Scalable Distributed Graph Processing

The graph structure is one of the most studied structures in mathematics and computer science. However, much of the traditional literature regarding algorithms for efficient computations using graph structures do not take the practical constraints of graph size into account. Considerations for memory constraints or even disk constraints on a single machine have become ever more important with the proliferation of use cases involving extremely large datasets. Indeed, by only allowing computation to occur on a single machine, the computations that can be performed or the size of the graph are necessarily limited. Worse still are the complications that running in a distributed system introduce to graph processing. As Malewicz et al describe, one must create a new distributed architecture, a decidedly non-trivial task; co-opt an existing architecture to support the operations one wants to perform, only to sub-optimal efficiency; or use parallel graph systems which would lack the major components necessary for a large-scale distributed system, thus also limiting its full potential.

Consider the case of Twitter, the micro-blogging service that had of July 2009 amassed over 40 million users, nearly 1.5 billion relations, and over 100 million tweets, according to Kwak et al, and today has achieved more than three times the number of users. With the sheer size of the social graph and the associated directed graph that Twitter's asymmetric follow model creates, many reasonable queries would require distributed computation in order to achieve answers in reasonable amounts of time.

Twitter is a particularly good medium for sociological as well as computational study because of its unique operational qualities. Assuming a user has a public profile, any other individual can choose to follow her. However, the original user subsequently does not have to follow the original user; i.e, follows are, as in the vernacular, one-way indications of a relationship, or directed edges on the twitter graph. As a result, the connections on Twitter are thought to represent societal fame, popularity, and influence. Particularly famous people are generally going to have many followers (at the time of this writing, performer Lady Gaga has 29.9 million followers and President Barack Obama has 20.3 million followers), and those individuals with the most followers are generally going to have the most influence. There is also at least one other way that popularity can be measured on Twitter, namely through retweets. A tweet can be retweeted by anyone who can see the original tweet or anyone who can see any retweet; in this way, multiple retweets can create a path between users that itself is likely more informative than the pure “following” relationship that necessarily connects those two users.

Depending upon the query, there is a varying amount of parallelizability that can be achieved in a distributed environment. Consider a simple query that wants to determine the number of individuals with a particular name on the Twitter graph. A Map Reduce cluster could easily solve this problem by simply having each node look through its subset of the graph to determine the number of individuals it has with a given name (Dean and Ghemawat). The data is then shuttled back to a single reduce node that would need to do nothing more than sum its responses. This
query is embarrassingly parallelizable, and hence simply adding nodes would automatically improve the performance of this query. Note that this query, while completely unrelated to graph structure, exposes a problem with typical methods to partition graphs. Many graphs will use a simple hash function on vertices or tuples that could create a situation where the distribution of relevant graph nodes will be uneven with respect to name; hence, even such a simple query would gain no advantage from parallelization (Huang, Abadi, and Ren).

For this project, I will focus on the improvement of large-scale graph processing by focusing on the manner in which graph data is partitioned across a distributed system. In particular, I will optimize the partitioning of an existing subset of the Twitter graph for a representative set of queries. The exact types of queries to be formulated are still being determined through conversations with current employees of Twitter; however, at present there are at least three types of queries: Determining the shortest-path between a given pair of vertices in the graph; identifying “triangles” between vertices (a triple of vertices that form a cycle); and measuring “reach” or “influence” of a particular vertex.

