Performance Analysis Of Graph Processing Frameworks

By

KOMPELLY HARSHAVARDHAN REDDY

Department of Computer Science
INDRAPRASTHA INSTITUTE OF INFORMATION TECHNOLOGY

A dissertation submitted to the Indraprastha Institute of Information Technology in accordance with the requirements of the degree of MASTER OF TECHNOLOGY in the Department of Computer Science.

AUGUST 2015

Abstract

Graphs have always been an interesting structure to study in both mathematics and computer science , and have become even more interesting in the context of online social networks, recommendation networks whose underlying network structures are nicely represented by graphs.The graphs are massive: Facebook social graph has billions of vertices and web graphs are much larger.With “large” graphs comes the desire to extract meaningful information from these graphs. In the age of multi-core CPUs and distributed computing, concurrent processing of graphs proves to be an important topic.

Graph processing frameworks are being increasingly used to perform analysis on the enormous graphs like follower graphs in online social networks ,web graph ,recommendation graphs etc.Graphlab, FlashGraph ,PowerGraph,X-stream are few frameworks are used to compute metrics such as pageank ,shortest path etc on graphs. The lack of access locality when traversing edges makes it difficult to achieve good results in graph analysis.

To gain an understanding of how graph processing frameworks perform, we conduct a study to experimentally compare FlashGraph and Graphlab Create using several metrics.The systems are compared with three different algorithms (PageRank,weakly connected components,and Triangle counting) on single machine.Our evaluation shows that Graphlab create is performing better than FlashGraph.

Dedication and acknowledgements

I want to thank my advisor Dr.Vikram Goyal, an open minded professor who always motivated me to think harder and research deeply into any topic and who also trusted me with my results.I am extremely lucky to have a work with him. I also want to express my gratitude towards Dr.Pushpendra Singh. I am grateful to both of them for our interesting discussions about the problem I studied and also for their active participation.

AUTHOR'S DECLARATION

I declare that the work in this dissertation was carried out in accordance with the requirements of the University's Regulations and Code of Practice for Research Degree Programmes and that it has not been submitted for any other academic award. Except where indicated by specific reference in the text, the work is the candidate's own work. Work done in collaboration with, or with the assistance of, others, is indicated as such. Any views expressed in the dissertation are those of the author.

SIGNED: .... DATE: ....

Table of Contents

Page

List of Tables **ix**

List of Figures **xi**

1 Introduction **1**

2 Graph Processing and Challenges **3**

3 Computing Models **5**

3.1 Vertex Centric Programming Model 5

3.1.1 Edge Centric Programming model 6

4 Factors effecting performance of Graph Processing Frameworks **7**

5 Graph Processing Frameworks **9**

5.1 FlashGraph 9

6 Algorithms **15**

6.0.1 PageRank 15

6.0.2 WCC 16

6.0.3 Triangle Counting 16

7 Dataset **17**

8 Experimental Setup and Results **19**

8.0.4 Evaluation Metrics 19

8.0.5 Results 20

9 Future Work And Conclusion **31**

9.0.6 Conclusion 32

Bibliography **33**

Bibliography **35**

List of Tables

Table **Page**

6.1 Algorithms 15

7.1 Graphs 17

8.1 Load Time 21
8.2 Computation Time 21
8.3 Total Time 22
8.4 Max Main Memory used 23
8.5 Cpu Utilization 23
8.6 Computation Time 24
8.7 Total Time 25
8.8 Main Memory 26
8.9 CPU Utilization 27
8.10 Computation Time 27
8.11 Total Time 28
8.12 Main Memory 29
8.13 CPU Utilization 29

LIST OF FIGURES

FIGURE Page

5.1 Architecture of FlashGraph . . . 9
5.2 Data representation of directed graph in Flashgraph . . . 11
5.3 GraphLab create architecture . . . 12

8.1 graph . . . 20
8.2 Computaton Time . . . 21
8.3 Total Time . . . 22
8.4 Main Memory usage . . . 23
8.5 CPU utilization . . . 24
8.6 Computation Time . . . 24
8.7 Total Time . . . 25
8.8 Main Memory usage . . . 26
8.9 CPU Utilization . . . 27
8.10 Computation Time . . . 28
8.11 Total Time . . . 28
8.12 Main Memory usage . . . 29
8.13 CPU Utilization . . . 30

Chapter 1

Introduction

Real world graphs are massive. These graphs may contain millions to billions of vertices and edges. Large number of these graphs are directed graphs such as online social networks, the web graphs,recommendation graphs and others.Extracting meaningful information from these graphs is challenging and interesting. Efficient graph processing is very challenging task. Graph processing frameworks are being increasingly used to perform analysis on the large graphs that surrounded us today. Systems like Graphlab, LfGraph,X-Stream,FlashGraph, Graphlab create are used to compute metrics such as PageRank,Shortest path and to perform operations such as clustering and matching.These frameworks are vertex centric or edge centric and processing is iterative.

There are two types of graph analytic frameworks : i) Distributed graph analytic frameworks ii) Disk based frameworks .

Processing graph on cluster of machines is called Distributed graph processing.Graphlab,Powergraph,Lfgraph are some examples of distributed graph analytical frameworks. In distributed graph processing graph will be partitioned on cluster of machines and each machine will compute parallely. Second type is graph processing can be done on single machine where graph will be stored in secondary memory. X-Stream,FlashGraph are some examples of disk based graph processing frameworks. In disk based frameworks the input graph is partitioned on cores of machine ,and each core computes parallely.

Graph analytic frameworks must pay head to five essential aspects : i) computation ii)communication, iii)preprocessing, iv)memory, v)scalability

1. **Computation** : Computation overhead must be low and load balanced across the servers or cores of machine. It determines algorithm's computation time.It will be effected by number and distribution of edges across machines or cores of a system.

2. **Communication** : Communication overhead must be low and load balanced across the workers. This also determines per iteration time thus overall job completion time.It is affected by the quantity and distribution of data exchange among vertices across the machines.

3. **Preprocessing** : Prior to the first iteration Graphs need to be partitioned across workers. Partitioning time must be low.It determines the completion time of algorithm.

4. **Memory** : The memory footprint of machine must be able. With less memory should be able to process large graphs.

5. **Scalability** : Smaller cluster of machines or single system with less main memory should be able to load and process the graphs. By increasing main memory for large networks the computation time should be low.

Most of today's graph processing frameworks falls short in at least one of the above categories. In this thesis, we did performance analysis between two disk based graph processing frameworks namely Flashgraph and Graphlab Create. We evaluated the performance of frameworks based on metrics such as computation time, load time,total time, maximum Cpu utilization and Maximum main memory. We conducted pagerank,triangle counting and weakly connected components experiments on Amazon graph, Google web graph and Com-Live journal graph. We observed that Graphlab create is giving better results compared to FlashGraph in all the experiments.

In Section 2 we will discuss about how graph processing will be done ? ,challenges in graph processing. Section 3 describes about computing models.Sections 4 includes factors effecting performance of graph analytic frameworks. In section 5 we gave a brief description of the frameworks used. Section 6 includes algorithm,section 7 describes about data sets and section 8 includes experimental setup, results and Section 6 includes future work and conclusion.

