A Multi-layer Framework for Graph Processing via Overlay Composition

Alessandro Lulli[2], Patrizio Dazzi[1(✉)], Laura Ricci[2], and Emanuele Carlini[1]

[1] Istituto di Scienza e Tecnologie dell'Informazione "A. Faedo", Consiglio Nazionale delle Ricerche (ISTI-CNR), Pisa, Italy
{patrizio.dazzi,emanuele.carlini}@isti.cnr.it
[2] Dipartimento di Informatica, Università di Pisa, Pisa, Italy
{lulli,ricci}@di.unipi.it

Abstract. The processing of graph in a parallel and distributed fashion is a constantly rising trend, due to the size of the today's graphs. This paper proposes a multi-layer graph overlay approach to support the orchestration of distributed, vertex-centric computations targeting large graphs. Our approach takes inspiration from the overlay networks, a widely exploited approach for information dissemination, aggregation and computing orchestration in massively distributed systems. We propose Telos, an environment supporting the definition of multi-layer graph overlays which provides each vertex with a layered, vertex-centric, view of the graph. Telos is defined on the top of Apache Spark and has been evaluated by considering two well-known graph problems. We present a set of experimental results showing the effectiveness of our approach.

1 Introduction

The current production of data is far beyond what has been experienced before. For example, in 2012 was created 2.5 exabytes (2.5×10^{18}) of data every day [14]. This data comes from multiple and heterogeneous sources, ranging from scientific devices to business transactions. In many of these contexts, data is modelled as a graph, such as social network graphs, road networks and biological graphs. Clearly, data of this size makes often infeasible to process these graphs by exploiting the computational and memory capacity of a single machine. Indeed, these problems are usually faced by exploiting parallel and distributed computing architectures.

Many solutions for the parallel and distributed processing of large graphs have been designed so far. Most of the methodologies currently adopted fall in two main approaches. On the one hand, low-level techniques (such as send/receive message passing or, equivalently, unstructured shared memory mechanisms) are often complex, error-prone, hard to maintain and, usually, their tailored nature hinder their portability. On the other hand, it can be observed a wide use of the MapReduce paradigm proposed by Dean and Ghemawat [9], and inspired by the well-known map and reduce paradigms that, across the years, have been provided by a number of different frameworks [1,7,8].

S. Hunold et al. (Eds.): Euro-Par 2015 Workshops, LNCS 9523, pp. 515–527, 2015.
DOI: 10.1007/978-3-319-27308-2_42

The MapReduce paradigm is often used in contexts that are different from the ones for which it has been conceived. In fact, some of the most notable existing implementations of such paradigm (e.g. Apache Hadoop and Apache Spark) are often used to implement algorithms which could be more fruitfully implemented by means of different parallel programming paradigms or different ways to orchestrate their computation. This is especially true when dealing with large graphs [13]. In spite of this, some MapReduce based frameworks achieved a wide diffusion due to their ease of use, detailed documentation and very active communities of users. As a consequence, in the last years, several solutions based on the MapReduce have been adapted for performing analysis on large graphs in a native way, supporting data streams, large graphs analysis, etc. Some of these solutions have been inspired by the BSP bridging model [20], such as the Pregel framework [13], GraphX [24] and Giraph [6]. A common trait shared by these frameworks is that they provide the possibility to describe graph processing applications from the point of view of a vertex in the graph. Every vertex processes the same function independently and can only access its local context which limits its knowledge to its neighbourhood, without having a global view on the graph. In vertex-centric frameworks the complexity of the distribution and communication is cut off from the data scientist, who can only focus on the algorithmic issues.

In this paper we propose Telos, a high-level multi-layer programming environment that raises the level of abstraction of vertex-centric graph processing frameworks by supporting the composition of graph-based overlays. By means of its support to compositionality it also promotes the reuse and the combination of existing solutions and algorithms. Our approach takes inspiration by the similarities existing between large graphs and massively distributed architectures, e.g., P2P systems. In fact, an effective, efficient and interesting approach adopted in these systems to orchestrate the computation and spread the information strongly relates with (multi-level) overlay networks. An overlay can be thought as an alternative network, built upon the existing physical network, connecting nodes by means of logical links established according to a well-defined goal. It can be noticed how, by construction, the building blocks of overlays match the key elements of graphs. In fact, vertices can be seen as networked resources and the edges as links. This view gives the possibility to dynamically define graph topologies different from the original one, so enabling each vertex to choose the most promising neighbours to speed-up the convergence of the distributed algorithm.

