HBP: Hotness Balanced Partition for Prioritized Iterative Graph Computations

Shufeng Gong, Yanfeng Zhang, Ge Yu
Northeastern University, Shenyang, China
gongsf@stumail.neu.edu.cn, {zhangyf, yuge}@mail.neu.edu.cn

***Abstract*—Existing graph partition methods are designed for round-robin synchronous distributed frameworks. They balance workload without discrimination of vertex importance and fail to consider the characteristics of priority-based scheduling, which may limit the benefit of prioritized graph computation. To accelerate prioritized iterative graph computations, we propose Hotness Balanced Partition (HBP) and a stream-based partition algorithm Pb-HBP. Pb-HBP partitions graph by distributing vertices with discrimination according to their hotness rather than blindly distributing vertices with equal weights, which aims to evenly distribute the hot vertices among workers. Our results show that our proposed partition method outperforms the state-of-the-art partition methods, Fennel and HotGraph. Specifically, Pb-HBP can reduce 40-90% runtime of that by hash partition, 5-75% runtime of that by Fennel, and 22-50% runtime of that by HotGraph.**

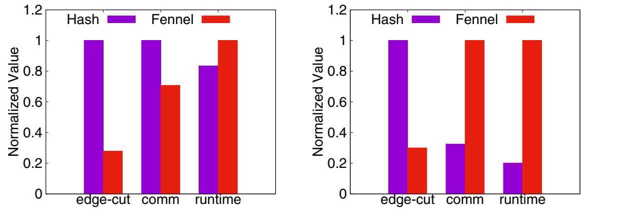
***Index Terms*—Hotness balance partition, Graph partition, Distributed computing, Prioritized Computation**

I. Introduction

To handle massive graphs, distributed graph processing systems partition the graph data into multiple graph partitions and process them on a cluster of workers. During the distributed graph computation process, 1) heavy communication cost between workers due to a large number of edge/vertex cuts and 2) idle workers due to unbalanced workload may exist, which degrades the performance of distributed computing. To reduce the communication cost and idle workers, many research efforts [1], [2] have been put on finding smart graph partition methods, aiming at minimizing connections between partitions and making workload evenly distributed among partitions.

Recently, a number of research works pay attention to asynchronous parallel processing, such as GraphLab [3], GRAPE+ [4], and Maiter [5]. In asynchronous distributed frameworks, the global synchronous barriers are removed. Thus there is no waiting time between workers, and the vertices/edges can be processed at any time. With the elimination of global barriers, a smart scheduling can be used during asynchronous iterations. Recent studies [6], [3] show that some of the vertices do play important roles in determining the final converged outcome. During iterations, high execution priorities are assigned to these important vertices, so that they are processed more frequently than other vertices, which showed much better performance for a class of graph algorithms [6].

Motivation. Prior graph partition algorithms [1], [2] are designed based on the synchronous parallel processing model. In the synchronous parallel model, there is a global synchronous barrier after each iteration (super step). In each super step, each vertex is only processed once, and each edge only delivers one message. Prior graph partition algorithms aim at balancing the workload and minimizing the number of edge/vertex cuts. While in asynchronous frameworks with priority scheduling, prioritized graph computation leads to the discrimination of vertex hotness. Some hot vertices are updated more frequently, and more messages are propagated from these hot vertices. The graph partition should take this property into account for computation effectiveness. Furthermore, due to priority scheduling, the number of updates on each vertex is not consistent, and the number of messages passed through each edge is not consistent. Even with few edge cuts/vertex replicas, it may still result in a large amount of communication cost. For example, if an edge of a high priority vertex is cut, there still can be a large number of messages delivered along this edge. As shown in Fig. 1, the number of edge cuts by Fennel is much less than that by hash partitioning, but the communication cost by Fennel is more than hash partitioning. Thus, a new partition method designed for prioritized graph processing is desired.



(a) 8 partitions (b) 12 partitions

Fig. 1: Hash partition vs. Fennel partition for prioritized graph computation (Pagerank on LiveJournal dataset).

