Future Generation Computer Systems 32 (2014) 324–337

Contents lists available at ScienceDirect

Future Generation Computer Systems

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

Parallel processing of large graphs☆

Tomasz Kajdanowicz*, Przemyslaw Kazienko, Wojciech Indyk

Wroclaw University of Technology, Poland

HIGHLIGHTS

- We compared three parallel computing techniques in terms of large graph processing.
- MapReduce, map-side join and Bulk Synchronous Parallel tested for two distinct problems.
- Iterative graph processing with the BSP implementation significantly outperforms MapReduce.
- Map-side join design pattern may improve the original MapReduce performance.

ARTICLE INFO

Article history:
Received 9 March 2013
Received in revised form
30 July 2013
Accepted 2 August 2013
Available online 28 August 2013

Keywords:
Large graph processing
Parallel processing
Big data
Cloud computing
Collective classification
Shortest path
Networked data
Bulk Synchronous Parallel
MapReduce

ABSTRACT

More and more large data collections are gathered worldwide in various IT systems. Many of them possess a networked nature and need to be processed and analysed as graph structures. Due to their size they very often require the usage of a parallel paradigm for efficient computation. Three parallel techniques have been compared in the paper: MapReduce, its map-side join extension and Bulk Synchronous Parallel (BSP). They are implemented for two different graph problems: calculation of single source shortest paths (SSSP) and collective classification of graph nodes by means of relational influence propagation (RIP). The methods and algorithms are applied to several network datasets differing in size and structural profile, originating from three domains: telecommunication, multimedia and microblog. The results revealed that iterative graph processing with the BSP implementation always and significantly, even up to 10 times outperforms MapReduce, especially for algorithms with many iterations and sparse communication. The extension of MapReduce based on map-side join is usually characterized by better efficiency compared to its origin, although not as much as BSP. Nevertheless, MapReduce still remains a good alternative for enormous networks, whose data structures do not fit in local memories.

1. Introduction

Many technical and scientific problems are related to data with the networked nature, which can be relatively simply represented by means of graph structures. Graphs provide a very flexible abstraction for describing relationships between discrete objects. Many practical problems in scientific computing, data analysis and other areas can be modelled in their essential form by graphs and solved with the appropriate graph algorithms.

In many environments graph structures are so big that they require specialized processing methods, especially parallel ones. This becomes particularly vital for data collections provided by users leaving their traces in various online or communication services, such as multimedia publishing portals or social networking sites, e.g. YouTube or Facebook. Additionally, these datasets reflect various user behaviour, so their graph representation may be complex with multiple relationships linking network nodes. This requires analytical methods dealing not only with simple graphs but also hypergraphs or multigraphs.

As graph problems grow larger in scale and more ambitious in their complexity, they easily outgrow the computation and memory capacities of single processors. Given the success of parallel computing in many areas of scientific computing, parallel processing appears to be necessary to overcome the resource limitations of single processors in graph computations. Parallel graph computation is, however, challenging [1] and before the advent of cloud computing and Hadoop, programmers had to use ill-suited distributed systems or design their own systems, which required additional effort to provide fault-tolerance and to address other problems related to parallel processing [2]. The rise of the MapReduce concept and Hadoop—its open source implementation—provided researchers with a powerful tool to process large data collections. Recently, Hadoop has become a de facto standard in

* Corresponding author. Tel.: +48 713203609.
E-mail address: tomasz.kajdanowicz@pwr.wroc.pl (T. Kajdanowicz).

0167-739X/$ – see front matter
http://dx.doi.org/10.1016/j.future.2013.08.007

academia and a significant solution for parallel processing in industry. It has been used in various areas, including some graph processing problems [3].

The MapReduce model is, however, badly suited for iterative and graph algorithms. There has been a lot of research in creating design patterns improving MapReduce performance for graphs like [4,5], or building systems that would aid iterative processing on MapReduce [6–10]. Google reacted to that with the development of Pregel [2]—an alternative system that implements the Bulk Synchronous Parallel (BSP) programming model [11] for graph processing.

The main difference in the processing of regular data structures (tables) and relational models (graphs) relies on different problem decomposition. Processing table structures is composed of handling of individual records (rows in the table). For the networked data, single processing of a graph vertex usually requires access to the neighbourhood of this vertex, which for most algorithms remains fixed for the whole processing time. This data may be either accessed at every algorithm iteration via a distributed file system (e.g. HDFS), as in the case of MapReduce, or preserved locally for the entire processing, the case for BSP.

