Md Majid Jahangir | SBU ID - 112077145
AMS 560 - Big Data Systems, Algorithms and Networks

Assignment 3 (Paper Reading)
Pregel: A System for Large-Scale Graph Processing

Motivation : With Web 2.0, the web graph grew exponentially. We saw social networks, transportation routes, citation relationships among published research work, etc, among few of the examples of graph network and algorithms such as shortest path, minimum cut, connected components, etc. Processing such large graphs efficiently is quite challenging and there was no scalable general purpose system that would implement graph algorithms over graph representations in a large scale distributed environment. There were existing systems that would provide graph algorithm library but they would work on a single computer and were not distributed; and even though there were parallel graph systems, they would not address fault tolerance or any other issues that were inherent for a large scale graph processing distributed systems. Hence, to address this issue, this paper [1] proposes a scalable and fault-tolerant platform with an API called Pregel, that is flexible enough to address and express arbitrary graph algorithms on arbitrary graph representations enabling distributed processing of large scale graphs.

Prior Solutions: Some of the prior solutions are Parallel Boost Graph Library (Parallel BGL) and CGMgraph. ParallelBGL specifies various key generic concepts to define distributed graphs and provides implementations based on Message Passing Interface (MPI) and it also provides a lot of implementations of algorithms based on MPI as well. It implements property maps that holds informations about vertices and edgesi in graph and uses ghost cells to hold values related to remote components; however this can lead to scaling problems if too many references to many remote components are required. Pregel differs from Parallel BGL by providing fault tolerance to handle failures during computations enabling it to work in a clustered environment.
CGMraph also provides a number of parallel graph algorithms using Coarse Grained Multicomputer (CGM) model based on MPI. The main agenda of it is to provide implementations of algorithms than an infrastructure to be used to implement them. The difference between Pregel and CGMgraph is that Pregel employs the use generic programming style whereas CGMgraph employs the use of object-oriented programming style.

Key Ideas of Proposed Solution: The key ideas of the proposed solution are -

1. Model of Computation - A pregel computation model consists of input as a directed graph where source and target vertices are identified by a vertex identifier and edges are also associated with a modifiable value; a sequence of supersteps (sequence of iterations) separated by global synchronization points till the algorithm terminates and returns with an output. For each superstep, the vertices are computed in parallel and each of them executes the same user defined function that is basically the logic of the given algorithm. At superstep 0, every vertex will be in active state and they participate in computation for any given superstep. At every superstep, a vertex can modify its state or of its outgoing edges, receives messages that were sent to it in previous superstep or sends a message to other vertices or can even modify the topology of the graph. Whenever a vertex has no work to do, it will not be computed in further supersteps unless it again receives a message and it will deactivate itself by voting to halt. The algorithm in its entirety will terminate when all of the vertices are in inactive state and there are no messages in transit.
2. C++ API - Pregel provides a C++ API for implementing any graph algorithms. In order to implement an algorithm using this API, the developer has to subclass a predefined existing class called Vertex which provides several methods such as GetValue() to get the value associated with a vertex or to modify its value by MutableValue() and other methods to send or receive messages to or from other vertices. It also contains a Compute() method which can be overridden to implement the logic of any given graph

algorithm to be executed at each active vertex in every superstep. Also, the values associated with each vertex and edges persists across supersteps.

3. Message Passing - Vertices communicate with each other directly by sending messages consisting of message value and the name of the destination vertex. A vertex can send any number of messages to any vertices,not necessarily its neighbours, in any supersteps. It is guaranteed that the messages will not be duplicated, however the order of messages is not guaranteed.
4. Combiners - Combiners are components that are used to combine multiple messages received at a vertex into one message. These are not enabled by default and can only be enabled by subclassing a class called Combiner and then overriding its virtual Compute() method. However, combiners should only be enabled for commutative and associative operations.
5. Aggregators - Aggregators are used as a mechanism to communicate or monitor and data sharing globally. It works in a way where each vertex can provide a value to an aggregator in superstep S, the system will combine those values using a reduction operator and then the final reduced value is available to all vertices in superstep s+1. Certain aggregators by default are min, max, sum, etc.
6. Topology Mutations - Compute() method can also issue requests to add or remove edges or vertices from graph, for e.g. for minimum spanning tree computation or clustering use-case.
7. Input/Output - Task of interpreting an input and graph computation is decoupled such that input can be taken in any format be it from file, or database or from BigTable. Similarly, output can also be written in any format. For such use-cases, a default class called Reader and Writer has been provided which can be subclassed to implement one's own way of interpreting inputs or writing an output.

