J. Parallel Distrib. Comput. ▮ (▮▮▮▮) ▮▮▮–▮▮▮

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

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

Accelerating breadth-first graph search on a single server by dynamic edge trimming

Guangyan Zhang *, Shuhan Cheng, Jiwu Shu, Qingda Hu, Weimin Zheng

Department of Computer Science and Technology, Tsinghua University, Beijing, China

HIGHLIGHTS

- A new approach that performs breadth-first graph search efficiently on a single server.
- The impact of graph diameters (e.g., web graphs and social graphs) on the performance of breadth-first graph search.
- The benefit of dynamic edge trimming: lower memory consumption, fewer computing tasks, and fewer disk I/Os.
- A follow-up improvement according to the feedbacks we got during the presentation of FastBFS on IPDPS 2016.

ARTICLE INFO

Article history:
Received 28 July 2017
Accepted 17 September 2017
Available online xxxx

Keywords:
Graph computing
In-memory computing
I/O optimization
BFS

ABSTRACT

Breadth-first graph search (a.k.a., BFS) is one of the typical in-memory computing models with complicated and frequent memory accesses. Existing single-server graph computing systems fail to take advantage of access pattern of BFS for performance optimization, hence suffering from a lot of extra memory latencies due to accessing no longer useful data elements of a big graph as well as wasting plenty of computing resources for processing them. In this article, we propose FastBFS, a new approach that accelerates breadth-first graph search on a single server by leverage of the access pattern during iterating over a big graph. First, FastBFS uses an edge-centric graph processing model to obtain the high bandwidth of sequential memory and/or disk access without expensive data preprocessing. Second, with a dynamic and asynchronous trimming mechanism, FastBFS can efficiently reduce the size of a big graph by eliminating useless edges in parallel with the computation. Third, FastBFS schedules I/O streams efficiently and can attain greater parallelism if an additional disk is available. We implement FastBFS by modifying the X-Stream system developed by EPFL. Our experimental results show that FastBFS can attain speedups of up to 7.9× and 10.4× in the computing speed compared with X-stream and GraphChi respectively. With an additional disk, the performance can be further improved.

1. Introduction

1.1. Motivation

Many large scale applications use graph structure to represent their data, such as web search, social networks [33], chemistry [34] and so on. In order to analyze those data efficiently, the research community is paying more and more attention to big graph computing. Many big graph computing approaches partition the storage and computation over a cluster made up of plenty of machines. There are many distributed graph processing systems such as Pregel [22], PowerGraph [9], GraphX [10] and so on. These systems suffer from problems such as load imbalance [14], high fault tolerance costs [32], etc. Recent studies on single server graph systems [11,17,19,26,35,36] indicate that with deliberately designed data representation and scheduling strategies, a single machine can solve very big graph problems with competitive performance. Moreover, with faster single server computing, same graph problems can be solved with fewer machines in a shorter time. Among big graph applications, breadth-first search (a.k.a., BFS) is a fundamental graph traversal algorithm. BFS is one of the building blocks of many graph analysis algorithms, e.g., shortest-paths, hence attracts lots of efforts on optimizing its performance on a variety of architectures [1,25]. Moreover, BFS is used as a representative graph traversal kernel [27] in the Graph500 benchmark suite [24], a metric for evaluating supercomputer performance.

BFS computation along with other graph applications share the same poor locality problem, which results in expensive memory accesses in shared-memory systems and frequent communications in clusters [2]. The lack of locality tends to have more significant influence for BFS in its low computation density phases, in which

* Corresponding author.
E-mail address: gyzh@tsinghua.edu.cn (G. Zhang).

https://doi.org/10.1016/j.jpdc.2017.09.007
0743-7315/

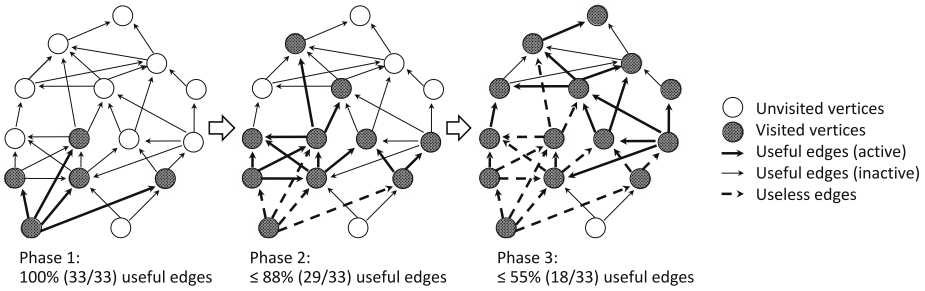

Fig. 1. A simple BFS example.

