International Innovative Research Journal of Engineering and Technology

ISSN NO: 2456-1983

International Conference on Emerging Innovation in Engineering and Technology

ICEIET-2017

Graph Processing On A Distributed System For A Massive Scale Using Epicg

P. Mathivanan[1],S.Divya Bharathi[2],B.Hemalatha[2], G.Gayathri[2]

[1]Assistant Professor, [2]Student, Department of Information Technology, Manakula Vinayagar Institute of Technology, Pondicherry

I. ABSTRACT

A large numbers of focused graph processing systems have been developed to cope with the increasing stipulate on graph analytics. Most of them require users to systematize a new framework in the cluster for graph processing and toggle to other systems to accomplish non-graph algorithms. This increases the involvedness of cluster management and results in unnecessary data transfer and redundancy. In this paper, we propose our graph processing engine, named as epiCG, which is built on top of epiC, an flexibility data processing system. The core of epiCG is a new unit called GraphUnit, which is able to not only perform iterative graph processing streamlined, but also collaborate with other types of units to accomplish any complex/multi-stage data analytics. The epiCG supports both edge-cut and vertex-cut partitioning methods, and for the latter method, we propose a novel light-weight greedy strategy that enables all the GraphUnits to generate vertex-cut partitioning in parallel. moreover, disparate existing graph processing systems, malfunction revitalization in epiCG is absolutely automatic. We compare epiCG with several graph processing systems via extensive experiments with real-life dataset and applications. The results show that epiCG possesses high orderliness and scalability, and performs tremendously well in large dataset settings, Proclaim its correctness for large-scale graph processing.

Keywords: EpiCG, epic, Vertex-cut, partitioning, Graph Unit, graph processing systems

II. INTRODUCTION

The increasing insists of graph analytics is the foreseeable effect of the growing large scale and magnitude of graph data. Big graph examples including social network graphs, email and illustration message graphs typically involve billions of vertices and edges. For example, Facebook has over 1.5 billion monthly active users at present. In recent years, a large number of particular distributed graph processing systems such as Pregel [1], Power- Graph [2], Giraph [3] and GPS [4] have been proposed to handle complex graph analytics tasks. These specialized systems gain their popularities for two reasons. First, they follow the vertex centric programming model introduced by Pregel that allows users to articulate various graph algorithms in a natural way. Second, most graph processing systems are designed for iterative computation and are in-memory processing systems [5] since they hold the graph data in memory during iterations. Therefore, such systems outperform the general purpose distributed systems such as MapReduce [6,7] and its open-source implementation Hadoop [8] that typically flush data to the distributed file system at the end of each iteration and reload data into memory in the beginning of the next iteration.

The epiC system

epiC [12] was proposed to address data variety challenge in Big Data. It adopts the Actor-like programming model and provides a simple yet competent unit crossing point to support various computation models. In epiC, users can express different computation logics by central different units. Each processing unit performs computation in parallel with other units and communicates with other units through message passing. Messages, however, cannot be sent directly between two units; each message is sent to the master network (which is responsible for routing messages) and then forwarded to the corresponding units. A unit becomes active if it receives messages from the master network. After parsing the message, the unit will load data from the underlying storage and apply the user defined computation logic to process the data accordingly. When the unit completes its computation, it flushes the output to the storage and becomes inactive until it receives a new message.epiC adopts the master-worker architecture. There is only a single master node in epiC (this is different from the nodes in the master network which are mainly responsible for message routing). The master node runs a master daemon which monitors the healthy statues of the slaves and commands workers to execute tasks. Each worker node establishes a worker tracker daemon. The worker tracker manages a pool of worker processes which accept and execute assigned unit tasks. In epiC, we assign one unit task to one worker process.

Distributed graph processing

The key idea of distributed graph processing is a vertex-centric programming model proposed by Pregel [1], which abstracts graph algorithms as the computation for every vertex and message ex- change between different vertices. Typically, the execution of a graph job consists of three phases: data loading, iterative computation and the output. In the data loading phase, an input graph is loaded from the underlying storage system and distributed among the compute nodes. During iterative computation, each compute node sequentially scans its received vertices and executes a user- defined compute function for each of them. Every vertex can update its value and send messages to other vertices during the computation. Pregel-like systems follow Bulk Synchronous Parallel (BSP) model [13] and all the compute nodes will proceed to the next iteration synchronously, while some systems such as Power- Graph [2] perform computation in

