A Distributed Vertex-Centric Approach for Pattern Matching in Massive Graphs

Arash Fard, M. Usman Nisar, Lakshmish Ramaswamy, John A. Miller, and Matthew Saltz

Computer Science Department
The University of Georgia
Athens, USA
{ar, nisar, laks, jam, saltzm}@cs.uga.edu

Abstract—Graph pattern matching is fundamentally important to many applications such as analyzing hyper-links in the World Wide Web, mining associations in online social networks, and substructure search in biochemistry. Most existing graph pattern matching algorithms are highly computation intensive, and do not scale to extremely large graphs that characterize many emerging applications.

In recent years, graph processing frameworks such as Pregel have sought to harness shared nothing clusters for processing massive graphs through a vertex-centric, Bulk Synchronous Parallel (BSP) programming model. However, developing scalable and efficient BSP-based algorithms for pattern matching is very challenging because this problem does not naturally align with a vertex-centric programming paradigm.

This paper presents novel distributed algorithms based on the vertex-centric programming paradigm for a set of pattern matching models, namely, graph simulation, dual simulation and strong simulation. Our algorithms are fine-tuned to consider the challenges of pattern matching on massive data graphs. Furthermore, we introduce a new pattern matching model, called strict simulation, which outperforms strong simulation in terms of scalability while preserving its important properties. We investigate potential performance bottlenecks and propose several techniques to mitigate them. This paper also presents an extensive set of experiments involving massive graphs (millions of vertices and billions of edges) to study the effects of various parameters on the scalability and performance of the proposed algorithms. The results demonstrate that our techniques are highly effective in alleviating performance bottlenecks and yield significant scalability benefits.

Keywords—data-intensive computing; query graphs; distributed algorithms; graph simulation; subgraph isomorphism

I. INTRODUCTION

Graphs are fundamental to representing and analyzing relationships in many diverse domains. Although graph storage, querying and mining have been extensively studied over the past several decades, massive scales of modern application domains such as social networks, genomics and the World Wide Web have reigned interest in highly scalable graph processing platforms and algorithms [1][2][3].

Graph pattern matching forms a uniquely important class of graph queries. Conceptually, pattern matching algorithms seek to find subgraphs of a data graph that are similar to a given query graph. While subgraph isomorphism returns the strictest matches for graph pattern matching in terms of topology [4], the problem is NP-complete [5] and thus is impractical for even large-scale graphs. Graph simulation provides a practical alternative to subgraph isomorphism by relaxing its stringent matching conditions. This allows matches to be found in polynomial time. Furthermore, some researchers [1][6][7][8] argue that graph simulation is more appropriate than subgraph isomorphism for modern applications such as social network analysis because it yields matches that are conceptually more intuitive.

In recent years, several variants of the graph pattern matching paradigm have been proposed including graph simulation, dual simulation and strong simulation. These models form a spectrum with respect to the stringency of the matching conditions – graph simulation is the least restrictive whereas strong simulation is the most restrictive. In other words, strong simulation is conceptually closest to subgraph isomorphism whereas graph simulation is the farthest. It is also noteworthy that, in general, the more restrictive the model, the less scalable the corresponding algorithm.

While polynomial time algorithms exist for graph, dual and strong simulations, they still do not scale to the massive graphs that characterize modern applications such as social networks. Therefore, a natural question is whether these algorithms can be parallelized and implemented efficiently on shared nothing clusters. In recent years, several graph processing frameworks have been proposed that enable programmers to easily and efficiently implement distributed graph algorithms on shared nothing clusters. These include Pregel [2], Giraph [9] and GPS [10] which are characterized by the BSP programming paradigm. In these platforms, the programmer expresses algorithms in terms of the actions performed by a generic vertex of the graph. In recent years, researchers have designed vertex-centric algorithms for various graph problems. Surprisingly, to the best of our knowledge, there are no studies on vertex-centric approaches for graph simulation models. While there is some recent work on distributed basic graph simulation [11], the proposed algorithm is not based on the vertex-centric programming paradigm. Moreover, there is no distributed algorithm for dual or strong simulation.