Contribution. In this paper, we propose the idea of *Hotness Balanced Partition (HBP)*, which partitions graph according to vertex's hotness. We first propose three objectives of graph partition for prioritized processing and then propose a heuristic stream-based graph partition algorithm, *SPb-HBP*, which only requires one pass of the graph data. Our experimental results show that our SPb-HBP can reduce 40-50% runtime of that by hash partition, 5-75% runtime of that by Fennel, and 22-31% runtime of that by HotGraph.

II. Preliminaries

Asynchronous DAIC. *Asynchronous Delta-based Accumulative Iterative Computation (asynchronous DAIC)* is a typical asynchronous computation model, which has been applied in many distributed frameworks such as Maiter [5] and its variants [7] [8]. DAIC [5] updates the vertices by accumulating the "changes" between iterations. By DAIC, we can process only the "changes" (Δ_v) of vertices to avoid the negligible updates. Furthermore, we can perform DAIC asynchronously to bypass the high-cost synchronous barriers in heterogeneous distributed environments. In the asynchronous DAIC model, the vertices accumulate the received messages from neighbors and can be updated at any time. After being updated, they send messages to their neighbors immediately.

Prioritized Execution. By using the DAIC model, the computation of any vertex can be performed at any time point. In other words, the vertex updates can be scheduled in any order. The scheduling order is crucial to computation effectiveness. In [5], a heuristic is provided to evaluate the profit of updating a specific vertex. They pick the vertex that can maximize the "change" of graph state as the scheduling candidate. In DAIC, the state values (e.g., the rank value in PageRank) of vertices are monotonically increasing/decreasing, thus a great "change" implies a big move to the fixed point that makes the current state closer to the final state. Therefore, the execution priority of each vertex depends on the amount of its change. Performing computations on the high priority vertices will accelerate the convergence. For the details of DAIC, please refer to [5].

III. Hotness Balance Partition

As discussed in Section I, the traditional partition methods fail to meet the requirements of priority scheduling frameworks. In priority scheduling frameworks, some vertices are given higher execution priority, so they become **hot vertices** in priority scheduling execution systems. The frequency of vertex updates is defined as vertex's **hotness**.

Given a graph with hotness $G = (V, E, H)$ and a partition number k, where V is vertex set, E is the edge set and $H = \{h_v, v \in V\}$ contains the hotness values of all vertices, an edge-cut graph partition aims to find a partition scheme $\mathcal{G} = \{G_1, G_2, \cdots, G_k\}$, where $G_i = (V_i, E_i, H_i)$ is a partition of G. V_i is the set of vertices in G_i such that $V = \bigcup_{i=1}^{k} V_i$ and $V_i \bigcap V_j = \varnothing$, and E_i is the set of edges whose source vertices are in V_i, i.e., $E_i = \{(u, v) | u \in V_i\}$. Then, each partition is assigned to a worker for parallel processing.

A. Hotness Estimation

The precondition of hotness balance partition is that we have obtained the hotness of vertices and the communication cost between two vertices. According to the priority scheduling introduced in Section II, the priority value of a vertex is determined by its Δ_v value ("change"), and Δ_v is collected from its in-neighbors $IN(v)$. Thus, if vertex v has a strong ability to collect Δ_v, vertex v is likely to be with a higher execution priority and is likely to be hot. We estimate the hotness of vertex v as follows.

$$h_v = \sum_{u \in IN(v)} \frac{w_{u,v}}{\sum_{w \in OUT(u)} w_{u,w}} \tag{1}$$

where $w_{u,v}$ is the weight of edge (u, v). If there is no weight on edges, we assume $w_{u,v} = 1$.

