Improving Streaming Graph Processing Performance using Input Knowledge

Hyperthreads (**6**)
Sumon Nath
(21Q050007)

I. Summary

Authors identify the problem of input feature agnostic streaming-graph processing techniques. In particular they find the shortcomings of batch reordering, which is a pre-update operation and shows how it degrades performance for graphs with low degree distribution. To mitigate this problem they propose an adaptive version of Batch reordering along with two complementary techniques. These techniques targets the update step in graph processing. They also propose input-aware work aggregation to improve the graph computation efficiency.

II. Details

A. Motivation - Batch reordering

- The problem with baseline updating methods lies in the fact that they are edge-centric and requires locking mechanism for shared memory operations which degrades performance.
- Batch reordering(RO) eliminates the use of locks by using vertex-centric updates.
- The authors show that RO improves performance for workloads with input-batches having high degree distribution, but degrades performance when degree distribution is low. This is due to the added overhead of sorting used to form vertex clusters.
- It is shown that RO spends about 33% of time on graph updates with workloads having low degree distribution compared to 9% in case of baseline.

B. Adaptive Batch Reordering

- Mitigates the problem of batch reordering by taking into account the nature of input batch, particularly the degree distribution.
- Authors propose a new metric called order-lambda clusterable average degree(CAD) which is quantifies the average degree of the top-degree vertices in an input batch. It is used to predict if an input batch is RO-friendly or RO-averse.
- It is an online technique which adaptively reorders input batches depending on the degree distribution.
- According to the nature of the input batch, i.e., RO-friendly or RO-averse, ABR invokes one of the complementary solutions proposed by the authors which will be discussed in the next section.

III. Positives

- ABR + USC leads to significant performance gains in reorder friendly cases.
- ABR + HUA mitigates the problem of reorder-averse input batches and improves performance

Improving Streaming graph processing performance using Input knowledge

→ Input knowledge must be taken into account.
→ Batch-reordering - does not take input into account.
→ Proposed: input aware batch reordering.
  ↳ complementary idea: update graphs dynamically < S/W / H/W.
  - computation improvement.

→ graph update improvement: 4.55× & 2.6×.
  graph compute " : 1.26×

Intro

→ Input sensitivity is critical to update & compute performance.
  - Input batches has varying properties like - degree dist.
    - locality characteristics.

→ Software solution - input-aware adaptive.
  Hardware solution - complementary to sofl. sol.

→ RO (Batch reordering) leads to performance improvement in some but in some benchmarks it leads to perf. degradation (let these be RO averse bench.).
  - RO's update perf. depends on the degree dist. of input batches.
  - proposal: Adaptive RO (ABR) - adaptively decides to Reorder based on the degree distribution.

↳ complementary proposals to ABR: USC for RO-friendly cases (S/W)
  HAU for RO-averse cases (H/W).
  ↳ reduces search ops during edge updates.

--> Problem with RO - averse batches

1. Lock-based updates — sol: map each update task specific core.
2. overheads of update search operations.

sol: dedicated logic to ~~scan~~ scan edge data ca... liner.

Graph computation.

- proposes input-aware work aggregation.

Opportunity: consecutive batches has large overlaps in graph modifications.

sol: for high locality btw batches, aggregates the computation.

Novelty.

ABR - ~~make~~ works on dynamic graphs rather than static graphs
- makes online decisions.
- improves update performance.
- uses a CAD metric to predict if an i/p batch is RO-friendly/averse.

Background

graph processing. update (batch of inputs) → compute

- Why RO? (edge-centric updates)

→ problem with baseline - lock operations. degrades perf.

RO - eliminates lock by using vertex centric updates.

problem with RO - sorting overhead.

trade-off - RO ~~performs~~ improves perf for batches with high degree dist., but degrades where degree dist. is low.

Lower degree leads to high overhead ~~for RO~~.

when RO is applied

7% to 33% time spent on RO.