Performance: Several performance tests were conducted with the single-source shortest paths (SSSP) implementations on a cluster of 300 multicore PCs. Runtimes for binary trees were reported to study the scaling properties and runtimes for log-normal random graphs were reported to study the performance in a realistic environment by using graphs of various sizes with weights of all edges set to 1. A speedup of 10 times was observed by using 16 times as many Pregel workers (from 50 to 800) to compute shortest path for a binary tree with a billion vertices, the drop in runtime from 174 seconds (for 50 Pregel workers) to 17.3 seconds (for 800 Pregel workers).
Also, a fixed number of worker tasks (800) were scheduled on 300 multicore machines to compute shortest paths for binary trees varying in size from a billion to 50 billion vertices. It was observed that the runtime increased linearly in graph size for graphs having a low average outdegree.
However, in real life, a binary tree representation of graph is never encountered, therefore even though Pregel scales in workers and graph size for such binary trees, experiments were also conducted on random graphs that use a log-normal distribution of outdegrees which resembles most of the real world large scale graphs such as social network,where most of the nodes have relatively low outdegrees but having some outliers having more than hundred thousand outdegrees. Computing shortest path on a graph having a billion vertices using 800 worker tasks on 300 multicore machines took about 10 minutes. It's also been reported that a naive algorithm to find the shortest path was used for all the experiments and therefore the above mentioned results should not be considered as the best because using a more advanced algorithm would result in an even better result. The experiments were merely conducted to show that a satisfactory result could be obtained with very less coding effort.

What are the advantages of Pregel comparing with other options? [Key Question]:

1. One of the alternatives for parallel large scale graph processing is Parallel Boost Graph Library (Parallel BGL). Parallel BGL specifies various key generic concepts to define distributed graphs and provides implementations based on Message Passing Interface (MPI) and it also provides a lot of implementations of algorithms based on MPI as well. It implements property maps that holds informations about vertices and edges in graph

and uses ghost cells to hold values related to remote components; however this can lead to scaling problems if too many references to many remote components are required. However, Pregel differs from Parallel BGL by providing fault tolerance to handle failures during computations enabling it to work in a clustered environment.

2. CGMraph also provides a number of parallel graph algorithms using Coarse Grained Multicomputer (CGM) model based on MPI. The main agenda of it is to provide implementations of algorithms than an infrastructure to be used to implement them. The difference between Pregel and CGMgraph is that Pregel employs the use generic programming style whereas CGMgraph employs the use of object-oriented programming style. By using a generic programming style, Pregel could be catered to multiple distributed systems infrastructure and catered to handle multiple graph inputs of various formats and could also write outputs in various formats.
3. MapReduce had been used in past for mining large scale graphs. Even though it is a distributed computing platform, it is ill-suited for graph processing. On the contrary, Pregel provides a graph API that is efficient to implement several graph algorithms and express several graph representations and have much more efficient support for iterative computations over the graph.
4. Several graph algorithm libraries such as LEDA, NetworkX, BGL, etc also exists, however they are just single computer graph library, unlike Pregel which is a scalable and fault tolerant distributed graph processing system.

What is superstep in Pregel? [Key Question]: A superstep is a sequence of iterations in Pregel's computation model. During each superstep, the vertices are computed in parallel and each of them executes the same user defined function that is basically the logic of the given algorithm. The user defined function species behavior observed at a single vertex 'v' and a single superstep 's' and it can read messages sent to vertex 'v' in superstep 's'- 1, send messages to other vertices that will be received at superstep 's' + 1, and modify the state of V and its outgoing edges.

How is Pregel implemented on Google cluster? [Key Question]:

Pregel has been designed to run on Google's cluster architecture. Copies of user programs is deployed on a cluster of machines; where one of the machine is the master and it coordinates worker's activities. All the other machines are called workers and they use a naming service to identify and locate master and then registers themselves with the master. Pregel library divides the input graph into partitions, each of which consists of a set of vertices and its outgoing edges. Master decides the number of partitions that a graph will have and assigns each of the partition to each worker thus ensuring parallelism. Then each worker is instructed by the master to perform a superstep. A worker will use just one thread for each partition and will iterate over active vertices in its partition. Messages are sent asynchronously but it is ensured that it will be delivered before the end of the superstep. At the end of superstep, the worker will inform the master about the number of vertices that will be active in the next superstep; the master will repeat this process as long as there are active vertices or any messages being in transit. To ensure fault tolerance, checkpointing is done at the beginning of each superstep and heartbeat mechanism is used to detect failures. On failures of any worker, the master will reassign that graph partition to the available set of workers and those worker will reload the partition state from the most recently available checkpoint that was done at the beginning of each superstep.

