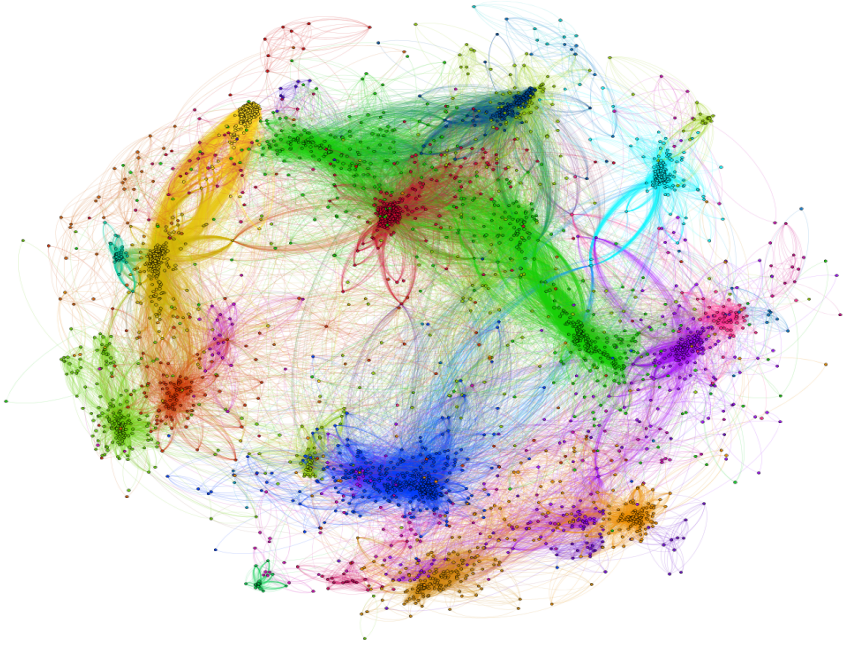
DOCTORAL THESIS IN SOFTWARE AND COMPUTER SYSTEMS
STOCKHOLM, SWEDEN 2014

Gossip-Based Algorithms for Information Dissemination and Graph Clustering

FATEMEH RAHIMIAN



KTH ROYAL INSTITUTE OF TECHNOLOGY

INFORMATION AND COMMUNICATION TECHNOLOGY

Gossip-Based Algorithms for Information Dissemination and Graph Clustering

FATEMEH RAHIMIAN

Doctoral Thesis in
Information and Communication Technology
Stockholm, Sweden 2014

TRITA-ICT/ECS AVH 14:09
ISSN 1653-6363
ISRN KTH/ICT/ECS/AVH-14/09-SE
ISBN 978-91-7595-108-9

KTH School of Information and
Communication Technology
SE-164 40 Kista
SWEDEN

Akademisk avhandling som med tillstånd av Kungl Tekniska högskolan framlägges till offentlig granskning för avläggande av teknologie doktorandexamen i datalogi Torsdag den 22 Maj 2014 klockan 13:00 i sal E i Forum IT-Universitetet, Kungl Tekniskahögskolan, Isafijordsgatan 39, Kista.

Swedish Institute of Computer Science
SICS Dissertation Series 68
ISSN 1101-1335.

© Fatemeh Rahimian, May, 2014

Tryck: Universitetsservice US AB

Abstract

Decentralized algorithms are becoming ever more prevalent in almost all real-world applications that are either data intensive, computation intensive or both. This thesis presents a few decentralized solutions for large-scale (i) data dissemination, (ii) graph partitioning, and (iii) data disambiguation. All these solutions are based on gossip, a light weight peer-to-peer data exchange protocol, and thus, appropriate for execution in a distributed environment.

For efficient data dissemination, we make use of the publish/subscribe communication model and provide two distributed solutions, one for topic-based and one for content-based subscriptions, named *Vitis* and *Vinifera* respectively. These systems propagate large quantities of data to interested users with a relatively low overhead. Without any central coordinator and only with the use of gossip, we build a novel topology that enables efficient routing in an unstructured overlay. We construct a hybrid system by injecting structure into an otherwise unstructured network. The resulting structure resembles a navigable small-world network that spans along clusters of nodes that have similar subscriptions. The properties of such an overlay make it an ideal platform for efficient data dissemination in large-scale systems. Our solutions significantly outperforms their counterparts on various subscription and churn scenarios, from both synthetic models and real-world traces.

We then investigate how gossiping protocols can be used, not for overlay construction, but for operating on fixed overlay topologies, which resemble graphs. In particular we study the NP-Complete problem of graph partitioning and present a distributed partitioning solution for very large graphs. This solution, called *Ja-be-Ja*, is based on local search and does not require access to the entire graph simultaneously. It is, therefore, appropriate for graphs that can not even fit into the memory of a single computer. Once again gossip-based algorithms prove efficient as they enable implementing light-weight peer sampling services, which supply graph nodes with partial knowledge about other nodes in the graph. The performance of our partitioning algorithm is comparable to centralized graph partitioning algorithms, and yet it is scalable and can be executed on several machines in parallel or even in a completely distributed peer-to-peer overlay. It can be used for both edge-cut and vertex-cut partitioning of graphs and can produce partition sizes of any given distribution.

We further extend the use of gossiping protocols to find natural clusters in a graph instead of producing a given number of partitions. This problem, known as graph community detection, has extensive application in various fields and communities. We take the use of our community detection algorithm to the realm of linguistics and address a well-known problem of data disambiguation. In particular, we provide a parallel community detection algorithm for cross-document coreference problem. We operate on graphs that we construct by representing documents' keywords as nodes and the co-location of those keywords in a document as edges. We then exploit the particular nature of such graphs, which is coreferent words are topologically clustered, and thus, can be efficiently discovered by our community detection algorithm.