Chapter 2

Graph Processing and Challenges

Initially the input graph is distributed or partitioned across multiple machines/worker threads. There are different types of graph partitioning methods like Hash based partitioning , Greedy partitioning method. Graph partitioning is used to decrease the computation time and communication overhead. Graph partitioning is NP- Complete problem . Frameworks like Lfgraph uses Hash based partitioning method. Powergraph uses Greedy partitioning method. Graph partitioning will try to provide same amount of workload on each system or thread.
Vertices send each other messages to perform computation. Computation is divided into iterations or super steps. In each iteration vertex executes some code and then communicates with its graph neighbors. So each iteration includes computation plus communication. Frameworks will shows the results once after completion of user specified number of iterations.
Efficient graph processing is a very challenging task.The following are the various challenges in efficient graph processing.

1. The data in graphs are unstructured and highly irregular. Most of the real world graphs follows power law distribution. The irregular structure of graph makes it difficult to extract parallelism by partitioning the graph. Scalability can be limited by unbalanced computation loads because of poor graph partitioning.

2. Graphs represent relationships between entities and these relations may be irregular and unstructured. The computations and data access patterns tend not to have locality.The lack of access locality when traversing edges makes it difficult to achieve good results in graph analysis.

3. Graph computations are often data driven.Structure of the computations in algorithm is not known before starting the algorithm. Because of this reason concurrent execution based

on partitioning of computation can be difficult.

4. Many graph algorithms are based on exploring the structure of a graph in preference to performing large numbers of computations on the graph data. As a result, there is a higher ratio of data access to computation than for scientific computing applications.Because of poor data locality , computation time can be dominated by the wait for memory fetches

5. Allocating the optimal resources to run the algorithm without failure is difficult. Poor allocation of resources may lead to high cost or high computation time.

Chapter 3

Computing Models

There are mainly two computing models which are used by many frameworks. They are i) Vertex centric Programming model ii) Edge Centric Programming model

3.1 Vertex Centric Programming Model

Frameworks like FlashGraph, Pregel, Lfgraph uses vertex centric programming model.Vertex centric programming model mainly contains three steps:i)update vertex data value based on received messages ii) Generate new messages for outgoing messages .iii)Send out messages to neighbors and vote for halt.
Vertex centric programming model runs in iterations. The start of each iteration is synchronized across servers or threads. During each iteration the vertex program for vertex v reads the value of incoming edges and performs computation specified by user defined function. If v value changes then it marked as active otherwise it marked as in active. The framework transmits active values to the servers or threads containing neighboring vertices of v. The computation terminates either at the first iteration when all vertices are in active or after pre-specified number of iterations.

Algorithm 1 Vertex centric programming

```
1: function VERTEXCENTRICPROGRAMMING(Vertex v)
2:     value[v] <- f(value[u]) , u belongs to in neighbor of v
3:     if value[v] is updated then
4:         send messagefor neighbor (value[v])
5:     end if
6: end function
```

3.1.1 Edge Centric Programming model

Algorithm 2 Edge centric programming

```
1: function SCATTER(Edgelist)
2:     for each edge e do
3:         if e.source has updated then
4:             scatter update on e
5:         end if
6:     end for
7: end function
8: function GATHER(update list of edge e)
9:     for each update u on edge e do
10:        if e.source has updated then
11:            e.destination = e.destination + u
12:        end if
13:    end for
14: end function
```

Frameworks like X-Stream uses Edge centric programming model to exploit sequential access.Edge centric programming model reads the edges in sequential order and access the vertices in random order. Generally in real world graphs number of edges will be more compared to vertices. So edge centric programming model decreases the random access. Edge centric programming model also runs in iterations.

The scatter method takes edge as input based on the edge and source vertex the function will return the value which will be propagated to destination vertex. The Gather method takes the update message as input and based on this value it recomputes the state of destination vertex.

Chapter 4

Factors effecting performance of Graph Processing Frameworks

Following are the various factors that will effect the performance of graph analytic frameworks.

1. Nature of Graph

   The sparsity structure of natural graphs presents a unique challenge to efficient graph parallel computation. In natural graphs most vertices have relatively few neighbors while some have many neighbors. Under power law degree distribution the probability that a vertex has degree d is given by

   $$p(d) \;\alpha\; d^{-\alpha}$$

   where the exponent α is a positive constant that controls the skewness of the degree distribution. Higher α implies lower density i.e majority of vertices will have lower degree. The skewed degree distribution implies that a small fraction of vertices are adjacent to a large fraction of edges.

   Natural graphs are difficult to partition. Many frameworks will depend on graph partitioning to minimize communication . The skewed degree distribution of vertices leads to communication asymmetry across the workers. The power law degree distribution can lead to substantial work imbalance in graph parallel abstractions that treat vertices symmetrically.

2. Type Of Algorithm

   Algorithms or applications also effects the performance of the graph analytic frameworks. There are different kind of algorithms like random walk, parallel traversal, sequential

traversal,graph mutations. In algorithms like random walk we will perform computations on all vertices in random walk model. We need to traverse all the vertices in random order.Sequential traversal algorithms find one or more paths in a graph according to some optimization criteria without updating or mutating the graph structure. Such algorithms typically start at one source vertex and, at super step i, process vertices that are distance i from the source. In parallel traversal,algorithms start by considering all vertices at once. Multiple vertices participate in each super step until they acquire some common information. In graph mutation algorithm algorithms graph structure will be changed during the computation. DMST algorithm is the example of Graph Mutation. Generally NP -Complete Algorithms will take more main memory and much computation time. Computation time directly depends on the type of algorithms ,complexity of algorithm .Communication overhead also increases in graph mutation algorithms.

3. Hardware

   Many real ‚Äì world graphs are massive. Facebook's social graph has billions of vertices.Google's web page graph has tens of billions of vertices. Analyzing or processing these kind of graphs require high random access memory.It is very hard to explore data locality in a graph. The lack of access locality when traversing edges make it difficult to achieve good results in graph analysis. Efficient graph analysis require high random access memory. Allocating optimal resources to a process is very tough task to do. Allocating more resources to a process may decrease the computation time but increases the cost of the operation. Allocating less resources my increase the response time and computation time. Allocating resources based on trial method is not feasible because it is cost consuming.

   System load also effects the performance of the framework. If load on the system is high, workers will take more computation time. Graph partitioning may leads to high load on some machines or workers less on some systems. If the graph partitioning is done in optimal way also the load on the systems vary because of the background processes running on those systems. High system load increases computation time and communication overhead.High system load can cause computation imbalance.

CHAPTER 5

GRAPH PROCESSING FRAMEWORKS

5.1 FlashGraph

FlashGraph is a semi external memory graph engine optimized for any fast I/O device such as Fusion I/O or arrays of solid state drives(SSD).FlashGraph enables us to process very large graphs in a single machine. It stores the edge lists of vertices on SSD's and maintains vertex in memory. FlashGraph runs on top of set associative file system to fully utilize the high I/O throughput provided by SSD array.

