Real-Time Partition of Streamed Graphs for Data Mining over Large Scale Data

Víctor Medel and Unai Arronategui
Escuela de Ingeniería y Arquitectura
University of Zaragoza
Zaragoza, Spain
{vmedel, unai@unizar.es}

***Abstract*—Mining data in real-time from large graphs requires a lot of memory to obtain a good distribution of information. Current state of the art solutions for streamed graphs are not scalable and they work with a single stream source. We propose a new reduced memory model to partition large graphs over big streams to improve mining algorithms. The aim of our work is to give support to data mining algorithms over large-scale structured data (e.g., Web structure, social networks) to minimise communication among partitions. In our architecture, the incoming graph elements are sampled to reduce total memory usage and the information in each partitioner is updated in a feedback scheme to allow multiple entry points. We have made experimentation with real-world graphs and we have discussed about the suitability of different sampling strategies depending on the graph structure. In addition, we have executed the PageRank algorithm over the partitioned graph, in order to measure the influence of the partition in the execution of a mining algorithm.**

Keywords–Big Graphs; Data Streaming; Graph Partition; Sampling.

I. Introduction

Mining from social networks, from Wikipedia or from World Wide Web compels to deal with a huge amount of information modelled by a graph. Information is continuously growing, so it has to be processed as is generated. The data stream paradigm [1] fits well with these kind of applications. As we want a real-time processing, the resource needs are very huge.

The underlaying graph is so large that it cannot be stored in a single machine, thus it has to be distributed among several machines before we can make some analytics over it. For example, Yahoo! Dataset [2] is 120 GB in size, and a web network graph of 50 billion vertices and 1 trillion edges, like the one used by Google in Pregel experimentation [3], needs 25 TB of free storage space.

Typical graph analytics algorithms in these domains (e.g., PageRank [4], Community Discovering [5], Triangle Counting [6], [7]) need a lot of communication among vertices, so the number of edges among partitions (cutting edges) will condition network traffic and therefore execution time. In other words, the quality of the partition solution has a direct impact in the execution of task over the graph.

Partitioning a graph in a streaming scenario is a novel application with a few works [8],[9]. Proposed algorithms do not scale well. They make an intensive use of memory because they need to have knowledge of previous elements of the graph. In addition, they only consider a single stream source, consequently their incoming rate is bound by network capacity.

In this work, we focus on graph partition problem in a streaming environment with hard resource constraints, in order to guarantee real-time data management. We consider a single-pass streaming algorithm, with multiple stream sources and we reduce the total memory usage, to propose a scalable model. Over the obtained partitions, we have executed the PageRank algorithm to show the trade-off between used memory in partition phase and total time of the execution of an analytic. We have used the PageRank algorithm to compare our results as it has been the reference analytic used in previous works.

We have achieved our aims by sampling incoming elements and by updating, periodically, the information in each partitioner. Our model is high scalable, it is not bind by network capacity and it allows multiple streaming inputs.

The paper is structured in six parts, with this Introduction as first section. In Section II, we synopsise fundamental notions in graph partition problem in a streaming scenario. In Section III, we talk about the state of the art in the domain and in Section IV, we present our architecture. The analysis of our model is made in Section V. In Section VI, experimental results from a real scenario are shown. Finally, the conclusion and future directions of the research are presented in Section VII.

II. Background

In this section, we present the fundamental notions used in this paper. We start showing how to model a graph in a streaming environment and the graph partition problem. At the end of the section, we analyse the requirements to guarantee real-time processing in data streams.

A. Graphs on data stream.

We consider that the graph arrives in a data stream way. A Data Stream A [10] is an ordered sequence of $a_1, a_2, ..a_n$ elements. In informal terms, the system has no control over the arriving model; streams are potentially unbound in size and once an element has been processed it cannot be retrieved.

We denote $G = (V, E)$ an undirected graph with vertex set $V = \{v_1, v_2, v_3, ...v_n\}$ and an edge set $E =$

$\{e_1, e_2, e_3, ..., e_m\}$. Note that n is the number of vertices, m the number of edges and $e_i = (v_j, v_k), \quad v_j, v_k \in V$.

A vertex graph stream, T, is a sequence of $t_1, t_2, ..., t_n$ where $t_j = (v_j, v_{j_1}, v_{j_2}, ..., v_{j_d})$, $v_j, v_{j_i} \in V$, $(v_j, v_{j_i}) \in E$ and $deg(v_j) = d$, for $i = 1, ..., d$ and for $j = 1, ..., n$.

