View Reviews

Paper ID

841

Paper Title

Hazard-Adaptive Query Processing Using Hardware-Assisted Runtime Profiling

Track Name

Research Paper 4th Submission Round

Reviewer #1

Questions

2. Suitability to the selected track

Correct track.

4. Novelty

No novelty

6. Summary of contribution (in a few sentences)

The authors present hazard-adaptive query processing, a technique that chooses different operator implementations based on hardware performance counters. They classify operator implementations into "robust" and "affected" and propose a simple architecture to switch between these classes. The evaluation shows that adaptivity to data characteristics achieves better results than using a single implementation.
While the idea is nice, the technical depth of this paper is limited. The evaluation is too narrow and it is not clear how well this approach generalizes. Important details about the collection and design of heuristics are missing, which makes it hard for readers to actually apply this in other settings.

7. Describe in detail all strong points, labeled S1, S2, S3, etc.

S1: Fully utilizing the hardware in databases is an important topic and the idea to adapt operators fits nicely into this space.

S2: The evaluation shows that performance counters can be used to adapt operator implementations to data characteristics.

S3: The paper is well-written and easy to follow.

8. Describe in detail all opportunities for improvement, labeled O1, O2, O3, etc.

O1: The insights presented in this paper are very limited and have been shown in previous work.

O2: The evaluation is limited to a very narrow scope.

O3: It is not clear how well this approach generalizes.

O4: The adaptivity approach adds a lot of complexity to a system with only limited performance impact in some cases.

9. Please rank the three most critical strong points and/or opportunities for improvement that led to your overall evaluation rating, e.g., "{S1, O7, O4}"

O1, O2, O3

10. Inclusive writing: does the paper follow the SIGMOD inclusive writing guidelines? See https://dbdni.github.io/pages/inclusivewriting.html for more information. If not, please provide suggestions on how to improve the writing of the paper.

I agree

11. D&I compliance

The line charts are not readable in a B/W print. The authors should improve this for better readability.

12. Availability: We have asked authors of all papers to make their code, data, scripts, and notebooks available, if this is possible. You can find this information in the summary of the paper on CMT when clicking on the Paper ID next to the title. In the text box provided below, please provide (a) constructive feedback to the authors about these provided artifacts (if any), (b) how this was useful in your evaluation of the submission, and (c) what changes are needed in order to make the code/data/scripts/notebooks shareable with the community (if any).

The artifacts look fine and would help readers to understand the authors' implementation.

13. Minor remarks. Use this field to describe minor remarks that are not listed as opportunities for improvement, e.g., to highlight typos, formatting problems, or minor technical issues.

D1: The insights from this paper are rather limited. It is well known that adaptivity to data characteristics improves performance, as the authors point out in their related work. While the approach is slightly different by using performance counters to adjust robustness along a scale, it is fundamentally similar to previous work discussed by the authors.

D2: The authors state that a calibration phase must be run to gather information about the heuristics but then do not discuss or evaluate this further. I consider this to be a, if not the crucial part for this design to work well in practice. It is hard to say if the authors hand-tuned the heuristics for their server or if they were gathered. The authors should discuss this calibration in more detail and evaluate its robustness.

D3: It is not possible to fully understand the performance impact in OLAP queries by looking at single-threaded individual operators. The authors should at least show a full, multi-threaded query containing multiple operators. Without this, it is not clear how much of an impact these results have on an end-to-end setup. Ideally, they could integrate this into a real system to show the impact.

D4: Making decisions based on performance counters is highly dependent on the microarchitecture of each system. The authors only evaluate a single server. Do these insights translate to other systems or are they artifacts of this particular server?

D5: The CPU in the evaluation server is *very* old. It was introduced in 2013 and has very different characteristics to modern CPUs, e.g., memory bandwidth is significantly lower. It is unclear whether an evaluation on a modern server would yield the same results. The authors should run the evaluation on a newer system, that is more representative of today's server landscape.