Considering the unique benefits offered by vertex-centric programming platforms, this paper is dedicated to exploring
the following questions. (1) How well do graph simulation in general, and various simulation models in particular fit into the vertex-centric distributed processing paradigm? (2) If certain simulation models are not suitable for the vertex-centric programming paradigm, what are the major bottlenecks? (3) Can we develop new simulation models that are conceptually similar to the existing ones, but are better suited to the vertex-centric paradigm? (4) How do various graph properties such as the number of vertices and the density of a graph impact the performance and scalability of the distributed graph simulation algorithms?

Towards answering these questions, we have designed vertex-centric distributed algorithms for various graph simulation models. We have comprehensively studied their behaviors through an extensive set of experiments. The main technical contributions of this paper are summarized below.

• We identify the bottlenecks that severely limit the scalability and performance of strong simulation on distributed graph processing platforms. Towards ameliorating these limitations, we propose a new graph simulation model called strict simulation. Strict simulation is vastly more amenable to vertex-centric distribution, and surprisingly, the quality of its result is the same or in some cases better than strong simulation.

• We introduce vertex-centric distributed algorithms for all four simulation models. Our algorithms include several techniques to mitigate performance bottlenecks.

• We present a detailed experimental study on the benefits and costs of the proposed distributed algorithms for various graph simulation models. The experiments validate that our strict graph simulation model is highly scalable and very well suited for vertex-centric distribution.

The rest of the paper is organized as follows. In section II, we briefly explain some requisite terminology and the main concepts involved in pattern matching models and vertex-centric graph processing. Section III introduces strict simulation, a new stringent and scalable pattern matching model. Section IV is dedicated to explanation of our distributed algorithms. We cover an experimental study about the behavior of our algorithms in section V. We also present some related work in VI, followed by conclusions in section VII.

II. BACKGROUND

In this section, we provide a brief overview of concepts that are fundamental to our work, namely, vertex-centric graph processing and graph pattern matching models.

A. Vertex-centric Graph Processing

In recent years, several platforms have been designed with the aim of effectively harnessing shared nothing clusters for processing massive graphs. The most popular platforms such as Pregel [2], GPS [10] and Giraph [9] are based upon BSP-based vertex-centric programming model.

Bulk Synchronous Parallel (BSP) was first proposed by Valiant [12] as a model of computation for parallel processing. Computation in this model is a series of supersteps. Each superstep contains three ordered stages: (1) concurrent computation, (2) communication, and (3) barrier synchronization. In the vertex-centric programming model proposed in Pregel, each vertex of the data graph is a computing unit which can be conceptually mapped to a process in the BSP model. At the time of loading, the data graph is partitioned among several workers. Each worker is a real process which handles the computation for a partition of vertices. Each vertex initially knows only about its own label and its outgoing edges. Then, vertices can exchange messages through successive supersteps to learn about each other or to accomplish a computing task. Within each superstep, vertices are executing the same predefined function. When a vertex believes that it has accomplished its tasks, it votes to halt and goes to inactive mode. A vertex remains inactive until it is triggered externally by a message from another vertex. When all vertices become inactive the algorithm terminates.

The vertex-centric approach is a pure message passing model in which users focus on a local action for each vertex of the data graph. Its usage of the BSP model makes it inherently free of deadlocks. Moreover, it can provide very high scalability, and by design it is well-suited for distributed implementation.

B. Pattern Matching Models

The goal of a pattern matching algorithm is to find all the matches of a given graph, called a query graph, in an existing larger graph, called a data graph. To define it more formally, assume that there is a data graph \( G(V, E, l) \), where \( V \) is the set of vertices, \( E \) is the set of edges, and \( l \) is a function that maps the vertices to their labels. Given a query graph \( Q(V_q, E_q, l_q) \) the task is to find all subgraphs of \( G \) that match the query \( Q \). Here, we assume all vertices are labeled, all edges are directed, and there are no multiple edges. Without loss of generality, we also assume a query graph is a connected graph because the result of pattern matching for a disconnected query graph is equal to the union of the results for its connected components. In this paper, we use pattern and query graph interchangeably.

