Paper Summary

PowerGraph: Distributed Graph-Parallel Computation on Natural Graphs

Dong Chen(dongchen), Huanyu Zhao(huanyuzh), Jinxiaoyu Zhi(jinxiaoy)

Problem and Motivation

The demand of large-scale graph-structured computation has driven the development of graph-parallel abstractions in specialized graph processing systems. graph-parallel abstractions rely on each vertex having a small neighborhood to maximize parallelism and effective partitioning to minimize communication.

However, graphs derived from real-world typically have power-law degree distributions, which are difficult to partition and represent in a distributed environment, limiting performance and scalability. To address the challenges, we need a new abstraction to achieve greater parallelism, lower network communication and storage costs.

Another problem is that specialized graph processing systems leads to unnecessary data movement, duplication, and lack of fault tolerance in the larger analytics process which often combines graphs with unstructured and tabular data. Therefore, we want to unify advances in graph processing systems with advances in dataflow systems enabling a single system to address the entire analytics pipeline.

Hypothesis

The first paper introduces PowerGraph abstraction which exploits the structure of vertex-programs and explicitly factors computation over edges instead of vertices. PowerGraph combines the best features from both Pregel and GraphLab, including data-graph and shared-memory view of computation and the commutative, associative gather concept.

The second paper introduces GraphX, which is an embedded graph processing framework built on top of Apache Spark. It is implemented using only a few basic dataflow operators (e.g., join, map, group-by) and recasts graph-specific optimizations as distributed join optimizations and materialized view

maintenance.

Solution Overview :

A new graph-parallel abstraction that can eliminate the degree dependence of the vertex-program by directly exploiting the GAS(Gather, Apply, Scatter) decomposition to factor vector-programs over edges.

1. Combining the best feature from both Pregel and GraphLab

   Data-graph, shared-memory view of computation and the commutative, associative gather concept are all included which means PowerGraph supports both the highly-parallel bulk-synchronous Pregel model of computation and the computationally efficient asynchronous GraphLab model of computation.

   1) When bulk synchronous execution:

   A super-step includes a lot of minor-steps like gather, apply and scatter. After each super-step, a barrier will stop it until all the super-steps finish and then will active all the vertices for next step.

   2) When asynchronous execution:

   To prevent non-determinism in results, PowerGraph will enforce the serializability so that every parallel execution of vertex-programs has a corresponding sequential execution.

2. Delta caching

   To avoid update the whole system with only a little change, the PowerGraph maintains the cache of the accumulator from the previous gather phase for each vertex, therefore only additional accumulator delta occur, the vertex related to it will be updated.

3. Distributed Graph Placement

   1) Allowing a single vertex-program to span multiple machines

   By evenly assignment edges to machines and allowing vertices to span machines, each machine only stores the edge information for the edges assigned to that machine. Since each edge is stored exactly once, changes to edge data do not need to be communicated. However, changes to vertex must be copied to all the machines it spans, thus the storage and network overhead

depend on the number of machines spanned by each vertex.

2) Greedy Vertex-Cuts

To de-randomizing the edge-placement process, the algorithm will place the next edge on the machine that minimizes the conditional expected replication factor.

Limitations

1. PowerGraph can only deal with fix graph but not dynamic graph.
2. Difficult to cooperate with unstructured data.
3. For some specific graph processing application which often combines graphs with unstructured and tabular data, PowerGraph leads to unnecessary data movement, duplication and lack of tolerance. So GraphX is introduced to solve the problems.

Class Summary

1. Some graph algorithms: Pregrel and GraphLab
2. Power Law and challenges of high degree vertices. For Pregrel, it sends many identical messages. And for GraphLab, the locking schema is unfair to high degree vertices. Moreover, high degree vertices bring work-Inbalance and storage issues.
3. PowerGraph is introduced to address the challenges of Power Law. It exploits the structure of vertex-programs and explicitly factors computation over edges instead of vertices. And GAS decomposition is composed of Gather, Apply and Scatter.
4. Different Graph Placement for Pregrel and PowerGraph: Pregrel uses Edge-cut and PowerGraph uses Vertex-cut. The replication factor for Vertex-cut is better than that of edge-cut.
5. Graph-parallel systems such as Pregrel and PowerGraph do not address the challenges of graph construction and transformation which are often just as problematic as the subsequent computation.
6. GraphX is a layer on top of Spark for graphs and graph-parallel computation. It extends the Spark RDD by introducing a new Graph abstraction: a directed

multigraph with properties attached to each vertex and edge. To support graph computation, GraphX exposes a set of fundamental operators and an optimized variant of the Pregel API.

7. GraphX is built on Spark, which provides lineage-based fault tolerance with negligible overhead as well as optional dataset replication. So GraphX achieves low-cost fault tolerance by leveraging logical partitioning and lineage.