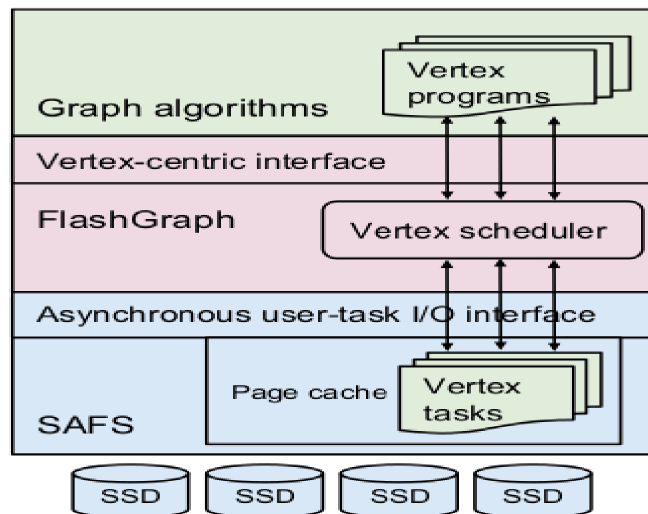

Figure 5.1: Architecture of FlashGraph

Figure 5.1 describes the architecture of FlashGraph. A graph algorithm in FlashGraph is

composed of many vertex programs that run inside the graph engine.Each vertex program represents a vertex and has its own user defined state and logic.The execution of vertex programs is subject to scheduling by FlashGraph.when vertex programs need to access data from SSD FlashGraph issues I/O request to SAFs n behalf of the vertex programs and pushes part of their computation to SAFS.

FlashGraph executes the algorithms in super steps(iterations).There are three possible states for a vertex: i) running, ii)active, iii) inactive. Vertex can be activated either by graph engine or other vertices.An active vertex enters running state when it is scheduled. Once active vertex scheduled, it remains in running state until it finishes task in current super step and becomes inactive.In each super step FlashGraph processes the vertices activated in previous super step.Algorithm ends when there are no active vertices in next super step. FlashGraph uses message passing to push the data into other vertices.It hides explicit synchronization from users and provides a more user friendly programming interface.

FlashGraph splits a graph into multiple partitions and assigns a worker thread to each partition to process vertices.Each worker thread maintains a queue of active vertices with in its own partition and executes user defined vertex program on them.The order of execution is done by FlashGraph's scheduler and it also guarantees fixed number of running vertices in a thread. The worker threads send and receive messages on behalf of vertices and buffer messages to improve the performance.To reduce memory consumption, FlashGraph process messages and pass them to vertices when buffer accumulates a certain number of messages. To avoid unnecessary duplication of messages FlashGraph uses multi-casting.With multi-cast, FlashGraph simply copies the same message once to each thread. FlashGraph provides a dynamic load balancer to identify computational imbalance created by high degree vertices in scale graphs. Once a thread processes active vertices in its own partition , it steals active vertices from other threads and processes them. This process continues until no threads have active vertices in current iteration.

Figure 5.2 describes the data representation of directed graph in FlashGraph. FlashGraph uses compact data representations both in memory and on SSDs.A smaller in- memory data representations allows FlashGraph to process a larger graph and use a larger SAFS page cache to improve performance.A smaller data representation on SSDs allows FlashGraph to pull more edge lists from SSDs in the same amount of time, resulting in better performance.

FlashGraph uses compact data representations both in memory and on SSDs. A smaller in-memory data representation allows us to process a larger graph and use a larger SAFS page cache to improve performance. A smaller data representation on SSDs allows us to pull more edge lists from SSDs in the same amount of time,resulting in better performance

FlashGraph maintains the following data structures in memory:(i)a graph index for accessing edge lists on SSDs;(ii)user-defined algorithmic vertex state of all vertices;(iii)vertex status used by FlashGraph;(iv)per thread message queues. To save space, FlashGraph choose to compute some vertex information at run time, such as the location of an edge list on SSDs and vertex ID.The

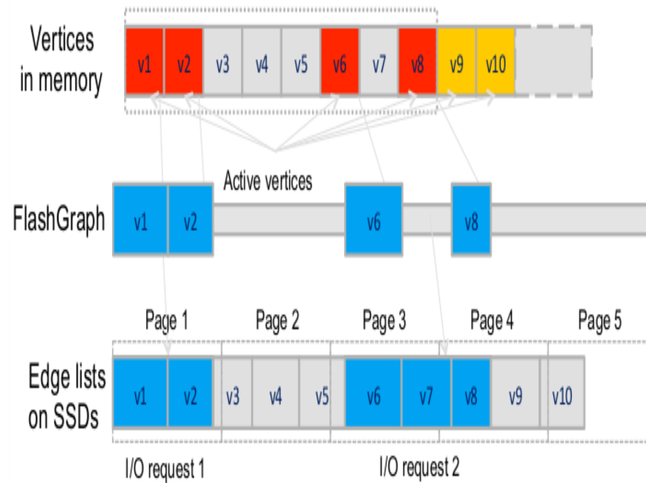

Figure 5.2: Data representation of directed graph in Flashgraph

graph index stores a small amount of information for each edge list and compute their location and size at run time . FlashGraph stores edges and edge attributes of vertices on SSDs. FlashGraph uses a single external memory data structure for all algorithms to reduce SSD wear out.When FlashGraph runs in the semi-external memory mode, it still has performance comparable to the state-of-art in-memory graph engines such as Galois and Ligra, while significantly faster than PowerGraph.

5.1.0.1 Graphlab Create

Graphlab Create works on a single multi-core machine. Graphlab Create is used to analyze terabyte scale data at interactive speeds, on single system.Graphlab Create is built on top of state-of-the-art technology in scalable data structures, powerful machine learning methods, intuitive visualization, and flexible deployment options. It is written in C++ for the best possible performance, with a Python interface for easy accessibility.

Figure 8.13 describes Graphlab create architecture. Graphlab Create's graph handling combines technologies and lessons learnt from GraphChi, PowerGraph and GraphX.Graphlab create provides more scalability than those frameworks.A vertex program on Graphlab create executed similar to Graphlab and Powergraph.Graphlab create exploits sequential access while accessing edge list from disks. PowerGraph is Distributed Graph analytic framework that eliminates the degree dependence of the vertex program by directly exploiting gather and scatter (GAS) decomposition to factor vertex programs over edges. Powergraph combines best features from both Pregel and Graphlab. From Graphlab Powergraph inherited the data graph and shared memory view computation eliminating the need for users to architect the movement of information .From Pregel, Powergraph inherited the commutative, associative gather concept . Powergraph uses

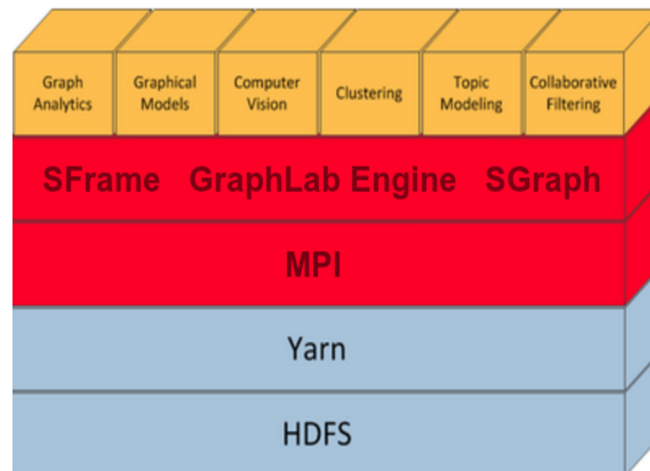

Figure 5.3: GraphLab create architecture

vertex centric programming model. Graphchi is a disk based system for computing efficiently on graphs with billions of edges. Graphchi uses vertex centric programming model. Graphx is an embedded graph processing framework built on top of apache spark distributed data flow system. Graphx also uses GAS programming model.

