Future Generation Computer Systems 78 (2018) 474–480

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

# DPM: A novel distributed large-scale social graph processing framework for link prediction algorithms

Alejandro Corbellini *, Daniela Godoy, Cristian Mateos, Silvia Schiaffino, Alejandro Zunino

*ISISTAN-CONICET, UNICEN, Paraje Arroyo Seco - Campus Universitario, CP7000, Tandil, Buenos Aires, Argentina*

## HIGHLIGHTS

- A novel framework that supports link-prediction algorithms is proposed.
- The programming style is similar to the fork-join style and thus, easy to use.
- Experiments showed that the framework is fast, compared to other two frameworks.
- However, network usage is slightly higher than other frameworks.

## ARTICLE INFO

*Article history:*
Received 20 June 2016
Received in revised form
16 January 2017
Accepted 12 February 2017
Available online 16 February 2017

*Keywords:*
Distributed graph processing
Recommendation algorithms
Online Social Networks

## ABSTRACT

Large-scale graphs have become ubiquitous in social media. Computer-based recommendations in these huge graphs pose challenges in terms of algorithm design and resource usage efficiency when processing recommendations in distributed computing environments. Moreover, recommendation algorithms for graphs, particularly link prediction algorithms, have different requirements depending of the way the underlying graph is traversed. Path-based algorithms usually perform traversals in different directions to build a large ranking of vertices to recommend, whereas random walk-based algorithms build an initial subgraph and perform several iterations on those vertices to compute the final ranking. In this work, we propose a distributed graph processing framework called Distributed Partitioned Merge (DPM), which supports both types of algorithms and we compare its performance and resource usage w.r.t. two relevant frameworks, namely Fork-Join and Pregel. In our experiments, we show that in most tests DPM outperforms both Pregel and Fork-Join in terms of recommendation time, with a minor penalization in network usage in some scenarios.



## 1. Introduction

The suggestion of friends, contacts or followees in social networks is one of the most prominent problems in today's Online Social Networks (OSNs). This type of recommendation serves multiple purposes, which include reducing users effort in the creation of their own personal networks, improving the quality of user engagement with social sites, favoring information spreading and contributing to the network expansion. In fact, the "Who to Follow" followee recommender service of Twitter is responsible for more than one-eighth of all new connections and it has been one of the major drivers of the company revenue [1].

The problem of suggesting users in an OSN is usually casted to a link prediction problem [2], which tries to infer a non-existent or missing relationship between two persons that is likely to occur in the future. Methods for link prediction use topology-based similarity metrics that can be categorized into path-based, neighbor-based (neighbor-based can be seen as a special case of path-based algorithms of length two) and random walk-based. Several social network recommendation algorithms based on these notions can be found in the literature [1,3,4].

Computing link prediction algorithms on real-world, large-scale social networks poses challenges regarding algorithm scalability considering the inherent huge resource needs (i.e. memory, CPU cores) and performance requirements (e.g. providing fast, real-time recommendations). Most link prediction algorithms,

* Corresponding author.
*E-mail addresses:* alejandro.corbellini@isistan.unicen.edu.ar (A. Corbellini), daniela.godoy@isistan.unicen.edu.ar (D. Godoy), cristian.mateos@isistan.unicen.edu.ar (C. Mateos), silvia.schiaffino@isistan.unicen.edu.ar (S. Schiaffino), alejandro.zunino@isistan.unicen.edu.ar (A. Zunino).

http://dx.doi.org/10.1016/j.future.2017.02.025
0167-739X/

both commercial ones and those developed in the academia, have been implemented as single-machine, single-threaded applications [1,5]. In consequence, these implementations struggle with scalability issues as the underlying social graph grows, which is commonplace in OSNs populated by a myriad of users.

The natural choice to process large amounts of social data are distributed graph frameworks. In particular, the implementation of link prediction algorithms can be adjusted to different classic distributed processing models, such as MapReduce [6], BSP [7] or, specifically for graphs, Pregel [8]. Such processing models prescribe certain primitives that govern how sub-computations are created and coordinated. However, in terms of graph operations, path-based algorithms have completely different requirements than random walk (RW)-based algorithms. The first ones perform graph traversal operations for a small amount of steps, while the second ones run a successive number of iterations over subgraphs. Hence, counting with the adequate processing framework directly impacts on the performance of the recommendation algorithm.

For graph processing, a generic model such as Fork-Join (FJ) [9] provides a classic divide-and-conquer programming style, in which vertex processing is managed by a parent job that creates and distributes independent tasks and merges their computed results. On the other hand, models like Pregel provide a vertex-centric programming style and distribute the task of merging results among all computing nodes. These frameworks have the disadvantage of being oriented to certain types of algorithms. FJ is usually a good choice for algorithms that execute over a small amount of steps, whereas in the case of iterative algorithms, FJ suffers from the bottleneck produced by the centralized join of results. Pregel, instead, performs well with iterative algorithms but it is not a good fit for algorithms that traverse paths for a small amount of steps. Moreover, algorithm code using the vertex-centric model is usually harder to develop and comprehend due to the message-based communication of results [10].

In this paper, a novel graph processing model called Distributed Partitioned Merge (DPM) is proposed. DPM is a hybrid model since it combines the simplicity of the FJ programming style and the performance and scalability provided by the Pregel framework. In this regard, the main objective of DPM is to quickly compute path-based and RW-based link prediction algorithms, while still providing an easy-to-use programming style as well as a seamlessly integration with a platform specifically designed for providing support for the development of recommender systems in OSNs [11].

