Future Generation Computer Systems 89 (2018) 698–712

Contents lists available at ScienceDirect

Future Generation Computer Systems

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

A highly cost-effective task scheduling strategy for very large graph computation

Yongli Cheng[a,b,*], Fang Wang[b], Hong Jiang[c], Yu Hua[b], Dan Feng[b], Yunxiang Wu[b], Tingwei Zhu[b], Wenzhong Guo[a,d,e]

[a] College of Mathematics and Computer Science, FuZhou University, China
[b] School of Computer, Huazhong University of Science and Technology, WuHan, China
[c] Department of Computer Science & Engineering, University of Texas at Arlington, USA
[d] Key Lab of Spatial Data Mining & Info. Sharing, Min. of Education, Fuzhou, China
[e] Fujian Provincial Key Laboratory of Network Computing and Intelligent Information Processing, Fuzhou University, Fuzhou, China

HIGHLIGHTS

- The reduced the number of supersteps due to the high convergence speed.
- The eliminated synchronization overheads due to the pipeline-based task scheduling.
- The high flexibility due to the network ecosystem friendliness.

ARTICLE INFO

Article history:
Received 18 October 2017
Received in revised form 19 June 2018
Accepted 10 July 2018
Available online 18 July 2018

Keywords:
Graph computation
Cost-effectiveness
Very large graphs

ABSTRACT

Existing distributed graph-processing frameworks, e.g., Pregel, GPS and Giraph, handle large-scale graphs in the memory of clusters built of commodity compute nodes for better scalability and performance. While capable of scaling out according to the size of graphs up to thousands of compute nodes, for graphs beyond a certain size, these frameworks would usually require investments of machines that are either beyond the financial capability of or unprofitable for most small and medium-sized organizations, making the deployment of their large-scale graph-computing jobs difficult if not impossible. At the other end of the spectrum of graph-processing frameworks research, the single-node disk-based graph-computing frameworks, such as GraphChi and XStream, handle large-scale graphs on just one commodity computer, leading to high efficiency in the use of hardware but at the cost of low user performance and limited scalability. Motivated by this dichotomy, in this paper we propose a pipeline-based task scheduling strategy with high cost-effectiveness. We use this scheduling strategy to design and implement a distributed disk-based graph-processing framework, called DD-Graph, that can process very large graphs with trillions of edges on a small cluster while achieving the high performance of existing distributed in-memory graph-processing frameworks. The evaluation of DD-Graph prototype, driven by very large graph datasets, shows that it saves 73% of GPS' hardware costs while running 1.34x faster than GPS.

1. Introduction

In recent years, there has been a recent surge of interest in extracting valuable information from graph structures in both academia and industry. Today, in many problem domains that require graph computation, the graphs are becoming larger than ever before. These graphs, such as social networks, can have billions of vertices and up to trillions of edges [1,2].

Due to the fact that many graph algorithms exhibit irregular access patterns [3], most graph-processing frameworks require that the graphs fit entirely in memory [4–8], necessitating either a supercomputer or a very large cluster to process very large graphs [4,9,10]. The excessive investment of a very large cluster or a supercomputer discourages and possibly prevents many small and medium-sized organizations from deploying their large-scale graph-computing jobs.

In order to reduce hardware costs and improve efficiency, several graph-processing frameworks, e.g., GraphChi [11] and XStream [12], have been proposed to process graphs with billions of edges on just one commodity computer, by relying on secondary storage [11,12]. However, the performance of these frameworks is limited by the limited secondary storage bandwidth of a single compute node [13] and the significant difference in the access

* Correspondence to: 2 Xue Yuan Road, University Town, Fuzhou, Fujian 350108.
E-mail address: chengyongli@hust.edu.cn (Y. Cheng).

https://doi.org/10.1016/j.future.2018.07.010
0167-739X/© 2018 Elsevier B.V. All rights reserved.

speeds between secondary storage and main-memory [14]. Furthermore, the limited amount of storage of a single commodity computer can potentially limit the scale of the processed graphs, since graphs continue to grow rapidly in size [10].

The key difference between the in-memory graph-processing frameworks and single-node secondary storage based graph-processing frameworks lies in the trade-off between the hardware cost and performance, with the former trading off hardware cost for performance while the latter doing the exact opposite. In this paper, we propose a *distributed disk-based graph-processing framework*, called DD-Graph that has the salient feature of both the low hardware cost and high performance.

Distributed disk-based graph-processing frameworks target efficient big graph processing with a small cluster of commodity PCs that is affordable to most common users. However, it is challenging to design an efficient distributed disk-based graph-processing system since the total resources of a small cluster are limited. This is also evidenced by most recent research results, such as Chaos and Pregelix [10,15]. GraphD [16] is proposed recently to hide the disk I/O cost by overlapping the disk I/O with the communication inside each compute node of the small cluster, improving the utilization of the resources in each compute node. However, this solution is efficient only for the network ecosystem with low bandwidth [16].