Both different parallel processing methods, i.e. MapReduce and BSP, along with the map-side join MapReduce modification, have been implemented in the Hadoop environment—all three were used in the experiments presented in this paper. Each approach was independently applied to solve two distinct graph analytical problems: single source shortest path (SSSP) calculation and collective classification of network vertices with Relational Influence Propagation (RIP). The graph algorithms had an iterative nature, which enabled testing their various parallel implementations in the following steps. The iterative computation was carried out in cloud environments containing various numbers of machines to compare scalability of Bulk Synchronous Parallel and MapReduce. Additionally, all approaches were tested on several large graph data sets coming from various domains.

The initial version of the paper was presented at the ICDM 2012 conference [12].

The following Section 2 provides a short state-of-the art study on graph problem solutions by means of cloud computing. The main architectures for graph processing, including distributed memory and shared memory, are presented in Section 3. Two parallel processing models MapReduce and Bulk Synchronous Parallel (BSP) are sketched in Section 4. Some discussion on their similarities as well as potential improvements is provided in Section 5. Also in this section, an important and experimentally verified MapReduce modification based on map-side join design patterns is proposed for graph processing. Two iterative graph algorithms: single source shortest path computation and collective classification are described in more depth in Section 6. Experimental setup and cloud environment, including data set profiles can be found in Section 7. The results of experiments are presented in Section 8. Discussions on the results and solutions of some problems, which arose during research, are depicted in Section 9. The last section, Section 10 contains general conclusions and further directions for work.

2. Related work

The dynamic development of distributed and cloud computing has led to stable solutions for massive data processing. Nowadays, there is an intensified focus on new models useful for specific kinds of processing. On top of distributed storage systems many solutions dedicated for particular tasks are located, for example fast random access data, pipeline processing, graph computations, etc. [13].

There are several concepts for parallel processing in clusters. Two of them are widely used in offline batch analysis systems and merit special attention: MapReduce and the less popular Bulk Synchronous Parallel (BSP). The former is especially very popular and applied to many real solutions [13].

The general idea behind the Bulk Synchronous Parallel (BSP) method was first coined and studied in early 90s [11,14]. Recently, it was adapted by Google to graph processing in clouds in the Pregel system [2]. Pregel's idea of using BSP for graph processing in clouds inspired others to create similar systems, some of which are open source e.g. [15,16].

The overview of large-scale graph engines is presented in [17], which contains graph systems designed to achieve different goals—from offline analytics system to online low-latency systems.

An empirical comparison of different paradigms for large-scale graph processing is presented in [18]. However, the presented paradigms require a proprietary and/or prototypical platforms, while, in this paper, we focus on approaches which are available on Hadoop, a highly popular, open source platform, which can be run on a set of commodity hardware.

Pace et al. [19] provided a theoretical comparison of BSP and MapReduce models. In terms of graph processing, they noticed that the Breadth First Search algorithm (for the shortest path computation) cannot be efficiently implemented by means of the MapReduce model. In this paper, we go forward and focus more on an empirical comparison for the real world data sets, using available implementations as well as evaluation for an additional graph problem—collective classification. The general conclusions remain the same: BSP usually appears to be a better model for solving graph problems than MapReduce. The results included in this paper provide quantitative analyses supporting that statement.

3. Parallel architectures for graph processing

Regardless of the nature of a particular computational problem it can be parallelled and scaled well when the overall solution is balanced in terms of the problem solution, the algorithm expressing the solution, the software that implements the algorithm and hardware. The algorithms, software, and hardware that worked properly for standard parallel applications are not necessarily effective for large-scale graph problems. In general, graph problems have specific properties that make them difficult to fit in existing distributed computational solutions. Among others, the following characteristics of graph processing causes challenges in effective parallel processing [20]:

- *Computation driven by relational data*. The majority of graph algorithms are executed according to the structure of a graph, where computation for each next vertex is strictly dependent on the results calculated for all antecedents. It means that the algorithm relies on the graph structures rather than on explicitly stated sequential processing. This implies that the structure of the whole computation is not known at the beginning of execution and efficient partition is hardly possible.
- *Diverse and unbalanced data structures*. Usually graph data is highly unstructured or irregular, which do not give many options for parallel processing based on partitioning. Additionally, a skewed distribution of vertex degrees makes scalability difficult, limiting it to unbalanced computational loads.
- *High overload for data access in comparison to computation*. Algorithms often explore graphs rather than performing complex computations on their structure, e.g. the shortest path problem requires only single arithmetic operations in path cost calculation but requires the performance of many data queries. Runtime can be easy dominated by the wait for memory access, not by computational activities.

Due to the fact that commercially available computer appliances have varying capabilities there can be distinguished several processing architectures suitable for distinct hardware. Depending on the amount of available storage and memory for computation the data might be processed in a different manner, reducing or increasing the latency. There can be distinguished distributed memory architectures and shared-memory architectures.