During the computation, when a vertex is updated, it will send a message to its outgoing neighbors, so that the number of messages passed through an edge is proportional to its source vertex's update times. In other words, the edge communication cost can be estimated as its source vertex's hotness. Thus, the communication cost of edge (u, v) is estimated as follows.

$$com_{u,v} = h_u. \tag{2}$$

B. Partition Goals

In asynchronous frameworks with priority scheduling, the hotter the vertices are, the more computation resources they need. We should assign computation resources according to vertex hotness. In other words, we should balance the hotness among partitions so that the same amount of computation resources are assigned to each partition, and we call this kind of partition as Hotness Balance Partition (HBP). Therefore, *our partition scheme aims to assign the same amount of vertex hotness to workers, which is the first goal.*

The assumption in the above analysis is that a global priority scheduler is used. However, in practice, a global scheduler is very expensive in large scale clusters. Thus, we use a local scheduler that runs on each worker to simulate the global scheduler, i.e., the priority scheduling works in parallel and on a per-worker basis. Note that, it is possible that a few super hot vertices are in a partition and a lot of cool vertices are in another partition, though the sums of hotness values are equal between partitions. Local scheduling will bring troubles in such a case, since a worker might always schedule cool vertices. Thus, to maximize the computation effectiveness with local scheduler, the local hotness distribution on each worker should be consistent with the global hotness distribution.

We use hotness histogram to describe hotness distribution. The vertex hotness values are divided into z continuous non-overlap intervals. Each interval corresponds to a bin $\mathcal{H}_j$ holding the vertices whose hotness values are in the jth interval, then we have $\max\{h_v | v \in \mathcal{H}_{j-1}\} < \min\{h_v | v \in \mathcal{H}_j\}$ and $\max\{h_v | v \in \mathcal{H}_j\} < \min\{h_v | v \in \mathcal{H}_{j+1}\}$. The height of the histogram bar is the sum of hotness values of vertices in each bin, i.e., $\sum_{v \in \mathcal{H}_j} h_v$. Then the probability distribution of hotness histogram in original G is defined as $P(\mathcal{H}_j) = \frac{\sum_{v \in \mathcal{H}_j} h_v}{\sum_{v \in V} h_v}$. Similarly, the probability distribution of hotness histogram in partition G_i is defined as $P_i(\mathcal{H}_j) = \frac{\sum_{v \in \mathcal{H}_{ji}} h_v}{\sum_{v \in V_i} h_v}$ where $\mathcal{H}_{ji} = \mathcal{H}_j \cap V_i$. We use **Hotness Jensen-Shannon distance (HJS)** [9] to measure the variance between hotness distributions of partition G_i and graph G as follows.

$$\begin{aligned} HJS(P || P_i) = & \frac{1}{2} \Bigg[\sum_{j=1}^{z} P(\mathcal{H}_j) log \left(\frac{P(\mathcal{H}_j)}{\frac{P(\mathcal{H}_j) + P_i(\mathcal{H}_j)}{2}} \right) + \\ & \sum_{j=1}^{z} P_i(\mathcal{H}_j) log \left(\frac{P_i(\mathcal{H}_j)}{\frac{P_i(\mathcal{H}_j) + P(\mathcal{H}_j)}{2}} \right) \Bigg] \end{aligned} \tag{3}$$

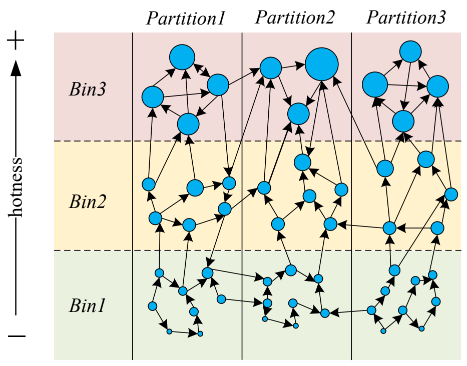


Fig. 2: Pb-HBP graph partition.

Therefore, with a fixed z, *our second goal is to minimize the HJS distance between each partition and the original graph.*

As known in distributed prioritized iterative computation, network communication has a great impact on the performance of distributed computing. Heavy network traffic may lead to message blocking when sending the important messages that help accelerate prioritized computation. Therefore, *our third goal is to minimize the communication cost between partitions.*

C. Per-Bin Hotness Balanced Partition

According to the above discussion, there are three objectives we want to minimize in HBP. We observe that if each bin is partitioned with balanced hotness, we can achieve both of the first and second goals at the same time.