The main concept behind improving a partitioning scheme is that for a given node, for the typical queries, the nodes of interest are also in close proximity (i.e the nodes are a limited number of hops apart). Hence, partitioning the graph based to determine tuple placement, rather than ignoring the graph structure and hashing, should yield much improved performance. For this project, I will consider the $n$-hop directed and undirected guarantee as defined by Huang, Abadi, and Ren, which ensure that for a given node, nodes up to $n$ hops away will be contained on the same machine. Necessarily, this implies that a given node may actually appear on multiple machines (assuming that the graph is connected). This method is not without trade-offs: storage cost will increase as a result of satisfying the $n$-hop guarantee, and depending on the connectedness of the graph, the storage costs can be significantly higher using this methodology. Another important problem to keep in mind is that maintaining duplicate tuples will incur significant overhead in the case that the graph structure itself needs to be updated in any way; for the purposes of this project, the structure of the graph will be considered unchanging. In spite of these deficiencies, there are still significant advantages to maintaining duplication. In densely connected graphs, as is the Twitter graph, it is fairly likely that two arbitrary vertices may be fairly close. While the two individuals may not be only two hops apart, by the virtue of the $n$-hop guarantee, far fewer machines would be necessary to operate to determine the shortest path between those two individuals. This result not only implies that more queries may be run in parallel by the system at higher throughput, but also that the latency of each query may be much smaller because joining the data across this small number of nodes will be far less expensive than trying to join many more partitions.

There are a number of practical applications of improving this performance with respect to the Twitter graph or other similar social networking sites. Consider LinkedIn, the self-branded “World’s Largest Professional Network”, which allows users to see work histories of other individuals. Central to LinkedIn’s user experience is the ability to “connect” with other users for some professional advancement. Hence, if a user is browsing an arbitrary other individual’s profile, LinkedIn would like to deliver to that user a path of individuals by which the two users are connected. Of course, of utmost importance is the actual path that LinkedIn would like to display. A several-hop path is far less likely to yield actual contact between users than is a one-
or two-step path. Instead, LinkedIn would optimally prefer a path of, if possible, mutual acquaintances, in order that the original user could leverage that acquaintance to connect with the other individual. If LinkedIn’s path recommendations yield tangible, real-world results, the user will continue to rely on LinkedIn in order to meet other professionals. Hence, it is not only essential for LinkedIn to be able to perform shortest-path calculations quickly, but in the amount of time it takes to load a webpage. Additionally, LinkedIn will have to provide all such routes in the event that there are multiple minimally short paths.

For Twitter itself, detecting triangles is important for multiple reasons. The first is that if a triangle exists, it is likely that those three individuals overlap one or more interests. Concerned with the user experience, Twitter may use this information to recommend to any of these three individuals a follower that both the other individuals have followed (and similarly, Twitter could recommend to other users this user if the latter is the only of the three that the former has not followed). Additionally, Twitter can use this as additional information about which brands might be most interesting to a user, thus increasing the reach of any promotional campaign for a paying Twitter client.

It is more obvious why the reach or influence of a user would like to be measures. A business that would like to advertise among its Twitter followers may turn to Twitter to determine which individuals it should seed a review unit or coupon code. The business would ideally like to achieve the largest “bang for its buck” and for the fewest number of units or codes, reach the maximum number of people. Partitioning the graph to maintain an $n$-hop guarantee allows for quickly determining the individuals that, for a given number of levels, reach the most individuals. This naturally has important implications for business that would like to reach as many individuals as possible at cheapest cost.

In order to come up with an improved partitioning scheme, I will be determining the most meaningful structural links between users and weighting them (i.e. considering retweet links to be stronger than follow links, etc.) as well as employing semantic analysis to determine relatedness of Tweets. An existing problem that will need to be determined with the Twitter graph is how to handle extremely popular profiles, like the aforementioned Gaga. Because of the sheer volume of following profiles, these vertices must be treated differently: At present the two prevailing ideas are either to split the vertices such that there are multiple Lady Gaga’s that share no followers or they can be ignored completely.

References

Grzegorz Malewicz, Matthew H. Austern, Aart J. C. Bik, James C. Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski: “Pregel: A System for Large-Scale Graph Processing”

Haewoon Kwak, Changhyun Lee, Hosung Park, and Sue Moon: “What is Twitter, a Social Network or a News Media?”

Jiewen Huang, Daniel J. Abadi, Kun Ren: “Scalable SPARQL Querying of Large RDF Graphs”
Jeffrey Dean and Sanjay Ghemawat: “MapReduce: Simplified Data Processing on Large Clusters”

Twitter: http://www.twitter.com/

LinkedIn: http://www.linkedin.com