International Journal of Electronics, Communication & Soft Computing Science and Engineering
ISSN: 2277-9477, Volume 3, Issue 7

Review on Complex Analytics on Large Scale Graph Structure Data in Cloud

Harsha J. Kolhe **Amitkumar Manekar**

Abstract — **Nowadays with the advent of the internet there is an increasing interest in executing rich and complex analysis tasks over large scale graphs. Examples of these tasks are ego network analysis, motif counting, finding social circles, anomaly detection and so on. This work are not well served by the vertex centric approach and by using this approach there is a problem of high communication, scheduling and memory overheads.**
There is a NScale, a novel end to end graph processing framework that is used to enables the distributed execution of complex sub-graph centric analytics over large scale graphs in the cloud.

Key Words- **cloud computing, graph analytics, NScale, Vertex centric framework.**

I. INTRODUCTION

Graphs are very attractive when it comes to modeling real world data , because they are flexible more than tables and rows in a RDBMS. Representing information network data as a graph is most natural with nodes representing the entities and edges denoting the interaction between them. There is growing need for executing complex analytics over such graph data to get valuable insights into the networks functional abilities, for scientific discovery, for event or anomaly detection and so on. As the world is moving towards an evolution of Big Data, representing such data in forms of graphs and performing graph analytics over such large volumes of graph data has become a crucial task.
Developing distributed graph processing frameworks for such tasks is being adopted widely everywhere. Consider a simple example of social network which uses graphs for its representation. Social networks are naturally modeled as graphs where entities are the nodes, and an edge connects two nodes if the nodes are related by the relationship that characterizes the network. The existing frameworks use vertex-centric approach but underperform due to large communication overheads and slow iterative convergence. Instead a novel approach called NSCALE[8] is suggested which uses sub-graph-centric framework. This approach allows users to write programs at neighborhood or sub-graph level. NSCALE uses Apache YARN[8], a state-of-the-art data processing framework, for efficient and fault-tolerant distribution of data and computation.

II. LITERATURE SURVEY

A large number of complex analysis tasks on graphs needs to be applied to data concerning online social networks, road transportation , citation networks, biological networks, IP traffic data, communication and messaging networks, financial networks, and many others to extract different types of results like ego network analysis[7], motif counting[2], finding social circles[1], personalized recommendations, link prediction, anomaly detection[6],, and so on. There are various approaches adopted for large graph processing like Pregel[3], Apache Giraph, Grace[5].
In these frameworks, user write vertex level programs that are executed by the framework in asynchronous fashion. Communication between vertices is done using message passing or shared memory & parallelism is controlled by the chosen consistency model. But the models used in all these approaches limit the user program's access to a single vertex's state or sometimes to the neighbor's local state in addition.
example, to analyze a small part of a graph say a neighborhood of a node, we have to gather all the information from neighbors through message-passing and reconstruct those neighborhoods locally in the vertex program local state. This involves a huge communication overhead and high memory requirements which arise from duplication of state. These overheads will increase with increasing size of the network to be processed.

III. NSCALE APPROACH

NSCALE allows users to write programs at the level of a subgraph rather than a vertex level[8]. It allows users to specify:
(a) a set of sub-graphs or neighborhoods of interest, using a high level specification language, and
(b) a user-specified program that should be executed on those sub-graphs, potentially in an iterative fashion.

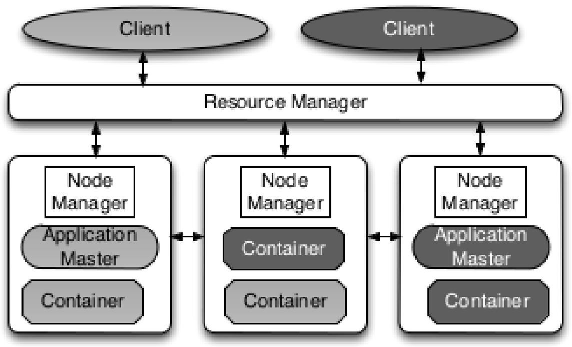

Fig.1. YARN Architecture

The user program is written against a general graph API (specifically, BluePrints), and has access to the entire state of the sub-graph[13] against which it is being executed. The approach can be used to compute the connected components in all the sub-graphs of interest. It uses specification format that allows users to specify sub-graphs of interest as k-hop neighborhoods around a set of query vertices, followed by a filter on the nodes and the edges in the neighborhood. It also allows selecting subgraphs induced by certain attributes of the of the nodes.
NScale ensures that each of the sub graphs of interest was entirely in memory at one of the machines while it is being executed against. It focuses on one-pass analytics that do not require iterative execution unlike the previous approaches. GEL is the graph extraction and loading layer which extracts the relevant data from the underlying graph, and employs a cost-based optimizer for data replication and placement. In doing this it aims to minimize the number of machines needed and balances load amongst them.
NScale uses a distributed execution engine that executes user-specified computation on the subgraphs in distributed memory. The execution engine finds out the overlap between subgraphs and then employs several optimizations that reduce the memory duplications without compromising correctness. NScale avoids iterative execution by using one-pass analytics. NScale uses YARN which is architecture for distributed execution as shown in fig 1. It is used for computation and data distribution.
The YARN framework provides transparent data and computation distribution, and fault tolerance for NScale. YARN(Yet Another Resource Negotiator) supports multiple workloads.
The Resource Manager is responsible for resource allocation. Node Manager forms data computation framework. A framework specific library called Application Master is responsible for individual applications execution and monitoring. Frameworks can interact with YARN using Hadoop. Thus YARN provides better scalability and fault tolerance for NSCALE.