D6: How does this approach transfer to non x86 servers? For example, do ARM servers, which are becoming more prevalent, provide the same performance counters used in this work? This should be discussed by the authors to show how well this approach generalizes.

D7: It is well known that 2 MiB huge pages improve the performance of radix partitioning because of fewer TLB misses. However, the authors only evaluate 4 KiB pages. It is not clear how well their adaptive approach translates to 2 MiB pages, as the overall number of misses will be reduced regardless of the number of bits.

D8: From the evaluation, it seems like this adaptivity nearly never pays off for GROUP BY. These results indicate that the authors' approach is not as widely applicable as they claim.

D9: Bandle et al. [4] have shown that radix joins only perform well in very narrow setups, which we often do not see in the real world. Based on this insight, it is not clear how the JOIN-adaptivity translates to other join operators, which may be more "robust" than radix joins in the first place. The authors should at least discuss this, if not implement other join variants for a better comparison.

D10: The authors distinguish between hazard-robust and -affected operators. However, they only present a robust one for SELECT. For JOIN, they only use affected variants and for GROUP BY, they trade a cache-miss-affected hash table for a TLB-miss-affected radix sort. So overall there seems to be only a single robust operator. I'm not sure this binary classification makes sense if there are no robust implementations for other operators than SELECT.

D11: I did not fully understand the "adjustment" part in the operator interface. Is this just an integer that says: "be more robust by X" and once it reaches an internal threshold, the operator switches the implementation? The authors could make this more clear in the text.

Reviewer #2

Questions

2. Suitability to the selected track
The paper is suitable for the research track.

4. Novelty
No novelty

6. Summary of contribution (in a few sentences)
This paper describes an approach to query execution that uses performance counters to identify the presence of micro-architectural hazards (such as cache misses and branch mispredictions) and adaptively change to a different algorithm for the current operator on-the-fly. For example, for the SELECT operator, the adaptive system can swap between a branch-based implementation (hazard affected) and a predication-based implementation (hazard robust) when it detects that branch mispredictions have crossed a threshold. A similar approach is presented for GROUP BY and JOIN in case of last-level cache misses or sTLB store misses. The evaluation shows that the adaptive approach can select the best algorithm adaptively as the data changes, however it does not convincingly show that this approach is practical in the general case and it does not compare to alternative approaches.

7. Describe in detail all strong points, labeled S1, S2, S3, etc.
S1. The paper is well written and easy to read. The solution is generally well-motivated, and the benefits of the solution for each operator (SELECT, GROUP BY and JOIN) are well explained.

S2. The evaluation shows that this approach is beneficial. It allows SQL execution operators to achieve near best-case performance while avoiding the worst-case even as the data changes. In many cases the performance improvement is significant, and close to one order of magnitude in some cases.

S3. The solution appears to be well-engineered, and the availability of the code will be valuable to researchers and practitioners.

8. Describe in detail all opportunities for improvement, labeled O1, O2, O3, etc.
O1. The evaluation does not compare to any other approaches. The Introduction and Section 7 mention several other adaptive and non-adaptive solutions, but there are no experiments showing that the adaptive approach in this paper is superior. At a minimum, I would expect a comparison with Zeuch et al. ([36] in the paper), since that approach seems the most similar to the proposed approach.

O2. The novelty is limited as the approach does not seem sufficiently different from [36]. Using heuristics instead of a cost model and reoptimization is certainly simpler, but it's not clearly better or more extensible, as heuristics also require tuning.

O3. The performance of this approach relies on well-tuned heuristics: (1) to identify the right thresholds for switching algorithms, and (2) to identify the "adaptive period" (frequency of reading performance counters and performing probes). There is only a minimal discussion of how these heuristics would be set in practice and no experiments showing the robustness of the selected values to different data or workloads (e.g., variations in the size of the database / memory pressure). Given that these heuristics are critical to the performance of the system and the key differentiator from [36], I would have expected more of a focus on them in the evaluation.