1) Subgraph isomorphism: Subgraph isomorphism is the most famous model for pattern matching. It preserves all topological features of the query graph in the result subgraph. However, finding all subgraphs that are isomorphic to a query graph is an NP-hard problem in the general case [5]. By definition, subgraph isomorphism describes a bijective mapping between a query graph \( Q(V_q, E_q, l_q) \) and a subgraph of a data graph \( G(V, E, l) \), denoted by \( Q \subseteq_{iso} G \).

2) Graph Simulation: Another model, graph simulation, permits faster algorithms by relaxing some restrictions on matches.
Definition: Pattern \( Q(V_q, E_q, l_q) \) matches data graph \( G(V, E, l) \) via graph simulation, denoted by \( Q \preceq_{sim} G \), if there is a binary relation \( R \subseteq V_q \times V \) such that (1) \( (u, u') \in R \), then \( l_q(u) = l(u') \); (2) \( \forall u \in V_q, \exists u' \in V : (u, u') \in R \); (3) \( \forall (u, v) \in R, (u, v) \in E_q \Rightarrow \exists v' \in V : (v, v') \in R \cap (u', v') \in E \).

Intuitively, graph simulation only preserves the child relationships of each vertex. The result of pattern matching is a maximum match set of vertices. The maximum match set, \( R_m \subseteq V_q \times V \), is the biggest relation set between \( Q \) and \( G \) with respect to \( Q \preceq_{sim} G \). The result match graph, \( G_r(V_r, E_r) \), is a subgraph of \( G \) that can represent \( R_m \) [7]. By definition, \( G_r \) is a subgraph of \( G \) which satisfies these conditions: (1) \( (u, u') \in R_m \Leftrightarrow u' \in V_r \); (2) \( \forall (u, u'), (v, v') \in R_m [(u, v) \in E_q \Leftrightarrow (u', v') \in E_r] \).

A quadratic time algorithm for graph simulation was first proposed in [13] with applications to the refinement and verification of reactive systems. This model and its extensions have been studied especially in recent years [6], [11] because of their new applications in analysis of social networks [1].

3) Dual Simulation: Dual simulation improves graph simulation by taking into account not only the children of a query node, but also its parents.

Definition: Pattern \( Q(V_q, E_q, l_q) \) matches data graph \( G(V, E, l) \) via dual simulation, denoted by \( Q \preceq_{sim} G \), if (1) \( Q \preceq_{sim} G \) with a binary match relation \( R_D \subseteq V_q \times V \), and (2) \( \forall (u, u') \in R_D [(w, u) \in E_q \Rightarrow \exists w' \in V : (w, w') \in R_D \cap (u', w') \in E] \). (adapted from [7])

In dual simulation as in graph simulation, we maintain the concept of a maximum match set and a result match graph. Here, the result match graph is also a single graph which might be connected or disconnected. In [7] a cubic algorithm for dual simulation is proposed.

4) Strong Simulation: Strong simulation adds a locality property to dual simulation. Shuai Ma et al. [7] introduce the concept of a ball to define locality. A ball \( b \) in \( G(V, E, l) \), denoted by \( G[v, r] \), is a subgraph of \( G \) that contains all vertices not further than a specified radius \( r \) from a center \( v \in V \); moreover, the ball contains all edges in \( G \) that connect these vertices (i.e., it is an induced connected subgraph). Given two vertices \( u \) and \( v \) in a connected graph, the distance from \( u \) to \( v \) is defined as the minimum number of edges in an undirected path from \( u \) to \( v \). The diameter of a connected graph is then defined as the greatest distance between any pair of nodes in the graph.

Definition: Pattern \( Q(V_q, E_q, l_q) \) matches data graph \( G(V, E, l) \) via strong simulation, denoted by \( Q \preceq_s G \), if there exists a vertex \( v \in V \) such that (1) \( Q \preceq_s G \), with maximum dual match set \( R_D \) in ball \( b \) where \( d_Q \) is the diameter of \( Q \), and (2) \( v \) is member of at least one of the pairs in \( R_D \). The connected part of the result match graph of each ball with respect to its \( R_D \) which contains \( v \) is called a maximum perfect subgraph of \( G \) with respect to \( Q \). (adapted from [7])

In contrast to the previous types of simulation, strong simulation may have multiple maximum perfect subgraphs (MaxPGs) as its result. As indicated by [7], although strong simulation preserves many of the topological characteristics of a pattern graph, its result can be computed in cubic time. Moreover, the number of MaxPGs is bounded by the number of vertices in a data graph, while subgraph isomorphism may have an exponential number of match subgraphs.

