Future Generation Computer Systems 104 (2020) 15–31

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

H^{2}Pregel : A partition-based hybrid hierarchical graph computation approach

Qiang Liu, XiaoShe Dong, Heng Chen *, Xingjun Zhang

School of Electronics and Information Engineering, Xi'an Jiaotong University, Xi'an 710049, China

ARTICLE INFO

Article history:
Received 20 March 2018
Received in revised form 2 August 2019
Accepted 10 September 2019
Available online 16 September 2019

Keywords:
Graph computing
Hybrid computation
BSP model
Hierarchical

ABSTRACT

A partition-based hybrid hierarchical graph computation approach, called H^{2}Pregel is proposed to address the redundant supersteps and inefficient computation problems due to low access locality. The H^{2}Pregel preprocesses the input graph through a distributed recode algorithm to ensure the continuity and sequence of vertex ids, then employs a hybrid approach to combine the advantages of both synchronous and asynchronous models, and hierarchically computes the high proportion of interior messages generated by high quality partition algorithms. Moreover, H^{2}Pregel leverages configurable parallel threads to accelerate local computation by "sub-supersteps", and employs an exterior messages stealing optimization to avoid extra communication overheads between tasks. We implemented H^{2}Pregel on Giraph, a classic open source system based on Pregel. The evaluation results on large-scale graphs show that, compared with Pregel in three partition algorithms, H^{2}Pregel can achieve average speedups by 1.12–4.52 times and decrease average communication messages by 23.5%-55.5%, and average supersteps by 15.8%-82.0%.

1. Introduction

As the era of "Big Data" is coming, the processing of large-scale graph-structured data has becoming increasingly important in a wide range of domains such as social networks, recommendation systems and data analytics. To address these challenges, a number of graph analytic systems have emerged, e.g., Pregel [1], GraphLab [2], GraphX [3] and GridGraph [4]. Most existing graph processing systems employ synchronous or asynchronous modes, and usually follow the "think like a vertex (TLAV)" philosophy, which address the iterative graph computation problems by vertex centric or edge centric approaches.

However, these distributed systems are not suited for all purposes, and sometimes even suffer from low access locality and poor performance [5] due to different realistic characteristics, like power-law distribution [6] or long hops [7]. Moreover, some graph applications, like BP neutral network often needs thousands of iterations [8], which result serious performance degradation.

To overcome these shortcomings, some researchers proposed a hybrid approach to leverage the advantages of both synchronous and asynchronous models, such as Mizan [9], PowerSwitch [10] and GRACE [11], which follows a heuristic workload migration algorithm based on runtime metrics, or supporting for both two modes by separating computation logic and scheduling orders. However, these dynamic swapping systems basically follow an "Afterwards Controlling" approach, which lacks considerations of access locality, and are often demonstrated not worth the high overhead [12]. Worse even, the flexibility for mode switching can also confuse programmers in selection, which may lead to infinite running for graph algorithms.

Moreover, in response to TLAV limitations and low access locality, some researchers adopt a coarse-grained granularity for storage and computation [5,13,14], which is modest between vertex and graph. Some works employ high quality partition algorithms [15], which have a small edges cut and similar communication workloads between tasks, as a "Beforehand Processing" approach, can efficiently improve unbalanced workload, especially for some slow converged applications or graphs with "supervertex". However, existing graph processing systems are primarily based on the random hash based partitioning algorithm [14], which partitions original graph into disconnected components based on vertex or edge ids, and causes large redundant communication across partitions. Moreover, a well-balanced graph partition by high quality underlying algorithms can even lead to a decrease of the overall performance in existing systems [15].

This paper proposes a novel partition-based hybrid hierarchical graph computation approach, called H^{2}Pregel, which leverages the advantages of both synchronous and asynchronous mode, and performs local computations on interior edges to avoid inefficient computations and network traffic between tasks. Moreover, H^{2}Pregel leverages the high proportion of interior messages

* Corresponding author.
E-mail address: hengchen@xjtu.edu.cn (H. Chen).

https://doi.org/10.1016/j.future.2019.09.021
0167-739X/

hierarchically by sub-supersteps to accelerate the convergence. We have built H^{2}Pregel on Giraph, a classic open source system based on Pregel, and constructed an implementation system called H^{2}Giraph. The evaluation results on large-scale graphs show that, compared with Pregel in three partition algorithms, H^{2}Pregel can achieve average speedups by 1.12–4.52 times and can reduce average communication messages by 23.5%–55.5%, and can reduce average supersteps by 15.8%–82.0%.

The remaining paper is organized as follows. Section 2 describes the related works on graph computation. Section 3 introduces the motivation of H^{2}Pregel. The computation abstraction and system implementation details are described in Sections 4 and 5, respectively. Section 6 evaluates the experiments results. Finally, we conclude this paper in Section 7.