O4. It seems like the extensibility of this approach may be limited since it appears that a custom solution must be implemented for each set of physical operators in each database on each supported architecture.

O5. The introduction and Section 7 both claim that non-adaptive approaches that measure the sortedness of the data ([1, 5, 14] in the paper) are "complicated and brittle". It's not clear why this is the case, and I think it deserves more discussion, as it is key to the argument motivating the need for an adaptive algorithm.

O6. The paper argues that certain high-level metrics such as cardinality can only be determined after processing is complete, and are therefore not usable. However, this ignores the fact that most query optimizers rely on such statistics that are collected offline. Although the stats may be slightly stale, they are generally accurate enough to be useful.

O7. In some places (e.g., the end of Section 3.6) the paper implies that the hazard-robust algorithm is run after all processing with the hazard-affected algorithm is complete. But this conflicts with the concept of "probing" to potentially switch back to the hazard-affected algorithm. I think this should be clarified.

O8. Footnote 3 says that "we include a single pass to identify the number of bits required...". Isn't that inefficient? Does the evaluation account for this?

O9. It's not clear what Figure 6 is demonstrating. I think it would help to show the tipping points on the chart and/or include the performance of the foundational algorithms.

9. Please rank the three most critical strong points and/or opportunities for improvement that led to your overall evaluation rating, e.g., "{S1, O7, O4}"

O1, O2, O3

10. Inclusive writing: does the paper follow the SIGMOD inclusive writing guidelines? See https://dbdni.github.io/pages/inclusivewriting.html for more information. If not, please provide suggestions on how to improve the writing of the paper.

I agree

11. D&I compliance

It follows the guidelines, although the authors could consider replacing the terms "black box" and "white box" in the introduction with "opaque"/"clear".

12. Availability: We have asked authors of all papers to make their code, data, scripts, and notebooks available, if this is possible. You can find this information in the summary of the paper on CMT when clicking on the Paper ID next to the title. In the text box provided below, please provide (a) constructive feedback to the authors about these provided artifacts (if any), (b) how this was useful in your evaluation of the submission, and (c) what changes are needed in order to make the code/data/scripts/notebooks shareable with the community (if any).

The authors provided an anonymous code repository. The code seems to be reasonably well organized, but the lack of a README makes it difficult to know how to reproduce the results.

13. Minor remarks. Use this field to describe minor remarks that are not listed as opportunities for improvement, e.g., to highlight typos, formatting problems, or minor technical issues.

M1. Calling SELECT, GROUP BY, and JOIN the three most prominent relational operators is somewhat debatable. What about PROJECT? Why not just call these 3 "important"?

M2. Figure 2 is unnecessarily large. This is one example of some space that could be reclaimed to make room for a more comprehensive evaluation.

M3. I think Section 3.3 should be "Hazard-Adaptive", not "Hazard-Affected". This mistake is repeated in the first sentence of the section as well.

M4. Acronyms such as DOP should be defined before using them. I think this is "degree of parallelism", but I don't think this acronym is so common that everyone will know it.

Reviewer #3

Questions

2. Suitability to the selected track

Good fit for regular paper in research track

4. Novelty

Novel implementation and evaluation of previous solutions to existing problem

6. Summary of contribution (in a few sentences)

The paper addresses the problem that hazards, e.g. many branch miss predictions or many data cache misses, can result in performance degradations.
For some of these hazards counter measures are known, e.g. branchless evaluation of filter conditions. Similar to prior work, the paper suggests
to collect low-level hardware information as they are provided by most CPUs today and detect and react if hazards are identified by switching to
alternative algorithms while an operator processes its input data.
While these ideas and concepts sound theoretically reasonable, they raise a number of practical questions that need to be solved and which are "just"
stated as assumptions, e.g. knowledge about exact cardinalities or dealing with concurrent workload. I also have concerns regarding how efficient
the operators are implemented as they seem to fully materialize their results. Furthermore the paper misses to analyze critical implementation
alternatives or aspects for all considered operators. The experiments are conducted on an outdated platform, and hence the relevance of these
results on modern hardware remains unclear.