We highlight three main aspects that can be adopted to graph processing: (i) *Local knowledge:* each node maintains a limited amount of information and a limited neighbourhood. During the computation each node relies only its own data and the information received from such neighbourhood; (ii) *Multiple views:* the definition of multi-layer overlays drives the node neighbourhood selection accordingly to specific goals, a concept successfully exploited in peer-to-peer networks [2,10]; (iii) *Approximate solutions:* algorithms running on top of overlays usually are conceived to deal with approximated data and to find approximated

solutions. Telos defines a distributed framework able to support all previous strategies. Finally, the contributions of this paper can be summarized as the following:

- the definition of a high-level multi-layer programming framework targeting computations on large graphs;
- the presentation of our publicly available implementation of the framework[1] on top of Apache Spark [26] providing both a high level API to define custom layers and some built-in layers.
- a threefold evaluation of our framework to assess the scalability and two proof-of-concepts directed to exploit a multi layer evolving topology and to improve the quality of the result of a state of the art balanced k-way partitioning algorithm.

2 The Telos framework

The vertex-centric model is an approach to define computation for processing large scale graphs that have become more and more adopted in the last years, even in very different contexts. According to such model each computation is organised in a BSP-like fashion [20]. A BSP computation consists of three main pillars. *Concurrent computation* every participating computing entity (a vertex in our case) may perform local computations, i.e., computing by making use of values stored in the local memory/storage. *Barrier synchronisation* after conducting its local computation every vertex waits until all other ones have reached the same point. *Communication* vertices exchange data between themselves before they reach the barrier. Each message sent at superstep S is received at superstep $S + 1$.

During a superstep, each vertex receives messages sent in the previous iteration and executes a user-defined function that can modify its own state. Once the computation of such function is terminated, it is possible for the computing entity to send messages to other entities. The communication model is based on a barrier at the end of each superstep. Each message sent at superstep S is received at superstep $S+1$. In other words, before the superstep $S+1$ can begin, all vertices must have executed the superstep S.

This approach has been fruitfully exploited for computing data analysis on large graphs. Indeed, in some ways, programming according to the vertex-centric model recalls the definition of epidemic (or gossip) computing. A widely used approach in massively distributed systems that leads the nodes of a network to work independently but having a common, overall, aim. Indeed, according to a gossip protocol, nodes build network overlays by means of the information they exchange one each others, similarly to the vertices in the vertex-centric model that communicate to each other thought their graph neighbourhood.

An overlay consists of a logical communication topology, built over an underlying network, that is maintained and used by nodes. Many gossip protocols are

[1] https://github.com/hpclab/telos.

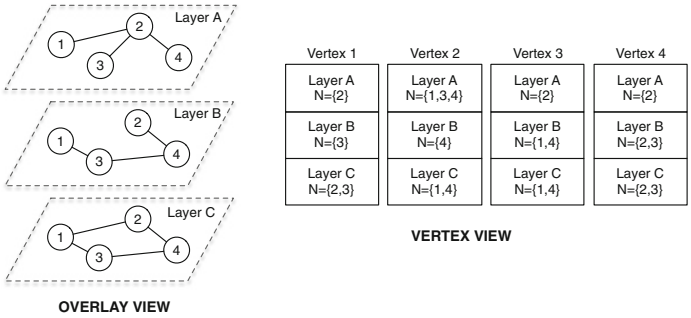


Fig. 1. Layered architecture and interactions

combined into layers [12]. The communication can be exploited over multiple overlays with each overlay devoted to the computation of a peculiar task. As an example, Vicinity [22] is a two-layered protocol aimed at discovering similar nodes in a network in a fully distributed fashion. In Vicinity, one layer obtains a random sampling of nodes from the network, and the other layer keeps the nodes more similar with a given context.