For a given graph $G(V, E, H)$ with z hotness interval bins $H\{\mathcal{H}_1, \cdots, \mathcal{H}_z\}$, if we partition each bin $\mathcal{H}_j$ into k hotness-balanced bin partitions $\{\mathcal{H}_{j1}, \cdots, \mathcal{H}_{jk}\}$ and merge the bin partitions that have the same partition id into a graph partition $V_i = \bigcup_{j=1}^{z} \mathcal{H}_{ji}$, the hotness of each graph partition is balanced, $\sum_{v \in V_i} h_v = \frac{\sum_{v \in V} h_v}{k}$, and the HJS distance between G and G_i is minimized, $HJS(P||P_i) = 0$.

Based on the above discussion, the first two HBP goals can be combined into one: balancing the hotness of each bin partition within each bin

$$\min \sum_{i=1}^{k} \left| \sum_{v \in \mathcal{H}_{ji}} h_v - \frac{\sum_{v \in \mathcal{H}_j} h_v}{k} \right| \tag{4}$$

We call this variant of HBP as Per-Bin Hotness Balanced Partition (Pb-HBP). The idea of Pb-HBP is illustrated in Figure 2. The vertices in different hotness bins (with different levels of hotness) are partitioned separately, at the same time the communication cost between partitions is minimized.

D. Stream-based Heuristic Algorithm

Based on the idea of Pb-HBP, we derive a heuristic streaming algorithm by greedily assigning vertices to partitions. The greedy assignment is performed as follows. Given that the current vertex partitions are $\{V_1, V_2, \ldots, V_k\}$, a newly scanned vertex is assigned to a partition such that the goals of Pb-HBP are most likely to be satisfied. Suppose our cost function to be *minimized* is f, where $f(V_i \cup \{v\})$ is the cost when assigning v to V_i. Vertex v is assigned to partition i such that

$$f(V_i \cup \{v\}) \le f(V_j \cup \{v\}), \forall 1 \le j \le k. \tag{5}$$

The greedy vertex assignment requires only one pass of the graph data, which is quite efficient for the large graph. However, the key is to design a cost function that describes the goals of Pb-HBP.

There are two goals of Pb-HBP, which describe the per-bin hotness balancing requirement as shown in Equation (4) and the communication minimization requirement. We unify them in a single cost function and greedily choose the best partition i that results in the least cost.

$$\begin{aligned} i = \mathop{\arg\min}_{i:\{\boldsymbol{h}_{ji} \le \tau \frac{\sum_{v \in \mathcal{H}_j} h_v}{k}\}} & \alpha \cdot \left((\boldsymbol{h}_{ji} + h_v)^{\gamma} - \boldsymbol{h}_{ji}^{\gamma} \right) \\ & + (1-\alpha) \cdot \Big(\sum_{u \notin V_i} com_{u,v} + \sum_{w \notin V_i} com_{v,w} \Big) \end{aligned} \tag{6}$$

where $0 \le \alpha \le 1$, $\tau > 1$, $\gamma > 1$, and $\boldsymbol{h}_{ji} = \sum_{v \in \mathcal{H}_{ji}} h_v$ denotes the current sum of hotness values of partition G_i in the jth bin. In this cost function, the first part describes the imbalance cost, while the second part describes the communication cost. Parameter α controls the weight of imbalance cost and communication cost. The imbalance cost is minimized when $H_{ji} = \frac{\sum_{i=1}^{k} H_{ji}}{k}$ for each i. Parameter τ is a small constant that defines the tolerance to hotness imbalance. Parameter γ controls how much preference to assign vertex to low hotness partitions since adding vertex to high hotness partitions may increase the risk of generating over-hot partitions, where larger γ results in higher cost when assigning vertex to higher hotness partitions.