7. Describe in detail all strong points, labeled S1, S2, S3, etc.

S1: Robustness in query processing is an important and relevant topic.
S2: The experiments indicate that using performance counters to switch between implementation variants of operators can be beneficial.
S3: The paper is well-written und easy to understand.

8. Describe in detail all opportunities for improvement, labeled O1, O2, O3, etc.

O1: The paper should be more explicit about the assumptions made for the query execution engine. It is mentioned, that the system is assumed
to use micro-batching. But it seems that these micro batches are also buffered s.t. they can be merged before being passed to the next operator.
The paper mentions that multiple "passes" are made without explaining this concept. It should be clarified if this kind of buffering is done
for every operator. Please also explain how query processing techniques like JIT-compiled query execution or vectorized processing
(using SIMD instructions) work in the target system.
O2: Using performance counters as proposed in the paper requires access to the PMU which is usually not possible from a VM in a hyper scaler. Hence,
the solution seems to require bare-metal setups - please clarify. Also, the number of events that can be collected is typically limited; how are
these limitations considered in the presented architecture?
O3: In addition to O2, performance counters collect system-wide events. Hence, multi-tenant setups with multiple databases (or other applicaions) on

the same host are noisy neighbors for the database workload. It makes it difficult to judge if the database cause the hazards or the noisy neighbor.
Similar for concurrent workload in the database: the parallel execution of statements in the database makes it difficult to attribute hazards to
a particular query. The paper shall explain how this is handled in the system, i.e. why they make the tipping points discussed in section 3.4 static.
O4: Switching the operator implementations as discussed in the paper seems to have impact on physical properties, e.g. sortedness of data or
preserving the tuple order. The adaptation techniques seem to be in conflict wth exploiting these physical propertis in the query optimizer. Please explain.
O5: The paper states in section 3.5. that "the number of tuples to be processed by the hazard-robust algorithm is known without actually processing the tuples.".
Similarly section 4.2. assumes that the output size of grouping is known, but estimating the number of grouping columns is challenging.
How should the precise cardinalities be made available for operators in the middle of a query execution plan? ... where cardinality estimates may be far off.
O6: The paper uses a very small system with very old processor (Ivibridge from 2012). Please conduct the experiments on contemporary hardware as, e.g. the
cache hierarchy has significantly changed in the meantime and cache sizes have increased. It is important to understand if the caims made in the paper
still hold on modern hardware. Similarly, the used Linux kernel and C++ compiler are quite old.
O7: An isolated analysis of operators as done in the experiments is only reasonable when operators fully materialze intermediate results. In a
pipelined query engine (as most state-of-the-art systems use) the code of multiple operators is executed concurrently resulting in interference between
operators.

9. Please rank the three most critical strong points and/or opportunities for improvement that led to your overall evaluation rating, e.g., "{S1, O7, O4}"
O1-O7.

10. Inclusive writing: does the paper follow the SIGMOD inclusive writing guidelines? See https://dbdni.github.io/pages/inclusivewriting.html for more information. If not, please provide suggestions on how to improve the writing of the paper.
I agree

11. D&I compliance
compliant

12. Availability: We have asked authors of all papers to make their code, data, scripts, and notebooks available, if this is possible. You can find this information in the summary of the paper on CMT when clicking on the Paper ID next to the title. In the text box provided below, please provide (a) constructive feedback to the authors about these provided artifacts (if any), (b) how this was useful in your

evaluation of the submission, and (c) what changes are needed in order to make the code/data/scripts/notebooks shareable with the community (if any).

I failed to access the URL provided by the authors

13. Minor remarks. Use this field to describe minor remarks that are not listed as opportunities for improvement, e.g., to highlight typos, formatting problems, or minor technical issues.

