standard HHJ algorithm [10] to be memory-adaptive, interacts frequently with the resource manager for memory budget adjustments through two mechanisms: 1) Event-based: When extra memory is needed, such as to avoid disk spilling, the HHJ operator will request it from the resource manager. 2) Frame-Interval: The algorithm initiates a check with the resource manager at intervals of every X frames processed by the HHJ. It is important to note that the resource manager is responsible for deciding memory allocation for operators to optimize performance. Our focus is solely on crafting an MA-HHJ responsive to memory changes. Our resource manager is a simulated component described in Section III. Each interaction with the resource manager results in one of the following three responses: 1) an increase, 2) decrease, or 3) no change in memory allocation.

A. Build Phase.

MA-HHJ can respond to memory decrease responses by releasing unused memory and spilling partitions, if necessary. Such partition spillings are simple since records are not inserted in the hash table yet.

In case of a memory increase response, MA-HHJ will accept the extra memory offered only if it can use it immediately. Before building the hash table, the MA-HHJ tries to reload spilled partitions into memory if enough memory is available. Listing 1 illustrates how our algorithm handles memory budget changes during the Build Phase.

```
Frame-Interval Memory Update:
for every frame interval:
  update = check with the resource manager
  if (update == decrease): spill partitions
  else: continue

Event-Based Memory Update:
  update = check with the resource manager
  if (update == decrease): spill partitions
  else if ( update == increase):
    accept and avoid spilling
  else: continue;

After Processing All Records:
  update = check with the resource manager
    if (update == decrease): spill partitions
    else if (update == increase): reload partitions
    else: continue;
  build hash table
```

Listing 1. Pseudocode for Build Phase

B. Probe Phase.

In case of memory increment, spilled partitions may be reloaded into memory during the Probe Phase if enough memory is available; however, their state is different from memory-resident partitions since there might have been some tuples from the Build input that probed the hash table while such partitions were disk-resident; thus they were written to disk as well to be processed later. To handle this situation, an additional step is necessary by the end of the Probe Phase to probe such "missed" tuples by probing them against the reloaded partitions. It is important to note that restoring disk-resident partitions during the Probe Phase may avoid unnecessary rounds of Joins. Listing 2 delves into this possibility.

In case of memory decrement, partition spellings trigger a hash table rebuilt to avoid the complexity of maintaining this data structure.

```
Frame-Interval Memory Update:
for every frame interval:
  update = check with the resource manager
  if (update == decrease): spill partitions
  else if (update == increase):
    while (memory is available):
      try to reload a partition from the disk
      if (partition reloaded)
        rebuild hash table

Event-Based Memory Update:
  update = check with the resource manager;
  if (update == increase):
    try to reload one or more partitions;
    if (any partition reloaded):
     rebuild hash table;
    else: do not accept memory;
  else if (update == decrease): spill partitions;
  else: continue;

After Processing All Records:
  reload tuples belonging to inconsistent partitions
     that were spilled to the disk & probe them
```

Listing 2. Pseudocode for Probe Phase.

III. Experiments and Results

We used a single join query to study the responsiveness of our MA-HHJ algorithm in a single Apache AsterixDB node configuration running in an 8-core 11th Gen Intel® Core™ i7 with 64GB of RAM and a 2TB NVME storage. The build and probe relations each have 1 GB of data, and each entry in the build relation matches exactly one entry in the probe relation.

To account for different memory contention scenarios, the resource manager assigns the memory budget for the operator according to the following distribution: During 80% of the time, the memory was chosen randomly from 80% to 100% of the maximum memory for join and the rest of the time between the minimum possible (20 frames based on [10]) to 100% of the maximum required memory for join.

As Figure 1 shows, more frequent check-ins with the resource manager improve the responsiveness of MA-HHJ to memory changes with the expense of a higher amount of I/O, which can lead to higher execution time for the operator.

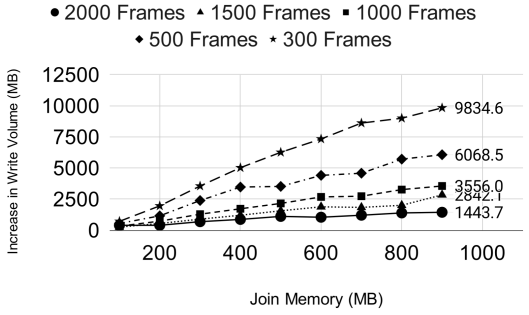

Fig. 1. Impact of Frame Interval on I/O Volume in MA-HHJ

Due to space limitations, we only present the results of Frame-based experiments. The increase in I/O operations and volume directly impacts the operator's execution time, Figure 2 presents the impact of memory adaptivity over the operator's execution time.

