RESEARCH REVIEW JOURNALS
Volume-03
Issue-12
December -2018

ISSN: 2455-3085 (Online)
RESEARCH REVIEW International Journal of Multidisciplinary
www.rrjournals.com [UGC Listed Journal]

A Systematic Survey on Graph Analytics on Big Data: Current State and Future Challenges

[1]Sai Prasad Padavala & [2]Dr. Suresh Chand Tyagi

[1]*Research Scholar Of Sri Satya Sai University*
[2]*Executive Director IDC Foundation*

ARTICLE DETAILS

Article History
Published Online: 10 December 2018

Keywords
graph analytics; big data; graph simulation; parallel and distributed algorithms

ABSTRACT

Graphs enjoy profound importance because of their versatility and expressivity. They can be effectively used to represent social networks, web search engines and genome sequencing. The field of graph pattern matching has been of significant importance and has wide-spread applications. Conceptually, we want to find sub graphs that match a pattern in a given graph. Much work has been done in this field with solutions like Sub graph Isomorphism and Regular Expression matching. With Big Data, scientists are frequently running into massive graphs that have amplified the challenge that this area poses. We study the speedup and communication behavior of three distributed algorithms for inexact graph pattern matching. We also study the impact of different graph partitioning's on runtime and network I/O. Our extensive results show that the algorithms exhibit excellent scalable behavior and min-cut partitioning can lead to improved performance under some circumstances, and can drastically reduce the network traffic as well.

1. Introduction

Graphs are of utmost importance in Computer Science because of their expressivity and the ability to abstract a huge class of problems. They have been successfully used to study and model numerous problems in different fields. These applications vary from software plagiarism detection, web search engines, study of molecular bonds to the modeling of social networks like Facebook, LinkedIn and Twitter [1]. Graph pattern matching is one of the most important and widely studied class of problems in graphs. A considerable amount of research has been put into this area, sprouting concepts like Subgraph Isomorphism, Regular Expression matching [2] and Graph Simulation [3]. Conceptually, pattern matching algorithms seek to find subgraphs of a given graph that are similar to the given query graph [4]. Though, Subgraph Isomorphism returns the strictest matches for graph matching in terms of topology [5], the problem is NP-complete [6], and thus does not scale well. Graph simulation, on the other hand, provides a practical alternative to subgraph isomorphism by relaxing the stringent matching conditions of subgraph isomorphism, and allowing matches to be found in polynomial time. Some researchers [7], [8] even argue that graph simulation is more appropriate than subgraph isomorphism for modern problems like social network analysis because it yields matches that are conceptually more meaningful. With the rapid advent of Big Data, graphs have transformed into huge sizes and are rapidly getting out of the grasp of conventional computational approaches. In this paper, we address the problem of graph pattern matching on such big graphs.

2. Graph Analytics: Current State and Future Challenges

Big data computing, already a market of seven billion dollars in 2011 is projected to increase to 50 billion dollars within six years [1]. It is crucial to the success of not only internet companies, e.g. Amazon, Twitter and Facebook, but also traditional business such as Wal-Mart and Bank of America, as well as government agencies. Furthermore, big data computing has become such a powerful paradigm that enables scientists across different disciplines to tackle challenging research problems. Two of most important big data applications are machine learning and graph analytics. For example, machine learning algorithms, e.g., collaborative filtering and topic modeling, are often used to improve user experience and increase the revenue. In the meantime, graph algorithms, such as Breath-First Search (BFS) and betweenness centrality, can be utilized for social network analysis and computational biology.Current big data computing systems fall into two major categories: batch processing (e.g., MapReduce and GraphLab) is able to analyze large volumes of on-disk data, but the processing time can be as long as several days and weeks; and streaming processing (e.g., Storm) can analyze in-memory data in a short period to time like milliseconds. While batch processing focuses on the large amount of historical data (Volume), streaming processing deals with the instantly generated data streams (Velocity). Both also need to address the issues like different data types (Variety) and uncertainty (Veracity).

Recently the Lambda Architecture shown in Figure 1 is proposed to combine the capability of batch and streaming processing for next-generation big data computing systems. The insight (the result of big data processing) is generated by merging the results from both pipelines. The lambda architecture, albeit an innovative design in itself, needs to tackle multiple challenges as big data continue to grow at an unexpected speed.