Each tuple represents a vertex with its adjacency list. The size of a tuple depends on the degree of the vertex d, so processing time per tuple is variable. As we consider undirected graphs, each edge appears implicitly twice. In storage terms, the graph size is bound by $O(n + 4m)$.

Although in the general Data Stream model elements arrive in a random order; some specific models have been proposed for graph problems [9]. In Breadth First Search (BFS) model, one vertex of the graph is selected and, from that vertex, a breadth first search strategy is performed to generate the following vertices. A Depth First Search (DFS) strategy could be also performed. These orders have full sense in graph applications. For example, if a web crawler follows links with a BFS strategy, the elements of the graphs are generated with that order.

B. *Graph partition problem.*

Given a graph $G = (V, E)$, we define a k partition set P, where $P = \{S_1..S_k\}$ such as $S_i \subset G$ and $\bigcup_{i=1}^{k} S_i = G$. We define the graph partition problem as finding an optimal P^* such that for all possible partitions P such that $|P| = k$, $f(P^*) \geq f(P)$, for a determinate function.

Our objective is to obtain a partition set P which minimises the communication cost among partitions S_i and consequently processing time of a mining algorithm over the partition. Thus, f depends on the following metrics:

$$\lambda = \frac{\textit{number of cutting edges}}{\textit{total edges}} = \frac{\Lambda}{m} \tag{1}$$

$$\rho = \frac{Max\{|S_i|, \forall i \in \{1, ..., k\}\}}{\frac{n}{k}} \tag{2}$$

An edge $(v_i, v_j) \in E$ is a cutting edge for a k partition set P if $v_i \in S_q$ and $v_j \in S_r$, with $i, j \in \{i, ..., n\}$ and $q, r \in \{1, ..., k\}$ and $r \neq q$.

The λ parameter (equation (1)) gives the possible overhead of needed communication among partitions when graph processing tasks are executed. The ρ value (equation (2)) is the balanced factor of the solution partition set P. In the processing phase, having too disbalanced partitions might increase the processing time (some machines have to do a heavy process and others might be idles).

This problem is NP-Complete [11], so we have to consider approximations to the optimal solution. In [9] [8], light heuristics are used to compute an approximation of the best partition.

C. *Real time streaming.*

Incoming elements are processed as they arrive. We cannot store each new element because the stream is unbound. The partition algorithm cannot belong to $O(n)$ used memory algorithms. In an informal way, if we consider the graph partition problem, storing each incoming element would build the entire graph in a local or distributed memory. However, this is the same graph that has been partitioned. If the system needs to access the distributed memory for each vertex, additional time is lost. Consequently, if incoming elements have to be processed as they arrive, the total number of distributed partitioners must be increased. This increment would suppose to multiply the number of partitioners by a factor which would depend on the response time of the distributed memory.

Another possibility could be to query to partitions for each arriving vertex. This solution implies that incoming rate could be at most half of total network capacity, so we could not guarantee real-time. Moreover, partition algorithm cannot belong to $O(n)$ process time algorithms and it cannot do more than one pass over the stream.

The fact that we develop a single-pass algorithm with hard memory usage restrictions, makes our solution approximate. We compute a (ε, δ)-approximation of λ^* which means that $P[\lambda \leq (1 + \varepsilon)\lambda^*] \leq 1 - \delta$.

III. Related Work

Graph algorithms have been widely treated in literature. Graph streaming model has been described in [1] in a theoretical way. It represents the sequential access to graph elements instead of random access, due to the size of the graph. In this regard, several papers propose how to adapt graph algorithms to streaming paradigm [12] [13]. They take considerations about the complexity of different typical algorithms (triangle count [14], property checking, connectivity, etc.) and they calculate the required space bound and number of times an element of a stream is processed. In several works, they relax some data stream restrictions in order to obtain more flexible models: Semi-stream [1], W-Stream [15], Best-Order Stream [16], Sort-Stream, etc.