Following the evolution of the gossip protocols, the main idea of Telos is to augment the classic vertex-centric frameworks by adding the support to a multiple layers architecture. Each vertex is associated with multiple protocols (i.e. one for each user-defined functions) which are organised into layers. In Fig. 1 is depicted a visual representation of this concept in the overlay view. For each protocol, a vertex v maintains a local context and a neighbourhood, the former represents the "state" of the vertex v considering a given protocol, whereas the latter represents the set of vertices that are exchanging messages with v in that protocol. Both the context and the neighbourhood can change during the computation and across the supersteps, given the possibility of building evolving "graph overlays".

Telos enables three different types of interactions that involve the vertices belonging to the graph: (i) *intra-vertex:* it is the access of a vertex v when executing the protocol p to the context of a protocol different from p on v; Fig. 1 represents this interaction in the vertices by showing that the context of each vertex is treated separately. (ii) *intra-protocol:* it is a message from a vertex v to a vertex u sent to the same protocol p. Intra-protocol messages sent during the super-step S will be received by the vertex u at super-step $S + 1$ (following the BSP model). (iii) *extra-protocol:* a vertex v when executing the protocol p requesting the context of protocol m on vertex u, with $p \neq m$, sends a request message to u. Upon the reception of the message, the context of vertex u at protocol m is sent back. This kind of messages are handled directly by Telos and no additional operations must be provided by the users.

Telos brings to the vertex-centric model many of the typical advantages of a layered architecture: (i) *modularity*, protocols can be composed and it is easy to improve functionality by adding layers; (ii) *isolation*, a modification of a protocol

does not affect the logic of the protocols on the other layers; (iii) *reusability*, protocols can be general enough to be reused for many and possibly different computations. More in detail, Telos ease the task, for programmers, of combining different protocols. Each protocol is managed independently by Telos and all the communications and the organisation of the records are all in charge of Telos, which it manages in a fully distributed manner. Currently, the Telos framework is built on top of Spark [26]. To realise data management and distribution, the Spark framework exploits the Resilient Distributed Dataset (RDD) [25], on top of which are defined collective operations such as map, reduce and join. We developed Telos on top of the standard Spark's API, as an additional abstraction that organises the layered vertex-centric view. All the tasks for managing RDDs (including checkpointing and persistence) are transparent to the programmers and managed by our framework. The framework coordinates the protocols and masquerades the underlying support to ease the application development. The framework handles all the burden required to create the initial set of vertices and messages and provides to each vertex the context it requires for the computation. Communications are completely hidden to the programmer as well. Telos handles data dispatch to the target vertices (and the corresponding layers). In addition, the framework manages also minor activities like the control on the maximum number of steps to perform and the persistence of intermediate data in case of failures.

2.1 Protocols

Protocols are first-class entities in Telos, they drive the computation and organise the topologies in each layer of the framework. Each protocol orchestrates the context of the vertices, such as their local state and their neighbourhood, in a vertex-centric view of the graph. In fact, each protocol is in charge of (i) modifying the state of the vertices, (ii) defining a representation of the state of the vertices, which are eventually sent as messages to other vertices, and (iii) defining custom messages aimed at supporting the orchestration of the nodes.

Table 1 reports the API that a Protocol object can implement. The Protocol interface abstracts the structure of the computation running on each vertex. The core logic of a Protocol is contained in the `compute()` method. Once called, say at superstep S, the `compute()` method can access to the state of the vertex at superstep $S-1$ and all the messages received by such vertex in step S, as well. The contract of the `compute()` method requires to return a new vertex context and a set of messages that will be dispatched to the target vertices at the superstep $S+1$. Note that the receivers of the messages are not necessarily part of the neighbourhood of vertex v at super-step S, namely the vertex v can send messages also to vertices not belonging to its neighbourhood. The termination of a Protocol is coordinated by a "halt" vote. At the end of its computation each vertex votes to halt, and when all vertices voted to halt, the computation terminates.