To Amir Hossein

Acknowledgements

It is a long journey, the life of a PhD student, full of ups and downs, joys and disappointments, failures and successes. But you should never lose hope, specially if you are, like me, surrounded with amazing people who generously give their time, support and advice to you.

Foremost, I would like to express my deepest gratitude to my advisor, Professor Seif Haridi. Your broad knowledge and expertise, your incredible enthusiasm and your continuous fatherly support, provided me with a great opportunity and excellent atmosphere for doing research. Without you this dissertation would never have been materialized. I will be forever indebted to you. *Thank you sir!*

I would like to thank my second advisor Dr. Šarūnas Girdzijauskas. I could not imagine having a better mentor than you. You generously gave me all the time and support I needed, showed me how to do research and gave me the most insightful guidance. Words can not explain how grateful I am to you, for you bore with me through all ups and downs. *Labai ačiū!*

I also would like to thank my co-advisor, Dr. Jim Dowling. I am extremely grateful to you for the illuminating discussions, your constructive feedbacks and invaluable support. *Go raibh maith agat!*

My deepest appreciation goes to Dr. Amir Hossein Payberah. I am thankful to you not only for your enormous contributions to the work presented in this thesis, but also for your generous help and support during the whole course of my studies. *Yek donya Mamnoonam!* I dedicate this work to you with all my love and gratitude.

I would like to thank Professor Márk Jelasity for our collaboration on graph partitioning. It has been an honor working with you and benefiting from your extensive knowledge. *Köszönöm!*

I am very much obliged to Professor Amr El-Abbadi and Dr. Ceren Budak for broadening my knowledge in the field of streaming data processing. Thank you!

My special thanks go to Dr. Sverker Janson at SICS. You are an amazing manager, always full of inspiration and great advices. *Tack så mycket!*

I am grateful to Martin Neumann for many fruitful and enthusiastic discussions we had on the topic of graph processing. *Dankeschön!*

I am thankful for the assistance given to me by Thinh Le Neuyen Huu for the work on locality awareness in publish/subscribe systems.

I thank Dr. Vladimir Vlassov, Professor Christian Schulte, Professor Alberto Montresor, Dr. Ali Ghodsi, Dr. Tallat Shafaat, Dr. Cosmin Arad, Dr. Ahmad Al-Shishtawy, Dr. Ian Marsh, Dr. Nima Dokoohaki, Dr. Roberto Roverso, Dr. Raul Jimenez, Niklas Ekström and Alex Averbuch. It has been a great opportunity working along side you and benefiting from your advices, experiences and warm encouragements. *Thank you all!*

I am also obliged to my colleagues and fellow students at KTH and SICS, for contributing to such a great environment for doing research.

I gratefully acknowledge the financial support I have received from End-to-End Clouds project funded by the Swedish Foundation for Strategic Research (SSF)

under the contract RIT10-0043.

Big thanks to my dear friends, without whom I could not get through the hardships of this rather bumpy way. And finally, my dear family! How can I ever thank you? Your endless love and support fills me with life. You are my source of inspiration and I am forever indebted to you. Loads of love and thanks to you all!

Contents

Part I

Thesis Overview

Chapter 1

Introduction

"640K is enough for anyone, and by the way, what's a network?"
- William Gates III, President of Microsoft Corporation, 1984

FAR from the early predications, in the world we live today, in the heart of almost every real-world application lies an essential need for big data management and decentralized computations over a network of computers.

Since the emergence of Web 2.0 applications, users of Internet do not merely consume data, but also produce data at a considerable rate. People share photos and videos on social networks, write blogs, even advertise and sell personal items on-line. The huge quantities of data that we have to deal with, does not contain independent and unrelated data items. Quite the contrary, data in today's world is highly connected. The friends that we have in a social network connect us to the rest of the people. Our common interests group us together. Even far from the realm of social network, in a rather microscopic level of science, down to genomes and protein networks, we can observe connected data.

Networks are mathematically modelled with graphs. A graph consists of nodes/vertices and links/edges. For example, in a social network nodes represent users and links represent the friendship relations. In some other use cases graphs are to be constructed on the fly. For example, in a content dissemination network the links are established for data exchange. Whether a graph models the actual data or the path through which data can be transferred, it can grow very big. It may not even fit into the memory of a single computer. Graph databases, for instance, will soon require to span multiple machines not only for efficient storage and retrieval, but also for efficient computations over data. Needless to say, traditional centralized systems can no longer keep up to provide such resource demanding services. Also, centralized algorithms that require frequent global operations over the entire graph become prohibitively costly. Hence, decentralized solutions are becoming prevalent in most of real-world applications. More precisely, we need mechanism that replace the global operations with local ones and can work with only partial information.

Thanks to the research in the field of distributed systems and peer-to-peer networks, such a mechanism exists. It is called *gossiping*, a.k.a. *epidemic protocol*. Gossiping is a light weight protocol for data exchange between neighboring nodes in a graph or network. Just like a virus can be transfered from one guy to another when they meet and spread in the whole society, two neighboring nodes of a graph can infect each other with pieces of information. Over time more and more nodes get infected and the information spreads in the graph. If these pieces of information are effectively aggregated at the nodes, they can be utilized to replace the global operations that exist in the centralized algorithms. In this thesis we employ gossiping to (i) build graphs, (ii) partition graphs, and (iii) extract information from the graphs.