The main disadvantage of these works is that these restrictions cannot be made in an on-line environment with real-time considerations. As graph partition problem in a streaming environment is an NP-Complete problem [11], it is not feasible to compute the optimal solution, so we compute an approximated one. In [9] [8], some approximated solutions are obtained via light heuristics. The solution provided by Fennel [8] is quite good for real graphs. For some graphs, the obtained partition is as good as Metis [17], an offline partition algorithmn. In their experiments, the worst case is for *amazon0312* and they get a 6% more cutting edges. The problem of these kind of solutions is the linear size of the used memory, which makes difficult to scale the heuristic. In addition, it only considers a single input stream, which binds incoming rate.

IV. Proposed Model

We propose the decentralised architecture that is illustrated in Figure 1. We have uncoupled the different processing stages in order to distribute them. There are several loaders which continuously send elements to partitioners. They execute the partition algorithm to select the most suitable partition, and they send the element to that partition. The partition algorithm has to be simple, in computational terms, and it has to select

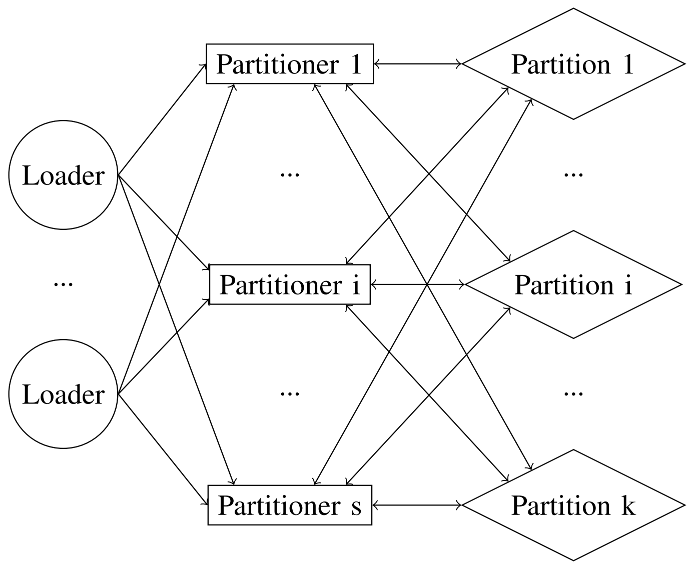

Figure 1. Proposed Architecture

the partition based on partial information. Its local information is updated by the partitions in a feedback scheme.

Memory size restriction has been solved sampling incoming vertices. We propose to group vertices in sets, in order to reduce total memory. We only have to store each sampling-set, which will be used by the algorithm to calculate the best partition. As the proposed framework is distributed, sampling functions cannot have knowledge of the entire graph. Each partitioner has to assign the same vertex to the same summary and all elements of a summary are assigned to the same partition. In order to maintain information consistency in each partitioner, each partition sends to them the set of sampling-sets stored periodically.

Let it be a graph $G = (V, E)$, a sampling size l and a sampled graph $G' = (\Psi, \Phi)$, where $\Psi = \{\Pi_1 ... \Pi_u\}$, $\Pi_i \subset V$, $\Phi = \{(\Pi_r, \Pi_q) \in \Psi,\ r, q = 1...u\}$ with $u = \frac{n}{l}$.

We define two surjectives sampling functions, g and h, where: $g : V \to 1...u$, $h : E \to \Phi$, such as:

i. $\forall v \in V,\ \exists \Pi_q \in \Psi \mid g(v) = q \Leftrightarrow v \in \Pi_q$
ii. $\forall i, j \in \{1, ..., n\},\ \forall q, r \in \{1, ..., u\},\ \forall v_i, v_j \in V,$ $\exists \Pi_r, \Pi_q \mid g(v_i) = \Pi_q,\ g(v_j) = \Pi_r,$ and $(v_i, v_j) \in E \Leftrightarrow h((v_i, v_j)) = (\Pi_q, \Pi_r)$.

We can conclude that for $i, j = 1, ..., n$, and for $q, r = 1, ..., u$, $\forall (\Pi_r, \Pi_s) \in \Phi$, $\exists v_i, v_j \in V$ such as $g(v_i) = \Pi_r,\ g(v_j) = \Pi_q, h((v_i, v_j)) = (\Pi_r, \Pi_q)$.

Figure 2 illustrates the partition algorithm with this notion. The set M represents the main memory, where $M = \{(q, j),\ q \in \{1...u\},\ j \in \{1...k\}\}$. Note that a sampling-set Π_q is assigned to a partition S_j through its index.