5) Comparing the result of the models: Figure 1 provides an example to show the difference in the results of the mentioned pattern matching models. This example is about finding a team of specialists on a social network like LinkedIn, inspired by an example in [11]. A vertex in this example represents a member, and a label represents a member’s profession. A directed edge from a member \( a \) to a member \( b \) indicates that member \( a \) has endorsed member \( b \). Because the query nodes here have distinct labels, we only specify data graph node ID’s for a match rather than specifying the full match relation. Given the pattern and the data graph, the maximum match set for graph simulation consists of all vertices except \( \{2, 13\} \). It is clear that vertices 1, 3, and 14 are not appropriate matches either. Dual simulation removes these inappropriate vertices; its maximum match set contains all vertices except \( \{1, 2, 3, 13, 14\} \). However, there are still some vertices that do not provide very meaningful matches. For example, the cycle \( \{15, 16, 17, 18, 19, 20\} \) creates a very big subgraph match, which is not desirable for such a small pattern. Applying strong simulation then shrinks the result match graph to a reasonable size; the result here is the set of vertices \( \{4, 5, 6, 7, 8, 9, 10, 11, 12\} \). In contrast, there are two isomorphic subgraphs corresponding to these two sets of vertices: \( \{4, 6, 7, 8\} \) and \( \{5, 6, 7, 8\} \).

III. STRICT SIMULATION - STRINGENT AND SCALABLE

Though the polynomial asymptotic time complexity of strong simulation and the closeness of its results to subgraph isomorphism are quite appealing, it is still computationally expensive for very large data graphs. Researchers have proposed some optimizations to increase the efficiency of the algorithm [7], but the locality restriction as it is defined in strong simulation creates a bottleneck because the size of
balls can be potentially very big. Here, the size of a ball means the number of vertices that it contains. Proportional to the diameter of $Q$ and average degree of vertices in $G$, each ball in strong simulation can be fairly bulky. Furthermore, due to communication overhead during ball creation, the problem is exacerbated in distributed systems where a data graph is partitioned among different nodes. In order to mitigate this overhead, we introduce a new pattern matching model, named strict simulation, which is more scalable and preserves the important properties of strong simulation. It is shown in [7], [14] that: (1) if $Q \preceq_{\text{iso}} G$, then $Q \preceq_{\text{sim}}^D G$; (2) if $Q \preceq_{\text{sim}}^S G$, then $Q \preceq_{\text{sim}}^D G$; and (3) if $Q \preceq_{\text{sim}}^D G$, then $Q \preceq_{\text{sim}}^S G$. It is traditionally thought that when the result of a pattern matching model becomes closer to subgraph isomorphism, the corresponding algorithm will become computationally less efficient. However, we show that strict simulation is not only more efficient than strong simulation, but also more stringent; i.e., its result is the same or closer to subgraph isomorphism.

A summary of the centralized algorithm proposed in [7] for strong simulation is shown in figure 2a. In this algorithm, the match relation of dual simulation, $R_D$, is computed first. Then, a ball $G_{D}[v, d_Q]$ is created for each vertex $v$ of the data graph contained in $R_D$. Members of the ball are selected regardless of their membership in pairs of $R_D$. At the next step, the dual match relation is projected on each ball to compute the result of strong simulation. Finally, the maximum perfect subgraph (MaxPG) is extracted from each ball in strong simulation on a ball with the same center. Consequently, the result of strict simulation is closer to subgraph isomorphism in comparison to strong simulation. Figure 3a shows an example that highlights the difference between the results of these two types of simulation. In this example, for strong simulation, a ball centered at vertex 2 would contain all the vertices in the data graph. In contrast, because the dual match graph would not contain vertex 11, the corresponding ball for strict simulation would only contain the set of vertices $\{2,3,4,10\}$. Therefore, the MaxPG resulting from strong simulation will contain all vertices except 11, while the one resulting from strict simulation will contain only vertices 1, 2, and 3.