Algorithm 1 Streamed Per-Bin Hotness Balanced Partition

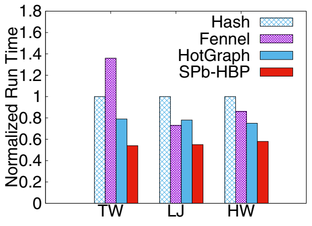
```
Input: Graph $G(V, E)$, number of partitions $k$, number of intervals $z$;
Output: Graph partitions $\{V_1, \ldots, V_k\}$;
 1: Estimate hotness values $H$ of all vertices;
 2: Init $\mathcal{H}_j$ based on histogram of $H$, $1 \le j \le z$;
 3: Init $V_i = \emptyset$, $1 \le i \le k$;
 4: Init $\boldsymbol{h}_{ji} = 0$, $1 \le i \le k$, $1 \le j \le z$;
 5: for each $v$ in $V$ do
 6:     Find $\mathcal{H}_j$ where $v$ resides;
 7:     for each $V_i$ do
 8:         if $\boldsymbol{h}_{ji} > \tau \cdot \frac{\sum_{v \in \mathcal{H}_j} h_v}{k}$ then
 9:             $c_i = +\infty$;
10:         end if
11:         $c_i = \alpha \cdot \left( (\boldsymbol{h}_{ji} + h_v)^{\gamma} - \boldsymbol{h}_{ji}^{\gamma} \right) + (1 - \alpha) \cdot \left( \sum_{u \notin V_i} com_{u,v} + \sum_{w \notin V_i} com_{v,w} \right)$;
12:     end for
13:     $i = \arg\min_i c_i$;
14:     $V_i = V_i \cup v$;
15:     $\boldsymbol{h}_{ji} = \boldsymbol{h}_{ji} + h_v$;
16: end for
```

Based on the cost function depicted in Equation (6), we propose Stream-based Pb-HBP (SPb-HBP), which sequentially read each vertex v and assign it to the partition i that results in the minimum cost. The detail of SPb-HBP is shown in Algorithm 1. We first estimate the hotness values H of all vertices (Line 1) and initialize z bins $\{\mathcal{H}_1, \ldots, \mathcal{H}_z\}$ based on the hotness histogram of H (Line 2). Line 3 initializes vertex partitions, and Line 4 initializes the sum of hotness values in each bin partition. We then perform the vertex assignment operation. We measure the increased cost when assigning a vertex to each partition (Line 11). If a bin partition is over-hot, it will not be assigned with the new vertex (Line 8 - 10). The partition that results in the minimum increased cost will be selected for assigning the vertex (Line 14), and the bin partition's hotness is correspondingly increased (Line 15).

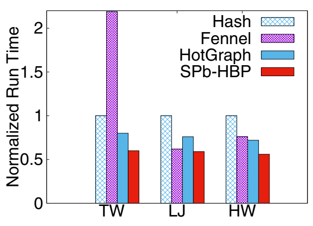
IV. Experimental Evaluation

Preparation. We first partition input graphs as a preprocessing step and then distributively run prioritized graph algorithms on Maiter [5], an implementation of the asynchronous DAIC framework and supports priority scheduling. We compare our proposed SPb-HBP with the state-of-the-art stream partition methods Fennel [2], HotGraph [10] and Hash partition. We perform the evaluation on two representative prioritized graph algorithms, PageRank and Penalized Hitting Probability (PH-P). We select three different real graphs with different types, Twitter (TW), Hollywood (HW) and LiveJournal (LJ). Our experiments are conducted on a cluster of machines on Alibaba Cloud, which consists of 4 ecs.cs.large nodes with one additional node as Master.

Compare with Other Partition Methods. We compare SPb-HBP with other state-of-the-art partitioning methods. In SPb-HBP, we simply set the number of bins $z = 2$. We partition the input graph into 4 partitions by using different partition methods. We then use 5 workers (1 master and 4 slaves) to perform PageRank and PHP algorithms on these partitions (DSPb-HBP runs distributed partition on 4 nodes). Fig. 3 and 4 shows the normalized runtime and communication cost on three graphs (TW, HW, and LJ). It is noticeable that the runtime and communication cost are resulted from graph algorithm's computation but not graph partitioning. We can see that SPb-HBP always results in shorter runtime than other methods no matter for PageRank or PHP computation. SPb-HBP can reduce 40-90% runtime of that by hash partition, 5-75% runtime of that by Fennel, and 22-50% runtime of that by HotGraph.