In addition, a thorough comparison of FJ, Pregel and DPM for supporting the distributed computation of various link prediction algorithms from the literature was carried out considering the requirements of each type of algorithm (path-based vs. RW-based) in terms of recommendation time and resource consumption. These experiments were performed on a large, real-world, snapshot of the Twitter social network [12] containing 40 million users and 1.4 billion relationships.

The rest of this article is organized as follows. Section 2 discusses related work regarding distributed large-scale graph processing for recommender systems. Section 3 describes FJ, Pregel, and the proposed model. Section 4 reports the experimental setting and results obtained. Finally, conclusions are stated in Section 5.

## 2. Related work

There are several studies that consider ad-hoc solutions as well as framework-based solutions for distributed graph processing. [13] analyzed the challenges in general-purpose distributed graph processing and considered two ad-hoc implementations over different distributed memory architectures. [14] built an ad-hoc implementation of low-rank approximation of graphs and applied it to link prediction. Unfortunately, without proper background, ad-hoc implementations are usually hard to reuse and maintain.

There are several general-purpose frameworks that have been used to process large-scale graphs in link prediction. Frameworks such as MapReduce [6], Fork-Join [15] or RDD (Spark) [16] have been applied to various graph-related processing problems [17–19]. Other frameworks, like Pregel [8] or GraphLab [20] are specifically designed for graph-based algorithms [21,22]. DPM differs from other frameworks in its ability to support both types of link prediction algorithms.

The comparison of such processing frameworks is difficult due to their design differences. For example, the original MapReduce and Pregel specifications based their failure recovery mechanisms on checkpointing to persistent storage. Thereby, in these frameworks the penalty of using I/O is very high in comparison to RDD-based solutions such as GraphX. Nevertheless, there are some studies that compare processing frameworks, disregarding this unfairness. [23] performed a thorough comparison of the BSP and MapReduce frameworks both in terms of performance and design drivers. Similarly, [24] restricted the performance comparison to Pregel-like framework implementations. The work presented by [25] focused on vertex-centric frameworks (a.k.a. "think like a vertex" frameworks) and makes a distinction from the classic BSP-based frameworks and graph databases. In contrast, the experiments carried out in this work try to establish a common ground of comparison by executing the three selected frameworks (FJ, Pregel and DPM) over the same distributed computing platform [11], sharing the same network communication and graph storage support. The fairness in the comparison is of major importance for determining the right framework for each algorithm.

## 3. Distributed Partitioned Merge

Developing distributed link prediction algorithms is a difficult task. Thus, the development of these algorithms is often based on distributed processing platforms that provide abstractions that isolate the user from the actual underlying distributed support (both software and hardware). In fact, most distributed platforms expose a programming model or framework that simplifies algorithm development while, at the same time, enforces the correct use of the platform. Even so, the choice of which framework is better for a given algorithm depends on the performance requirements and the ease of use of the programming model. One of the drivers involved in this decision is the type of link prediction algorithm being developed.

For example, path-based link prediction algorithms fit naturally under the divide-and-conquer strategy and, thus, a Fork-Join [15] framework (also known as Split and Merge or Split and Reduce in some frameworks[1]) may be a good fit. The main entity in a distributed FJ algorithm is the FJ job, which represents the computation as a whole. In the fork stage, the FJ job is responsible of creating (or *forking*) parallel tasks to be executed in remote nodes. Once a child task finishes its execution, it sends its results back to the parent job. In this so-called *join* stage, the parent job performs awaits for all the child tasks to finish and then merges their results. The main disadvantage of this framework is that merging sub-results in a single node produces a bottleneck that may negatively impact on algorithms with multiple steps, such as RW-based link prediction algorithms.

[1] For example, GridGain's TaskSplit, http://www.gridgain.com/api/javadoc/org/gridgain/grid/compute/GridComputeTaskSplitAdapter.html.

RW-based algorithms, on the other hand, may benefit from frameworks that support iterative algorithms. Most of these frameworks are based on the "think like a vertex" or vertex-centric programming models. Pregel [8], one of the most relevant vertex-centric frameworks, provides a message-oriented model that defines a vertex function interface that the user may realize to process his/her graph algorithm. The vertex function is applied to each vertex separately. From this perspective, a vertex computes its associated results and communicates them by sending messages to other vertices. Although by following this model some algorithms are simple to implement, developing under this type of model is generally harder to understand than a simple fork and join model, because vertex-centric algorithms are modeled as a distribution or *flood* of messages between vertices [10].

Pregel also prescribes a framework for distributed graph processing. At its core, Pregel uses a logical synchronization barrier that establishes the iteration boundaries and avoids inconsistent access to results, a technique based on the well-known BSP (Bulk-Synchronous Parallel) [7] framework. A Coordinator component is responsible for managing the whole computation, notifying workers when the next iteration must start. On startup, each worker is assigned a graph partition and activates (i.e. it is scheduled for execution). This activation mechanism allows computing each vertex for a number of iterations until the vertex function calls the *voteToHalt* function, which deactivates a vertex. A vertex may be reactivated if it receives a new message. Finally, after the computation is completed, workers send the sub-results to their peers.