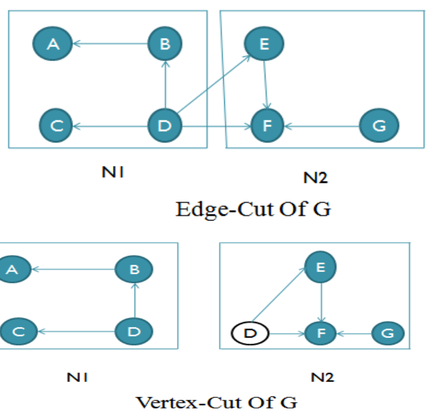

Fig. 1. Two graph partitioning methods.

an asynchronous manner. Finally, in the output phase, compute nodes flush the results (e.g., the computed values of the vertices) to the storage system. Essentially, distributed graph processing distributes graph data and parallelizes computation tasks among a cluster of compute nodes, thus accelerating the processing significantly.

III. RELATED WORK

Distributed graph processing. There have been a large number of systems proposed for graph processing. Pregel [1] follows the Bulk Synchronous Parallel (BSP) model and introduces a vertex- centric programming model that allows users to express graph algorithms in a natural way. Later on, various implementation of Pregel have been developed. Giraph [3] treats the computation as a map-only job in MapReduce framework [6] where the in- put and output data are stored in HDFS. Bu et al. [18] proposed Pregelix, which runs iterative dataflows dealing with computations, to conduct graph analysis. Yan et al. implemented Pregel+, a C/C++ graph system eliminating serialization cost introduced by Java. Graphlab introduces a shared memory abstraction which allows the adjacent vertices and edges to be accessed by the local vertex and hence enables users to concentrate on the sequential computation by hiding the details of data movement between vertices.
Since synchronization is indispensable in BSP model, the stragglers, i.e., the workers who run much slower than others, can significantly slow down the synchronization process. Hence, Sal- ihoglu et al. [4] introduced GPS which follows the same storage design with Giraph but provides two optimizations. The first one is to maintain dynamic partitions among workers, and the second one is to divide the adjacency list of high-degree vertices into different workers to balance the loads of workers. Graph lab and PowerGraph [2] employ Gather-Apply-Scatter (GAS) model and can run in synchronous or asynchronous mode. Asynchronous mode migrates the stragglers by eliminating global synchronization cost. However, this increases system complexity due to the concurrency control for serializability.
Recent efforts focused on leveraging existing platforms for graph processing. Simmen et al. [24] introduced Aster 6 which provides SQL-like interface for graph analytics. Similarly, Fan et al. [17] proposed Grail, a syntactic layer for querying graph on top of RDBMS, which translates graph queries into SQL queries. GraphX [9] is built on Spark [10], which implements Pregel by leveraging general dataflow operators in Spark. To align the performance with the specialized graph processing systems, various optimizations for dataflow operators have been proposed. This is different from our work as we develop epiCG as one unit in epiC for graph processing and reply on epiC to leverage different units for complex analytics query processing. As an independent unit, epiCG allows us to implement all the optimizations proposed for the specialized graph processing systems.

Graph partitioning. Graph partitioning is critical to distributed graph processing. While various edge-cut partitioning methods [14] have been proposed to balance the computation workload among multiple compute nodes and try to minimize the network communication cost, less attention has been paid to vertex-cut partitioning. PowerGraph [2] proposed a greedy strategy to generate vertex-cut partitioning, which require to use a single compute node to load the entire graph into memory, thus restricting the size of graph that can be handled. SBV-cut and JA-BE- JA-VC are two recent works for distributed vertex-cut partitioning. However, both of them requires iterative computation over the entire graph, which does not allow multiple compute nodes to generate a vertex-cut partitioning in parallel.

