Kunal Baweja | UNI: kb2896

Paper Review: GraphX - A Resilient Distributed Graph System on Spark

Introduction

Computations on graphs have traditionally been cumbersome and inefficient due to the high complexity of graphs as a data structure, but these have had varied and important applications, social networks being the biggest example of all. However, the existing graph parallel systems lack in fault tolerance while efficiently making computations over graphs and that is what GraphX introduced in this paper stresses on. GraphX attempts to provide scalability as well as reliability for graph computation problems by combining the advantages of graph-parallel and data-parallel systems by expressing the graph computation problem within the Spark data-parallel framework.

Motivation and Basic Concept of GraphX

Most of the graph parallel systems are well suited for a certain limited sub-domain of graph computation problems and because each of these frameworks relies on a specific-runtime, it is difficult to compose all of these into a single abstraction.

On the other hand, data-parallel systems like *MapReduce* and *Spark* are well suited for large scale data processing as well as graph construction. These frameworks also provide excellent fault tolerance and big data handling capabilities, but most of the abstractions expressions naively describe graph computations within the scope of these frameworks, which involves a large overhead of moving around excessive data, but fails to exploit the structure of graphs.

GraphX proposes to combine the advantages of graph parallel systems and data parallel systems by expanding upon the *Resilient Distributed Dataset (RDD)* of *Spark* framework, to introduce *Resilient Distributed Graph (RDG)* that associates records with vertices and edges of the graph. *GraphX* uses this implement existing graph parallel frameworks on top of *Spark* and also introduces new operations to view, filter and transform graphs. It exploits the graph structure to minimize the movement of data and communication overhead within *Spark* by making following major contributions:

1. *RDG* supports a wide range of graph operations on top of resilient data parallel Spark framework.
2. A tabular representation of efficient vertex-cut partitioning and data parallel partitioning heuristics.

RDG (Resilient Distributed Graphs)

Similar to data parallel computations which adopt a record centric view of data, graph parallel computation adopts a vertex centric or edge centric view of data for computation i.e it represents data in form of edges and vertices of a graph.

GraphX defines graph transformations such that each operation yields a new graph, hence the core data structure of graph is immutable in *GraphX* system. GraphX consists of *directed adjacency structure* as well as user defined attributes for *vertices* in graphs and programs describe transformations from one graph to another via operators which transform vertices or edges or both, in in context of their adjacent vertices and edges.

Partitioning (Edge Cuts and Vertex Cuts)

Most graph partitioning schemes adopt the *edge-cut* strategy to partition the graphs, but obtaining a precise edge-cut partitioning is a very expensive operation due to which many graph-parallel systems resort to *random edge-cut* strategy. Although this strategy achieves a nearly optimal work balances, it also causes a worst-case communication overhead as almost all of the edges are cut in the graph.

In contrast, *vertex-cuts* evenly assign edges to different machines in the system and thus allow vertices to span across multiple machines. The communication and storage overhead of a vertex cut is hence directly proportional to the sum of number of machines across which a vertex spans, thus reducing both of these. This further ensures that the number of machines spanned by a vertex is minimized and the computation gains efficiency.

Vertex-Cuts as Tables in GraphX

The *GraphX RDG* provides a vertex-cut representation for a graph using three unordered horizontally partitioned tables implemented as Spark RDDs:

1. ***EdgeTable(pid, src, dst, data)***: Each edge is represented as a tuple consisting of the source vertex id, destination vertex id, and user-defined data as well as a virtual partition identifier (pid)
2. **VertexMap(id, pid):** a mapping from the id of a vertex to the ids of the virtual partitions that contain adjacent edges
3. ***VertexDataTable(id, data):*** This tables stores the vertex data, in the form of a *vertex (id, data) pairs*. The vertex data table is indexed and partitioned by the *vertex id*.

GraphX uses a three-way join to bring gather the information related to an edge by combining source vertex data, edge data and target vertex data from the above three RDDs.

Graph Parallel Computation (with GraphX)

GraphX interface is designed to enable the construction of new graph-parallel APIs, rather than acting as one itself. By composing operations in the RDG interface *GraphX* compactly expresses several of the most widely used graph-parallel abstractions, such as *Pregel* and *PowerGraph*.

References

1. GraphX: A Resilient Distributed Graph System on Spark
https://amplab.cs.berkeley.edu/wp-content/uploads/2013/05/grades-graphx_with_fonts.pdf