The frequency at which a protocol is activated (w.r.t. the supersteps) is regulated by `getStartStep()` and `getStepDelay()` methods. These methods are

Table 1. The Protocol API

Function	Description
`setStartStep()`	Defines the first step on which the `compute()` is called
`setStepDelay()`	Defines the delay in terms of steps between two successive calls of `compute()`
`beforeSuperstep()`	Defines aggregators and combiners to be run before the `compute()`
`afterSuperstep()`	Defines aggregators and combiners to be run after the `compute()`
`compute()`	Defines the protocol behaviour. Receives messages sent at the previous step, modifies the state, creates new messages and eventually votes to halt
`init()`	To define custom initialization procedures
`createContext()`	Sets the initial context of a vertex
`createInitMessages()`	Sets the initial messages

useful when a computation involves different protocols characterised by different converge time, i.e., the amount of steps it needs to converge to a useful result. By means of these methods it is possible to regulate the activations of the protocols with respect to the amount of elapsed supersteps. Figure 2 graphically depicts the behaviour of these methods. In the figure, *Protocol A* is activated at each superstep, i.e., if invoked, its implementation of `getStepDelay()` method, will return 1, whereas the very same call on *Protocol B* will return 2. In a pretty similar fashion the method `getStartStep()` drives the *first* activation of a protocol. Going back to the aforementioned figure, when called on *Protocol A* it will return 0, whereas it will return 1 if called on *Protocol B*. Finally, `beforeSuperstep()` and `afterSuperstep()` allow to define aggregators and combiners to be executed before and after the execution of the protocol. Telos comes with two built-in protocols for building dynamic topologies or exploiting properties of graphs:

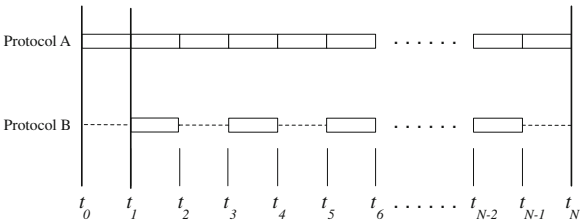


Fig. 2. The effect of `getStartStep()` and `getStepDelay()` methods

Random Protocol. The aim of this protocol is to provide to each vertex a random vertex identifier upon request. The vertex identifier must be taken uniformly and randomly in the space of identifiers of graph vertices. Telos provides two versions

of this protocol, the first implementing a gossip random peer sampling protocol [21], the second implementing a distributed random number generator[2].

Ranking Protocols. These protocols are widely exploited in gossip frameworks to create and manage topology overlays [10,22]. The overlays are created and maintained according to a ranking function which measures the similarities between two vertices. In Telos we implemented a generic ranking protocol able to take as input the ranking function. It dynamically keeps in the neighbourhood of each vertex the more similar vertices according to the user defined ranking function. As an example, if every vertex is represented as a point in a two dimensional space and the similarity metrics is the euclidean distance, the ranking protocol eventually keeps in the neighbourhood of each vertex the k closest vertices.

3 Evaluation

To evaluate the effectiveness of our approach we conducted a set of three experiments. In Sect. 3.1 we validate the framework, in Sect. 3.2 we evaluate its scalability, and finally Sect. 3.3 tests our framework in a real scenario presenting a multi-layer solution for graph partitioning.