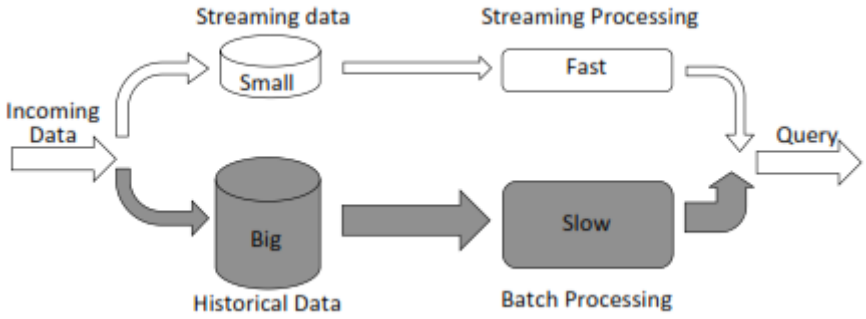


Fig. 1. Overview of the Lambda Architecture

3. Distributed algorithms for graph, dual and strong simulation

In this section, we give an outline of distributed algorithms for graph, dual and strong simulation that are designed for a vertex-centric system. Our implementation of strong simulation is an optimized version of the original strong simulation.

Graph Simulation: In the designed distributed algorithm for graph simulation, the query graph is distributed among all vertices of the data graph, and then each vertex should find out its match set among the vertices of the query graph. Vertex u ′ of the data graph matches to vertex u of the query graph if the two vertices have the same label, and each child of u has at least one match among the children of u ′ . In a vertex-centric system, a vertex initially knows only about its own label and the id of its children. Therefore, each vertex needs to communicate with its neighbors to learn about their labels and status in order to evaluate the child relationship condition. A Boolean flag, called match flag, is dedicated to each vertex which indicates if the vertex has a potential match among the vertices of the query graph. This flag is initially false. In an example displayed in figure 2, all the vertices of the data graph labeled a, b, and c make their flag true at the first super step, and then vertices 1, 2, and 5 send messages to their children. At the second super step only vertices 5, 6, and 7 will reply back to their parents. At the third super step, vertices 1, 5, 6, 7, and 8 can successfully validate their match set, but vertex 2 makes its flag false, because it receives no message from any child. Therefore, vertex 2 sends a removal message to vertex 1. This message will be received by vertex 1 at super step four. It will successfully reevaluate its match set, and the algorithm will finish at super step five when every vertex has voted to halt (there is no further communication).

Dual Simulation: The algorithm for dual Simulation is very similar to graph Simulation. In addition to child relationship, we extend the algorithm to check parent relationship as well.

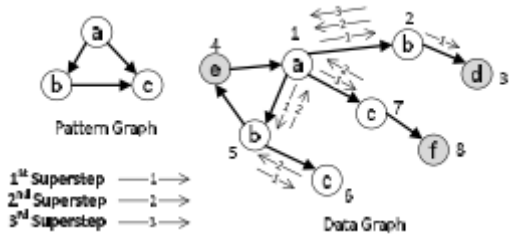


Figure 2: An example for distributed graph simulation

Strong Simulation: A version of strong Simulation is done in two phases: (1) we run dual simulation to find the match set R and then (2) for each vertex in R, we create a ball with a diameter of dq. Once we have all the balls ready, we run dual simulation on each to compute the final output of strong simulation.

The biggest challenge we faced in strong simulation was the creation of balls. Because of the scale of graphs, we may end up creating balls for many of vertices simultaneously which could bog down the whole system. Therefore, we tried two different approaches that we go through below

1) Depth-First Ball: As the name suggests, the ball creation process works in a depth-first fashion. In the first super step, each vertex left after dual simulation performs as the center of a ball and sends messages to its adjacent vertices with the specified ball size. The receiving vertices reply back with their labels and forward the message to their adjacent vertices with a decremented ball size. This process is repeated and these messages are removed from the system when the ball size reaches zero.

2) Breadth-First Ball: This approach works on a simple ping-reply model. In figure 3, let us suppose we want to create a ball around vertex X of radius 2. Initially, X only knows its adjacent node ids 1,7 and no label information. It starts off by sending a ping message to all of its adjacent nodes. In the second super step, all the recipient nodes replyback with their labels and the ids of their children and parents. Thus, in 3(b), X upon receiving these messages stores the gathered information about the vertices in its ball; e.g., labels and ids of its descendants and ascendants. In 3(c), it sends another ping message to all of its boundary vertices (vertices at a distance 2 of the center) which reply back with their labels and the ids of their children and parents, subsequently saved by the node X in 3(d). The drawback of this approach is that it results in almost twice the number of supersteps, yet it is much more efficient and performs much better than the other approach, therefore we adopted this approach for our strong simulation tests.