Graphlab Create can handle different kinds of data – graph, table, image, text, etc. Graphlab creates store the graph using SFrame infrastructure, so it is not exactly vertex centric or edge centric, it is a combination of both. Graphlab Create supports wide array of data types

1. SFrames

   SFrame is an efficient disk-based tabular data structure that is not limited by RAM. This lets you scale your analysis and data processing to handle terabytes of data, even on your laptop.SFrames act like a table by consisting of 0 or more columns. Each column has its own data type and every column of a particular SFrame must have the same number of entries as the other columns that already exist. There are two things that make SFrames very different from other data frames:

   a) Each column is an SArray, which is a series of elements stored on disk. This makes SFrames disk-based and therefore able to hold data sets that are too large to fit in your system's memory.

   b) An SFrame's data is located on the server that is running the Graphlab toolkits, which is not necessarily on your client machine.

   SFrame is essentially the keeper of references to columns (SArrays), so adding and deleting columns is a very cheap operation. However, the fact that SFrames store their data on disk produces some important limitations when thinking about editing an SFrame:

a) SFrames are immutable with respect to column size and data.

b) SFrames do not support random access of elements and are not indexed.

Sequential access is king on disk, and this is very useful to remember when working with SFrames. This means that inspecting a specific row would perform quite poorly and writing to a specific row is not possible.

2. SGraph

   SGraph is a disk-based graph data structure that stores vertices and edges in SFrames. an ideal structure for capturing sparse relationships between things. SGraphs are fundamental for efficient computation of many machine learning models.

Graphlab create also used to perform operations such as clustering, building recommendation graphs and graph analysis.Graphlab create provides scalability which lacks in Graphlab and it is significantly better than Hadoop and Spark.

Chapter 6

Algorithms

We consider three categories of graph algorithms: Random Walk, Parallel traversal, Sequential Traversal. Random walk algorithms perform computations on all vertices based on random walk model. Pagerank algorithm is an example of random walk algorithm. Parallel traversal algorithms start by considering all vertices at once. Multiple vertices participate in each super step until they acquire some common information. Weakly connected component is an example of Parallel traversal algorithms.Algorithms like triangle counting require a vertex to read many edge lists . Triangle counting algorithm generate more I/O intensive then the other algorithms and generate many random I/O access. Below table(Table 1) gives brief description of the algorithms used.

Algo.	Category	Similiar Algo.	CPU	Memory
PageRank	Random Traversal	HITS	Medium	Medium
WCC	Parallel Traversal	Label Propagation	Low	Medium
Triangle	Parallel Traversal	-	Medium	High

Table 6.1: Algorithms

6.0.1 PageRank

pagerank is an iterative algorithm to compute most influential nodes in a network. In each iteration, a vertex influence is measured as the sum of influence of the nodes that point at each other. Each node's score is updated until the scores converge , or until the user specified maximum iterations reached.All vertices start with a value of 1.0. At each super step, a vertex 'u' is updates its value to p=0.15+0.85x where x is the sum of its in-edges and sends p/deg+(u) along its out-

edges, where deg(u) is u's out degree. This gives us expectation value.Dividing it by the number of vertices gives the probability value.Pagerank requires only out-edge lists.The pagerank value characterizes the importance of a vertex in the graph using the following recursive definition:

$$pr(i) = resetprobability + (1 - resetprobability) \sum_{j \in N(i)} \frac{pr(j)}{outdegree(j)}$$

where N(i) is the set containing all vertices j such that there is an edge going from j to i. Self edges (i.e., edges where the source vertex is the same as the destination vertex) and repeated edges (i.e., multiple edges where the source vertices are the same and the destination vertices are the same) are treated like normal edges in the above recursion.

Page rank is a type Random walk algorithm where all vertices remain active throughout the computation ,communication is also stable across super steps. Pagerank algorithm is used to find most influential people in social networks like facebook,twitter etc. Google uses pagerank algorithm to rank web pages based on the idea that more important websites likely to receive more links from other websites.

6.0.2 WCC

Weakly connected components (WCC) is an algorithm that finds the maximal weakly connected components of a graph. A component is weakly connected every pair of vertices is mutually reachable when ignoring edge directions.WCC algorithm is an example of parallel traversal algorithm.

Initially all vertices will be active. Each vertex starts as its own component by setting its component ID to vertex ID. When a vertex receives a smaller component ID it updates its vertex value with the received ID and propagates that ID to neighbors.Weakly Connected Components(WCC) algorithm takes at most a number of super steps equal to graph's longest path. WCC algorithm requires both in-edge and out-edge lists. Unlike pagerank algorithm vertices can halt before others.This results in network communication that decreases monotonically with increasing super steps, allowing us to observe system performance under an other distinct communication pattern.

6.0.3 Triangle Counting

The number of triangles in a vertex's immediate neighborhood is the measure of the density of vertex neighborhood. A vertex computes the intersection of its own edge list and edge list of each neighbor to look for triangle. We count triangles on only one vertex in a potential triangle and this vertex then notifies other two vertices of the existence of the triangle through message passing. This requires both in edge and out edge lists. Triangle counting require a vertex to read many edge lists. Triangle counting algorithm more I/O intensive then the others and generate many random I/O accesses.

CHAPTER 7

DATASET

We obtained the data set from Stanford Network Analysis Project. All graph are real world data sets with millions of vertices and edges. Table 2 shows the brief description of the data sets. $|V|$ denote the number of vertices, and $|E|$ denotes number of edges.The data sets can be characterized by several properties like density. A graph is dense if $|E| = |V|^2$. So all of our data sets are sparse. A real-world dta sets tend to follow a power law degree distribution. The graphs contain vertices with high degree which may cause computation or communication bottlenecks due to wok load imbalances.We selected Amazon graph, Web-Google graph and com-Live journal graph for doing experiments because they belongs to different kind of networks. Table below(Table 2) describes different graphs used.

Graph Name	**Type of Network**	**Vertices ($|V|$)**	**Edges ($|E|$)**	**Diameter**
Amazon	Product Purchasing Network	403394	3387388	21
Google	Web Graph	875713	5105039	21
Com-Live Journal	Social Network	3997962	34681189	17

Table 7.1: Graphs

1. Amazon Graph

   Amazon co-purchasing network is an example of product purchasing networks. Network was collected by crawling amazon website. It is based on customers who bought this item also bought feature of amazon website.If a product ‘i’ is frequently co-purchased with product ‘j’, then graph contains directed edge from ‘i’ to ‘j’. Amazon graph contains 403394 vertices and 3387388 edges. The longest shortest path of graph is 21.

2. Web-Google Graph

   Web-Google graph is an example of web graph. Nodes represent web pages and directed edges represent hyperlinks between them. Web Google graph contains 875713 nodes and 5105039 edges. The longest shortest path of the graph is 21.

3. Com-Live Journal

   Com-Live Journal is an example of social networks . Live journal is a free on-line blogging community where users declare friendship with each other.Live journal also allows users to form a group which other members can then join.This groups are considered as ground-truth communities. Com-Live Journal graph contains 3997962 vertices and 34681189 edges. The longest shortest path of the graph is 17.