First we study how we can employ gossiping, to construct overlays that provide large-scale *publish/subscribe* services. Publish/subscribe communication model is a ubiquitous protocol for data dissemination in today's applications, in which users express their interest in form of subscriptions and get notified when some matching information is published. News syndication, multi-player games, social networks, and media streaming applications are a few examples of systems that are utilizing the publish/subscribe communication model. Depending on the application, the publish/subscribe service could be bandwidth intensive, as in streaming applications, or time critical, as in stock market applications, or may include a large number of subscriptions, as in social networks. We present two gossip-based distributed algorithms, namely *Vitis* and *Vinifera* for topic-based and content-based subscriptions, respectively. We use gossiping to cluster users with similar subscriptions together and at the same time embed a structure into this unstructured overlay, in order to make it navigable. The gossip protocol enables Vitis and Vinifera to operate without a central coordinator and any global knowledge.

Next, we use gossiping for partitioning a graph across multiple servers or clusters. This is a very important problem, because with the ever increasing size of the graphs, it is crucial to partition them into multiple smaller clusters that can be processed efficiently in parallel. Unlike the conventional parallel data processing, parallel graph processing requires each vertex or edge to be processed in the context of its neighborhood. Therefore, it is important to maintain the locality of information while partitioning the graph across multiple (virtual) machines. It is also important to produce equal size partitions that distribute the computational load evenly between clusters. We present a gossip-based distributed algorithm, called *Ja-be-Ja* for two types of graph partitioning, i.e., edge-cut and vertex-cut partitioning.

Finally, we use gossip for extracting information out of raw data. In particular, we address a linguistic problem that has to do with the classification of multiple documents with respect to an ambiguous term, such that each class of documents refer to a unique manifestation of the ambiguous term in reality. Note, this task may not always be a difficult task for humans. When one comes across Mercury in an article about the solar system, they instantly think of Mercury, the planet, and not about Mercury, the chemical element or Freddie Mercury. For a computer

though, such a disambiguation requires a considerable amount of processing. This problem, i.e., the task of disambiguating manifestations of real world entities in various records or mentions, is known as *Entity Resolution* or *Coreference Resolution*. We present a distributed solution to this problem, which is again based on gossip. We construct a graph out of the context words of the ambiguous mentions and then run a community detection algorithm to cluster this graph into several components, each referring to a distinct meaning of the ambiguous word.

Although all our solutions are based on gossip, they are not designed for exactly the same data distribution model. This is mainly due to the inherent differences in the problems in the first place. In the next section we introduce two models and point out which of our algorithms are designed for which model.

1.1 Models and Assumptions

In papers A, B and C, we frequently use the term *node* to refer to a processing unit, for example a computer in a peer-to-peer network. In papers D, E and F, *node* refers to graph nodes (or *vertices*). Since the peer-to-peer networks also resembles a graph, in the former case nodes can also be considered as nodes of a graph.

All the algorithms that are presented in this thesis work can be executed in a distributed environment. Using the graph processing terminology, all our algorithms are *vertex-centric*, meaning that they are executed by the graph nodes independently and in parallel. Two main models are considered:

One-host-multiple-nodes
This model is interesting for data centers or cloud environments, where each computer hosts thousands of nodes at the same time. The host periodically executes the algorithm over all the nodes that it holds. If an information exchange takes place with other nodes on the same host, the communication cost is negligible. However, information exchange across hosts is costly and constitutes the main body of the communication overhead.

One-host-one-node
In this model, each node could be placed either on a different host, or processed independently in a distributed framework. This model is appropriate for frameworks like GraphLab [1] or Pregel [2]. It can also be used in peer-to-peer (P2P) overlays, where each node is an independent computer. In both cases, no shared memory is required. Nodes communicate only using messages passing, and each message adds to the communication overhead.

Our work on publish/subscribe systems are designed for a peer-to-peer environment. Our partitioning algorithm can be executed in a one-host-one-node model, as well as the one-host-multiple-nodes model. Finally, our work on coreference resolution is designed for execution on one-host-multiple-nodes model. Note, the nature

of this problem does not match the one-host-one-node model. However, due to its complexity it requires parallel execution in face of large quantities of data. That is why our algorithm is designed for parallel execution on multiple machines, each hosting a subset of input data.

1.2 Outline

In the next chapter we explore the necessary background for the thesis. In Chapter 3 we elaborate our main contributions and also identify the delimitation of this thesis work. We conclude the work in Chapter 4. The complete publications, on which this thesis is based, are presented in Part II. Chapters 5, 6, and 7 describe our work on large-scale data dissemination. Our graph partitioning algorithms are presented in Chapters 8 and 9. Finally, Chapter 10 presents our solution for the coreference problem.

Chapter 2

Background

"Our knowledge can only be finite, while our
ignorance must necessarily be infinite."
- Karl Popper

Since all our algorithms make use of gossip-based protocols in one way or another, in this chapter we introduce gossiping protocols in the context of peer-to-peer networks, where they were first introduced. We explain how one can make use of gossiping protocols to provide a peer sampling service, a service that is utilized in most of our solutions. We also describe how a peer sampling service can be used to construct and maintain a desired topology. The topology construction is primarily important for our publish/subscribe solutions.

2.1 Peer-to-Peer Overlays

A peer-to-peer (P2P) overlay is an overlay network that exploits the existing resources at the edge of the network. Each node is represented by a peer, and plays the role of both client and server in the overlay network. Nodes cooperate to provide a distributed service, without the need for a single or centralized coordinator/server. The resources in such networks increase as more nodes join the network. Thus, peer-to-peer networks can potentially scale to a large number of participating nodes without having to dedicate powerful machines to provide the service. BitTorrent is a well-known example of such networks. In a peer-to-peer network nodes can join or leave the network continuously and concurrently. This phenomenon is called *churn*. Also network capacities change due to congestion, link failures, etc. Any such system, therefore, must handle churn in order to provide a reasonable quality of service.