The Distributed Partitioned Merge (DPM) framework proposed in this paper aims at reducing the performance bottleneck observed in the Fork-Join framework while still keeping a divide and conquer approach. The MapReduce [6] approach to avoid the merge bottleneck is to split – or "shuffle" – the output of each map operation in $R$ partitions so that workers may reduce sub-results themselves without losing locality. Similarly, in Pregel, the output of each worker is divided according to the assignment of graph partitions in order to exploit the locality of the results on each iteration.

Fig. 1 shows an example of a DPM computation step. In this simple setup, three workers (labeled $Worker_1$, $Worker_2$ and $Worker_3$) process a user-defined algorithm over a graph $G$ that consists of 30 vertices. The first stage of DPM is the Partitioning Stage (shown in (1) in the Figure), in which the DPM Job (the master component that controls the execution) splits the input vertex list $V = \{v | v \epsilon G\}$ into a set of partitions (from $P_1$ to $P_3$) that are assigned to each Worker, along with a copy of the user-defined UserTask (i.e. the user-defined link-prediction algorithm). It is worth noting that in our experiments, the partitioning strategy assigns vertices to where their data is stored, in order to keep data locality. In this example, $Worker_1$ stores vertices from $V_1$ to $V_{10}$, $Worker_2$ from $V_{11}$ to $V_{20}$, and so on. Optionally, a Result Combiner component may be passed to DPM to merge different subresults belonging to a single vertex, reducing memory and network usage.

Once the Partitioning Stage is completed, each worker may execute the UserTask on its assigned partition, beginning the Distributed Merge Stage. Fig. 1 shows the workflow for $Worker_1$. Initially, those (*vertex*, *subresult*) pairs emitted from the UserTask are combined into a map of vertices and results (2). In this example, $Worker_1$ generates results for all vertices, from 1 to 30, but depending on the algorithm, input vertices may not necessarily be part of the output (e.g. the UserTask may generate results for some vertices, or even no results at all). Once the execution is completed, the current sub-results are partitioned (3) and sent to their corresponding Workers (4). Naturally, those sub-results belonging to the current Worker are stored locally. In parallel, the list of vertices that were part of the output, are sent to the DPM Job (5) so that they can be processed in the next step. On any given moment, other workers may finish their execution and send their sub-results to other workers. In this example, $Worker_1$ receives results from $Worker_2$ and $Worker_3$, and combines them with its local sub-results (6).

In opposition to Pregel, the management of the active vertices is centralized, which produces a bottleneck as the DPM job needs to join the sets of active vertices into one final list, which will be used to start the next iteration. As a result, each worker stores the results of the computation and the worker responsible for the whole DPM Job keeps the list of active vertices.

Centralizing computation is often harmful in distributed systems, however, merging a set of vertex identifiers is much cheaper than merging a table of results (e.g. a vertex and its associated results). Additionally, this decision reduces the complexity of the worker implementation. In Pregel, workers not only manage active vertices but also handle individual messages sent to each vertex, resulting in a performance and code maintenance hotspot. On the other hand, the amount of vertices in real-world graphs is often exponentially smaller than the amount of edges [12]. Due to the fact that most graph algorithms are edge-based, the penalization from managing active vertices in a single location is often negligible in comparison to edge processing.

**Table 1**
Path-based algorithms considered in the experiments.

| Algorithm | Equation |
|---|---|
| TFR | $s_{xy}^{AR} = (((AA') \bullet T)\, A) \bullet T$ where $T = (1 - A - I)$ |
| LocalPath | $s_{xy}^{LP} = A^2 + \alpha A^3$ |
| Katz | $s_{xy}^{Katz} = \sum_{l=1}^{\infty} \beta^l \cdot \lvert path_{xy}^l \rvert = \beta A + \beta^2 A^2 + \beta^3 A^3 + \cdots$ |

## 4. Experimental evaluation

This section presents the experimental setup and the obtained results, being the latter organized into path-based algorithms and random walk-based algorithms.

### 4.1. Setting

The algorithms considered in this work were developed in Graphly [26],[2] a distributed multi-model platform written in Java, which already included the Fork-Join and Pregel programming models. Besides, Graphly abstracts many operations commonly used in recommendation algorithms and provides a simple distributed Graph storage that is used by the different models. In the context of the current work, DPM was developed as a third model, leveraging some of the networking and storage tools already implemented in Graphly. Consequently, the comparison between programming models is performed over the same support, providing a fair comparison scenario.

Three path-based link prediction algorithms, listed in Table 1, were selected as representatives of this type of algorithms for our experiments. The first algorithm by Armentano et al. [3] called TFR is a simple path-based algorithm to find similar users in Twitter by using the follower/followee graph structure in the social network of the target user. The algorithm rests on the hypothesis that users who share the same followees are similar and, thus, followees from one user can be recommended to another user. The second algorithm considered is the LocalPath (LP) index [27]. LP uses information of local paths with length 2 and length 3 in order

[2] Graphly's Source Code, https://github.com/acorbellini/jLiME/tree/master/graphly.