Chapter 8

Experimental Setup and Results

We conducted all experiments on a single multi-core machine. It has 4 cores and have two threads for core. Machine has 4 GB main memory and Intel core i5 processor with 2.60 ghz speed. All experiments conducted on Ubuntu 12.04 (64 bit) with kernel Linux 3.11.0-26-generic. We used FlashGraph and Graphlab create to conduct the experiments. We provided similar kind of system load for both the frameworks while running the algorithms. While running the algorithms we ensured no process is started by user other than the processes started by booting operating systems to do fair analysis. We conducted experiments multiple times and considered mean of resultant values as final results.

8.0.4 Evaluation Metrics

We measure the performance through the following metrics: total time, load time,computation time ,CPU usage and memory usage.

Total time is the total running time start to finish. Total time is sum of load time and computation time. Load time is defined as the time taken to load and partition the input graph.Computation time includes vertex computation ,barrier synchronization and communication.

Computation time , in combination with cpu utilization can help to identify message processing overheads.We focused on maximum memory usage. This gives us minimum memory resources all machines need to run experiment with out failure. Using different data sets enables us to investigate scalability of the system. How performance scales with larger graphs on the system. To track these metrics, we use total ,load time reported by all systems. For more fine grained statistics , and to compute memory usage , we rely on one second interval sar,top and free.These are started locally and killed after each experiment . We conducted same experiments multiple

times and take mean of the resultant values as final results.

8.0.5 Results

In this section, we present and analyze our experimental results. For all plots,each bar represents a system tested. Bars are grouped by data sets/graphs.Time Plots are splitted into computation time and load time. We discuss the experimental results of both the frameworks for each algorithm.

8.0.5.1 PageRank

Pagerank algorithm is used to find the influential nodes in the network. We performed pagerank algorithm on three graphs with max iterations=10 and observed load time,computation time,maximum cpu utilization, maximum main memory used.Table 3 shows the summary of results of pagerank algorithm.

1. Load Time

   Load time is defined as time taken to load and partition the graph. Graph lab create Load time for all graphs is less compared to FlashGraph. Graphlab create is at least 3x faster than the FlashGraph. For Amazon product purchasing network, both the frameworks load time almost equal. In case of Amazon Graph FlashGraph's load time is 2.17 seconds Graphlab

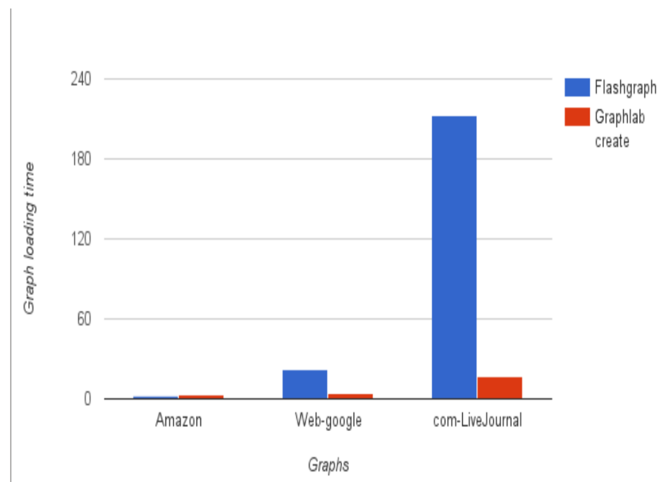

Figure 8.1: graph

   create load time is 3.08 seconds.For Google web graph FlashGraph took 32.34 seconds while Graphlab create took 4.23 seconds to load the graph. For com-live journal social network,FlashGraph took 222.4 seconds while Graphlab create took 16.42 seconds. By increasing the size of the graph the difference between load time of Graphlab create and

FlashGraph is increasing. In case of google web graph Graphlab create is 2X faster than FlashGraph.In case of com-live journal social graph Graphlab create is 7X faster than flshgraph.

Load Time	Flash Graph	Graph Create
Amazon	2.17	3.08
Google web Graph	32.34	4.23
Com-Live Journal	222.4	16.42

Table 8.1: Load Time

2. Computation Time

   Computation time of Graphlab create for all graphs is less than FlashGraph. The computation times of Flashgraph for Amazon graph, google graph and com-live journal graph are 3.85s, 7.56s,37.18s respectively. The computation times of Graphlab create for Amazon graph.Google graph,com-live journal graph for pagerank algorithm are 2.43s,4.58s,13.70s. You can notice Graphlab create is at least 2X faster than Flashgraph. By scaling

Computation Time	Flash Graph	Graph Create
Amazon	3.85	2.434
Google web Graph	7.56	4.58
Com-Live Journal	37.18	13.78

Table 8.2: Computation Time

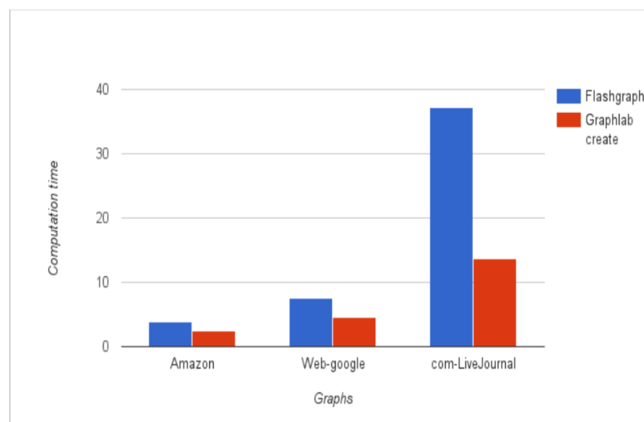

Figure 8.2: Computaton Time

the size of the graph you can observe the difference between computation time is increasing. For Google web graph Graphlab create is 2X faster than flashgraph, but in case of com-live

journal graph Graphlab create almost 3x faster than flashgraph.

FlashGraph's total time of Amazon graph, google web graph and com-live journal are 5.95s,29.56 and 259.58 seconds respectively.Graphlab create's total time of amazon graph,google web graph and com -live journal are 5.51s,8.62 s and 30.12 respectively. In case of amzon graph both frameworks total time almost equal but in case of amazon graph and com -live journal graphs graphlab create is 3X and 8x faster than Flashgraph respectively.

Total Time	Flash Graph	Graph Create
Amazon	5.95	5.51
Google web Graph	29.56	8.61
Com-Live Journal	259.58	30.12

Table 8.3: Total Time

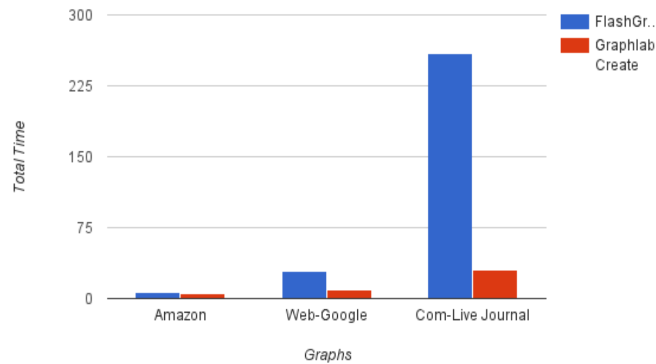

Figure 8.3: Total Time

3. Main Memory Usage

By observing the maximum main memory used we can find out minimum main memory required for a framework to run an algorithm with out any failure. FlashGraph uses less main memory compared to Graphlab create. FlashGraph uses maximum of 6.3 percent of main memory to run pagerank algorithm on Amazon graph while Graphlab create uses 23.2 percent of main memory for the same graph. FlashGraph uses maximum 8.6 percent of main memory to run pagerank algorithm on Google web graph while Graphlab create uses 32.1 percent of main memory for same graph. Flashgraph uses maximum of 20.12 percent main memory for com-live Journal while Graphlab create uses 32.15 percent of main memory for same graph. In case of amazon and google web graph graphlab create

uses 4x main memory compared flashgraph. In case of com-live journal Graphlab create uses 1.5X main memory compared to flashgraph.