Peer-to-peer overlays are mainly categorized into (i) structured, (ii) unstructured, and (iii) hybrid overlays. In a structured overlay, nodes acquire an identifier

from a globally known identifier space and are arranged to form a well defined topology. Such overlays should provide navigability, that is every node should be able to route to any other node in few, usually logarithmic, number of steps. This is achieved by utilizing a greedy distance-minimizing lookup service over the topology. Chord [3], Pastry [4], Kademlia [5], Symphony [6], CAN [7], One-hop DHT [8] and Oscar [9–11] are examples of the structured overlays.

One the other hand, unstructured overlays usually do not have a predefined topology and nodes randomly discover and select each other to link with. Lookup in these overlays usually takes the form of either flooding or random walk. Gnutella [12] and Kaza [13] are two examples of unstructured overlays.

While structured overlays are more efficient in routing, they need to be constantly maintained in the presence of churn in the network. On the other hand, unstructured overlays are very robust and automatically adapt to the changes in the network, though they can not guarantee a bounded routing time. Hybrid overlays exploit the best of the two worlds and are optimized for specific purposes. In such an overlay, some links are chosen with predefined criteria that lead to better routing performance, while some other links are selected randomly or based on other characteristics that are important for the application. For example, in our publish/subscribe systems, we construct hybrid overlays that are specially designed for connecting users with similar subscriptions together.

2.2 Gossiping Protocol

The basic gossiping protocol is based on a symmetric information exchange between pairs of nodes in a network. Each node has a local state, which is determined by the logic of the application. For instance, it could indicate the node's load or cluster identifier. Whatever the state is, a node periodically shares it with its neighbors, i.e., the nodes that are directly connected to it. The basic protocol is illustrated in Algorithms 1 and 2. Every node p has two different threads, one active and one passive. The active thread periodically initiates an information exchange with a random neighbor q, i.e., node p sends a message containing its local state S_p to node q. The neighbor selection policy that implements the function $GetNeighbor()$ depends on the nature of the application. Node p then waits for a response from q, and when the response is received, p will update its current state. The passive thread at each node, for example node p, waits for messages sent by other nodes, and as soon as it receives a message, it will send back a reply containing its own state S_p, and also updates its state with the newly received value. Note, the information exchange is symmetric, since both participants send their states and receive each other's state. When nodes have both states, they run the $Update()$ method, which is again application dependent.

In the next section, we will see a few options for implementing this generic function for building peer sampling services (PSS). We use PSSs in our publish/-subscribe systems (Papers A to C) as well as our graph partitioning algorithms

Algorithm 1 Generic Gossiping Protocol - Active thread at node p

```
1: procedure GOSSIP
2:     q ← GetNeighbor()
3:     Send S_p to q
4:     Recv S_q from q
5:     S_p ← Update(S_p, S_q)
6: end procedure
```

Algorithm 2 Generic Gossiping Protocol - Passive thread at node p

```
1: procedure RESPONDTOGOSSIP
2:     Recv S_q from q
3:     Send S_p to q
4:     S_p ← Update(S_p, S_q)
5: end procedure
```

(Papers D and E). For the latter algorithms, in addition to a PSS, we have another gossiping protocol, in which nodes' states contains the partition they belong to as well as the partitions they neighbors belong to. These information is exchanged between pair of node and accordingly nodes update their state based in the received information. Moreover, we use gossiping in our diffusion-based community detection (Paper F), where the state of nodes is their color inventory and the cluster they belong to. In each round, nodes communicate not with a single neighbor, but with all their neighbors. The update method is also implemented such that it takes into accounts the incoming values from all neighbors. For more specific information about nodes states and the update function for each algorithm, please refer to the corresponding paper(s).

2.3 Peer Sampling Services

Peer sampling services (PSS) have been widely used in large scale distributed applications, such as information dissemination [14], aggregation [15], and overlay topology management [16–19]. The main purpose of a PSS is to provide the participating nodes with uniformly random sample of the nodes in the system. Gossiping algorithms are the most common approach to implementing a PSS [20–26]. In a gossip-based PSS, protocol execution at each node is divided into periodic cycles. In each cycle, every node selects a node from its partial view and exchanges a subset of its partial view with the selected node. Subsequently, both nodes update their partial views. Implementations of a PSS vary based on a number of different policies [21]:

1. *Node selection*: determines how a node selects another node to exchange information with. It can be either randomly (*rand*), or based on the node's age (*tail*).

Algorithm 3 T-Man - Active Thread

```
1: procedure ExchangeRT
2:     neighbor ← selectRandomNeighbor()
3:     buffer ← getSampleNodes()          ▷ provided by the peer sampling service
4:     buffer.merge(RT)                   ▷ RT is the local routing table
5:     Send [buffer] to neighbor
6:     Recv newBuffer from neighbor
7:     buffer.merge(newBuffer)
8:     RT ← selectNeighbors(buffer)
9: end procedure
```

Algorithm 4 T-Man - Passive Thread

```
1: procedure RespondToRTExchange
2:     Recv buffer from neighbor
3:     newBuffer ← getSampleNodes()
4:     newBuffer.merge(RT)
5:     Send [newBuffer] to neighbor
6:     newBuffer.merge(buffer)
7:     RT ← selectNeighbors(newBuffer)
8: end procedure
```

2. *View propagation*: determines how to exchange views with the selected node. A node can send its view with or without expecting a reply, called *push-pull* and *push*, respectively.

3. *View selection*: determines how a node updates its view after receiving the nodes' descriptors from the other node. A node can either update its view randomly (*blind*), or keep the youngest nodes (*healer*), or replace the subset of nodes sent to the other node with the received descriptors (*swapper*).

In our work, we employ a light-weight peer sampling service, for providing each node with a uniformly random set of existing nodes in the system. Such service allows our systems to work without the need for any global knowledge at any point.