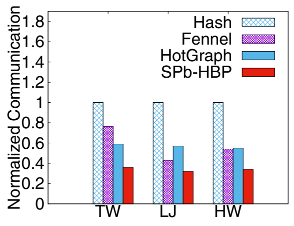


(a) PageRank

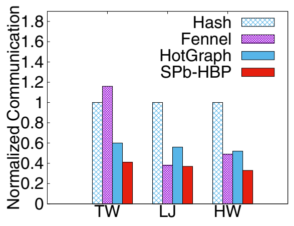


(b) PHP

Fig. 3: Runtime comparison.



(a) PageRank



(b) PHP

Fig. 4: Communication cost comparison.

V. Conclusion

The traditional k-balanced graph partitioning fails to work in prioritized asynchronous iterative frameworks. In this paper, we propose a novel graph partition idea, Hotness Balanced Partition (HBP), tailored to prioritized scheduling frameworks. We propose a Stream-based Per-bin Hotness Balance Partition (SPb-HBP) algorithm to efficiently partition graphs. Our results show that our proposed graph partition scheme can greatly improve the performance of prioritized graph computations and at the same time is quite efficient to partition large-scale graphs.

Acknowledgment

This work was partially supported by National Key R&D Program of China (2018YFB1003404), National Natural Science Foundation of China (61672141, U1811261) and Fundamental Research Funds for the Central Universities (N181605017, N181604016, N171606005).

References

[1] F. Petroni, L. Querzoni, K. Daudjee, S. Kamali, and G. Iacoboni, "Hdrf:stream-based partitioning for power-law graphs," in *CIKM2015*, 2015, pp. 243–252.
[2] C. Tsourakakis, C. Gkantsidis, B. Radunovic, and M. Vojnovic, "Fennel:streaming graph partitioning for massive scale graphs," in *WSDM2014*, 2014, pp. 333–342.
[3] Y. Low, D. Bickson, J. Gonzalez, C. Guestrin, A. Kyrola, and J. M. Hellerstein, "Distributed graphlab: a framework for machine learning and data mining in the cloud," in *VLDB2012*, 2012, pp. 716–727.
[4] W. Fan, P. Lu, X. Luo, J. Xu, Q. Yin, W. Yu, and R. Xu, "Adaptive asynchronous parallelization of graph algorithms," in *SIGMOD2018*, 2018, pp. 1141–1156.
[5] Y. Zhang, Q. Gao, L. Gao, and C. Wang, "Maiter: An asynchronous graph processing framework for delta-based accumulative iterative computation," *IEEE TPDS*, vol. 25, no. 8, pp. 2091–2100, 2014.
[6] ——, "Priter: a distributed framework for prioritized iterative computations," in *SOCC2011*, 2011, pp. 1–13.
[7] J. Yin and L. Gao, "Scalable distributed belief propagation with prioritized block updates," in *CIKM2014*, 2014, pp. 1209–1218.
[8] Z. Wang, L. Gao, Y. Gu, Y. Bao, and G. Yu, "A fault-tolerant framework for asynchronous iterative computations in cloud environments," in *SOCC2016*, 2016, pp. 71–83.
[9] D. M. Endres and J. E. Schindelin, "A new metric for probability distributions," *IEEE Transactions on Information theory*, vol. 49, no. 7, pp. 1858–1860, 2003.
[10] Y. Zhang, X. Liao, H. Jin, L. Gu, G. Tan, and B. B. Zhou, "Hotgraph: Efficient asynchronous processing for real-world graphs," *IEEE TOC*, vol. 66, no. 5, pp. 799–809, 2017.