Max Main Memory used	Flash Graph	Graph Create
Amazon	6.3	23.2
Google web Graph	8.6	32.1
Com-Live Journal	20.12	32.51

Table 8.4: Max Main Memory used

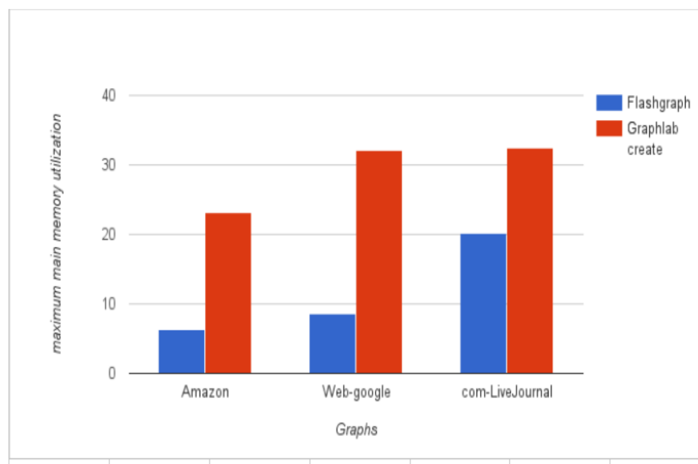

Figure 8.4: Main Memory usage

4. CPU Utilization

CPU utilization with combination of computation of time will help to find out the scalability of the system. Maximum CPU utilization in both the systems almost similar in Amazon and web google graph but in case of com-live journal graph Graphlab create cpu utilization is 2x more than the flashgraph. Flashgraph's maximum cpu utilization while running pagerank algorithm on all graphs are 97.3, 95.66 and 32.18 respectively. Graphlab create's maximum cpu utilization while running pagerank algorithm on all graphs are 97.3, 95.66 and 32.18 respectively.

Cpu Utilization	Flash Graph	Graph Create
Amazon	97.3	98.2
Google web Graph	95.66	97.2
Com-Live Journal	32.5	68.3

Table 8.5: Cpu Utilization

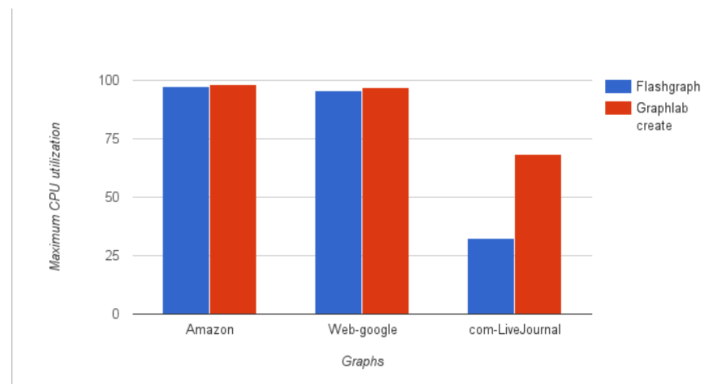

Figure 8.5: CPU utilization

8.0.5.2 WCC

Weakly connected components (WCC) is an algorithm that finds the maximal weakly connected components of a graph. We performed WCC algorithm on three graphs and observed load time,computation time maximum cpu utilization, maximum main memory used.

1. Computation Time

Computation Time	**Flash Graph**	**Graph Create**
Amazon	1.21	0.37
Google web Graph	1.70	1.25
Com-Live Journal	51.63	9.30

Table 8.6: Computation Time

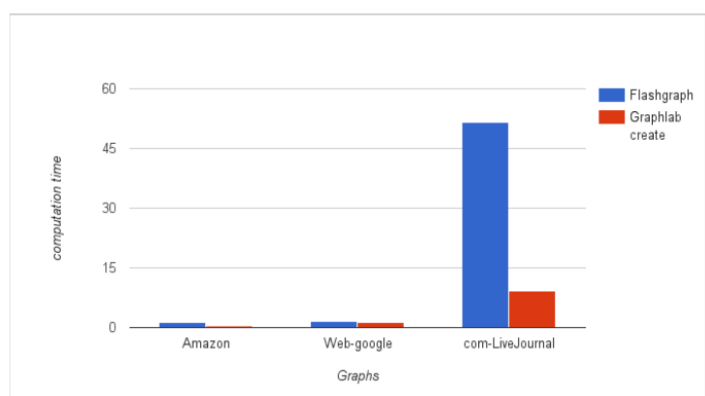

Figure 8.6: Computation Time

Computation time of Graphlab create for all graphs is less than FlashGraph. The computation times of Flashgraph for Amazon graph, google graph and com-live journal graph are

1.28s, 1.7s,51.63s respectively. The computation times of Graphlab create for Amazon graph ,Google graph,com-live journal graph for WCC algorithm are 0.37s,1.25s,9.30s. In case of Amazon graph and Google web graph both the framework computation times are almost equal. But in case of Com-Live journal graph , Graphlab create 6x faster than Flashgraph. By scaling the size of the graph you can observe the difference between computation time is increasing.

2. Total Time

   As graphs are same , Load time of both the frameworks are not changed. Total time of Flashgraph for Amazon, Google and Com-Live journal graphs are 3.38s,34.04,273.6s respectively. Total time of Graphlab create for Amazon,Google and Com-Live Journal graphs are 3.45s,5.28 and 25.72 respectively. In case amazon graph both the frameworks total are almost equal but in case of google web graph and com -Live journal graph Graphlab create is 5x and 10X faster than Flashgraph.

Total Time	**Flash Graph**	**Graph Create**
Amazon	3.38	3.45
Google web Graph	34.04	5.28
Com-Live Journal	273.6	25.72

Table 8.7: Total Time

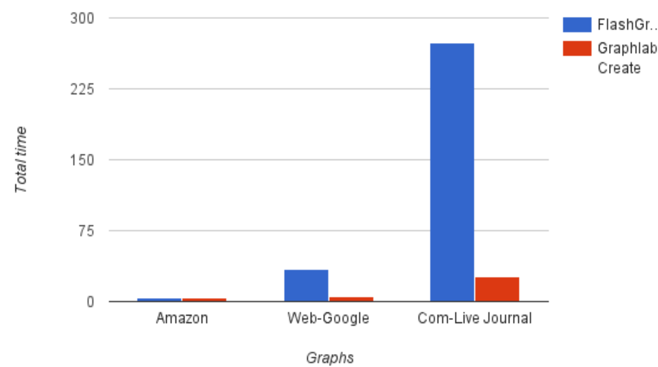

Figure 8.7: Total Time

3. Main Memory Usage: FlashGraph uses less main memory compared to Graphlab create. FlashGraph uses maximum of 2.2 percent of main memory to run WCC algorithm on Amazon graph while graphlab create uses 22.4 percent of main memory for the same graph. FlashGraph uses maximum 8.3 percent of main memory to run WCC algorithm

on Google web graph while graphlab create uses 16.4 percent of main memory for same graph. Flashgraph uses maximum of 21.4 percent main memory for com-live Journal while graphlab create uses 31.8 percent of main memory for same graph. In case of amazon and google web graph graphlab create uses 8x and 2x main memory compared to flashgraph. In case of com-live journal Graphlab create uses 1.5X main memory compared to flashgraph.