IV. PROPOSED SYSTEM ARCHITECTURE

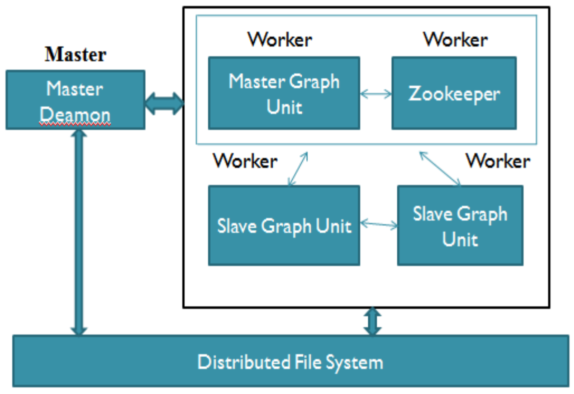

Figure 1: Proposed System Architecture

The architecture of epiCG. epiCG deploys the distributed file system (DFS) as its underlying storage. Typically, DFS contains the initial graph data to be processed by epiCG and the final results produced by epiCG. epiCG follows the single master- multiple worker architecture of epiC. The master runs a master daemon which maintains the healthy statuses of all the workers and instructs workers to execute *unit* programs. We divide workers into three categories to execute *masterGraphUnit*, *slaveGraphUnit* and *Zookeeper*, respectively. The *masterGraphUnit* coordinates su-persteps among all the workers who execute slaveGraphUnit *and* the *Zookeeper* maintains information shared among these workers, e.g., which worker has finished the execution of the current su-perstep, how many workers have dumped a checkpoint. Typically, given a set of workers, we choose one worker to run the *master- GraphUnit* program. The functionalities of the GraphUnits are listed as follows.**MasterGraphUnit.** the *MasterGraphUnit* program performs two tasks:

1. Partition and distribute the input graph among the workers that run *SlaveGraphUnit program*.
2. Coordinate the SlaveGraphUnit *workers* to perform super steps synchronously.

To perform these tasks correctly, *MasterGraphUnit maintains* several important objects.

- **MasterPartitioner:** generate the vertex-to-partition mapping list several important data structures managed by each for the input graph.
- **MasterClient:** notify workers of the newly computed global aggregated values.
- **MasterAggregator:** retrieve local aggregated values from the workers and generate the global aggregated ones.

SlaveGraphUnit. The SlaveGraphUnit program is responsible for the following four tasks.

- **WorkerServer:** Retrieves and manages the graph data that is assigned to the worker.
- **WorkerPartitioner:** Maintains partition information for the vertices residing in the worker.
- **WorkerClient:** Forwards messages to the zookeeper and other workers.
- **WorkerAggregator:** Computes aggregated values and writes to the zookeeper.

SlaveGraphUnit maintains four important objects:

1. Load its assigned graph data and flush computation results from/to the storage system.
2. Loop over vertices and execute *compute ()* function.
3. Forward messages generated during the computation.
4. Generate aggregated values and write to the zookeeper.

Once a graph job is submitted to epiCG, all the workers will be activated immediately. At the beginning of the execution, epiCG establishes pair wise connections between GraphUnits. This is different from epiC where units cannot communicate with each other directly, but rely on the message service provided by the master network. As most graph applications such as PageRank and shortest path computation involve a large number of messages, setting up direct connections between units allows them to communicate with each other more streamlined and prevents the master network being the bottleneck

.

ALGORITHM 1:GENERATE VERTEX CUT

```
INPUT:  v, a vertex
        φp, PartitionOwnerList
        n, the number of Partitions
OUTPUT:M:list of(copy,partition)pairs
1   M← θ;
2   E_V←v.GetEdges();
3   if |E_V|≤θ then
4   M←{(v,GetPartition(v.vid))};
5   Else
6   /* generate vertex-cut for v*/
```

```
7  $N \leftarrow \theta$;
8  foreach Edge(v,u)€ $E_V$ do
9  W←GetWorkerInfo(GetPartition(u.vid), $\varphi p$);
10 $v^r \leftarrow$N.get(w);
11 If $v^r$= null then
12 $v^r \leftarrow$CreateVertex(v);
13 $N \leftarrow N \cup \{(W, v^r)\}$;
14 $v^r$.AddEdge(($v^r$,u));
15 /* select master vertex and assign copies to partition */
16 $W^r \leftarrow$GetWorkerInfo(GetPartition(v.vid), $\varphi p$);
17 $v_m \leftarrow$null; $p_m \leftarrow \theta$;
18 Foreach(w,$v^r$)€ N do
19 If W=$W^r$ then
20 $V^r$.isMaster←true;
21 $M \leftarrow M \cup \{v^r$,GetPartition(v.vid)$\}$;
22 $v_m \leftarrow v^r$;
23 Else
24 $v^r$.isMaster←false;
25 P←ChooseOnePart ition(W);
26 $M \leftarrow M \cup \{(V^r,P)\}$;
27 $P_m \leftarrow P_m \cup \{$P.pid$\}$;
28 $V^r$.#allEdges← $|E_V|$;
29 If W /€ N.key Set() then
30 $v^r \leftarrow$CreateVertex(v);
31 $M \leftarrow \{v^r$,GetPartition(v.vid)$\}$;
32 $V_m$.AddMirrorPartitionIds($P_m$);
```