IV. NSCALE ARCHITECTURE

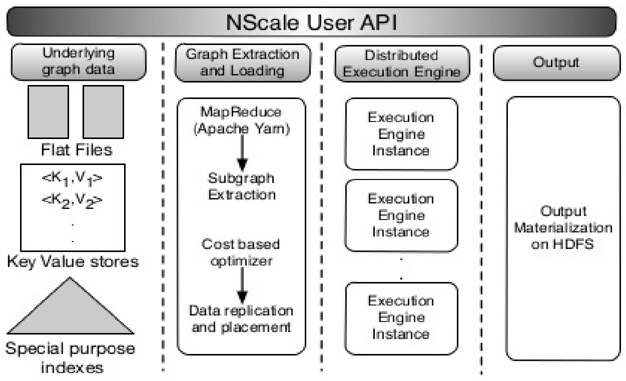
NScale User API
Underlying graph data
Flat Files
<K1,V1>
<K2,V2>
Key Value stores
Special purpose indexes
Graph Extraction and Loading
MapReduce (Apache Yarn)
Subgraph Extraction
Cost based optimizer
Data replication and placement
Distributed Execution Engine
Execution Engine Instance
Execution Engine Instance
Execution Engine Instance
Output
Output Materialization on HDFS

Fig. 2. High level overview of NScale

The main components of NScale architecture (as shown in fig.2) are GEL, the graph extraction and loading module and the distributed execution engine which are woven together by the YARN framework[8] .The topmost layer is the NScale user API to used by developers to specify the subgraphs of interest and the kernel computation that needs to be run on the subgraphs. NScale supports the storage of the underlying graph in a variety of different formats including edge lists, adjacency lists, and in a variety of different types of persistent storage engines including key-value pairs, specialized indexes stored in flat files, relational databases, etc.

Graph Extraction Engine-
GEL is a phase in which graph extraction and loading is performed which extracts the relevant portions of the graph and utilizes a cost-based optimizer to partition and load the graph onto distributed memory using as few machines as possible to minimize the communication cost. GEL processes the raw graph data and extracts the relevant portions, i.e., the sub-graphs of interest from the underlying graph. GEL acts as a 2-stage MapReduce (MR) job over YARN. Graph compression and sub-graph extraction is used to reduce the size of the graph.

Distributed Execution Engine-
Another major component of NScale is Distributed Execution Engine. It runs as a runtime library inside each reducer and uses master-slave concept. The partition-master identifies the subgraphs specified by the user and owned by the partition using parameters passed by the reducer.

CONCLUSION

Previous vertex-centric are limited in their ability to express and efficiently execute complex and rich graph analytics tasks. NScale proposes a sub-graph-centric framework where the users can write computations against entire sub-graphs or multi-hop neighborhoods in the graph and provide ease-of-use and efficiency. Also the graph extraction and loading phase saves total execution time for small where when Apache Giraph fails.

REFERENCES

[1] J. McAuley and J. Leskovec. Learning to Discover Social Circles in Ego Networks. In NIPS, 2012.
[2] R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, and U. Alon. Network motifs: Simple building blocks of complex networks. Science, 2002.
[3] G. Malewicz, M. H. Austern, A. J.C Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski. Pregel: a system for large-scale graph processing. In SIGMOD, 2010.
[4] G. Wang, W. Xie, A. J. Demers, and J. Gehrke. Asynchronous Large-Scale Graph Processing Made Easy. In CIDR, 2013.
[5] Leman Akoglu, Mary McGlohon, and Christos Faloutsos. OddBall: spotting anomalies in weighted graphs. In PAKDD, 2010.
[6] L. Backstrom and J. Leskovec. Supervised random walks: Predicting and recommending links in social networks. CoRR, 2010.
[7] Martin Everett and Stephen P Borgatti. Ego network betweenness. Social networks, 27(1):31–38, 2005.
[8] NScale: Neighborhood-centric Large-Scale Graph Analytics in the Cloud Abdul Quamar University of Maryland Amol Deshpande University of Maryland Jimmy Lin University of Maryland.
[9] Y. Tian, A. Balmin, S. A. Corsten, S. Tatikonda, and J. McPherson. From ”Think Like a Vertex” to ”Think Like a Graph”. PVLDB, 2013.
[10] J. Seo, S. Guo, and M. S. Lam. Socialite: Datalog extensions for efficient social network analysis. In ICDE, 2013.
[11] S. Salihoglu and J. Widom. GPS: a graph processing system. In SSDBM, 2013.
[12] Tomonori Izumi,ToshihikoYokomaru,Atsushi Takahashi, and Yoji Kajitani. Computational complexity analysis of set-bin-packing problem. IEICE TRANSACTIONS on Fundamentals of Electronics, Communications and Computer Sciences, 81(5):842–849, 1998.
[13] N. Kashtan, S. Itzkovitz, R. Milo, and U. Alon. Efficient sampling algorithm for estimating subgraph concentrations and detecting network motifs. Bioinformatics, 2004.