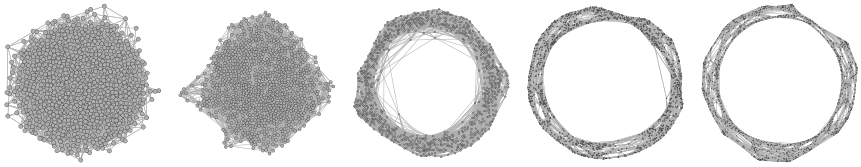
3.1 Torus Overlay

The aim of this experiment is to validate our approach. To this end we show a layered architecture built by means of our Telos framework to port concepts from the massively distributed systems to large graph processing solutions. As a proof-of-concept we built an experiment that organises the vertices and the edges of a graph according to a determined ranking function. Exploiting ranking functions to organise the overlay topology is a widely adopted approaches in highly distributed systems that we believe would be also useful in graph processing systems to drive the orchestration of the computation. In this case we a torus shaped overlay. The implementation is a two-layers approach. The bottom layer implements a random protocol, and the upper layer implements the ranking protocol. We tested the implementation on a graph made of $20K$ vertices. The results in Fig. 3 show the evolution of the graph in real time. The topology recalls the shape of a torus already at super-step 10. At super-step 20 no edges connects "distant" vertices in the torus. This experiment shows that, if properly instrumented, Telos can correctly manage multiple layers and can build the requested topology in a fewer number of super-steps.

3.2 Scalability

This experiment evaluates how the Telos framework manages larger input graphs when increasing the number of cores involved in the computation. For this experiment, we built two Erdos-Renyi random graphs (1 M vertices 5 M edges the first,

[2] http://www.cs.rit.edu/ark/pj2/doc/edu/rit/util/package-summary.html.



(a) step 0 (b) step 5 (c) step 10 (d) step 15 (e) step 20

Fig. 3. Evolution of Torus Computation at different Supersteps

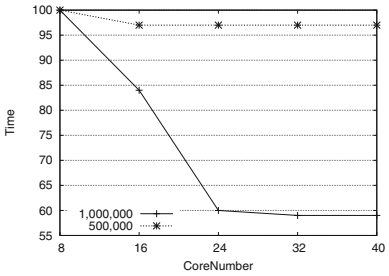


Fig. 4. Scalability as a function of the number of cores

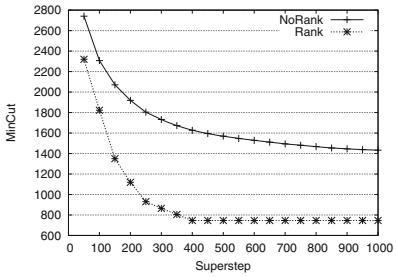


Fig. 5. Convergence of edge-cut over supersteps

500K vertices 1.5 M edges the second) generated with the Snap library [11]. We run the torus overlay experiment described in Sect. 3.1 with both graphs varying the number of cores: $\{8, 16, 24, 32, 40\}$, and measuring of the convergence time. Results are presented in Fig. 4. Values are normalised, independently for each graph, in the range [1 ; 100], with 100 being the highest execution time. Each value is the average of 3 independent runs computed by using quad-cores workstations. Considering the 500K graph, it can be observed that using more than 16 cores does not brings any benefit. When considering the larger graph, we achieve performance benefits till 32 cores. In detail, with respect to 8 cores, 16 cores cut 15 % of the convergence time, and 24 cores cuts the 40 %.

3.3 Graph Partitioning

JA-BE-JA [16] is a distributed algorithm for resolving the balanced k-way graph partitioning problem and it works as the following. Initially, it creates k logical partitions on the graph by randomly assigning colours to each vertex, with the number of colours equals to k. Then, each vertex attempts to swap its own colour with another vertex according to the most dominant colour among its neighbours, by: (i) selecting another vertex either from its neighbourhood or from a random sample, and (ii) considering the "utility" of performing the colour swapping operation. If the colour swapping would decrease the number of edges between vertices characterised by a different colour, then the two vertices swap their colour; otherwise, they keep their own colours. In a previous work [3] we