DD-Graph is different from several recently proposed distributed disk-based graph-processing frameworks [10,15–17], which improves the overall runtime of the graph-computing job significantly by using the pipeline-based task scheduling strategy that provides three key features as follows.

1. *High convergence speed*. By pipelining the tasks of the graph-computing job, our scheduling strategy can reduce the number of supersteps of the graph-computing job significantly. Since when computation Ci of task i has done, the computation Ci +1 of task i +1 can use the computation result immediately. A task is defined as the execution process of a partition in one superstep. This is important since, in distributed disk-based graph-processing frameworks, the runtime of each superstep is usually time-consuming when processing a very large graph.
2. *Eliminated synchronization overheads*. During the execution process of pipeline-based task scheduling strategy, there is not a clear division between any two consecutive supersteps. Furthermore, when the stage of loading partition has finished, the compute node can immediately execute the computation stage of task t currently being launched if the computation stage of task $t-1$ has finished and the computation result of task $t-1$ has arrived, eliminating the costly synchronization overheads.
3. *Network ecosystem friendliness*. The performance of Chaos [10] and Pregelix [15] relies heavily on the assumption that network bandwidth far outstrips storage bandwidth. At the other end of the spectrum of distributed disk-based graph-processing frameworks research, GraphD [16] is designed for the low-bandwidth networks. Thus, the disk I/O can be hidden by the communication. However, DD-Graph is network ecosystem friendly. It can hide almost the entire communication time and the full disk I/O time by overlapping the disk I/O and communication of each compute node with the computations of other compute nodes, if a cluster with an appropriate scale is available. DD-Graph also provides two optimizations to further improve the communication and the disk I/O efficiencies, as detailed in Sections 4.1 and 4.3.

The rest of the paper is structured as follows. Background and motivation are presented in Section 2. Section 3 introduces the pipeline-based task scheduling strategy. System design and optimization are presented in Section 4. Experimental evaluations of the DD-Graph prototype are presented in Section 5. We discuss the related work in Section 6 and conclude the paper in Section 7.

2. Background and motivation

In this section, we first present a brief introduction to distributed in-memory graph-processing frameworks. We then discuss the single-node disk-based graph frameworks. The significantly different characteristic between the two types of graph-processing frameworks help motivate us to propose a distributed disk-based graph-processing framework that has the characteristics of both the low hardware cost and high performance.

2.1. Pregel-like graph-processing frameworks

Pregel-like graph-processing frameworks, such as Pregel [4], GPS [5] and Giraph,[1] adopt a vertex-centric computation model. In these frameworks, an input graph is divided into partitions, each of which resides in the memory of a compute node during the execution process. A graph-computing job proceeds in a sequence of iterations (supersteps), which terminates when all vertices vote to stop the computation. In each iteration, a user-defined **vertex-program(v)** function is invoked for each vertex **v**, conceptually in parallel. Inside the **vertex-program(v)** function, the state of vertex **v** is updated by using the old state and the incoming messages that were sent to **v** in the previous iteration; then **vertex-program(v)** function generates messages based on the new state of **v** and sends them to **v**'s neighbors. At the end of each iteration, all compute nodes synchronize to ensure that all messages have been received successfully.

Due to the "think-like-a-vertex" philosophy, these frameworks are very user friendly for coding and debugging parallel graph algorithms. Furthermore, in recent years, several in-memory distributed graph-processing framework, such as Gemini [18] and BlitzG [19], have improved the system performance significantly by overcoming the performance bottleneck caused by the fine-grained and high-frequency communication. These frameworks are very useful for high-end users to deploy their large-scale and time-sensitive graph-computing job. For high-end users, such as big companies and banks, they usually have a large number of compute nodes necessitated by the combined requirement of large aggregate memory space with respect to the size of the large-scale graph and high system performance. However, high performance is not always the first thing. For most small to medium size companies and most research institutes, they usually need to meet the increasing rapidly needs of processing large-scale graph computations in a reasonable amount of time. In this case, a very large cluster is not easily accessible in most small to medium size companies and most research institutes, possibly preventing many organizations from deploying their large-scale graph-computing jobs.

2.2. Single-node graph-processing frameworks

In recent years, several graph-processing frameworks, such as GraphChi [11] and Xstream [12], have been proposed to process large-scale graphs on just a single commodity computer. However, due to the costly I/O latency, they routinely suffer from poor performance, leading to a long time used by users to wait for the graph-computing results. In order to address this problem, some techniques are proposed to reduce the I/O latency. GridGraph [20] breaks graphs into 1D-partitioned vertex chunks and

[1] Apache Giraph: http://giraph.apache.org.