2.4 Topology Management

The overlay topology management is one of the applications that benefits from peer sampling services. In this thesis, we utilize T-man [16], which is a generic protocol for topology construction and management. In T-man, each node, p, periodically exchanges its routing table (RT) with a neighbor, q, chosen uniformly at random among the existing neighbors in the routing table. Node p, then, merges its current routing table with q's routing table, together with a fresh list of the nodes, provided by the underlying peer sampling service (Algorithms 3, lines 2-7). The resulting list becomes the candidate neighbors list for p. Next, p selects a number of neighbors among the candidate neighbors and refreshes its current routing table. The same process will take place at node q (Algorithm 4). The core

idea of our topology construction is captured in the neighbor selection mechanism, referred to as *selectNeighbors* in Algorithms 3 and 4. Such flexibility in neighbor selection makes it possible to construct any desirable topology, from a single ring or random graph, to any complex topology like torus, etc.

2.5 Small-world Networks

The small-world phenomenon refers to the property that any two individuals in a network are usually connected through a short chain of acquaintances. The existence of such chains have been long studied by researchers in different sciences, ranging from mathematics and physics to sociology and communication networks.

In 2000, Kleinberg [27] argued that there exist two fundamental components to this phenomenon. One is that such short chains are ubiquitous and the other is that individuals are able to find these short chains, using only local information. Kleinberg introduced the notion of *distance* and showed that in a small-world network two nodes are connected not uniformly at random, but with a probability that is inversely proportional to their distance. More precisely, nodes u and v are connected to one another with probability $d(u,v)^{-\alpha}$, where $d(u,v)$ denotes the distance between the two nodes, and α is a structural parameter. Different values for α yields a wide range of small-world networks, from random to regular graphs. However, Kleinberg mathematically proved that a greedy routing algorithms works best only if α is equal to the number of dimensions in the network. In other words, navigation in a r-dimensional small-world network is most efficient only if nodes u and v are connected to each other with probability $d(u,v)^{-r}$.

Many peer-to-peer systems, such as Symphony [6], Oscar [9, 11] and Mercury [28], have already used Kleinberg's ideas to introduce overlay structures that are efficient in routing. We are also inspired by these works, in order to ensure a bounded routing complexity in our overlays.

Chapter 3

Thesis contribution

"Not everything that can be counted counts, and
not everything that counts can be counted"
- William Bruce Cameron, Sociologist, 1963

In this thesis we show how a decentralized gossiping protocol can be used to (i) construct overlays that are efficient for information dissemination, (ii) partition big graphs into partitions of any given number or size, and (iii) find communities in a graph that we construct to solve the cross-document coreference problem. In the next sections, we list our detailed contributions, as well as the delimitation of our work, in each of the mentioned fields separately.

3.1 Information Dissemination

The amount of data in the digital world surrounding us is increasing very rapidly. According to a study by IBM, "15 petabytes of data are created every day - 8 times the volume housed in all US libraries" [29]. Thus, finding the relevant information is becoming more like looking for a needle in a haystack. Publish/subscribe systems, or pub/sub systems for short, leverage this problem by providing users with only the information they are actually interested in. Users of such systems utilize a subscription service to express their interest in specific data by either subscribing to a priori-known categories of data or defining filters over the content of the information they want to receive. These subscription models are called *topic-based* and *content-based*, respectively [30]. In both models, whenever some new data appears in the system, the interested subscribers are notified.

Currently, the majority of these systems use a client/server model and rely on dedicated machines to provide subscribe services. However, with a rapidly growing number of users on the Internet, and a highly increasing number of subscriptions, it is becoming necessary to use decentralized models for providing such a service at a reasonable cost. Moreover, the centralized model raises a privacy problem, since

all the user interests are revealed to a central authority, while in the real life most users are reluctant to give away their personal interests for various privacy reasons. Therefore, researchers have turned to peer-to-peer overlays, as an alternative design paradigm to the centralized model. Peer-to-peer overlays, if well implemented, exploit the resources at the edges of the network to provide a scalable service at a low or almost no cost. The available resources in a peer-to-peer network grow/shrink when more nodes join/leave the system. However, continuous joins and fails in such networks should be gracefully handled in order to provide a reasonable quality of service. Many peer-to-peer publish/subscribe systems have been proposed so far. However, they either

- require a potentially unbounded number of connections per node, which renders the system unscalable, or
- are potentially inefficient in routing, which results in large message delivery latencies, or
- put a heavy and/or unbalanced load on the nodes, which could ultimately lead to rapid deterioration of the system's performance once the nodes start dropping the messages or choose to permanently abandon the system.

The main contributions on the subject of information dissemination are presented in form of two systems, *Vitis* [31, 32] and *Vinifera* [33], for topic-based and content-based publish/subscribe models, respectively. These contributions, which have been fully presented in Chapters 5, 6 and 7, can be summarized as follows:

- introducing novel algorithms for how to construct an overlay that adapts to user subscriptions and exploits the similarity of interests. With the use of gossip, we effectively cluster together the nodes with similar or overlapping subscriptions, while every node maintains only a bounded number of connections. These clusters are later exploited to reduce the amount of traffic overhead that is generated in the network.
- introducing a novel algorithm for leader election inside clusters by using only the undergoing gossiping protocol. These leaders, called *gateways* in Vitis terminology, are utilized to connect clusters of nodes with similar interest, while the generated traffic overhead is kept low.
- building efficient data dissemination paths over the clusters by enabling rendezvous routing over unstructured overlays. This is achieved by injecting structure into an otherwise unstructured overlay, using the ideas in the Kleinberg's model. We guarantee that the event delivery time complexity is in logarithmic order.
- introducing load balancing mechanisms that adapt the overlay structure to the load of the published messages.