presented an implementation of JA-BE-JA in Apache Spark, outlining the adaptations that have been required in order to efficiently adapt the algorithm to match a BSP-like structure. In its original formulation, JA-BE-JA assumes that each vertex has complete access to the context of its neighbours and also their neighbourhood. To enforce this assumption, we initially introduced specific messages to retrieve the neighbourhood of any vertex. However, we noticed that forcing a strong consistency of such information slows down the performance too much. As a consequence, we provided an alternative implementation (called GP-SPARK) that introduces a degree of approximation to accelerate the computation, in which vertices piggyback their neighbourhood information in other messages. This mechanism causes the vertices to apply the local heuristics on possibly stale data, but increases the performance of the original approach providing a comparable quality of results with respect to the original version. The original GP-SPARK works with a two layered architecture composed as following: (i) the *colour swapping* protocol, that attempts the colour swapping targeting either a vertex from the local neighbourhood or from the random sampling, and (ii) the random sampling protocol that provides some random vertices to the colour swapping protocol. Here, we present GP-TELOS, an improved version of GP-SPARK that introduces a new layer with the *border ranking protocol* aimed to boost the quality of results. The objective is to give to JA-BE-JA the possibility to select from a better set of vertices when attempting colour swapping. The new layer sorts the vertices according to a *ranking function* that favours vertices to be connected with others that represent good swapping candidates. For instance, a blue vertex having 3 red neighbours would be ranked high from a red vertex having 3 blue neighbours. The ranking protocol orders the neighbourhood vertices according to a function to sort vertices being better swapping candidates. We compared the performance of GP-TELOS and GP-SPARK by means of the following metrics: (i) *edge-cut:* the number of edges that cross the boundaries of each subgraph. This metrics gives an estimation about the quality of the cut, with lower values corresponding to a better cut, and (ii) *convergence:* the number of supersteps required to achieve a substantially definitive edge-cut result. Our aim is to show that Telos does not affect the performance in term of supersteps to find a solution with respect to the GP-SPARK implementation. The experiments have been conducted on two datasets taken from the Walshaw archive [23] (3elt and vibrobox) and from the Facebook social network[3]. Figure 5 shows the convergence time in term of supersteps for the 3elt dataset with $K = 2$. Results with other datasets and different values of K are not included due to space constraints but they exhibit similar trends. The results show evidence that convergence is similar between GP-SPARK and GP-TELOS, as in both cases they achieve an almost-definitive edge-cut around the 400th superstep. In particular, after the 400th superstep GP-TELOS is stable, whereas GP-SPARK is converging but it is improving the result in every step marginally. Also, GP-TELOS yields a much better quality of results, achieving half the edge-cut of GP-SPARK. Table 2 presents the edge-cut obtained by GP-TELOS and GP-SPARK, averaging the results

[3] http://socialnetworks.mpi-sws.org/.

Table 2. Edge-cut value for GP-TELOS and GP-SPARK with the three datasets

K	3elt			Vibrobox			Facebook		
	GP-TELOS	GP-SPARK		GP-TELOS	GP-SPARK		GP-TELOS	GP-SPARK	
2	750	**1,433**	**(+91 %)**	14,812	22,244	(+50 %)	75,690	80,971	(+7 %)
4	1,810	2,903	(+60 %)	30,432	40,358	(+33 %)	147,991	157,282	(+6 %)
8	3,048	4,473	(+47 %)	43,728	56,954	(+30 %)	256,902	**245,682**	**(−4 %)**
16	4,191	6,344	(+51 %)	54,339	75,051	(+38 %)	348,494	353,061	(+1 %)
32	5,241	8,491	(+62 %)	67,787	95,858	(+41 %)	415,315	457,257	(+10 %)
64	6,419	10,622	(+65 %)	88,953	116,149	(+31 %)	520,391	552,714	(+6 %)

of 5 runs. We executed multiple runs by varying the number of the graph partitions with the values $K = \{2, 4, 8, 16, 32, 64\}$. It can be noticed that GP-TELOS obtains a better edge-cut in all the datasets and in all the configurations but with the Facebook dataset and 8 partitions. However also in this configuration GP-TELOS provides an edge-cut similar (just 4 % less) to the one in the GP-SPARK version. Overall, the GP-TELOS version provides a value between the 47 % and the 91 % better in the 3elt dataset and always better that the 30 % in the Vibrobox dataset. These results suggest that the new layer helps improving the results, and a layered vertex-centric approach can be used to carry out graph processing computations.

4 Related Work

A comprehensive survey of the most important currently available vertex centric programming models and of the corresponding frameworks is presented in [15]. While most of these provide a set of basic functionalities for vertex-centric computations, several related frameworks and approaches have been developed with the goal of extending and/or optimizing the basic frameworks.