When an element t of the unbound stream T arrives, we obtain its vertex. If we have assigned the set which that vertex belongs to, its index will belong to M. So, the partitioner has to send the vertex and its edges, t, to the already assigned partition S_i ($S_i = S_i \cup \{t\}$). On the contrary, if it is the first

```
**Input:** Unbound stream T
  $M = \emptyset$
  **for all** $t \in T$ **do**
    let $v$ = get vertex $v \in V$ from $t$
    let $q = g(v)$
    **if** $\exists i \in \{1..k\} \mid (q, i) \in M$ where $S_i \in P$ **then**
      Send $t$ to $i$ partition node
      In partition node $i$ $S_i = S_i \cup \{t\}$
    **else**
      $i = PartitionHeuristic(t, P)$
      $M = M \cup \{(q, i)\}$
      Send $t$ to $i$ partition node
      In partition node $i$ $S_i = S_i \cup \{t\}$
    **end if**
  **end for**
```

Figure 2. Vertex Partition Algorithm

time a vertex of that set arrives, the partitioner computes the optimal partition for it, using the partition heuristic. Then, it adds the corresponding summary to M.

The main advantage is that the partition heuristic only depends on $\frac{n}{l}$ in memory terms and the algorithm has to partition $\frac{n}{l}$ vertex, instead of n.

The analysis of partition heuristic is out of the scope of this paper. The only requirement is that it has to be computed in constant time. In the experimentation phase, we have selected Fennel [8].

In our scheme, local information in each partitioner is updated with a frequency f; so, some information might be incoherent in this interval. In Section V, we show the relationship among the sampling function, the update period and the approximate solution.

A. Sampling functions.

Functions g and h, which are defined in Section IV, are light functions ($g, h \in O(1)$), and the information used to assign one element to one set has to be known *a priori* for each partitioner. In other words, as we want an easy distribution of the algorithm, the decision has to be made without taken into account the previous elements. Sampling function cannot depend on the arriving order. We consider the number of elements in a set constant, so $|\Pi_1| = |\Pi_2| = ... = |\Pi_{|\Psi|}| = l$

We propose the following sampling functions, which are based on the vertex identifier.

- **Hash function.** Each vertex $v_i \in V$ goes to a set depending on its identification $i, j = 1..n$ as follows:

$$g(v) = i \ mod \ l$$
$$|\Psi| = \frac{n}{l}$$
$$h((v_i, v_j)) = (\Pi_{i \ mod \ l}, \Pi_{j \ mod \ l}) \tag{3}$$

- **Consecutive assignation.** If the identification of a node has implicit an order (e.g., it is a number), we can build sampling-sets sequentially. In some situations (e.g., BFS and DFS model), this sampling function has more sense, because as elements arrive in a determined way, connected elements go to the same set and to the same partition.

$$\forall i,j = 1..n,\ g(v_i) = i\ div\ l$$
$$|\Psi| = \frac{n}{l}$$
$$h((v_i,v_j)) = (\Pi_{i\ div\ l}, \Pi_{j\ div\ l}) \qquad (4)$$

V. Analysis.

Once we have proposed a model, we are going to analyse how much memory it needs in the partition stage and how much messages are sent from partitions to partitioners. We do not take into account the normal stream traffic because it is implicit to the model.

A. Memory Bound

Theorem 1: Given a graph $G = (V, E)$, a sample size l and a single-pass partition algorithm ALG which produces λ' cutting edges with $O(n)$ memory bound; there exists an (ε,δ)-approximation of cutting-edge fraction λ, with a $O(\frac{n}{l})$ memory bound in each parallel partitioner and $\varepsilon \in O\left(\sqrt{\frac{3lkln(\frac{1}{\delta})}{m[(l-1)(k-1)+\lambda' k]}}\right)$.

Proof: Memory reduction is achieved sampling incoming vertices into sets, and the sets are used as input of the algorithm ALG. With a sampling size of l, the memory bound belongs to $O(\frac{n}{l})$. In order to calculate the accuracy of the solution, we define a set of random variables X_{ij}, where $X_{ij} = 1$ if $v_i \in S_p, v_j \in S_q$ and $q \neq p, i,j = 1..n, p,q = 1..k$. By the law of total probability,

$$P(X_{ij} = 1) = \frac{(l-1)(k-1)}{lk} + \frac{\lambda'}{l} = p \qquad (5)$$

As $\lambda' < p$, by Chernoff bound:

$$P\left[\frac{\sum X_{ij}}{m} \geq (1+\varepsilon)\lambda'\right] \leq e^{-\frac{\varepsilon^2}{2+\varepsilon}mp} \qquad (6)$$