**Figure 2. Flowcharts for strong and strict simulation**

In comparison, the centralized algorithm for strict simulation is depicted in figure 2b. The key difference between strict and strong simulation is that balls are created from the dual result match graph, $G_D$, rather than from the original graph. Though the algorithm for strict simulation includes an extra logical step for creating the match graph with respect to dual simulation, in practice, this operation can be folded into the creation of the balls. Furthermore, because the balls in strict simulation are often significantly smaller, this seemingly minor difference between the two algorithms has a profoundly positive impact on running time, and it improves the quality of the matches.

The resulting MaxPG extracted from each ball in strict simulation is always a subgraph of the result of strong simulation on a ball with the same center. Furthermore, its match set contains any possible subgraph isomorphic match on that ball. Consequently, the result of strict simulation is closer to subgraph isomorphism in comparison to strong simulation. Figure 3a shows an example that highlights the difference between the results of these two types of simulation. In this example, for strong simulation, a ball centered at vertex 2 would contain all the vertices in the data graph. In contrast, because the dual match graph would not contain vertex 11, the corresponding ball for strict simulation would only contain the set of vertices $\{2,3,4,10\}$. Therefore, the MaxPG resulting from strong simulation will contain all vertices except 11, while the one resulting from strict simulation will contain only vertices 1, 2, and 3.

**Figure 3. Comparing strong and strict simulation**

We formally define strict simulation as follows.

**Definition:** Pattern $Q(V, E, l)$ matches data graph $G(V, E, l)$ via strict simulation, denoted by $Q \preceq_{\text{sim}}^D G$, if there exists a vertex $v \in V$ such that (1) $v \in V_D$ where $G_D(V_D, E_D, l_D)$ is the result match graph with respect to $Q \preceq_{\text{sim}}^D G$; (2) $Q \preceq_{\text{sim}}^D G_D[v, d_Q]$ where $G_D[v, d_Q]$ is a ball extracted from $G_D$; and (3) $v$ is a member of the result MaxPG.

The definition of a MaxPG for strict simulation is similar to that of a MaxPG for strong simulation. Using the definition and the properties of dual simulation presented in [7], [14], properties of strict simulation can be proved as follows.

**Theorem 1:** For any query graph $Q$ and data graph $G$ such that $Q \preceq_{\text{sim}}^D G$, there exists a unique set of maximum perfect subgraphs for $Q$ and $G$.

**Proposition 1:** (1) If $Q \preceq_{\text{iso}} G$, then $Q \preceq_{\text{sim}}^D G$; (2) if $Q \preceq_{\text{sim}}^D G$, then $Q \preceq_{\text{sim}}^S G$.
Proposition 2: The number of maximum perfect subgraphs produced by $Q \lessdot_{\text{sim}}^G$, $G$ is bounded by the number of vertices in $G$.

Theorem 1 ensures that strict simulation yields a unique solution for any query/data graph pair. Proposition 1 indicates that strict simulation is a more stringent notion than strong simulation, though still looser than subgraph isomorphism. Proposition 2 gives an upper bound on the number of possible matches for a query graph in a data graph. The following theorem also shows that the asymptotic time complexity of strict simulation is the same as that of strong and dual simulation.

Theorem 2: For any query graph $Q$ and data graph $G$, the time complexity for finding all maximum perfect subgraphs with respect to strict simulation is cubic.

For any query graph $Q$ and data graph $G$, the number of possible matches for a query graph in a data graph is bounded by the number of vertices in the query graph. The theorem ensures that strict simulation yields a unique solution for any query/data graph pair. Proposition 1 indicates that strict simulation is a more stringent notion than strong simulation, though still looser than subgraph isomorphism. Proposition 2 gives an upper bound on the number of possible matches for a query graph in a data graph. The following theorem also shows that the asymptotic time complexity of strict simulation is the same as that of strong and dual simulation.

Theorem 2: For any query graph $Q$ and data graph $G$, the time complexity for finding all maximum perfect subgraphs with respect to strict simulation is cubic.

With the vertex-centric programming model, in contrast to usual graph programming models, an algorithm should be designed from the perspective of each vertex of a graph. This change in point of view makes algorithm design more difficult for problems like pattern matching that need a global overview of a graph.

In this section, we first present a distributed algorithm for graph simulation, and then we move on to discuss distributed algorithms for dual, strong, and strict simulation as the complexity of the problem increases.