D1: Section 3.1. states that hazard-affected algorithms "The defining principle of hazard-affected algorithms is the minimization of work i.e., CPU instructions executed.".
JIT-compiled query execution minimizes the number of CPU instructions executed per query and thereby reduces the risk of hazards in the caches. Please comment on this aspect.
Furthermore it states that hazard-affected algorithms "almost universally do not focus on micro-architectural efficiency, instead, they focus on work efficiency.".
Please give citations on specific implementations to be able to verify these claims.
D2: Please clarify the scope of robustness addressed in the paper. Besides hazards in the hardware there are a number of other factors that may case issues
with the robustness of query processing. For example the paper focuses only on in-memory processing, but, e.g., spill-to-disk processing addresses issues
with insufficient DRAM capacity during processing. Another source of robustness issues are bad cardinality estimates.
D3: The paper argues that memory initialization costs should be O(1), i.e. memory should only be initialized once used. But this may make memory allocations more expensive.
Hence, there is a delicate trade-off needed between lazy memory allocation and the cost of individual allocations. Please discuss this in more breadth.
D4: The discussion of the SELECT operation is somewhat incomplete, as it ignores the important class of "SARG-able" predicates, i.e. filters that can be pushed into
the table access operations which opens the discussion of access path selection.
D5: The probing phase testing alternative execution alorithms may introduce hazards, e.g. by increasing the instruction footprint increasing the number of instruction
cache misses. Please discuss this aspect.
D6: There are many alternatives of implementing the group-by operator. Please be more specific, which implementations are exactly considered, e.g. by citing corresponding
related work. From the paper it is difficult to jude how competitive the implementations in the paper are. For example, related to O4, a group-by implementation could
exploit the sortedness of the input, providing a very robust and simple implementation.
Please explain how the key-payload output produced by the group-by implementation fits into a query engine. Which (generic?) interface is used for exchanging
results between operators.
Notice also that in some cases a robust, static decision can be made. In TPC-H Q1

there can be at most 6 groups; hence a vector can be allocated that can keep all possible groups.
In that case also no probing for alternatives may be needed at runtime. Such strategies are not mentioned in the paper.
Section 5.1 mentions that grouping merges partial results in a sigle pass. This may not always be possible, e.g. when the fanout of partial results is too large. Please explain.
D7: The detection of hazards is based on LLC misses of hash-based grouping and joins, but as discussed in O3 there can be many other sources of LLS misses. This makes the hazard detection
for hash-based algrithms somewhat unreliable with concurrent workload.
D8: Section 4.2. mentions that first the hazard-affected (hash-based) is executed and afterwards the hazard-robust algorithm. At the end of the section it is then mentioned that the probing allows for switching between algorithms. These are contradictory statements - please explain.
D9: For the join operation it remains unclear why only hash-build is analyzed. The issues of hash-building a very similar to the ones for group-by. At same time, hash-probing adds
additional challenges, e.g. LLC misses for the lookup or varying numbers of results per probed tuple.
D10: Please support the claim that TLB misses are the main source of hazards of the radix join with profiling data or citations.
D11: Please provide a formal definition of partitionedness.
D12: The discussion on parallelism in sec 5.1. misses that more parallelism tends to increase the number of LLC misses. This is also confirmed in the analysis of Figure 6c.
D13: In section 5.2., please explain how exactly the lenght of the adaptive period is chosen.
D14: Figure 3b indicates that the sort-based implementation is only relevant for cases with more than 1 million groups.
Likely the tipping point is even higher on modern hardware with significantly larger caches.
D15: The analysis in figure 5a indicates that the switching of algorithms can actually make the system less robust.
D16: For reproducibility, please share which configuration is used for the experiments of Figure 6.
D17: Section 3.4. argues that parallelism does not need to be considered for adaptation because only high degrees of
parallelism lead to high bandwidth utilization. But the analysis in Figure 6a and b shows that already at DOP = 4
the bandwidth is saturated. This likely becomes more severe on modern CPUs with more than 50 CPU cores. Consequently,
parallelism is a critical dimension to consider for robust and adaptive query processing.