As $\lambda' < 1$ and $p > 0$, then $\varepsilon < 1$, so $-\frac{\varepsilon^2}{2+\varepsilon} < -\frac{\varepsilon^2}{3}$:

$$\varepsilon \in O\left(\sqrt{\frac{3lkln(\frac{1}{\delta})}{m[(l-1)(k-1)+\lambda' k]}}\right)$$

■

B. Number of sent messages

Theorem 2: Given a graph $G = (V, E)$, s distributed partioners, a sampling size l, an update frequency f and a single-pass partition algorithm ALG which produces a λ' cutting edges percent with a $O(n)$ memory bound; there exists an (ε,δ)-approximation of λ with a $O(\frac{n}{l})$ memory bound in each distributed partitioner and a $O(\frac{ns}{\sigma f})$ global distributed complexity, where σ is the incoming elements per time unit and $\varepsilon \in O\left(\sqrt{\frac{3ln(\frac{1}{\delta})}{m\left[1-\left(e^{\frac{-l\sigma f(\sigma f-1)}{2n}}\frac{\lambda'}{l}\right)\right]}}\right)$.

Proof: It is easy to see that the number of sent messages from partitions to partitioners depends on the update period f and on the number of partitioners s. In a f period, the system sends s messages. If $sigma$ elements arrive in one time unit, the entire graph arrives in $\frac{n}{\sigma}$. Then, in $\frac{n}{\sigma}$ periods, it sends $\frac{ns}{\sigma f}$.

Now let us calculate the accuracy of the solution. We consider that ALG will be better than a random partition algorithm ($\lambda' \in O\left(\frac{k-1}{k}\right)$). Therefore, the worst case happens when a vertex v_i whose $g(v_i)$ has been assigned in the same period f is assigned again by the random partition algorithm. The probability of get σf unique elements from n/l groups is (a.k.a. birthday problem):

$\frac{\frac{n}{l}-1}{\frac{n}{l}} \times \frac{\frac{n}{l}-2}{\frac{n}{l}} \times ... \times \frac{\frac{n}{l}-(\sigma f-1)}{\frac{n}{l}}$ and by a Taylor expansion its upper bound is $e^{\frac{-l\sigma f(\sigma f-1)}{2n}}$.

In order to simplify our calculus, we consider that the probability of generating a cutting edge by a random partition algorithm is about 1 (reasonable assumption for big k), so:

$$P(X_{ij} = 1) \approx 1 - \left(e^{\frac{-l\sigma f(\sigma f-1)}{2n}}\frac{\lambda'}{l}\right) = p \qquad (7)$$

And by Chernoff bound:

$$P\left[\frac{\sum X_{ij}}{m} \geq (1+\varepsilon)\lambda'\right] \leq e^{-\frac{\varepsilon^2}{2+\varepsilon}mp} \qquad (8)$$

As $\lambda < 1$ and $p > 0$, then $\varepsilon < 1$, so $-\frac{\varepsilon^2}{2+\varepsilon} < -\frac{\varepsilon^2}{3}$:

$$\varepsilon \in O\left(\sqrt{\frac{3ln(\frac{1}{\delta})}{m\left[1-\left(e^{\frac{-l\sigma f(\sigma f-1)}{2n}}\frac{\lambda'}{l}\right)\right]}}\right)$$

■

The number of sent messages per time unit is $\frac{s}{f}$. If we consider a distributed architecture with a distributed memory, the traffic between the memory and the partitioners per time unit is 2σ. For big values of σ, the bound of our system is better than the distributed memory approach.

VI. Evaluation

We have implemented our model on a real environment in order to test it. Among the available open source distributed Data Stream Management Systems that are available in a distributed and open source version, we have chosen Storm [18], due to its flexibility to deploy the distributed infrastructure over the machines. The implementation has been developed using Java. We have set a Storm cluster of eight workers and one master. Each machine has one core and 2 GB of memory size.

In addition, we have executed the PageRank mining algorithm over the obtained partition in a GraphLab cluster [19]. GraphLab is a distributed graph processing engine which provides several data mining algorithms. Moreover, PageRank [4] is a graph mining algorithm which is used to rank elements in a network.