unnecessary data access takes much more time than actual computing. Fig. 1 presents an example of BFS execution, where the useful edges keep reducing along with the traversal. Due to the limit of memory capacity, BFS performed over a large scale graph is often organized as several rounds of graph traversals [17,26]. Graphs are divided into separated partitions for scaling. A typical iteration of graph traversing procedure begins with loading the graph into memory one partition by another, generates updates from source vertices to destination vertices, applies changes to those vertices, and finally ends with writing the updated state back to the disk. We can see from Fig. 1 that vertices in the graph converge rapidly as the process goes on. This makes a large portion of the accessed edges useless for further computation. Based on this observation, we can exploit the convergence manners of this graph algorithm to improve the execution efficiency. By trimming sub-graphs that are no longer relevant to the subsequent computation, BFS traversal can benefit from lower memory consumption, fewer computing tasks, and fewer disk I/Os.

GraphChi and X-Stream are two representatives of graph computing systems over a single server. GraphChi partitions large scale graphs into disjoint vertex intervals and corresponding edge shardings. With each edge sharding sorted by the source vertices, GraphChi utilizes a parallel sliding window method to make the disk access sequential. However, the computing-intensive sorting operation needed for every sharding is very time-consuming. X-Stream is an edge-centric single server graph system. It scatters edges from the source vertices to generate updates in a streaming manner, and apply these updates in the gather phase. Thanks to the sequential access pattern of data streaming, X-Stream can compute edges and updates from the disk obtaining the maximum disk bandwidth. However, X-Stream cannot perform well on traversal algorithms [26] because of a large amount of irrelevant I/O and computations processing the converged vertices and their corresponding edges.

1.2. Our contributions

In this article, we propose FastBFS, a new approach that performs breadth-first graph search efficiently on a single server. First, FastBFS traverses the graph using an edge-centric graph processing model. Therefore it can obtain the high bandwidth of sequential memory and/or disk access without expensive preprocessing. Second, FastBFS eliminates useless edges in parallel with the graph traversal at a very low cost. With an asynchronous trimming mechanism, it can reduce the size of big graphs efficiently and improve the overall performance. Moreover, FastBFS uses a trimming trigger to dynamically turn on/off the edge trimming to improve the efficiency. Third, if an additional disk is available, FastBFS can schedule I/O streams efficiently in a parallelized manner, which improves the performance even more.

We implement FastBFS by modifying X-Stream [26], an open-source graph computing system developed by EPFL. We evaluate the performance of FastBFS by a comparison with X-Stream and GraphChi, two representative graph computing systems over a single server. Our experimental results show that FastBFS can attain speedups of up to 7.9× and 10.4× in the computing speed compared with X-stream and GraphChi respectively. With an additional disk, FastBFS can get even better speedups respectively.

1.3. Differences from our prior work

This article is based on our prior work presented at the 2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS'16) [6]. In this article, we provide the following important and new materials beyond the prior version.

1. We extend the coverage of the data sets in the experiments by including more real-world graphs, especially those with large diameters, such as web graphs. This is a follow-up improvement according to the feedbacks we got during the presentation of FastBFS on IPDPS 2016.
2. We introduce dynamic edge trimming to improve the performance. Web graphs normally have larger diameters than social graphs, which makes graph traversal inefficient for edge-centric systems. We introduce a dynamic trimming mechanism to reduce the cost of edge trimming.
3. Based on the new type of data sets and optimizations, we perform more experiments to make the performance evaluation of FastBFS more convincing and more adequate. Moreover, we give some analyses on the performance differences among different types of graphs on different systems.
4. Finally, we also add some new materials to make our motivation and contribution clearer and to help the reader better understand how FastBFS works.

1.4. Article organization

The remainder of this article is organized as follows. Section 2 introduces the design of FastBFS. We then describe the implementation of the FastBFS prototype in Section 3. The performance evaluation of FastBFS is described in Section 4. Finally, we review the related work in Section 5 and conclude this article in Section 6.

2. The FastBFS approach

FastBFS traverses the graph efficiently using an efficient trimming mechanism. Irrelevant edges are eliminated in parallel with the traversal without introducing much overhead.

2.1. FastBFS overview

FastBFS discovers vertices in a layered structure, originating from the root vertex. The root vertex is considered as the first

Please cite this article in press as: G. Zhang, et al., Accelerating breadth-first graph search on a single server by dynamic edge trimming, J. Parallel Distrib. Comput. (2017), https://doi.org/10.1016/j.jpdc.2017.09.007.