2. Related works

Graph theory has been applied in a wide range of fields, such as geometry, computer science, decision systems and information science. Some researchers employ graph-theoretic approaches as a catalyst for novel concepts and cloud applications, such as cybernetics in social networks [16], network function virtualization in edge computing [17] and multi-objective scheduling for scientific workflow [18]. Moreover, some works extend graph theory into decision-making systems by a mathematical modeling approach, such as rough fuzzy digraphs [19], intuitionistic fuzzy graphs [20] and m-polar fuzzy matroids [21].

In general, existing large-scale graph processing systems can be categorized into two classes: synchronous and asynchronous [12]. Most synchronous approaches expresses the computation as a sequence of synchronous iterations, which follows the Mapreduce-style [22], like PEGASUS [23] and Hadi [24], or BSP (Bulk Synchronous Parallel) [25] based systems, such as Pregel, Hama [26], GPS [27] and Giraph [28] that applies "Scatter–Gather (SG)" [29] programming model, force all tasks join the same superstep and suffer from the high, fixed cost of the global synchronization barrier until a global convergence is reached, or the number of supersteps exceeds the threshold. These synchronous approaches can reduce overheads caused by consistency and data race, but also introduces redundant iterations and workload imbalance problems, especially in some sequential scenarios, each iteration lasts as long as the slowest task and leads severe performance degradation, which are the so-called "data skew" or "stragglers" problems [30].

In contrast, asynchronous approach systems eliminate the global barrier and the "straggler" problem, but come at the expense of added complexity from distributed scheduling and locking mechanism to maintain data consistency. For instance, GraphLab implements a shared memory model and takes a fine-grained distributed lock to avoid concurrent conflicts between vertex and adjacent neighbors, but the additional complexity in consistency model and nondeterministic execution procedure also result difficulties for following programmers [31].

Some researchers propose a hybrid approach to leverage the advantages of both two execution models. PowerSwitch [10] constantly collects runtime execution statistics and leverages a set of heuristics to predict future performance and determine which mode could be profitable. GRACE [11] decouples the dependency between application logic and execution policies, and provides a carefully designed and implemented parallel execution engine for both synchronous and user specified built-in asynchronous execution policies. Mizan [9] achieves efficient dynamic load balancing by fine-grained vertex migration and distributed hash table (DHT) for looking up service and better adapt to changes in computing needs. Finishing Computation Serially (FCS) [5] monitors the size of the "active" graph at the end of execution, if it becomes small enough to fit in the memory of a single machine, an aggregation operation is triggered to collect the active graph to the master and performs a serial computation. FCS can eliminate almost 20%–60% supersteps. ExPregel [32] performs local synchronization for vertices of intra-node tasks to reduce communication between machines. GiraphUC [33] proposes a barrierless asynchronous parallel (BAP) model, which reduces both messages staleness and global synchronization.

Some researchers propose a coarse-grained granularity approach to provide a higher level abstraction and reduce the number of messages transmitted and buffered across supersteps. Storing Edges At Subvertices (SEAS) [5] optimizes computation by merging sets of sub-vertices to form "super vertices", and incurs communication between sub-vertices and super vertices instead of sending adjacency lists. Giraph++ [13] provides a subgraph-centric framework and follows a "think like a graph" philosophy, which dramatically outperforms "think like a vertex" framework by orders of magnitude both in computation and message traffic. PathGraph [14] partitions the graph at a "path" level granularity, with each partition represented as a forward tree and a reverse tree. Moreover, PathGraph follows a path-centric model in both computation and storage, and "Scatter–Gather (SG)" programming model, which utilizes locality through reduced memory usage and efficient caching.

Some researchers consider improving the performance through high quality partition algorithms. PAGE [15] proposes a partition aware graph computation engine to efficiently support computation tasks with different partitioning qualities, and handle both local and remote incoming messages adaptively based on partition-related runtime statistics. Xu [34] proposes a novel adaptive streaming graph partitioning approach to cope with heterogeneous environment, which formally models the heterogeneous computing environment with the consideration of the unbalance of computing and communicating ability for each node, and offers a new graph partitioning objective function to minimize the job execution time.

Our works is similar to the works in [13,32], but with some significant differences. Most similar papers mainly follow a hybrid approach, while this paper proposes a hybrid and hierarchical approach, which introduces the factor of partition algorithm and employs a slight synchronization between intra-task parallel threads instead of high synchronization overheads between tasks. Meanwhile, we also propose a user-defined option to control the number of local processing threads. Moreover, H^{2}Pregel provides better reusability that mostly retains the programming interface and fault tolerance model of Pregel, so that traditional vertex-based graph algorithms can be trivially ported, which avoids the complexity in writing subgraph-based algorithms.

3. Problem statement

3.1. Redundant supersteps

Fig. 1 shows the number of active vertices in each superstep of Connected Components (CC) algorithm on Web-BerkStan dataset, which totally costs 704 supersteps. As shown in Fig. 1, the number of active vertices exhibits a power-law distribution, and more than 97.6% vertices have already converged after 20 supersteps, which take only 2.84% of total number. Generally speaking, 2.4% vertices cost more than 97.2% supersteps, which lead significant redundant synchronization overheads and low efficiency computation.