Graph Processing Frameworks

Aris Paphitis
CUT

Key Point

Optimal graph analysis of big data obtained from Online Social Networks (OSN). OSNs collect a vast ammount of user-generated data from billions of users.This consequently fules the Big Data phenomenon. Need to develop an optimized cluster computing framework, tailored to big social data workloads. Goal is to augment the recently proposed platforms for large data processing [Mesos, Spark, Shark] that make use of the fast memoryof computer clusters as opposed to stable storage. MapReduce and its variations are not suitable for big social data analysis primarily due to their acyclic data flow model that does not capture many use cases [???]. Spark solution.... But, (more to say here ???) However Spark is not tailored to social graphs, thus it does not process them in an optimal fashion. PowerGraph is customized for graph-based computations and in several workloads outperfoms Spark and MapReduce. On the other hand ... Inherent data-parallelism Vs Minimum cut problems. Minimum cut problems have not been sufficiently parallelized. GraphChi reduces the I/O access when we process large graphs on a single machine by swapping data in memory and disk, but cannot scale to tens of TB and PB as the cluster computing frameworks do. (Read GraphX, derived from Hadoop+Pregel, running on Spark.)

0.1 Abstract

Many practical computing problems concern large graphs. Standard examples include the Web graph and various social networks. The scale of these graphs - in cases it can be billions of vertices with a trillion edges - poses challenges to their efficient processing. Frequently applied algorithms include shortest paths computations, different flavors of clustering, and variations of the page rank theme.

While distributed computational resources have become more accessible, developing distributed graph algorithms still remains challenging. Graph algorithms usually exhibit poor locality of memory access, very little work per vertex, and a changing degree of parallelism over the course of execution. Distribution of resources exacerbates the locality issue, and increases the probability that a machine will fail during execution.

Additionally, social networks, Web graphs and protein interaction graphs are particularly challenging to handle, because they cannot be readily decomposed into small parts that could be processed in parallel.

1 Pregel

`Pregel` was Google's answer to this task. In Pregel, programs are expressed as a sequence of iterations, in each of which a vertex can receive messages sent in the previous iteration, send messages to other vertices, and modify its own state and that of its outgoing edges or mutate graph topology. It was designed for efficient, scalable and fault tolerant implementation on clusters. Distribution related details are hidden behind an abstract API.

2 GraphLab

3 PowerGraph

The `PowerGraph` abstraction exploits the internal structure of graph programs to address the challenges of computation on natural graphs in a graph-parallel abstraction. Natural graphs commonly found in the real-world have highly skewed power-law degree distributions. PowerGraph exploits the structure of power-law graphs

4 Giraph

5 GraphX

`GraphX` is an embedded graph processing framework built on top of Apache Spark. It aims at unifying advances in dataflow systems with advances in graph processing systems, thus eliminating unnecessary data movement and duplication along analytics pipelines. The GraphX API enables the composition of graphs with unstructured and tabular data and permits the same physical data to be viewed both as data and as collections without data movement or duplication. Graph computations are reduced to a specific `join-map-group-by` dataflow pattern and additional optimizations are introduced to achieve performance parity with specialized graph processing systems.

6 GraphChi

`GraphChi` is a disk-based system for computing efficiently on graphs using a single consumer-level computer. It introduces a novel method, Parallel Sliding Window (PSW), for processing very large graphs from disk. PSW, unlike most distributed frameworks, implements the asynchronous model of computation, which has been shown to be more efficient than synchronous for many purposes.

6.1 Bulk Synchronous Parallel model

Both, Apache Giraphe and Google's Pregel are inspired by Bulk Synchronous Parallel model of distributed computation. The Bulk Synchronous Parallel (BSP) abstract computer is bridging model for designing parallel algorithms. It was developed by Leslie Valiant of Harvard University during the 1980s. A BSP computer consists of:

1. components capable of processing and local memory transactions
2. a network that routes messages between pairs of such components, and
3. a hardware facility that allows for the synchronization of all or a subset of components.

A computation proceeds in a series of global supersteps consisting of three components.

- `Concurrent Computation`: every participating processor may perform local computations, i.e., each process can only make use of values stored in the local fast processor memory. The computations occur asynchronously of all the others but may overlap with communication.
- `Communication`: The processes exchange data between themselves to facilitate remote data storage capabilities.
- `Barrier synchronization`: When a process reaches this point (the barrier), it waits until all other processes have reached the same barrier.

Notes