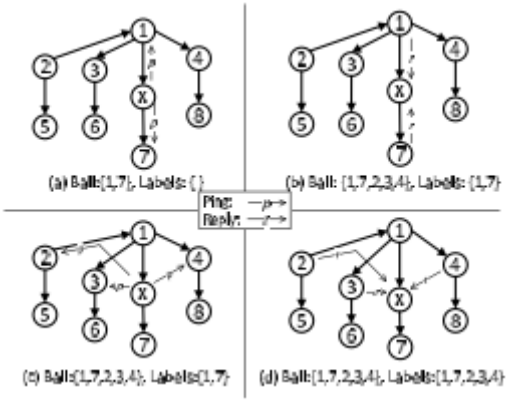


Figure 3: A breadth-first ball around vertex X with $d_q = 2$

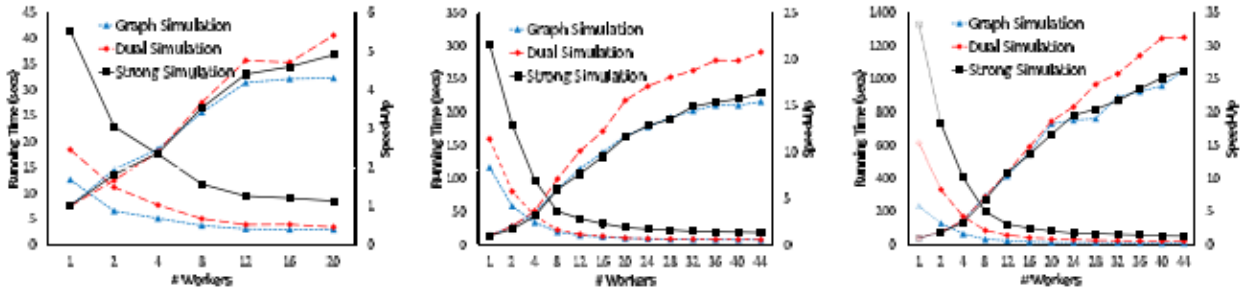
4. Implementation of distributed algorithms

In this section, we implement graph simulation on two different distributed computing infrastructures and compare their pros and cons.

A. GPS - Graph Processing System: As mentioned above, because all of the algorithms were designed with a BSP and vertex-centric model, we decided to use something akin to Pregel for the implementation. Of the numerous options available, we picked GPS for our algorithms since it offers everything that we desire of Pregel and is available as open-source. It is written in Java and also gives us an option to write a master.compute() method that proved to be quite handy in case of strong simulation. As in Pregel, there are two types of main components in the system: one master node and k worker nodes. A GPS job starts off by partitioning the data graph over all the participating workers. Every worker reads its partition and then distributes the vertices based on a round-robin scheme, i.e., vertex v gets assigned to worker W = v.id % k. The lifecycle of a GPS job can be summarized in following steps: (a) parse the input graph files, (b) start a new superstep, and (c) terminate computation when all the vertices have voted to halt and there are no messages in transit, otherwise go to (b).

B. Akka: Akka is a toolkit and runtime for building highly concurrent, distributed, and fault tolerant event driven applications on the JVM. It has an extended API that lets you manage service failures, load management (back-off strategies, timeouts and processing-isolation). It also scales well with increases in the number of cores and/or number of machines. The API is available both in Java as well as Scala.

C. GPS vsAkka: GPS delivers on its promise of Pregel. However, to enable the message passing for the custom types requires substantial effort that is not only time-consuming, but is also prone to errors. For example, if the message contains some complextypes, we need to be very careful with how the message is serialized at the sending vertex and then deserialized at the receiving end. Implementation detail of serializer and deserializer can be cumbersome, hard to maintain and not so readable.



(a) enwiki-2013, $|V| = 4x10^6$ (b) uk-2005, $|V| = 3.9x10^7$ (c) Synthesized, $|V| = 10^8, \alpha = 1.2$

Figure 4: Running times and speed-up for distributed algorithms, |Vq| = 25, αq = 1.2

Akka, unlike GPS is a general purpose toolkit for building highly concurrent and distributed applications and not something that is built ground-up only for graphs. It means there is some work that needs to be done to make Akka work in a fashion similar to BSP. Perhaps, the biggest edge that Akka has over other comparable models is its inherent ability and support for sending messages between actors. With Akka, the developers do not have to worry about serializing/deserializing of data. They can send messages wrapping complex types with extremely concise and terse syntax.