Salihoglu and Widom [18] propose a set of interesting optimisations specifically targeting graph processing in Pregel-like solutions. These include performing the computation sequentially on the master node when number of nodes in the active graph is under a given threshold and merging a sets of vertices to form supervertices in order to optimize the communication cost. Both of these focus on reducing the completion time and the communication volume by instrumenting the support with automated recognition features, activated to speed-up the computation. Another interesting approach that goes beyond the Pregel-like solutions has been proposed by Tian *et al.* [19]. Their idea is to shift the reasoning from the point of view of a single vertex to the point of view of an aggregated set of vertices. By means of this change of perspective they have been able, on selected problems, to give a notable speeds-up in the computation. Both this proposal and our approach are based on the idea of pushing, by design, a shift in the paradigm that allows the implementation of more efficient solutions.

An interesting strategy to speed-up the computation is considering approaches dealing with approximated data and returning approximated solution. Zhang *et al.* [27] proposed both an approximated and an exact solution for k-nearest neighbours join developed according to the MapReduce paradigm. Their approximated solution run orders of magnitude faster than the exact one. The algorithm proposed in [4] reduces the completion time of the computation of a well known graph algorithm, the problem of finding connected components, by dynamically reducing the graph. Another algorithm, still implemented in MapReduce, is presented by Riondato *et al.* [17]. The paper proposes a randomized parallel distributed algorithm to extract approximations of the collections of frequent item-sets and association rules from large datasets. The approximate solution still guarantees, with high probability, the quality of the results.

5 Conclusions

In this paper we presented Telos, a high-level programming framework that supports vertex-centric computations based on layered overlays aimed at large graph processing, implemented on the top of Apache Spark. Telos takes inspiration from approaches that have proved to be robust and efficient in massively distributed systems that, somehow, recalls the structure of large graphs. We conducted an experimental analysis to validate the feasibility of the approach and shows its scalability with respect to the computational resources exploited. The experiments demonstrated that dynamic topologies can be effectively exploited during the computation. We believe that the ability of supporting multiple dynamic layers can be useful in many contexts, as for example to cluster users that are closer in a virtual environment [5] or for classical graph analysis problems, such as connected components and centrality measures. As a future work we plan to conduct a comprehensive analysis of the performance of the framework and a comparison with other approaches targeting large graph computation. We also plan to implement Telos on top of different parallel programming frameworks to assess the performance of our proposed approach regardless the performances provided by Apache Spark.

References

1. Aldinucci, M., Danelutto, M., Dazzi, P.: Muskel: an expandable skeleton environment. Scalable Comput. Pract. Exp. **8**(4), 325–341 (2007)
2. Carlini, E., Coppola, M., Dazzi, P., Laforenza, D., Martinelli, S., Ricci, L.: Service and resource discovery supports over p2p overlays. In: International Conference on Ultra Modern Telecommunications and Workshops, ICUMT 2009, pp. 1–8. IEEE (2009)
3. Carlini, E., Dazzi, P., Esposito, A., Lulli, A., Ricci, L.: Balanced graph partitioning with Apache Spark. In: Lopes, L., et al. (eds.) Euro-Par 2014, Part I. LNCS, vol. 8805, pp. 129–140. Springer, Heidelberg (2014)