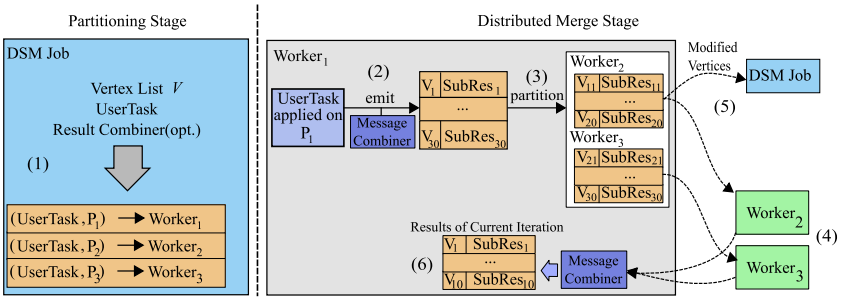



**Fig. 1.** Distributed Partitioned Merge model: Overview.

to exploit additional information of the neighbors within length 3 distances to current vertex. The third algorithm compared is Katz, which is based on the ensemble of all paths between two vertices [28]. The paths are exponentially damped by their lengths to give the shorter paths higher scores. The value of $\beta$ must be lower than the inverse of the largest eigenvalue of $A$, so that factor $\beta^l A^l$ with $l \to \infty$ is 0, and thus the Katz index converges.

Two well-known random walk-based link prediction algorithms were selected: HITS and SALSA. HITS stands for Hypertext Induced Topic Search [29]. HITS is rooted on a mutual reinforcing relationship between the notions of *hubs* and *authorities*. Authorities estimate the node value based on the incoming links, while hubs estimate the node value based on outgoing links. SALSA (Stochastic Approach for Link-Structure Analysis) [30] combines the random walk idea of PageRank [31] with the hub and authority notions of HITS. However, SALSA removes the dependency between hub and authority score computation by adding a two-step random walk. For example, the authority of a given vertex $v$ depends on the sum of authorities of vertices that have outgoing edges to those vertices that are pointed by $v$.

Regarding the computing cluster characteristics, an heterogeneous cluster of 8 nodes was used for running experiments, each having its own hardware characteristics. Three of these nodes had 16 GB of RAM whereas the remaining nodes had 8 GB of RAM. The software configuration of these nodes consisted of Ubuntu 14.04 and OpenJDK 7. Graphly was installed on each node and configured to run the necessary Pregel, Fork-Join and DPM components. Most parameters, such as number of thread and heap size, are automatically configured depending on the node characteristics. One of the manually-set parameters was the Coordinator Node parameter associated to Pregel, which was always set to the same node in order to avoid a random selection of the coordinator, which may alter the experimental results.

Experiments were carried out using a Twitter dataset[3] containing the complete follower-following network as of July 2009, provided by [12]. The dataset contains approximately 1400 million relationships between more than 41 million users. Since Twitter data are no longer available to researchers, it remains as the largest snapshot of Twitter accessible [32]. The dataset was stored on the cluster nodes using a modulo-based partitioning scheme in order to fairly distribute vertices across the cluster.

Three test groups of users were selected out of the complete dataset with the goal of illustrating the performance of the different frameworks. The first set consists on the top-10 users ordered by amount of followers, called the "Followers Set". The followers set, numbered from $Fol_1$ to $Fol_{10}$, is the most unbalanced group of users. For example, $Fol_1$ has 1.7M followers and only 563 followees. Only three users in this set have more than 200 followees (500, 400k and 770k followees respectively). Likewise, the "Followees Set" contains the top-10 users ordered by number of followees, numbered from $Fee_1$ to $Fee_{10}$. The followees set is more balanced than the followers set, having 1M followers and followees lists ranging from 100k to 500k. Finally, the third set called "Middle Set", numbered from $Mid_1$ to $Mid_{10}$, consists of users with a ratio of followee/follower between 0.4 and 0.6, ordered by number of followers. In this last group, the lists of followees/followers have an average length 100k.

**Table 2**
Subgraph size explored by each algorithm for each user group (in millions of vertices).

| | TFR | Katz | LocalPath | SALSA | | HITS |
|---|---|---|---|---|---|---|
| | | | | Hub | Authority | |
| *Followers* | 14–37 | 20–40 | 7–35 | 0.5–1.57 | 0.5–1.56 | 0.7–1.6 |
| *Followees* | 35–39 | ∼40 | 28–35 | 0.25–1.55 | 0.2–1.52 | 0.2–1.58 |
| *Middle* | ∼38 | ∼40 | 32–37 | 1.75–2.90 | 1.74–2.9 | 1.78–2.95 |

The rationale behind the selection of these groups is that the initial number of incoming or outgoing links may affect resource usage. Indeed, the size of the graphs explored by each algorithm varies for each group of users as shown in Table 2. For example, for path-based algorithms (TFR, Katz and LocalPath), the graph explored by users in the Followers group is usually half the size of the subgraphs explored by other group of users. The reason of this disparity is that most path-based algorithms process outgoing links, i.e. followees, and users in the Followers group have a small number of followees.

### 4.2. Results

The experiments were carried out as follows. A recommendation of 10 followees was computed for each combination of user, algorithm and framework. This procedure was repeated 10 times for each user to obtain the average recommendation time, network and memory consumption, along with their corresponding standard deviations. The results are presented below, grouped by the type of algorithm and user set selected.

#### 4.2.1. Path-based algorithms

Fig. 2 shows the results in terms of network usage, memory consumption and recommendation time, for the Followees Set and for the TFR, LocalPath and Katz algorithms. The values are expressed as a ratio of the obtained result in comparison to FJ, which was used as a baseline. For example, the recommendations issued to user *Fee*1 using the TFR algorithm presented +20% of memory usage for Pregel and +10% for DPM in comparison to FJ (top-left chart, first group of bars). Thus, in these charts, values below 0 are better than the FJ baseline values.

Surprisingly, FJ was the best performing framework in terms of network and memory usage in most path-based experiments. In the case of the TFR algorithm, in most cases DPM used less memory than Pregel (except for user *Fee*4), but had a clear tendency to use

[3] Twitter 2010 dataset, http://an.kaist.ac.kr/traces/WWW2010.html.