A. Distributed Graph Simulation

A summary of our algorithm for a vertex in distributed graph simulation is shown in figure 4. At load time, we distribute the query graph, $Q$, among all workers. The effect of distributing $Q$ to each worker is negligible because its size is small and the total number of workers is limited to the number of processing elements in the system. An if-else ladder is used to distinguish between different tasks in different supersteps, and the BSP framework ensures that all vertices are at the same superstep.

We define a boolean $\text{match}$ flag in each vertex to track whether, at the end of execution, this vertex matches a node in the query graph. Initially, it is assumed that a vertex is not a match. At the first superstep, the $\text{match}$ flag of each vertex becomes true if its label matches the label of a vertex in $Q$. In this case, a local match set, called $\text{matchSet}$, will be created to keep track of its potential matches in the query graph. Each vertex, then, learns about the $\text{matchSets}$ of its children in the first three supersteps and keeps them in a local list in order to evaluate graph simulation conditions. Each vertex removes any match from its local $\text{matchSet}$ that does not satisfy the simulation conditions and then informs its parents about any changes in its $\text{matchSet}$. Consequently, each vertex that receives changes in its children’s $\text{matchSets}$ reflects those changes in its list of match children and reevaluates its own $\text{matchSet}$. The algorithm terminates after the third superstep if no vertex removes any match from its $\text{matchSet}$. This procedure will continue in superstep four and beyond until no vertex makes any change in its $\text{matchSet}$. In order to make sure that the algorithm will terminate, any active vertex without an incoming message votes to halt after the third superstep. At the end of execution, the local $\text{matchSet}$ of a vertex contains the correct and complete set of matches between that vertex and the vertices of the query graph.

In an example displayed in figure 5, all the vertices of the data graph labeled $A$ and $B$ make their $\text{match}$ flag true at the first superstep, and then vertices 1, 2, 3, and 4 send messages to their children. At the second superstep vertices 1, 2, 3, and 4 will reply back to their parents. At the third superstep, vertices 1, 2, and 3 can successfully validate their match set, but vertex 4 makes its flag false because it receives no message from any child. Therefore, vertex 4 sends a removal message to vertex 3. This message will be received by vertex 3 at superstep four and will cause its flag to become false. Similarly, vertex 3 sends a removal message to vertex 2 which will be received at superstep five. Vertex 2 will successfully reevaluate its match set, and the algorithm will finish at superstep six when every vertex has voted to halt (there is no further communication).

The correctness of the proposed distributed algorithm can be derived from the following lemmas.
that arises from creating the balls. Recalling their definitions, simulation. The main challenge, however, is the overhead simulation will create a ball around itself to compute strong Each vertex that has successfully passed the filter of dual on top of the algorithm for distributed dual simulation.

C. Distributed Strong and Strict Simulation

The algorithm for distributed strong simulation is built on top of the algorithm for distributed dual simulation. Each vertex that has successfully passed the filter of dual simulation will create a ball around itself to compute strong simulation. The main challenge, however, is the overhead that arises from creating the balls. Recalling their definitions, each ball in strong simulation is an induced connected subgraph of the data graph; whereas, each ball in strict simulation is an induced connected subgraph of the dual match graph.

The distributed algorithms that we have designed and implemented for strong and strict simulation follow the flowcharts of figure 2, but in a distributed fashion. The first step for applying dual simulation is the same as the distributed algorithm for dual simulation. As mentioned, at the end of dual simulation each vertex of the data graph has a matchSet that contains the IDs of vertices of the query graph that match to that vertex. Each vertex with a non-empty matchSet makes a ball centered at itself in a breadth-first search (BFS) fashion that takes \(2(d_Q - 1)\) supersteps. For strong simulation, all the vertices in the neighborhood are considered as members of a ball. However for strict simulation, only vertices with the member flag set to true will answer the request for neighborhood information. Moreover, while the center vertex of the ball receives neighborhood information, it adds a vertex to the ball only if there is a corresponding edge in the pattern graph. Eventually, the center vertex of each ball will perform the rest of the computation on that ball for either strong or strict simulation in a sequential fashion. Because vertices are distributed among workers, this phase can be considered an embarrassingly parallel workload.