- combining multiple techniques from various fields, including gossiping, structured overlays and hashing techniques, to construct systems that outperform the existing state-of-the-art solutions.
- implementing and evaluating these systems in simulation, using both synthetically generated and real-world data traces.

Delimitations

- **Durability.** *Durability* refers to the property that a generated data item will survive in the system permanently. This property is of great importance for many applications, specially database systems. A publish/subscribe system can also be augmented by a durable storage system, which guarantees the persistency of events, as well as subscriptions. A lot of research is going on to design distributed storage systems and key-value stores which provide such guarantees. However, these works are orthogonal to our work and are considered out of the scope of this document.
- **Content filtering and matching techniques.** There are some interesting work on how to filter data content in the overlay networks [34–36]. However, these works are orthogonal and can be complementary to our solutions. In particular, we can utilize [34] on top of our dissemination trees in order to better filter out the published content.
- **Security attacks and byzantine behaviors.** Although security issues are practically important in real-world systems, the research work to address such issues are also orthogonal to our work and are considered out of the scope of this research. We assume all nodes behave in accordance with the protocols. However, node and link failures are handled in our systems.

3.2 Graph Partitioning

Finding good partitions is a well-known and well-studied problem in graph theory [37]. In its classical form, graph partitioning usually refers to *edge-cut* partitioning, that is, to divide vertices of a graph into disjoint clusters of nearly equal size, while the number of edges that span separated clusters is minimum. There are some studies [38–40], however, that show tools that utilize edge-cut partitioning do not achieve good performance on real-world graphs (which are mostly power-law graphs), mainly due to unbalanced number of edges in each cluster. In contrast, both theory [41] and practice [42, 43] prove that power-law graphs can be efficiently processed in parallel if *vertex-cuts* are used. In contrast to edge-cut partitioning, a vertex-cut partitioning divides edges of a graph into equal size clusters. The vertices that hold the endpoints of an edge are also placed in the same cluster as the edge itself. However, the vertices are not unique across clusters and might have to

(a) Edge-cut (b) Vertex-cut

Figure 3.1: Partitioning a graph into three clusters

be *replicated* (cut), due to the distribution of their edges across different clusters. A good vertex-cut is one that requires minimum number of replicas. Figure 3.1 illustrate the difference between these two types of partitioning.

We focus on processing extremely large-scale graphs, e.g., user relationship and interaction graphs from on-line social networking services such as Facebook or Twitter, resulting in graphs with billions of vertices and hundreds of billions of edges. The very large scale of the graphs we target poses a major challenge. Although a very large number of algorithms are known for graph partitioning [44–51], including parallel ones, most of the techniques involved assume a form of cheap random access to the entire graph. In contrast to this, large scale graphs do not fit into the main memory of a single computer, in fact, they often do not fit on a single local file system either. Worse still, the graph can be fully distributed as well, with only very few vertices hosted on a single computer.

We provide a distributed balanced graph partitioning algorithm, called Ja-be-Ja, both for edge-cut and vertex-cut partitioning. Choosing between edge-cut and vertex-partitioning depends on the application, and Ja-be-Ja, to the best of our knowledge, is the only algorithm that can be applied in both models. Ja-be-Ja is a decentralized local search algorithm and it does not require any global knowledge of the graph topology. That is, we do not have cheap access to the entire graph and we have to process it only with partial information. Each vertex of the graph is a processing unit, with local information about its neighboring vertices, and a small subset of random vertices in the graph, which it acquires by purely local interactions. Initially, every vertex/edge is assigned to a random partition, and over time vertices communicate and improve upon the initial assignment.

Our algorithm is uniquely designed to deal with extremely large distributed graphs. The algorithm achieves this through its locality, simplicity and lack of synchronization requirements, which enables it to be adapted easily to graph processing frameworks such as Pregel [2] or GraphLab [1]. Furthermore, Ja-be-Ja can be applied on fully distributed graphs, where each network node represents a single graph vertex.

To evaluate Ja-be-Ja for edge-cut partitioning, we use multiple datasets of different characteristics, including a few synthetically generated graphs, some graphs that are well-known in the graph partitioning community [52], and some sampled graphs from Facebook [53] and Twitter [54]. We first investigate the impact of different heuristics on the resulting partitioning of the input graphs, and then com-

pare JA-BE-JA to METIS [45], a well-known centralized solution. We show that, although JA-BE-JA does not have cheap random access to the graph data, it can work as good as, and sometimes even better than, a centralized solution. In particular, for large graphs that represent real-world social network structures, such as Facebook and Twitter, JA-BE-JA outperforms METIS [45].

For vertex-cut partitioning, we will compare our solution with a state-of-the-art system [55], and we show that JA-BE-JA not only guarantees to keep the size of the partitions balanced, but also outperforms its counterparts with respect to vertex-cut.

The main contributions on this subject are published in two papers, JA-BE-JA [56] and JA-BE-JA-VC [57], for edge-cut and vertex-cut partitioning, respectively. These papers are fully presented in Chapters 8 and 9.

Delimitations

- Edge-cut partitioning for the graphs with weighted vertices can not be readily addressed with JA-BE-JA. The reason is JA-BE-JA requires the initial color distribution to be an invariant. It is for exactly this reason, that graph vertices can not change their colors independently and have to find some other vertices to swap their color with. However, this will only work if the two swapping vertices are of equal weights. If vertices have different weights, the color distribution before and after the swap will not be the same. Likewise, JA-BE-JA-VC is not readily applicable to vertex-cut partitioning for graphs with weighted edges.