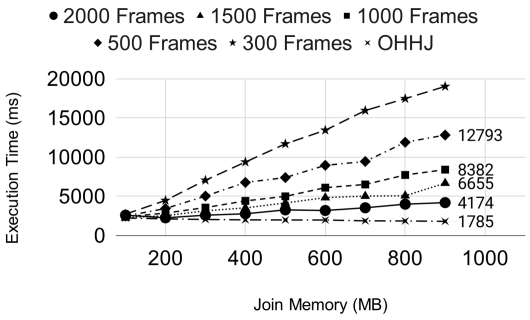

Fig. 2. Impact of Frame Interval on Execution Time in MA-HHJ

IV. Conclusion and Further Studies

The central contribution of this study lies in understanding how memory variance and the frequency of memory budget updates impact the I/O volume and, consequently, the execution time of an operator. We showed that the responsiveness of MA-HHJ is directly impacted by the frequency of operator check-ins with the resource manager. The future work of this study involves studying and designing dynamic memory assignment logic for the resource manager in order to maximize its performance goals as well as designing memory-adaptive versions of other memory-intensive operators such as sort.

References

[1] 2023. Amazon Redshift: WLM dynamic memory allocation. https://docs.aws.amazon.com/redshift/latest/dg/cm-c-wlm-dynamic-memory-allocation.html. Accessed: 2023-11-22.

[2] 2023. Apache AsterixDB. https://asterixdb.apache.org/. Accessed: 2023-11-22.

[3] 2023. Microsoft SQL Server: Memory management architecture guide. https://learn.microsoft.com/en-us/sql/relational-databases/memory-management-architecture-guide?view=sql-server-ver16. Accessed: 2023-11-22.

[4] 2023. Resource Consumption in Postgresql. https://www.postgresql.org/docs/current/runtime-config-resource.html. Accessed: 2023-11-22.

[5] Sattam Alsubaiee et al. "AsterixDB: A Scalable, Open Source BDMS". In: Proc. VLDB Endow. 7.14 (2014), pp. 1905–1916. DOI: 10.14778/2733085.2733096. URL: http://www.vldb.org/pvldb/vol7/p1905-alsubaiee.pdf.

[6] Diane L. Davison and Goetz Graefe. "Memory-Contention Responsive Hash Joins". In: VLDB'94, Proceedings of 20th International Conference on Very Large Data Bases, September 12-15, 1994, Santiago de Chile, Chile. Ed. by Jorge B. Bocca, Matthias Jarke, and Carlo Zaniolo. Morgan Kaufmann, 1994, pp. 379–390. URL: http://www.vldb.org/conf/1994/P379.PDF.

[7] Diane L. Davison and Goetz Graefe. "Memory-Contention Responsive Hash Joins". In: Very Large Data Bases Conference. 1994.

[8] David J. DeWitt et al. "Implementation Techniques for Main Memory Database Systems". In: SIGMOD'84, Proceedings of Annual Meeting, Boston, Massachusetts, USA, June 18-21, 1984. Ed. by Beatrice Yormark. ACM Press, 1984, pp. 1–8. DOI: 10.1145/602259.602261. URL: https://doi.org/10.1145/602259.602261.

[9] Laura M. Haas et al. "Seeking the Truth About ad hoc Join Costs". In: VLDB J. 6.3 (1997), pp. 241–256. DOI: 10.1007/S007780050043. URL: https://doi.org/10.1007/s007780050043.

[10] Shiva Jahangiri, Michael J. Carey, and Johann-Christoph Freytag. "Design Trade-offs for a Robust Dynamic Hybrid Hash Join". In: Proc. VLDB Endow. 15.10 (2022), pp. 2257–2269. URL: https://www.vldb.org/pvldb/vol15/p2257-jahangiri.pdf.

[11] Masaya Nakayama, Masaru Kitsuregawa, and Mikio Takagi. "Hash-Partitioned Join Method Using Dynamic Destaging Strategy". In: Proceedings of the 14th International Conference on Very Large Data Bases. VLDB '88. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1988, pp. 468–478. ISBN: 0934613753.

[12] Hwee Hwa Pang, Michael J. Carey, and Miron Livny. "Partially Preemptible Hash Joins". In: SIGMOD Rec. 22.2 (1993), pp. 59–68. ISSN: 0163-5808. DOI: 10.1145/170036.170051. URL: https://doi.org/10.1145/170036.170051.

[13] HweeHwa Pang, Michael J. Carey, and Miron Livny. "Partially Preemptive Hash Joins". In: Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, Washington, DC, USA, May 26-28, 1993. Ed. by Peter Buneman and Sushil Jajodia. ACM Press, 1993, pp. 59–68. DOI: 10.1145/170035.170051. URL: https://doi.org/10.1145/170035.170051.

[14] Leonard D. Shapiro. "Join Processing in Database Systems with Large Main Memories". In: ACM Trans. Database Syst. 11.3 (1986), pp. 239–264. DOI: 10.1145/6314.6315. URL: https://doi.org/10.1145/6314.6315.

[15] Hansjörg Zeller and Jim Gray. "An Adaptive Hash Join Algorithm for Multiuser Environments". In: 16th International Conference on Very Large Data Bases, August 13-16, 1990, Brisbane, Queensland, Australia, Proceedings. Ed. by Dennis McLeod, Ron Sacks-Davis, and Hans-Jörg Schek. Morgan Kaufmann, 1990, pp. 186–197. URL: http://www.vldb.org/conf/1990/P186.PDF.