A. Datasets.

The datasets we have used to test our system are in Table I. *PL*, *WS* and *BA* datasets are synthetic, created by the Networkx package. We have used these datasets because Web Network and social graphs can be modelled by a power law graph, [20]. WS is a Watts-Strogatzsgraph model [21] and BA is a Barabasi-Albert graph [22]. *Amazon**, *Wiki-talk* and *LiveJournal1* are real datasets. The *amazon** dataset represents co-purchasing information of Amazon. If a product i is co-purchased with a product j, there is an edge from i to j in the graph. The information was collected in March 12 2003, May 05 2003 and June 01 2003. In *Wiki-talk* dataset, each vertex represents a user, and an edge from i to j represents that the user i has edited, at least once, the talk page of user j. The information was recollected in January 2 2008. In *LiveJournal1* dataset, each link between vertices (users) represents a friendship relation.

TABLE I. LIST OF USED DATASETS

Dataset	Vertices	Edges
WS10000	10000	134944
WS100000	100000	3997464
BA10000	10000	134841
BA100000	100000	3548775
PL10000	10000	134766
PL100000	100000	4047486
amazon0312	400727	2349869
amazon0505	400727	2439437
amazon0601	400727	2443311
LiveJournal1	4843953	42845684
Wiki-talk	2388953	4656682

B. Experimentation and evaluation.

We use λ and ρ metrics (see equations (1), (2)) as measures of the quality of the obtained partition.

In experiments, we have used Fennel [8] as best partition heuristic. We have measured the relationship among the quality of the partition (through λ and ρ), the sampling size l and the number of partitioners s. In addition, we have measured the amount of used memory and the impact of these parameters in the execution of a mining algorithm (PageRank).

1) Partition quality: In Figure 3, we observe how sample size l affects λ value. We have partitioned *amazon0312* dataset into 32 partitions. The experiment has been made with different incoming orders (Random and BFS) and Hash and Consecutive sampling functions (see equations (3), (4)). The first measure, $l = 1$, is equivalent to the one obtained in Fennel partition algorithm, and the last one corresponds to the situation when there is only one set per partition, $l = \frac{n}{k}$, which is equivalent to a random partition strategy. With two elements per group, the number of cutting edges increases significantly compared to Fennel, but it is better than the Random partitioner. In our results, the kind of sampling function used affects the quality of the partition. In a BFS incoming ordering, the results are better for a consecutive assignment function. This kind of order is naturally obtained in social and web graphs because they are obtained by crawlers.

2) Used Memory: We can see that the maximum used memory depends on l. In Table II, the results for the *LiveJournal* dataset are shown. In this case, we have partitioned into

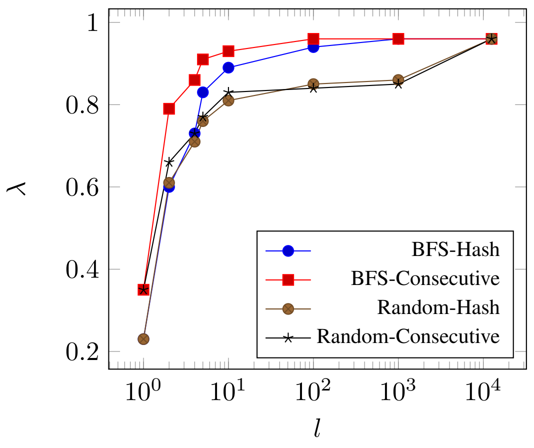

Figure 3. λ versus l in *amazon0312* dataset for different incoming orders and sampling functions.

TABLE II. VARIATION OF λ AND ρ IN LIVEJOURNAL1 DATASET WITH EIGHT PARTITIONS

l	λ	ρ	Used Memory (MB)
1	0.5	1.01	245.99
2	0.56	1.01	135.63
10	0.68	1.01	27.60
605495	0.96	1	0

eight partitions with a BFS order and a consecutive assignment function.

In Figure 4, we can see the RAM memory required to store the sampling sets. As it is natural, when the number of elements per group increases, the total memory decreases. Note that with $l = 1$, the total used memory is 20,8 MB. Approximately, this is 0.052 kB per element, so we cannot process a web network graph (50 billion vertices [3]) with the Fennel algorithm.