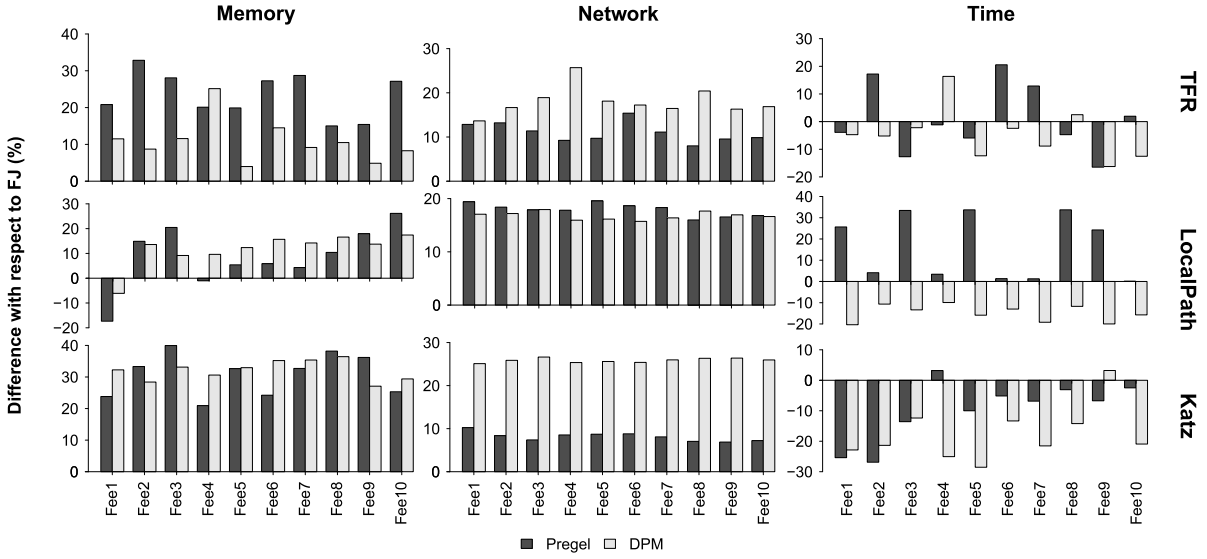



**Fig. 2.** Average memory, network and recommendation time for path-based algorithms and the Followees set of users.

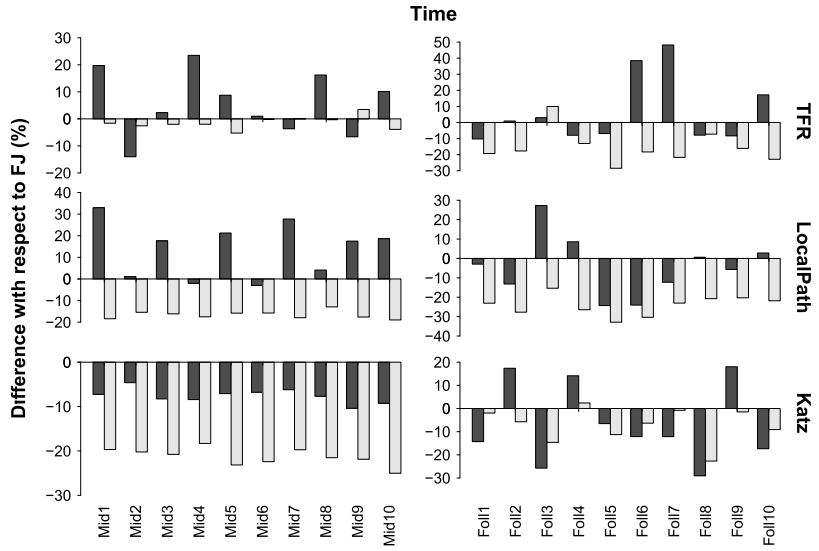



**Fig. 3.** Average recommendation time for path-based algorithms and the Followers and Middle groups.

more network bandwidth. In LocalPath, memory usage between DPM and Pregel was mixed and in Katz, it was approximately the same. Except for LocalPath, in which DPM had the same amount of network usage than Pregel, DPM transferred more data than Pregel, mainly due to the overhead introduced by the active vertex lists.

In terms of recommendation time, the results are mixed for "short" path-based algorithm like TFR or LocalPath, which compute paths up to length 3 or less. In this cases, Pregel performed worst than FJ and DPM showed improvements of +10% to +20% in comparison to FJ. In a "long" path-based algorithm like Katz, which in our experiments executed up to length 4 without filtering any vertex, the results clearly favor Pregel and DPM. These results can be explained by the number of vertices processed by Katz – Table 2 shows that it usually computes the whole graph – and the iterative support given by DPM and Pregel. FJ struggles with large sub-result merging and, due to its lack of support for iterative algorithms, it must re-partition results for each iteration.