Strength :

1. It is fault tolerant as it was designed to be run on Google cluster architecture. Checkpointing at the beginning of each superstep makes it fault tolerant such that even if a worker computing a small partition of a graph fails, another worker could resume the computations on that partition by resuming the work from the checkpoint.
2. It is scalable by the virtue of it being run on Google cluster architecture. With the concept of partitioning the entire large scale graph into multiple small graphs, and then making the user defined graph algorithm run on each partition in parallel makes it scalable.

3. The C++ API provided by Pregel is intuitive, flexible and easy to use and have been curtailed to graph processing use-case only. Concepts like that of aggregators, combiners, a unit testing framework and a single machine mode which helps in rapid debugging and prototyping has been useful and makes it very easy to be adopted. Application developers need no knowledge of parallel systems and instead, they just need to “think like a vertex” and write some functions that encapsulate the logic for what one graph vertex does. This generic vertex-oriented programming model has been found to ease the implementation of distributed graph algorithms to a great extent [5].
4. For cases, where the graph being dealt with is really dense or certain algorithms which has a dense communication over a sparse graph, Pregel can transform such algorithms to a variant which is more friendly for Pregel workflow by applying combiners, aggregators or topology mutations on them.

Weakness :

1. Currently the entire computation state of Pregel resides in physical memory at the same time, however for very large graphs which may not accommodate entirely in the physical memory at the same time, computation using Pregel becomes difficult.
2. Assigning vertices or partitions to worker machines to minimise inter-machine communication is a challenge. If the topology of the graph does not correspond to the message traffic, then partitioning of the graph based on topology may not be sufficient and in that case dynamic re-partitioning mechanisms should be employed.

Follow Up Works -

1. Towards highly scalable pregel-based graph processing platform with x10 [3] - The paper proposes a graph processing system based on Pregel called X-Pregel by using the state of the art PGAS programming language X10. The paper introduced two new features on top of Pregel - 1. an optimization to reduce the number of messages being exchanged between workers, (2) a dynamic re-partitioning scheme that effectively reassign vertices to different workers during the computation. They also demonstrated that their system processes large graph faster than prior implementation of Pregel.
2. PAGE: a partition aware graph computation engine [4] - In this paper, the authors found that in Giraph, even on a well partitioned graph the performance was 2 times worse than the simple partitions and it was because the local message processing cost in graph computing systems may surpass the communication cost in most cases. The authors analysed the cost of the parallel graph computing systems as well as the relationship between cost and graph partitioning and proposed a novel Partition Aware Graph computation Engine named PAGE. PAGE is equipped with two newly designed modules, i.e., the communication module with a dual concurrent message processor, and a partition aware one to monitor the system's status. The monitored information can be utilized to dynamically adjust the concurrency of dual concurrent message processor with a novel Dynamic Concurrency Control Model (DCCM). The DCCM applies several heuristic rules to determine the optimal concurrency for the message processor.
3. Pregelix: dataflow-based big graph analytics [5] - In this paper, the author proposed that instead of building a Pregel system from scratch, it is better to explore an architectural alternative — expressing Pregel’s semantics as database-style dataflows and executing them on a general-purpose data-parallel engine using classical parallel query evaluation techniques. The author used this approach to build Pregelix — a dataflow-based Pregel implementation for Big Graph analytics.

References -

1. Pregel: A System for Large-Scale Graph Processing - Grzegorz Malewicz, Matthew H. Austern, Aart J. C. Bik, James C. Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski
2. The morninig paper - https://blog.acolyer.org/2015/05/26/pregel-a-system-for-large-scale-graph-processing/
3. Towards highly scalable pregel-based graph processing platform with x10 - Nguyen Thien Bao, Toyotaro Suzumura - https://dl.acm.org/citation.cfm?id=2487984
4. PAGE: a partition aware graph computation engine - Yingxia Shao, Junjie Yao, Bin Cui, Lin Ma
5. Pregelix: dataflow-based big graph analytics - Yingyi Bu - https://dl.acm.org/citation.cfm?id=2525962