V. EXPERIMENTAL STUDY

The goal of this experimental study is to evaluate the four proposed distributed algorithms and to learn about their bottlenecks and design trade-offs with respect to properties of the inputs. We have used GPS [10], which can be considered a free distribution of Pregel, to implement our distributed algorithms. GPS is written in Java and has an extended API to provide a type of global communication among vertices.

Parameters that we consider for data graphs are the number of vertices, denoted by \(|V|\), the density of the graph, denoted by parameter \(\alpha\), where \(|V|^{\alpha}\) equals the number of edges, and the number of distinct labels, denoted by \(l\). In all experiments, \(l = 200\), unless it is mentioned explicitly. The only parameter considered for queries is the number of vertices, denoted by \(|V_q|\). Moreover, the number of workers used in each experiment is another parameter, denoted by \(k\).

A. Experimental setting

Both real world and synthesized datasets were used in our experiments. In terms of real world datasets, we used uk-2002 with 18,520,486 vertices and 298,113,762 edges; and journal-2008 which has 5,363,260 vertices and 79,023,142 edges\(^1\).

We used graph-tool [15] to synthesize small and medium size randomly generated data graphs, but because of its

\(^1\)http://law.di.unimi.it/datasets.php
physical memory limits we also implemented our own synthesizer to generate large semi-randomly generated graphs. The input parameters of our graph synthesizer are the number of vertices \( n \), the average number of outgoing edges \( d \), and the number of distinct labels \( l \). It selects an integer between 0 and \( 2d \) as the average number of outgoing edges for every vertex. Then, for each outgoing edge, the endpoint of the edge is randomly selected. The label of each vertex is also a randomly selected number between 0 and \( l \).

To generate a pattern graph, we randomly extract a connected subgraph from a given dataset. Our query generator has two input parameters: the number of vertices and the desired average number of outgoing edges. Unless mentioned otherwise, the average number of outgoing edges is set to such a value that \( \alpha = 1.2 \).

The experiments were conducted using GPS on a cluster of 12 machines. Each one has 128GB DDR3 RAM, two 2GHz Intel Xeon E5-2620 CPUs, each with 6 cores. The intra-connection network is 1Gb Ethernet. One of the machines plays the role of master.

B. Experimental results

The most important results achieved from the experiments are categorized in the following groups. It is important to mention that distributed strong simulation is so slow on large datasets that we could not run most of the experiments with that particular algorithm; therefore, in a separate set of experiments we compare strong and strict simulation.

**Experiment 1: Running time**

We examine the running time of the proposed algorithms with different numbers of workers to study their performance and scalability (figure 6). In the experiment displayed in figure 6a, we could not run the test on a single machine because of memory limits.

Like many other systems that are designed for processing large datasets, the BSP model scales very well on bigger datasets. Moreover, all types of simulation exhibit a filtering behavior, meaning that they start by processing a large set and then refine it; this causes light workload at the final supersteps.

**Experiment 2: Impact of density**

In the first experiment of this group (figure 7a), we compare the running time of different pattern matching models to the number of vertices in the data graph. As expected, the running time increases with growth in the size of data graph. The ratio (approximately twice) between the running times of graph simulation and dual simulation shows the difference between their computational complexity. The increasing difference between the running times of strict and dual simulation can be explained by the fact that the number of balls increases proportionally to the number of vertices in the result of dual simulation. Figure 7b shows the impact of the density of a data graph on running time. An increase in running time is to be expected. The unchanged cost of ball creation (difference between dual and strict simulation) reveals a good feature of strict simulation; because the density of the data graph does not have a big impact on the density of the dual match graph, the balls do not necessarily increase in size as the density of the data graph increases.

**Experiment 3: Impact of pattern**

Figure 8 shows that running time increases as the size of the query becomes bigger, which is not surprising. The behavior remains similar across datasets with different numbers of vertices. It can be observed that with an increase in the size of a query, the difference between graph simulation and dual simulation increases to a point, but then becomes unchanged. This is because, for big queries, it is very likely that the results of graph simulation and dual simulation will be the same. Although the growth in runtime of strict simulation slows at this point, its growth with respect to the growth of dual simulation increases due to an increase in ball size.