TABLE III. VARIATION OF λ AND ρ IN *amazon0312* DATASET WITH 6 PARTITIONERS AND 32 PARTITIONS

l	λ	ρ
2	0.8	1.23
5	0.81	1.1
10	0.83	1.12
100	0.85	1.19
1000	0.91	1.18
12532	0.96	1

3) Distributed Partitioners: The number of partitioners s affects the quality of the partition, because they manage local information. In Table III, we show the results for $s = 6$. We have used contiguous grouping strategy and BFS arrival ordering. Experimental results show that with bigger sets, the performance is similar to the single loader. This is because with bigger sets, the number of partitioned elements is small, so the balanced load factor decreases.

4) PageRank Processing: The last experiment we have made is to execute a graph algorithm over obtained partition in GraphLab. We have used the *LiveJournal1* dataset and the PageRank algorithm for testing our system in a real scenario. We have chosen PageRank because is a well-known analytic over a graph, and we can use it to compare ourselves with

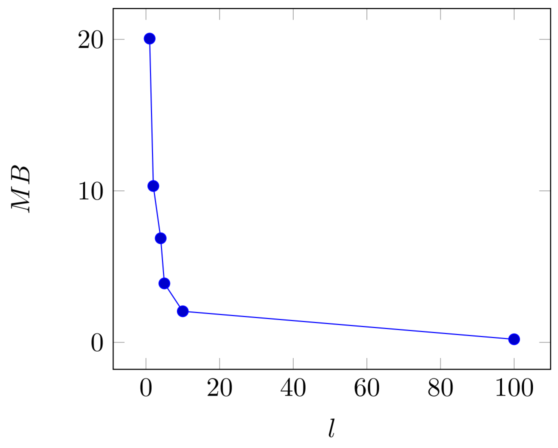

Figure 4. Required RAM memory to process *amazon0312* dataset versus sample size l

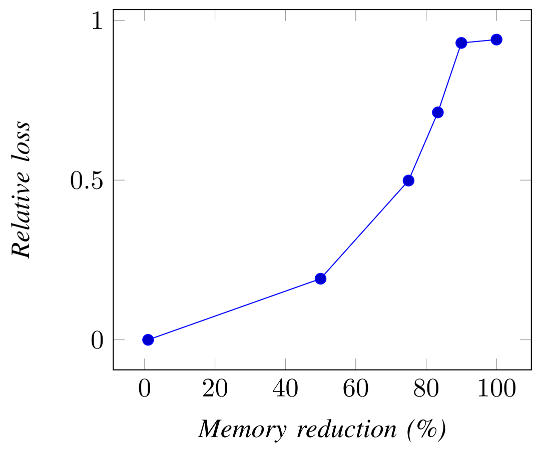

Figure 5. Relative loss of execution time of PageRank versus memory reduction.

previous works. The aim of the experiment is to measure the real trade-off between memory usage in partition stage and execution time of algorithm over the obtained partitions.

Figure 5 shows the loss in performance terms versus the memory reduction percent. We have calculated the loss using as reference the time the PageRank algorithm takes in a partition solution obtained by Fennel (equivalent to $l = 1$). The memory reduction has been calculated in the same way, and its relationship with l is straightforward. Last point refers to a Hash partition strategy, which does not use any memory, but almost doubles the execution time of PageRank. With a 50% memory reduction (equivalent to $l = 2$), the PageRank execution is only increased about 25%. For high values of l, we achieve a high memory reduction ($l = 10$ equals 90% of memory reduction), however the execution time of an analytic is similar to the obtained with a Hash partitioner.

VII. Conclusion and Future Work

In our work, we have proposed a scalable model which allows to partition large scale streaming graphs in an efficient way. To reduce memory usage, we have sampled vertex of incoming graph to compose a subgraph. We have used this subgraph to partition the original graph, with a single-pass generic algorithm. The information consistency is maintained updating local state in each partitioner with information from the partitions. In addition, we have calculated the memory bound, the introduced error and the distributed complexity of the model.

Our solution proposes a trade-off between available memory and processing time. With our sampling functions, not having a global knowledge of the graph does not cause a significant loss in performance terms. In our experiments, we show that a 50% reduction in RAM memory only increases the processing time of the PageRank algorithm a twenty five percent.

One future investigation line is to adopt the sampling model to more complex scenarios, like weighted or evolving graphs.