3.3 Coreference Resolution

Resolving entities in a text may not always be a difficult task for humans. When one comes across Mercury in an article about the solar system, they instantly think of Mercury, the planet, and not about Mercury, the chemical element or Freddie Mercury. For a computer though, such a disambiguation requires a considerable amount of processing. This problem, i.e., the task of disambiguating manifestations of real world entities in various records or mentions, is known as *Entity Resolution* or *Coreference Resolution*. The ambiguity arises from the fact that the same word can refer to multiple entities. Often disambiguation is required across multiple documents. For example, there are various articles across the web that contain news about Mercury. Given a set of such documents with an ambiguous *mention* (Mercury, for example), the *Cross-Document Coreference* problem seeks to group together those documents that talk about the same *entity* in real world (e.g., one group for the planet, one for the chemical element, etc.).

This problem is challenging because: (i) often the number of underlying entities and their identities are not known (e.g., we do not know how many different Mercuries are to be discovered), and (ii) the number of possible classifications grows exponentially with the number of input documents.

A widely used approach to this problem, known as *Mention-Pair model*, is to compute a pair-wise similarity value based on the common keywords that exist in each pair of documents [58]. If two documents are found similar more than a predefined threshold, they are classified together. However, this requires huge amount of computations. The high complexity of the Mention-Pair model renders it impractical for web-scale coreference, where we have to process millions of documents in a reasonable time. Even after the costly computation step, a clustering step is required to partition the mentions into coreferent groups. The clustering itself is a challenging task and is known to be NP-hard.

We propose a novel approach to coreference resolution, which does not require separate classification and clustering steps. Instead, we transform the problem to a vertex-centric graph processing task. This enables us to take advantage of the recent advances in graph processing frameworks, such as GraphChi [59] or GraphLab [1], and apply our algorithm to extremely large graphs.

To construct the graph, we create two types of nodes. One type represents the ambiguous word, which we assume is given in advance. Another type of nodes represents the unambiguous words that surround the ambiguous word in each document. Since we do not know whether or not different mentions of the ambiguous word are referring to the same real-world entity, we create as many nodes as the number of documents mentioning them. The unambiguous words might as well appear in multiple documents. For them, however, we do not create a new node, if they already exist. Finally, we add an edge between two nodes, if their corresponding words co-occurred in the same document. Consequently, each single document is represented by a full mesh, or *clique*, of all its keywords.

We will then observe that some cliques overlap, which indicates that their corresponding documents have a similar context. In fact, the main insight to our work is that the topological community structure of the constructed graph identifies similar contexts and thus, is an accurate indicator of the coreference classification. Based on this fact, we propose a novel community detection algorithm for coreference resolution. Our algorithm is diffusion based and exploits the fundamentals of flow networks. In such a network each node has a capacity and each edge can transfer a flow, just like a pipe, between two nodes. We envision multiple flows in our graph, one per community. To distinguish these flows, we assign a distinct color to each of them.

Initially each single document constitutes a distinct community, i.e., it will be assigned to a unique color. All the nodes that belong to a document will get a unit of the color of their document. Therefore, those nodes that are shared between documents, will receive multiple units of colors. However, each node always identifies itself with only a single color, which has the highest collective volume in its neighborhood, so-called the *dominant color*. Nodes continuously exchange parts of their colors with their neighbors by diffusing the colors through their links. Therefore, the available volume of color at nodes, and accordingly the dominant color in their vicinity, changes during the course of algorithm. We will show that with appropriate diffusion policies it is possible to accumulate one distinct color

in each of the well connected regions of the graph. Finally, the ambiguous nodes that end up having the same dominant color are considered to be coreferent. Since our constructed graph is sparse, the overhead of such computation remains low. Moreover, we can produce more accurate results, compared to the state of the art. This twofold gain is owed to the combination of two ideas, that constitute our main contributions:

- a technique for transforming the expensive coreference problem, into a graph problem, in which the coreferent words belong to the same topological community structure. The graph that we construct is sparse, because those documents that have dissimilar contexts, will have very few or even no direct connections. The computations on the graph are performed per edge basis, i.e., only if there is an edge between two nodes, they will communicate some flows. Hence, the irrelevant documents which are weakly connected, if not disconnected, will not impose any computation in the graph. At the same time, a more thorough search of the solution space is possible, as we are not limited to pair-wise similarity discoveries only. Instead, similarity between any number of documents is naturally captured within the community structures that emerge from the inter-linked context words.

- a novel node-centric diffusion-based community detection algorithm that mainly uses local knowledge of the graph at each node. Hence, it allows for highly parallel computations and usage of the existing graph processing frameworks.

We run our algorithm on different datasets, which are transformed to graphs with distinct structural properties. For example, on a baseline dataset for person name disambiguation, we produce a classification with an F-score 15% higher than that of the state of the art algorithm by Singh et al. [60]. Moreover, on a dataset provided in the Word Sense Induction task of SemEval 2010, we achieved as good F-score as the best reported result. However, we considerably outperform the other solutions with respect to a complementary accuracy metric, which measures the average number of items in each clusters. The full paper is presented in Chapter 10.

Delimitations

- To discover which word(s) is (are) ambiguous, is a different problem which is orthogonal to our work and out of the scope of this document.

- Tagging the classified entities, i.e., annotating the entities with the correct label requires external knowledge, for example from Wikipedia or DBpedia, and is orthogonal to our work, hence, out of the scope of this document.