**Experiment 4: Comparison of strong and strict simulation**

The difference in behavior between the algorithms for strong and strict simulation can be seen in 9. Because of the high cost of ball creation in distributed strong simulation, it was only possible to test it on fairly small datasets.

Figures 9a and 9b compare the running time of the two algorithms on two different datasets and a range of query sizes. The differences are huge on both datasets, though they are small for \( |V_q| = 10 \).

We also performed a set of experiments to compare the quality of the results of strict and strong simulation based on their closeness to the results of subgraph isomorphism. Though strict simulation in comparison to strong simulation has theoretically the same or higher quality in its result, in practice our experiments have yet shown no significant difference between them. The closeness value to subgraph isomorphism that we found for both strict and strong simulation is very similar to the value reported in [7].

Figure 9c shows the total size of the communication in the system per superstep for strong and strict simulation. The chart shows three phases for the procedure. The first phase is the dual simulation phase that occurs before superstep 6. Then, ball creation occurs between supersteps 6 and 19. Supersteps 20 and 21 correspond to processing balls and terminating the algorithms; therefore, no communication was observed in these supersteps.

There is a difference in communication at the first superstep because in strong simulation every vertex needs to learn about its neighborhood regardless of its matching status. The exponential increase of communication size in strong simulation during creating balls is because of the involvement of all vertices in that process. The jagged shape of communication is because of our BFS-style algorithm for discovering balls, which contains requests at one superstep and responses at the next.

Figure 9d shows the difference between the number
of active vertices in strong simulation and strict simulation which indirectly indicates execution overhead of workers. After the dual simulation phase, though the number of active vertices in strict simulation does not experience a significant change, the number of active vertices in strong simulation increases exponentially because inactive vertices become active again during ball creation.

VI. RELATED WORK

We surveyed related work in two categories: pattern matching models, and distributed algorithms for pattern matching on massive graphs.

Many matching models find similarity between graphs based on exact structural matching like subgraph isomorphism and its approximate algorithms. On the other hand, some matching models take into account the meaning of attributes of vertices and edges and their relationship in addition to graph structure. We are interested in the recent type which is also called semantic graph matching [4].

A few graph pattern matching models introduced during recent years to find a proper semantic graph match like p-homomorphism [16] and bounded simulation [6]. In [7], which introduces dual and strong simulation, the idea of a distributed algorithm is talked about in abstract, but is not based on vertex-centric and thus varies greatly from our approach. There are also very interesting theoretical investigations of different types of simulation presented in [14], but it never comes close to a distributed algorithm for strong simulation.

In [11] a distributed algorithm only for basic graph simulation is proposed. They have implemented and tested their algorithm on a cluster of 16 machines; however, neither the algorithm nor its implementation is adapted to a widely-used cluster computing platform, but it is implemented all in Python. To best of our knowledge, our algorithms are the first distributed algorithms for dual and strong simulation.

There are also some proposed approaches based on indexing to increase efficiency of answering graph pattern matching queries; for example [17] or [18]. However, creating and storing indexes for massive data graphs is not feasible.

In [3] a distributed algorithm is introduced for finding isomorphic subgraph matches in a huge data graph. They deploy graphs on Trinity [19] which is a commercial distributed memory cloud environment. To avoid the construction time and the storage capacity for complicated indices, they only use a simple string index which maps node labels to node IDs. Conceptually similar to our approach, their main idea is based on graph exploration, but they cannot avoid expensive joins because of the intrinsic limits of subgraph isomorphism.

VII. CONCLUSION

This paper discusses the development of (1) a new technique for graph pattern matching and (2) distributed implementations of four graph pattern matching techniques. The implementations follow a vertex-centric BSP approach which has been successful in other areas of graph analytics, but has not yet been used successfully in the area of graph pattern matching. Our new technique for pattern matching, strict simulation, has been shown to scale much better than strong simulation in a vertex-centric framework. In the future, we will explore the efficiency of similar techniques for graph pattern matching when the edges of the query and data graph are also labeled. We will also adapt algorithms to use incremental graph pattern matching techniques.

ACKNOWLEDGEMENT

Lakshmish Ramaswamy is partially supported by a grant from the NSF CSR program.
REFERENCES


