A Graph Processing System with Actors

Abstract

Graph-based applications become hotter and hotter due to the rising of the social-networks and other problems come up such as paths of disease outbreaks, or chemical compounds, or biological structures. A strong desire to process large graph motivates researchers to study on distribute memory machines. Unfortunately, developing distributed graph algorithm still requires some cost, especially to non-expert. While manufacturing technology improves, physical limits of semiconductor-based microelectronics have become a major design concern. A combination of increased available space and the demand for increased thread level parallelism led to the development of multi-core CPUs. Today, a single multi-core server already has a very powerful computing capabilities, which means we can exploit its capabilities to do more job.

In this paper, we present GPA, a graph processing system with actors on a single machine. With actors, we avoid the consumption caused by the frequent switching among threads. Additionally, we utilize the memory mapping for better performance. We show, through experiments and theoretical analysis, that processing large-scale graph on a modern PC with actors performs well.

Introduction

Graph algorithms are becoming increasingly important for solving many problems in scientiﬁc computing, data mining and other domains such as social networks, web graphs, chemical compounds, and biological structures. The scale of real graphs is so large that may consist of billions of vertices, trillions of edges. For Example, the Yahoo Web graph have 1.4 billion vertices and 6.6 billion edges. However, Graph processing is difficult because of the inherent complicated data structure of the graph and the extremely large size of the graph. Therefore, designing a scalable, fault-tolerant, robust system for processing the large-scale graphs is one of the most urgent problems facing systems researchers.

Motivated by the demands, there are already some solutions, which are able to process large scale graphs with distributed system such as Pregle, PowerGraph and GPS, proposed by other researchers. Nowadays, though distributed computional resources are more accessible than ever before, processing these graphs still remains many challenges. In distributed system, the first main problem is workload balancing which caused by partitioning the large scale graph into small partition to fit the cluster nodes. The second main issue is message passing. Messages exit among the different computional nodes or inside of the cluster node, the cluster nodes take cost to communicate with each other and the communication between nodes causes latency which matters in a BSP based graph processing system. Therefore, many researchers spend a lot of energy to study the distributed system based on Pregel to solve the problems mentioned above and some gain reasonable performance, such as Mizan which aims to the workload balancing, GPS focus on the messaging latency. But from a developer’s perspective, developing, debugging and optimizing distributed algorithm on distributed system is quite difficult because the user needs to be skilled at managing and tuning a distributed system in a cluster, which is a nontrivial job for the ordinary user. Besides, these distributed systems need many machines in a cluster which brings both money and energy cost.

Recently, some graph processing engines on single PC have been proposed to address the problems of the distributed graph systems. For example, Ligra is a lightweight graph processing framework this is specific for shared-memory multicore machine. Specially, Graphchi, a disk-based graph processing engine on a single modern PC, significantly outperforms all representative distributed graph engines. TurboGraph inspired by Graphchi focuses on parallelism and overlapping of CPU processing and I/O processing with a novel concept pin-and-slide.

However, we observe that none of the existed single PC engine underutilize the resource of the multicore machine or the operating system. Parallelism could improve the computing efficiency. Given the success of parallel computing in scientific computing, data analysis and other areas, parallel processing appears to be necessary to overcome the resource limitations of single processors in graph computations. While the inherent complicated characteristics of the graphs make them hard to match the current parallelism computational problem-solving approaches. Vertex-centric is a very brilliant idea for graph processing. In the mode, every single vertex is a little compute unit, which simplifies the process, and every vertex communicate with each other by message. Unfortunately, a modern PC could not support a large number of concurrent, because the concurrent implemented by thread has its limitation. Motivated by this issue, we present a totally different graph processing engine on a single modern PC. We implements the engine with actor/coroutine, actor/coroutine is an ultra-lightweight thread that could greatly improve concurrency. The higher the concurrency, the more suitable the vertex-centric is. We are bold attempt to map a vertex to an ultra-lightweight thread and transfer the communication between vertices into the communication of ultra-lightweight thread. Besides, we notice that the IO processing is the main efficiency killer, we exploit the ability of memory mapping provided by the operating system.

The rest of the paper is as follows. Section 2 reviews related work. In next section, we adopt the vertex-centric model to fit the ultra-lightweight thread. Section 4 presents the detailed implementation of the engine. Section 5 describes the experiments results. And in section 5, and section 6 summarizes and concludes the paper.

Related Work

Many scalable graph systems have recently been proposed. We survey some of the most relevant works, which may be broadly classified into single-machine approaches and distributed approaches depending on the size of the system.

**Distributed Systems:** Pregel is a synchronous vertex-centric programming model proposed by Google for graph system. In Pregel, each vertex is executed in parallel and the user rewrite the compute function invoked by the vertex. Pregel introduced the first bulk synchronous parallel (BSP) distributed message-passing system. In BSP model, all vertex kernels run simultaneously in a sequence of super-steps. Within a super-step, each vertex receives messages from in-neighbor of the last super-step and sends messages to out-neighbor of the next super-step. And a barrier is imposed between two super-steps to guarantee that all vertices finish processing messages. However, Pregel is not open-source and many other systems drown from Pregel such as GPS, PowerGraph, GraphLab, Mizan etc. Other systems Pegasus and gbase are based on MapReduce and support matrix-vector multiplication using compressed matrices.

**Single-machine Systems:** X-Stream is a system for processing both in-memory and out-of-core graphs on a single shared-memory machine. X-Stream take advantage of using an edge-centric model and streaming completely unordered edge lists rather than performing random access. Ligra is a lightweight graph processing framework this is specific for shared-memory multicore machine. Graphchi is a disked-based single-machine system following the asynchronous vertex-centric programming model. Graphchi proposed parallel sliding windows (PSW) to handling disk-based large-scale graphs and it updates values to the edges in Graphchi. PSW partitions the vertices into P execution intervals, and each execution interval contains a shard file that stores all in-edges sorted by their source vertices. PSW processes one shard at a time. During the processing, first of all, loading an execution interval into memory from the disk, then update the vertices and edges. At last, writing the updated content to disk.

Overview