Main Memory	Flash Graph	Graph Create
Amazon	2.2	22.4
Google web Graph	8.3	16.4
Com-Live Journal	21.4	31.8

Table 8.8: Main Memory

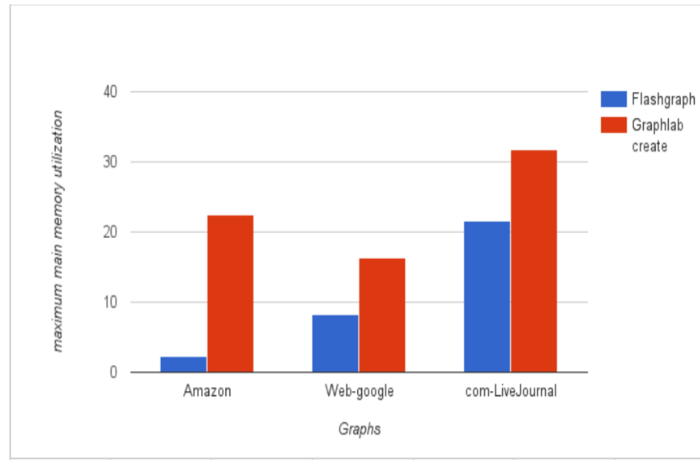

Figure 8.8: Main Memory usage

4. Cpu Utilization:

   Maximum CPU Utilization in both the systems almost similar in Amazon and web google graph but in case of com-live journal graph Graphlab create cpu utilization is 2x more than the flashgraph. Flashgraph's maximum cpu utilization while running WCC algorithm on all graphs are 12.2, 57.66 and 31.38 respectively. Graphlab create's maximum cpu utilization while running WCC algorithm on all graphs are 13.4, 58.61 and 81.12 respectively.

8.0.5.3 Traingle

Triangle counting algorithm is used to measure the cohesiveness of local community.The number of triangles in a vertex's immediate neighborhood is the measure of the density of vertex neighborhood.

Cpu Utilization	Flash Graph	Graph Create
Amazon	12.2	13.4
Google web Graph	57.6	58.61
Com-Live Journal	31.38	81.12

Table 8.9: CPU Utilization

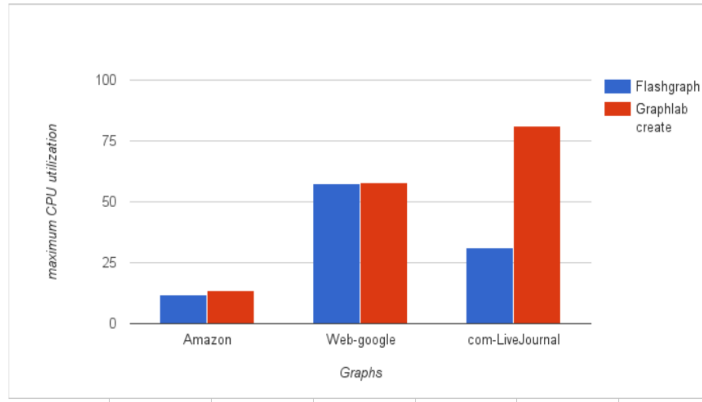

Figure 8.9: CPU Utilization

1. Computation Time:

Computation Time	Flash Graph	Graph Create
Amazon	2.66	14.85
Google web Graph	5.07	26.28
Com-Live Journal	382.3	178.6

Table 8.10: Computation Time

FlashGraph Computation time of triangle counting algorithm is less than the Graphlab create. FlashGraph is almost 7x faster than Graphlab create in Amazon graph and it is at least 5x faster than Graphlab create in case of Google web Graph. The computation time of Flashgraph for Amazon and Google web graph are 2.66 and 5.07. The computation times of Graphlab create for Amazon and Google web graph are 14.85 and 26.28. But in case com -live journal graph Graphlab create is almost 2x faster than flashgraph. The computation time of Flashgraph in case of com-live journal graph is 382.3s and Graphlab create computation time is 178.6s.

2. Total Time:

Graphlab create total times for google and com-live journal graphs less than flashgraph but in case of Amazon graph flashgraph is faster than Graphlab create.The total time of

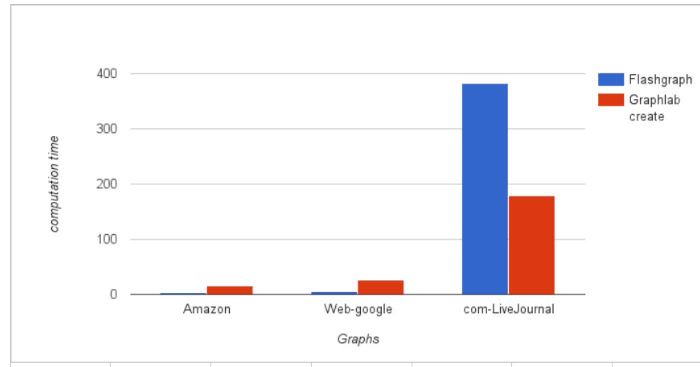

Figure 8.10: Computation Time

Flashgraph for Amazon ,google and com -live journal social graphs are 4.83,37.62,504.7s.The total time of Graphlab create for Amazon ,google and com -live journal social graphs are 17.30.9,195.02s.

Total Time	Flash Graph	Graph Create
Amazon	4.83	17.83
Google web Graph	37.62	30.9
Com-Live Journal	404.7	195.02

Table 8.11: Total Time

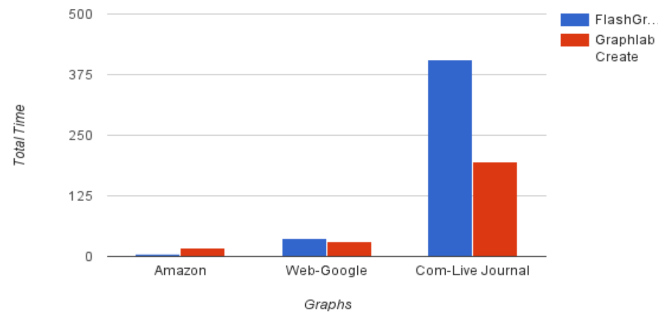

Figure 8.11: Total Time

3. Main Memory Used:

   FlashGraph uses less main memory compared to Graphlab create. FlashGraph uses maximum of 6.6 percent of main memory to run triangle counting algorithm on Amazon graph while graphlab create uses 25.7 percent of main memory for the same graph. FlashGraph

uses maximum 9.2 percent of main memory to run triangle algorithm on Google web graph while graphlab create uses 36.4 percent of main memory for same graph. Flashgraph uses maximum of 27.5 percent main memory for com-live Journal while graphlab create uses 72.1 percent of main memory for same graph. In case of amazon and google web graph graphlab create uses 4x main memory compared to flashgraph. In case of com-live journal Graphlab create uses 2.5X main memory compared to flashgraph.