To illustrate the differences between DPM and FJ, consider the graph used in the example of Fig. 1. In this example, a graph of 30 vertices is processed by 3 workers. The received vertices to be processed in the current step are kept in the Coordinator node, e.g. $Worker_1$. Additionally, DPM keeps the results distributed among the 3 workers by using a partitioning scheme (in our experiments, it is aligned with the way vertex data, such as followees and followers, is stored). This ultimately means that, if $Worker_1$ generates the result $v_1 \rightarrow 10$, $Worker_2$ generates $v_1 \rightarrow 20$ and $Worker_3$ generates $v_1 \rightarrow 5$, these three results for $v_1$ are merged in $Worker_1$, i.e. the worker assigned to store and process $v_1$. If the algorithm sums these results, then this merge would result in $v_1 \rightarrow 35$. The Coordinator, i.e. $Worker_1$, would receive $v_1$ and the other vertices reached by the current algorithm step, but only the vertex identifiers. FJ, on the other hand, keeps the current results in a single worker, for example $Worker_1$, and splits them among the 3 available workers in order to compute them. In this case, not only the results for $v_1$ are merged in $Worker_1$, but also the results of the other vertices. Thus, all workers send their sub-results to $Worker_1$ on each step, overloading $Worker_1$ and underusing the other workers capabilities. Conversely, DPM only centralizes the vertex identifiers and appends them to a list of vertices that must be processed in the next computation step.

In the experiments performed on the Middle and Followers sets, a similar behavior was observed in terms of network and memory

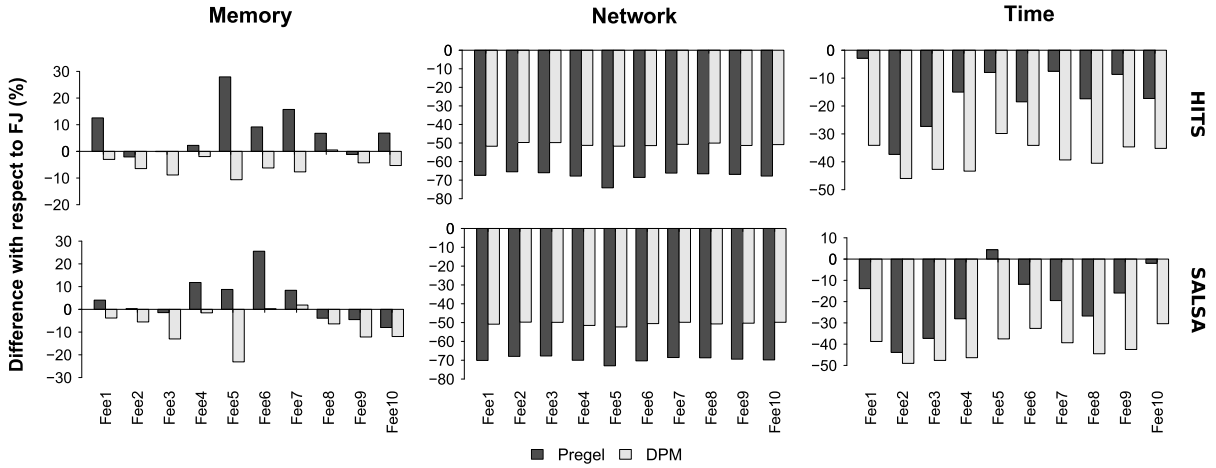



**Fig. 4.** Average memory, network and recommendation time for path-based algorithms and the Followees set of users.

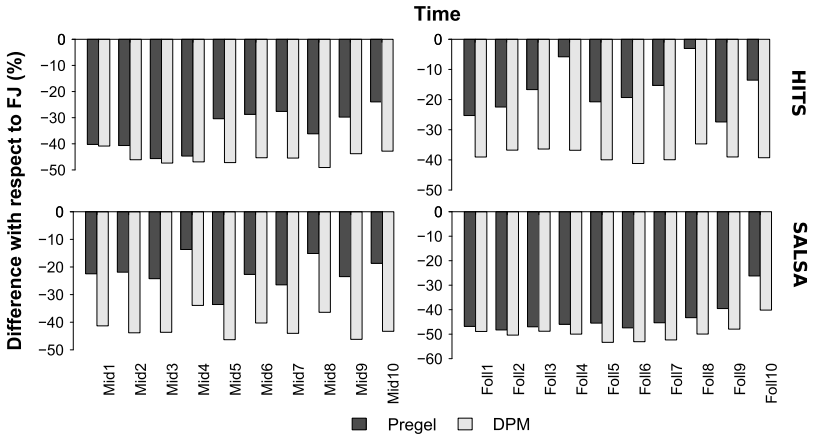



**Fig. 5.** Average recommendation time for RW-based algorithms for users in the Middle and Followers set.

consumption and, thus, the results were omitted. However, the Followers set showed different results, in particular for the Katz algorithm, as the size of the graphs explored was much smaller and, thereby, the penalization for FJ was lower. This behavior can be seen in the recommendation time measured for the Followers and Middle users, shown in Fig. 3.

#### 4.2.2. RW-based algorithms

The results obtained from running SALSA and HITS algorithms for 10 iterations were notably different from those obtained from path-based algorithms. The smaller sub-graphs explored and the iterative nature of the algorithms posed different requirements over the processing frameworks. The experimental results for the Followees set are shown in Fig. 4. In terms of memory usage, apart from some specific cases, the differences between the three frameworks are not high, although DPM network usage was always lower. However, regarding memory usage, Fork-Join suffers from its lack of support for iterative algorithms and quickly surpass the network usage of DPM and Pregel. In this category, DPM was the most network intensive, approximately +15% consumption than Pregel.

DPM and Pregel offered the best recommendation times, presenting times +30% to +40% faster than FJ. Moreover, DPM computed RW-based algorithms, in average, almost +15% to +20% faster than Pregel. This pattern is confirmed in the results obtained from the Middle and Followers set, as shown in Fig. 5. Similarly to the previous section, the results of memory and network usage for the Middle and Followers sets were omitted due to its similarity to the results presented for the Followees group.