Acknowledgement

This work was partially supported by the Spanish Ministry of Economy under the program "Programa de I+D+i Estatal de Investigación, Desarrollo e innovación Orientada a los Retos de la Sociedad", project identifier TIN2013-40809-R, and by COSMOS, research group recognized by the Aragonese Government. V.Medel was the recipent of a fellowship from Departamento de Industria e Innovación of the Diputación General de Aragón.

References

[1] S. Muthukrishnan, Data streams: Algorithms and applications. Now Publishers Inc, 2005.

[2] “Yahoo! AltaVista Web Page Hyperlink Connectivity Graph 2002,” 2015, URL: http://webscope.sandbox.yahoo.com/catalog.php [retrieved: May, 2015].

[3] G. Malewicz et al., “Pregel: a system for large-scale graph processing,” in Proceedings of the 2010 ACM SIGMOD International Conference on Management of data. ACM, 2010, pp. 135–146.

[4] L. Page, S. Brin, R. Motwani, and T. Winograd, “The pagerank citation ranking: Bringing order to the web.” Stanford InfoLab, Tech. Rep., 1999.

[5] M. E. Newman and M. Girvan, “Finding and evaluating community structure in networks,” Physical review E, vol. 69, no. 2, 2004, p. 026113.

[6] T. Schank and D. Wagner, “Finding, counting and listing all triangles in large graphs, an experimental study,” in Experimental and Efficient Algorithms. Springer, 2005, pp. 606–609.

[7] R. Pagh and C. E. Tsourakakis, “Colorful triangle counting and a mapreduce implementation,” Information Processing Letters, vol. 112, no. 7, 2012, pp. 277–281.

[8] C. Tsourakakis, C. Gkantsidis, B. Radunovic, and M. Vojnovic, “Fennel: Streaming graph partitioning for massive scale graphs,” in Proceedings of the 7th ACM international conference on Web search and data mining. ACM, 2014, pp. 333–342.

[9] I. Stanton and G. Kliot, “Streaming graph partitioning for large distributed graphs,” in Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2012, pp. 1222–1230.

[10] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom, “Models and issues in data stream systems,” in Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems. ACM, 2002, pp. 1–16.

[11] T. Feder, P. Hell, S. Klein, and R. Motwani, “Complexity of graph partition problems,” in Proceedings of the thirty-first annual ACM symposium on Theory of computing. ACM, 1999, pp. 464–472.

[12] Z. Bar-Yossef, R. Kumar, and D. Sivakumar, “Reductions in streaming algorithms, with an application to counting triangles in graphs,” in Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms. Society for Industrial and Applied Mathematics, 2002, pp. 623–632.

[13] J. Feigenbaum, S. Kannan, A. McGregor, S. Suri, and J. Zhang, “Graph distances in the streaming model: the value of space,” in Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms. Society for Industrial and Applied Mathematics, 2005, pp. 745–754.

[14] D. Garcıa-Soriano and K. Kutzkov, “Triangle counting in streamed graphs via small vertex covers,” Tc, vol. 2, 2014, p. 3.

[15] J. M. Ruhl, “Efficient algorithms for new computational models,” Ph.D. dissertation, Citeseer, 2003.

[16] A. D. Sarma, R. J. Lipton, and D. Nanongkai, “Best-order streaming model,” in Theory and Applications of Models of Computation. Springer, 2009, pp. 178–191.

[17] D. LaSalle and G. Karypis, “Multi-threaded graph partitioning,” in Parallel & Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on. IEEE, 2013, pp. 225–236.

[18] “Apache Storm Website,” 2015, URL: http://storm.apache.org [retrieved: May, 2015].

[19] Y. Low, J. Gonzalez, A. Kyrola, D. Bickson, C. Guestrin, and J. M. Hellerstein, “Graphlab: A new framework for parallel machine learning,” arXiv preprint arXiv:1006.4990, 2010.

[20] M. Faloutsos, P. Faloutsos, and C. Faloutsos, “On power-law relationships of the internet topology,” in ACM SIGCOMM Computer Communication Review, vol. 29, no. 4. ACM, 1999, pp. 251–262.

[21] D. J. Watts and S. H. Strogatz, “Collective dynamics of small-world networks,” nature, vol. 393, no. 6684, 1998, pp. 440–442.

[22] R. Albert and A.-L. Barabási, “Statistical mechanics of complex networks,” Reviews of modern physics, vol. 74, no. 1, 2002, p. 47.