4. Carlini, E., Dazzi, P., Lucchese, C., Lulli, A., Ricci, L.: Cracker: crumbling large graphs into connected components. In: 20th IEEE ISCC, International Symposium on Computer and Communications. IEEE (2015)
5. Carlini, E., Dazzi, P., Mordacchini, M., Ricci, L.: Toward community-driven interest management for distributed virtual environment. In: an Mey, D., et al. (eds.) Euro-Par 2013. LNCS, vol. 8374, pp. 363–373. Springer, Heidelberg (2014)
6. Ching, A.: Giraph: large-scale graph processing infrastructure on hadoop. In: Proceedings of the Hadoop Summit, Santa Clara (2011)
7. Danelutto, M., Dazzi, P.: A java/jini framework supporting stream parallel computations. In: Proceedings of the International Conference ParCo (2005)
8. Danelutto, M., Pasin, M., Vanneschi, M., Dazzi, P., Laforenza, D., Presti, L.: PAL: exploiting java annotations for parallelism. In: Gorlatch, S., Bubak, M., Priol, T. (eds.) Achievements in European Research on Grid Systems, pp. 83–96. Springer, New York (2008)
9. Dean, J., Ghemawat, S.: Mapreduce: simplified data processing on large clusters. Commun. ACM **51**(1), 107–113 (2008)
10. Jelasity, M., Montresor, A., Babaoglu, O.: T-Man: Gossip-based fast overlay topology construction. Comput. Netw. **53**(13), 2321–2339 (2009)
11. Leskovec, J., Sosič, R.: SNAP: A general purpose network analysis and graph mining library in C++, June 2014. http://snap.stanford.edu/snap
12. Lua, E.K., Crowcroft, J., Pias, M., Sharma, R., Lim, S., et al.: A survey and comparison of peer-to-peer overlay network schemes. IEEE Commun. Surv. Tutor. **7**(1–4), 72–93 (2005)
13. Malewicz, G., Austern, M.H., Bik, A.J., Dehnert, J.C., Horn, I., Leiser, N., Czajkowski, G.: Pregel: a system for large-scale graph processing. In: Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data, pp. 135–146. ACM (2010)
14. McAfee, A., Brynjolfsson, E., Davenport, T.H., Patil, D., Barton, D.: Big data. The management revolution. Harvard Bus. Rev. **90**(10), 61–67 (2012)
15. McCune, R.R., Weninger, T., Madey, G.: Thinking like a vertex: a survey of vertex-centric frameworks for large-scale distributed graph processing (2015). arXiv:1507.04405
16. Rahimian, F., Payberah, A.H., Girdzijauskas, S., Jelasity, M., Haridi, S.: Ja-be-ja: a distributed algorithm for balanced graph partitioning. In: IEEE 7th International Conference on Self-Adaptive and Self-Organizing Systems (SASO 2013), pp. 51–60. IEEE (2013)
17. Riondato, M., DeBrabant, J.A., Fonseca, R., Upfal, E.: PARMA: a parallel randomized algorithm for approximate association rules mining in mapreduce. In: International Conference on Information and Knowledge Management, CIKM 2012, pp. 85–94 (2012)
18. Salihoglu, S., Widom, J.: Optimizing graph algorithms on pregel-like systems. PVLDB **7**(7), 577–588 (2014)
19. Tian, Y., Balmin, A., Corsten, S.A., Tatikonda, S., McPherson, J.: From "think like a vertex" to "think like a graph". PVLDB **7**(3), 193–204 (2013)
20. Valiant, L.G.: A bridging model for parallel computation. Commun. ACM **33**(8), 103–111 (1990)
21. Voulgaris, S., Gavidia, D., Van Steen, M.: Cyclon: inexpensive membership management for unstructured p2p overlays. J. Netw. Syst. Manag. **13**(2), 197–217 (2005)

22. Voulgaris, S., van Steen, M.: VICINITY: a pinch of randomness brings out the structure. In: Eyers, D., Schwan, K. (eds.) Middleware 2013. LNCS, vol. 8275, pp. 21–40. Springer, Heidelberg (2013)
23. Walshaw, C.: The graph partitioning archive (2002). http://staffweb.cms.gre.ac.uk/~c.walshaw/partition/
24. Xin, R.S., Gonzalez, J.E., Franklin, M.J., Stoica, I.: Graphx: a resilient distributed graph system on spark. In: First International Workshop on Graph Data Management Experiences and Systems, p. 2. ACM (2013)
25. Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, M., Franklin, M.J., Shenker, S., Stoica, I.: Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing. In: Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation, p. 2 (2012)
26. Zaharia, M., Chowdhury, M., Franklin, M.J., Shenker, S., Stoica, I.: Spark: cluster computing with working sets. In: Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing, p. 10 (2010)
27. Zhang, C., Li, F., Jestes, J.: Efficient parallel kNN joins for large data in MapReduce. In: 15th International Conference on Extending Database Technology, EDBT 2012, pp. 38–49 (2012)