## 5. Conclusions

In this paper we presented DPM, a novel framework that resulted from analyzing the requirements of different link prediction algorithms. One of DPM's major advantages is that it offers a simple fork-join model while performing well for iterative algorithms. The vertex centric programming model provided by Pregel, is often harder to comprehend because the developer must think in terms of vertex to vertex messaging and how messages "flood" the graph. The Fork-Join framework offers a simpler programming model, but the join bottleneck severely reduces its scalability. DPM tries to reduce this costly bottleneck by distributing the merging of sub-results and centralizing only the list of currently active vertices.

As shown in the experiments, the merge of active vertices in DPM has almost no impact on the overall time for recommendation while it generates a moderate increase in network consumption. In terms of memory usage, both Pregel and DPM had a higher memory consumption than FJ, specially for the path-based experiments.

For path-based algorithms, DPM ranked as the fastest model. It is worth noticing that the differential in recommendation time achieved by DPM for a single user can significantly reduce the total time required for computing recommendation for a batch of users. Nonetheless, on limited network and memory scenarios, or in situations where network transferences are billed (as in paid Cloud environments), the Fork-Join alternative may be a better choice than Pregel or DPM.

Regarding RW-based algorithms, FJ required almost twice the time to provide recommendations and produced almost twice the

amount of network traffic than the other frameworks, enforcing the hypothesis that choosing the correct framework for a given type of algorithm is important to provide fast and resource-efficient recommendations.

Future work regarding DPM includes testing its performance using graph-wide iterative algorithms like PageRank and SimRank. Moreover, the comparison against other frameworks like GraphX or GraphLab is also in progress.

## References


[1] A. Goel, P. Gupta, J. Sirois, D. Wang, A. Sharma, S. Gurumurthy, The who-to-follow system at twitter: Strategy, algorithms, and revenue impact, Interfaces 45 (1) (2015) 98–107.

[2] D. Liben-Nowell, J. Kleinberg, The link-prediction problem for social networks, J. Amer. Soc. Inf. Sci. Technol. 58 (7) (2007) 1019–1031.

[3] M. Armentano, D. Godoy, A. Amandi, Followee recommendation in Twitter based on text analysis of micro-blogging activity, Inf. Syst. 38 (8) (2013) 1116–1127.

[4] A. Papadimitriou, P. Symeonidis, Y. Manolopoulos, Scalable link prediction in social networks based on local graph characteristics, in: Proceedings of the 9th International Conference on Information Technology: New Generations, ITNG, Thessaloniki, Greece, 2012, pp. 738–743.

[5] Y. Jing, X. Zhang, L. Wu, J. Wang, Z. Feng, D. Wang, Recommendation on Flickr by combining community user ratings and item importance, in: Proceedings of the IEEE International Conference on Multimedia and Expo, ICME 2014, Chengdu, China, 2014, pp. 1–6.

[6] J. Dean, S. Ghemawat, MapReduce: Simplified data processing on large clusters, Commun. ACM 51 (1) (2008) 107–113.

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

[8] G. Malewicz, M.H. Austern, A.J.C. Bik, J.C. Dehnert, I. Horn, N. Leiser, G. Czajkowski, Pregel: A system for large-scale graph processing, in: Proceedings of the 2010 International Conference on Management of Data, SIGMOD'10, Indianapolis, IN, USA, 2010, pp. 135–146.

[9] C. Mateos, A. Zunino, M. Hirsch, EasyFJP: Providing hybrid parallelism as a concern for divide and conquer Java applications, Comput. Sci. Inf. Syst. 10 (3) (2013) 1129–1163.

[10] S. Hong, J. Van Der Lugt, A. Welc, R. Raman, H. Chafi, Early experiences in using a domain-specific language for large-scale graph analysis, in: First International Workshop on Graph Data Management Experiences and Systems, ACM, 2013, p. 5.

[11] A. Corbellini, C. Mateos, D. Godoy, A. Zunino, S. Schiaffino, An architecture and platform for developing distributed recommendation algorithms on large-scale social networks, J. Inf. Sci. 41 (5) (2015) 686–704.

[12] H. Kwak, C. Lee, H. Park, S. Moon, What is Twitter, a social network or a news media?, in: Proceedings of the 19th International Conference on World Wide Web, WWW'10, Raleigh, NC, USA, 2010, pp. 591–600.

[13] A. Lumsdaine, D. Gregor, B. Hendrickson, J. Berry, Challenges in parallel graph processing, Parallel Process. Lett. 17 (1) (2007) 5–20.

[14] X. Sui, T.-H. Lee, J. Whang, B. Savas, S. Jain, K. Pingali, I. Dhillon, Parallel clustered low-rank approximation of graphs and its application to link prediction, in: Languages and Compilers for Parallel Computing, in: LNCS, vol. 7760, Springer, ISBN: 978-3-642-37657-3, 2013, pp. 76–95.

[15] C. Mateos, A. Zunino, M. Campo, An approach for non-intrusively adding malleable fork/join parallelism into ordinary JavaBean compliant applications, Comput. Lang. Syst. Struct. 36 (3) (2010) 288–315.

[16] R.S. Xin, J.E. Gonzalez, M.J. Franklin, I. Stoica, GraphX: A resilient distributed graph system on spark, in: Proceedings of the 1st International Workshop on Graph Data Management Experiences and Systems, GRADES'13, New York, NY, USA, 2013, pp. 2:1–2:6.