V. CONCLUSION

In this paper, we present our distributed graph processing engine epiCG. We develop epiCG as one extension of epiC to avoid extra configuration for a new framework. epiCG supports both edge-cut and vertex-cut partitioning, and a light-weight approach is employed in epiCG to parallelize the generation of vertex- cut partitions. epiCG also allows automatic failure detection and recovery. The experiments on real-life datasets illustrate the high efficiency and scalability of epiCG, compared with three state-of- the-art distributed graph processing systems such as, Giraph, PowerGraph and GiraphX.

REFERENCES

[1] G. Malewicz, M.H. Austern, A.J.C. Bik, J.C. Dehnert, I. Horn, N. Leiser, G. Cza- jkowski, Pregel: a system for large-scale graph processing, in: SIGMOD, 2010.

[2] K. Lang, Finding good nearly balanced cuts in power law graphs, Tech. Rep.YRL-2004-036, Yahoo! Research Labs, 2004. [3] http://giraph.apache.org/.

[4] S. Salihoglu, J. Widom, Gps: a graph processing system, in: SSDBM, ACM, New York, NY, USA, 2013, pp. 22:1–22:12.

[5] H. Zhang, G. Chen, B.C. Ooi, K.-L. Tan, M. Zhang, In-memory big data man- agement and processing: a survey, IEEE Trans. Knowl. Data Eng. 27 (7) (2015) 1920–1948.

[6] J. Dean, S. Ghemawat, Mapreduce: simplified data processing on large clusters, in: OSDI, 2004.

[7] F. Li, B.C. Ooi, M.T. Özsu, S. Wu, Distributed data management using mapre- duce, ACM Comput. Surv. (CSUR) 46 (3) (2014) 31.

[8] http://hadoop.apache.org/.

[9] J.E. Gonzalez, R.S. Xin, A. Dave, D. Crankshaw, M.J. Franklin, I. Stoica, Graphx: graph processing in a distributed dataflow framework, in: OSDI, 2014, pp. 599–613.

[10] M. Zaharia, M. Chowdhury, M.J. Franklin, S. Shenker, I. Stoica, Spark: cluster computing with working sets, in: HotCloud, 2010.

[11] J.E. Gonzalez, Y. Low, H. Gu, D. Bickson, C. Guestrin, Powergraph: distributed graph-parallel computation on

natural graphs, in: OSDI, 2012.

[12] D. Jiang, G. Chen, B.C. Ooi, K.-L. Tan, S. Wu, epiC: an extensible and scalable system for processing big data, Proc. VLDB Endow. 7 (7) (2014) 541–552.

[13] L.G. Valiant, A bridging model for parallel computation, Commun. ACM 33 (8) (1990) 103–111.

[14] G. Karypis, V. Kumar, Metis-unstructured graph partitioning and sparse matrix ordering system, version 2.0.

[15] R. Chen, J. Shi, Y. Chen, H. Guan, H. Chen, Powerlyra: differentiated graph com- putation and partitioning on skewed graphs, Tech. rep., 2013.

[16] Y. Shen, G. Chen, H.V. Jagadish, W. Lu, B.C. Ooi, B.M. Tudor, Fast failure recov- ery in distributed graph processing systems, Proc. VLDB Endow. 8 (4) (2014) 437–448.

[17] J. Fan, A.G.S. Raj, J.M. Patel, The case against specialized graph analytics engines, in: CIDR, 2015.

[18] Y. Bu, V.R. Borkar, J. Jia, M.J. Carey, T. Condie, Pregelix: big(ger) graph analytics on a dataflow engine, Proc. VLDB Endow. 8 (2) (2014) 161–172.

[19] C. Xie, R. Chen, H. Guan, B. Zang, H. Chen, Sync or async: time to fuse for distributed graph-parallel computation, in: PPoPP, 2015, pp. 194–204.

[20] L. Page, S. Brin, R. Motwani, T. Winograd, The pagerank citation ranking: bring- ing order to the web, Tech. rep., 1999.