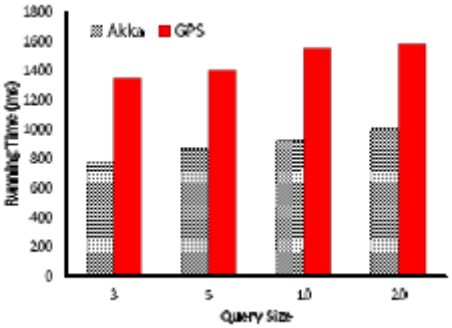


Figure 5: GPS and Akka on Graph Simulation

We implemented a prototype application for Graph Simulation using Akka. We were surprised by the power and ease provided by Akka for rapid development. Even with the most basic implementation, we were able to achieve much better times as we were getting from a system like GPS. The tests were conducted on the amazon-2008 [25] dataset and were run over a cluster of 5 machines. As can be seen in Figure 5, Akka ran almost twice as fast as GPS.

5. Conclusion

Graph pattern matching has been an important topic in the field of Computer Science and has been gaining prominence recently. It has become more challenging with the rapidly increasing size of graphs. In this paper, we study the three polynomial time pattern matching techniques in detail. In the future, we intend to look into different ways to improve the strong simulation running times on GPS and Akka. It is clear that we need to find out how we can improve the process of ball-creation. Instead of creating the ball on every dual-matched vertex, we intend to come up with techniques to reduce the total number of balls created without compromising the results. We also want to come up with new algorithms that do not suffer the synchronization bottlenecks of the BSP model, thus achieving better parallelism.

References

[1] A. Mislove, M. Marcon, K. P. Gummadi, P. Druschel, and B. Bhattacharjee, “Measurement and analysis of online social networks,” in Proceedings of the 7th ACM SIGCOMM conference on Internet measurement. ACM, 2007, pp. 29–42.

[2] Z. Sun, H. Wang, H. Wang, B. Shao, and J. Li, “Efficient subgraph matching on billion node graphs,” Proceedings of the VLDB Endowment, vol. 5, no. 9, pp. 788–799, 2012.

[3] M. R. Henzinger, T. A. Henzinger, and P. W. Kopke, “Computing simulations on finite and infinite graphs,” in Foundations of Computer Science, 1995.Proceedings., 36th Annual Symposium on. IEEE, 1995, pp. 453–462.

[4] A. Fard, U. Nisar, L. Ramaswamy, J. A. Miller, and M. Saltz, “Distributed algorithms for graph pattern matching,” http://www.cs.uga.edu/˜ar/abstract.pdf, Tech. Rep., 2013.

[5] B. Gallagher, “Matching structure and semantics: A survey on graph-based pattern matching,” AAAI FS, vol. 6, pp. 45–53, 2006.

[6] M. R. Garey and D. S. Johnson, Computers and Intractability; A Guide to the Theory of NP-Completeness. New York, NY, USA: W. H. Freeman & Co., 1990.

[7] J. Brynielsson, J. Hogberg, L. Kaati, C. Martenson, and ˚ P. Svenson, “Detecting social positions using simulation,” in Advances in Social Networks Analysis and Mining (ASONAM), 2010 International Conference on. IEEE, 2010, pp. 48–55.

[8] A. Fard, A. Abdolrashidi, L. Ramaswamy, and J. A. Miller, “Towards efficient query processing on massive time-evolving graphs,” in Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 2012 8th International Conference on, oct. 2012, pp. 567 –574.

[9] D. R. Zerbino and E. Birney, “Velvet: algorithms for de novo short read assembly using de bruijn graphs,” Genome research, vol. 18, no. 5, pp. 821–829, 2008.

[10] S. Ma, Y. Cao, W. Fan, J. Huai, and T. Wo, “Capturing topology in graph pattern matching,” Proceedings of the VLDB Endowment, vol. 5, no. 4, pp. 310–321, 2011.

[11] D. Conte, P. Foggia, C. Sansone, and M. Vento, “Thirty years of graph matching in pattern recognition,” International journal of pattern recognition and artificial intelligence, vol. 18, no. 03, pp. 265–298, 2004.

[12] J. R. Ullmann, “An algorithm for subgraph isomorphism,” Journal of the ACM (JACM), vol. 23, no. 1, pp. 31–42, 1976.

[13] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento, “A (sub) graph isomorphism algorithm for matching large graphs,” Pattern Analysis and Machine Intelligence, IEEE Transactions on, vol. 26, no. 10, pp. 1367–1372, 2004.

[14] D. Gregor and A. Lumsdaine, “The parallel BGL: A generic library for distributed graph computations,” Parallel ObjectOriented Scientific Computing (POOSC), 2005.

[15] A. Chan, F. Dehne, and R. Taylor, “Cgmgraph/cgmlib: Implementing and testing CGM graph algorithms on pc clusters and shared memory machines,” International Journal of High Performance Computing Applications, vol. 19, no. 1, pp. 81– 97, 2005.

[16] J. Dean and S. Ghemawat, “Mapreduce: simplified data processing on large clusters,” Communications of the ACM, vol. 51, no. 1, pp. 107–113, 2008.