Main Memory	Flash Graph	Graph Create
Amazon	6.6	25.7
Google web Graph	9.2	36.4
Com-Live Journal	27.5	72.1

Table 8.12: Main Memory

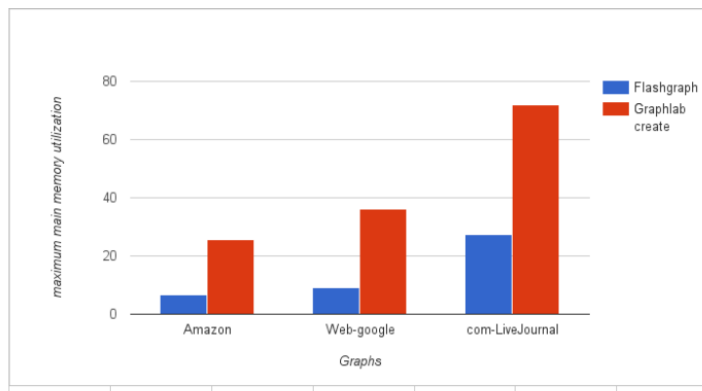

Figure 8.12: Main Memory usage

4. Cpu Utilization:

   Flashgraph's maximum CPU utilization while running triangle algorithm on all graphs are 64.2, 87.3 and 19.3 respectively. Graphlab create's maximum cpu utilization while running pagerank algorithm on all graphs are 73.5, 98.4 and 96.2 respectively.

CPU Utilization	Flash Graph	Graph Create
Amazon	64.2	73.5
Google web Graph	87.3	98.4
Com-Live Journal	19.3	96.2

Table 8.13: CPU Utilization

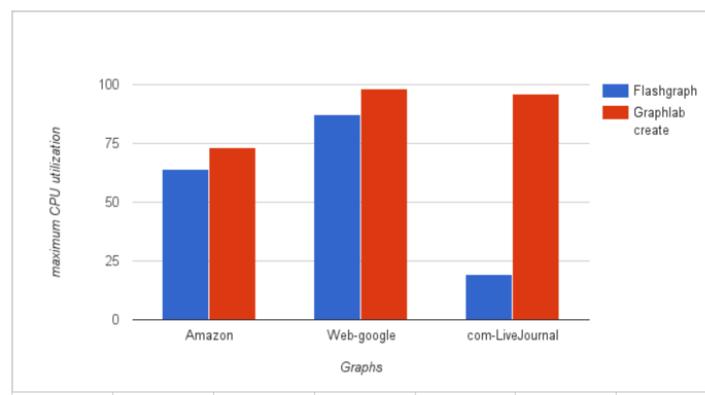

Figure 8.13: CPU Utilization

Chapter 9

Future Work And Conclusion

We are working on providing dynamic graph partitioning in frameworks LFgraph and Powergraph and want to compare the performance of these two frameworks. Graph analytic frameworks performs computations in iterations. In Graph analytic frameworks graph partitioning is the only to decrease the computation time and communication overhead. The graph partition done by the frameworks may not be optimal.Some workers may get high work load ,and some may get less workload.So the workers with more workload take more time for computation compared to other machines . But end of each iteration, all worker machines need to be synchronized. Because of this, other machines will wait until heavy workload machines completes their job. In these scenarios moving some vertices from heavy load workers to low load workers may result in decrease in communication overhead and workload imbalance. This is called dynamic partitioning of graph.

If suppose vertex 'i' is a high degree vertex. Assume that on partition, vertex i placed in machine1 and most of the neighbors of 'i' are placed in machine2. This results in high communication overhead between machine1 and machine2. If we move vertex 'i' to machine2 from machine1 it may decrease the communication overhead and workload imbalance between machine 1 and machine 2.In some cases dynamic graph partition may increase communication overhead and workload imbalance. Following are the various challenges in providing Dynamic graph partition.

1. Identifying the workload imbalance across the servers during the iterations.
2. Identifying the vertices to reassign.
3. Identifying the suitable time to move vertices to new workers
4. Locating reassigned vertices.

9.0.6 Conclusion

Graph processing systems are increasingly important as more and more problems require dealing with graphs.To this end, we presented a thorough comparison of two recent graph processing systems,FlashGraph and Graphlab Create on three graphs(Amazon graph, web-Google Graph, Com-Live Journal Graph) and three different algorithms:PageRank, WCC, and Triangle. We used single machine to conduct all algorithms and evaluated the performance using computation time,load time,total time memory usage, and Cpu Utilization metrics.We found that graphlab Create is performing better than Flashgraph in computation time.Flashgraph is using less main memory compared to Graphlab create. For small networks both the frameworks computation almost equal but in case of large graph the difference between computation time is increasing.We identified that scalability of FlashGraph should be improved to improve the performance.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] Da Zheng, Disa Mhembere, et al. *"FlashGraph: Processing Billion-Node Graphs on an Array of Commodity SSDs."*

[2] Low, Yucheng, et al. *"Graphlab: A new framework for parallel machine learning."*

[3] Gonzalez, Joseph E., et al. *"PowerGraph: Distributed Graph-Parallel Computation on Natural Graphs."* OSDI. Vol. 12. No. 1. 2012..

[4] *"Dato: Fast, Scalable Machine Learning Platform"*
URL - `https://dato.com/`

[5] Salihoglu, Semih, and Jennifer Widom. *"Gps: A graph processing system."* Proceedings of the 25th International Conference on Scientific and Statistical Database Management. ACM, 2013.

[6] Kyrola, Aapo, Guy E. Blelloch, and Carlos Guestrin. *"GraphChi: Large-Scale Graph Computation on Just a PC."* OSDI. Vol. 12. 2012.

[7] Malewicz, Grzegorz, et al. *"Pregel: a system for large-scale graph processing."* Proceedings of the 2010 ACM SIGMOD International Conference on Management of data. ACM, 2010.

[8] Hoque, Imranul, and Indranil Gupta. *"LFGraph: Simple and fast distributed graph analytics."* Proceedings of the First ACM SIGOPS Conference on Timely Results in Operating Systems. ACM, 2013.

[9] Khayyat, Zuhair, et al. *"Mizan: a system for dynamic load balancing in large-scale graph processing."* Proceedings of the 8th ACM European Conference on Computer Systems. ACM, 2013.

[10] Roy, Amitabha, Ivo Mihailovic, and Willy Zwaenepoel. *"X-stream: Edge-centric graph processing using streaming partitions."* Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles. ACM, 2013.

[11] Han, Minyang, et al. *"An experimental comparison of pregel-like graph processing systems."* Proceedings of the VLDB Endowment 7.12 (2014): 1047-1058.

[12] Xin, Reynold S., et al. *"Graphx: A resilient distributed graph system on spark."* First International Workshop on Graph Data Management Experiences and Systems. ACM, 2013.