3.4 Publications

- F. Rahimian, S. Girdzijauskas, A. Payberah, and S. Haridi, *Vitis: A gossip-based hybrid overlay for Internet-scale publish/subscribe*, in IEEE International Parallel & Distributed Processing Symposium, 2011.
- F. Rahimian, T. L. N. Huu, and S. Girdzijauskas, *Locality-awareness in a peer-to-peer publish/subscribe network*, in Distributed Applications and Interoperable Systems, Springer, 2012.
- F. Rahimian, S. Girdzijauskas, A. H. Payberah, and S. Haridi, *Subscription awareness meets rendezvous routing*, in AP2PS 2012, The Fourth International Conference on Advances in P2P Systems, 2012.
- F. Rahimian, A. H. Payberah, S. Girdzijauskas, M. Jelasity, and S. Haridi, *Ja-Be-Ja: A distributed algorithm for balanced graph partitioning*, in Proc. of the 7th IEEE International Conference on Self-Adaptive and Self- Organizing Systems (SASO), IEEE, Sep. 2013.
- F. Rahimian, T. L. N. Huu, and S. Girdzijauskas, *Distributed vertex-cut partitioning*, in Distributed Applications and Interoperable Systems, Springer, 2014.
- F. Rahimian, S. Girdzijauskas, and S. Haridi, *Parallel Community Detection for Cross-Document Coreference*, 2014.

List of the publications that are not included in this thesis.

- A. Payberah, J. Dowling, F. Rahimian, and S. Haridi, *gradientv: Marketbased p2p live media streaming on the gradient overlay*, in Distributed Applications and Interoperable Systems, Springer, 2010.
- A. Payberah, J. Dowling, F. Rahimian, and S. Haridi, *Sepidar: Incentivized market-based p2p live-streaming on the gradient overlay network*, in International Symposium on Multimedia, 2010.
- A. Payberah, J. Dowling, F. Rahimian, and S. Haridi, *Distributed Optimization of P2P Live Streaming Overlays*, Springer Computing, Special Issue on Extreme Distributed Systems: From Large Scale to Complexity (Computing), June 2012.

Chapter 4

Conclusions

"Success is not final, failure is not fatal: it is the courage to continue that counts."
- Winston Churchill

We believe that without parallel and distributed algorithms, big data processing will become prohibitively costly and the new advances in the emerging computing frameworks will not be effectively uitilized. Hence, developing decentralized algorithms that can operate on partial data on top of these computing frameworks is absolutely necessary. Our work is a small step in that direction.

We employed the power of gossip to solve various problems, including data dissemination, partitioning and disambiguation. Gossiping enables all our solutions to work with partial knowledge and with a very low communication overhead. Thus, our algorithms can be executed efficiently and effectively in a distributed environment.

Our solutions for data dissemination scale to very large networks, while producing a relatively low overhead. Data is delivered to the interested users (subscribers) reliably and fast. Even in the presence of failures the delivery rate of the system remains high. In particular, we presented Vitis, a topic-based publish/subscribe system, which scales with the number of nodes as well as the number of topics in the overlay. The main contribution of this work is a novel hybrid publish/subscribe overlay that exploits two ostensibly opposite mechanisms: unstructured clustering of similar nodes and structured rendezvous routing. We employed a gossiping technique to embed a navigable small-world network, which efficiently establishes connectivity among clusters of nodes that exhibit similar subscriptions. We also give a theoretical bound on the worst case delay. Moreover, we showed how we can embed locality information into the overlay topology, that is, how to make the connections between nodes to better reflect the physical network connections. We introduced a notion of distance in the neighbor selection mechanism of Vitis and studied how we can exploit locality-awareness in-line with the subscription correlations. Finally, we showed that Vitis adapts to biased rates of events that are

published on different topics, and builds more efficient groups for hot topics, thus, improving the overall performance of the event dissemination.

We also presented Vinifera, a content-based publish/subscribe system that uses a gossip-based technique to construct a topology that not only resembles a small-world network, but also connects the nodes with similar subscriptions together. On top of this hybrid overlay, we utilized a rendezvous routing mechanism to propagate node subscriptions in the overlay. Together with an order preserving hashing technique and an efficient showering algorithm we enabled range queries, and at the same time, we employed a load balancing technique to deal with the potential non-uniform user subscriptions. The combination of all these techniques are seamlessly integrated within a single gossiping layer, thus keeping Vinifera simple, lightweight and robust.

For graph partitioning, we provided Ja-be-Ja, an algorithm that to the best of our knowledge, is the first distributed algorithm for balanced edge-cut partitioning that does not require any global knowledge. To compute the partitioning, nodes of the graph require only some local information and perform only local operations. Therefore, the entire graph does not need to be loaded into memory, and the algorithm can run in parallel on as many computers as available. We showed that our algorithm can achieve a quality partitioning, as good as a centralized algorithm. We also studied the trade-off between the quality of the partitioning versus the cost of it, in terms of the number of swaps during the run-time of the algorithm.

We also presented Ja-be-Ja-vc, a distributed and parallel algorithm for vertex-cut partitioning. Ja-be-Ja-vc partitions edges of a graph into a given number of clusters with any desired size distribution, while the number of vertices that have to be replicated across clusters is low. In particular, it can create balanced partitions while reducing the vertex-cut. Ja-be-Ja-vc is a local search algorithm that iteratively improves upon an initial random assignment of edges to partitions. It also utilizes simulated annealing to prevent getting stuck in local optima. We compared Ja-be-Ja-vc with two state-of-the-art systems, and showed that Ja-be-Ja-vc not only guarantees to keep the size of the partitions balanced, but also outperforms its counterparts with respect to vertex-cut.

Finally, we presented a graph-based approach to coreference resolution. We showed that by using a graph representation of the documents and their context, and applying a community detection algorithm we can speed up the task of coreference resolution by a very large degree. The accuracy of coreference resolution could also be improved at the same time, because we are able to search beyond only pair-wise comparisons. The graph that we construct enables us to discover any existing closeness/similarity between any subset of documents. Thus, we can explore the solution space more freely and more smartly.