[17] L. Cao, B. Cho, H.D. Kim, Z. Li, M.-H. Tsai, I. Gupta, Delta-SimRank computing on MapReduce, in: Proceedings of the 1st International Workshop on Big Data, Streams and Heterogeneous Source Mining: Algorithms, Systems, Programming Models and Applications, (BigMine'12), ACM, Beijing, China, ISBN: 978-1-4503-1547-0, 2012, pp. 28–35.

[18] H. Lu, M. Halappanavar, A. Kalyanaraman, Parallel heuristics for scalable community detection, Parallel Comput. (ISSN: 01678191) 47 (0) (2015) 19–37.

[19] N. Buzun, A. Korshunov, V. Avanesov, I. Filonenko, I. Kozlov, D. Turdakov, H. Kim, EgoLP: Fast and distributed community detection in billion-node social networks, in: 2014 IEEE International Conference on Data Mining Workshop, 2014, pp. 533–540, ISSN 23759259.

[20] Y. Low, D. Bickson, J. Gonzalez, C. Guestrin, A. Kyrola, J.M. Hellerstein, Distributed GraphLab: A framework for machine learning and data mining in the cloud, Proc. VLDB Endow. 5 (8) (2012) 716–727.

[21] M. Han, K. Daudjee, K. Ammar, M.T. Özsu, X. Wang, T. Jin, An experimental comparison of pregel-like graph processing systems, Proc. VLDB Endow. (ISSN: 21508097) 7 (12) (2014) 1047–1058.

[22] B. Heitmann, An open framework for multi-source, cross-domain personalisation with semantic interest graphs, in: Proceedings of the Sixth ACM Conference on Recommender Systems - RecSys'12, 2012, p. 313.

[23] T. Kajdanowicz, P. Kazienko, W. Indyk, Parallel processing of large graphs, Future Gener. Comput. Syst. 32 (2014) 324–337.

[24] M. Han, K. Daudjee, Giraph unchained: Barrierless asynchronous parallel execution in Pregel-like graph processing systems, Proc. VLDB Endow. 8 (9) (2015) 950–961.

[25] R.R. McCune, T. Weninger, G. Madey, Thinking like a vertex: a survey of vertex-centric frameworks for distributed graph processing, Distrib. Parallel Cluster Comput. (ISSN: 15577341) (2015) 46556.

[26] A. Corbellini, A distributed platform to ease the development of recommendation algorithms on large-scale graphs, in: Proceedings of the 24th International Conference on Artificial Intelligence, Doctoral Consortium, AAAI Press, 2015, pp. 4353–4354.

[27] L. Lü, C.-H. Jin, T. Zhou, Similarity index based on local paths for link prediction of complex networks, Phys. Rev. E 80 (4) (2009) 046122.

[28] L. Katz, A new status index derived from sociometric analysis, Psychmetrika 18 (1) (1953) 39–43.

[29] J.M. Kleinberg, Authoritative sources in a hyperlinked environment, J. ACM (ISSN: 00045411) 46 (5) (1999) 604–632.

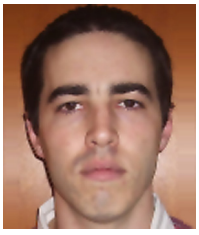
[30] R. Lempel, S. Moran, SALSA: The stochastic approach for link-structure analysis, ACM Trans. Inf. Syst. (ISSN: 10468188) 19 (2) (2001) 131–160.

[31] L. Page, S. Brin, R. Motwani, T. Winograd, The PageRank citation ranking: Bringing order to the Web, Tech. Rep. 1999-66, Stanford InfoLab, 1999.

[32] S. Faralli, G. Stilo, P. Velardi, Large scale homophily analysis in twitter using a twixonomy, in: Proceedings of the 24th International Conference on Artificial Intelligence, (IJCAI 2015), AAAI Press, Buenos Aires, Argentina, 2015, pp. 2334–2340.


**Alejandro Corbellini** is a postdoctoral researcher at CONICET. He received his Ph.D. in Computer Science in 2016 at UNICEN University. He is also a graduate teaching assistant at UNICEN. His main research interests are recommender systems, distributed programming and large-scale data mining.

**Daniela Godoy** received her Ph.D. degree in computer science from UNICEN University in 2005. She is a full-time professor in the Computer Science Department at UNICEN, member of ISISTAN Research Institute and researcher at CONICET. Her research interests include intelligent agents, user profiling and text mining.



**Cristian Mateos** received a Ph.D. degree in Computer Science from the UNICEN University in 2008. He is a full time Assistant Professor at the UNICEN and member of ISISTAN-CONICET. His main research interest are parallel/distributed programming, Grid middlewares and Service-oriented Computing.

**Silvia Schiaffino** is an Adjunct Professor at UNICEN University, Tandil, Argentina. She works at ISISTAN Research Institute and as an Independent Researcher at CONICET. She received her Master degree in Systems Engineering in 2001 and her Ph.D. in Computer Science in 2004, both from UNICEN. Her main research interests are intelligent agents, personalization, recommender systems, and data mining.

**Alejandro Zunino** has a Ph.D. in computer science from UNICEN University. He is an adjunct professor at UNCPBA and member of ISISTAN and CONICET. His research areas include grid computing, service-oriented computing, Semantic Web services, and mobile computing.