Alma Mater Studiorum – Università di Bologna

DOTTORATO DI RICERCA IN
Ingegneria Elettronica, Informatica e delle Telecomunicazioni
Ciclo XXVII

Settore concorsuale di afferenza: 09/H - INGEGNERIA INFORMATICA
Settore scientifico disciplinare: ING-INF/05 SISTEMI DI ELABORAZIONE DELLE INFORMAZIONI

DISTRIBUTED INFORMATION SYSTEMS AND DATA MINING IN SELF-ORGANIZING NETWORKS

Presentata da: Tommaso Pirini

Coordinatore Dottorato
Prof. Alessandro Vanelli-Coralli

Relatore
Prof. Gianluca Moro
Co-Relatore
Prof. Claudio Sartori

Esame finale anno 2015-2016

Contents

Chapter 1

Introduction

The development of network technology, sensors and portable electronic devices with increasing computational power, memory and connectivity at the same time with decreasing cost, size and energy consumption, has opened up new application scenarios. The possibility of transforming entire cities in smart organisms able of improving the quality of life, the environment, the effectiveness of services (e.g. tourism, social, health, transport...) creates new business opportunities. The diffusion of sensors and devices to generate and collect data is capillary. The infrastructure that envelops the smart city has to react to the contingent situations and to changes in the operating environment. At the same time, the complexity of a distributed system, consisting of huge amounts of components fixed and mobile, can generate unsustainable costs and latencies to ensure robustness, scalability, and reliability, with type architectures middleware. A critical point, therefore, is to exploit the intrinsic decentralization of the system by distributing much control as possible, leaving and aggregating the data collected on the device itself if necessary for processing in a distributed manner. To do this, the distributed system must be able to self-organize and self-restore adapting its operating strategies to optimize the use of resources and overall efficiency. This work involved the study of models, methods and algorithms to produce infrastructure software for decentralized control. It can generate self-organizing networks for smart city, with the use of techniques that maintain network security from existing innovative approaches. The infrastructure must offer efficient routing, management, research and automated analysis of the data collected and distributed, and must be able to scale to networks with millions of devices.

The management of huge amounts of data distributed across multiple

sites is becoming more and more important. Peer-to-peer systems (P2P) can offer solutions to face the requirements of managing, indexing, searching and analyzing data in scalable and self-organizing fashions, such as in cloud services and big data applications, just to mention two of the most strategic technologies for the next years. To this purpose P2P systems should overcome some drawbacks, such as the issue of the traffic load unbalancing across their autonomous and heterogenous nodes. Moreover, most P2P systems become popular in the literature, such as Chord and Tapestry, are inefficient in processing multi-dimensional exact and range queries over any arbitrary combination of data attributes because their indexing models do not support natively these kind of multi-dimensional queries.

In this thesis we present G-Grid, a multi-dimensional distributed data indexing able to efficiently execute arbitrary multi-attribute exact and range queries in decentralized P2P environments. Moreover G-Grid guarantees the completeness of the query results like any DBMSs. In G-Grid data can be distributed on various autonomous peers, without a centralized management. Peers organize themselves by local interactions among neighbors, and the total decentralization of control makes it suitable for intra-organisational applications, e.g. distributed data center of a single company, and inter-organisational applications, e.g. Bitcoin. G-Grid is adaptive regarding dynamic changes in network topology. G-Grid is a foundational structure and can be effectively used in a wide range of application environments, including grid computing, cloud and big data domains. We present several experiments to compare G-Grid with recent and efficient P2P structures presented in literature. The results show that G-Grid further improves network management and traffic overlay.

This efforts have focused primarily on improving the number of hops and the number of structure maintenance messages. However, the massive use of these P2P networks caused the emerging problem of the traffic load balancing. Two of the main causes that create inequalities in the traffic distribution are the non-uniform distribution of data and the hierarchical structures. In fact, G-Grid is basically organized as a binary tree and this causes an unbalanced traffic load, overburdening the peers as much as they are close to the tree root. To equally balance the load on this kind of hierarchical P2P systems, rather than imposing globally some load balancing rules, we seamless integrated a ring-based overlay in G-Grid that leads to the emergent property of self-balancing. Experiments show how drastically the new integrated overlay, which lead to the new P2P systems we call G-Grid Chord, improve

the balancing according to an almost uniform load distribution among nodes.

Nevertheless we detected in the last improvement on the structure a huge growing of maintenance system traffic into the network, so we decided to made a further step to enhance the G-Grid structure introducing a bit of randomness. It is obtained by merging G-Grid Chord with a Small World network. The Small World networks whereas are structures of compromise between order and randomness. These networks are derived from social networks and show an almost uniform traffic distribution. There are, in fact, a lot of algorithms introducing a bit of randomness to produce huge advantages in efficiency, cutting maintenance costs, without losing efficacy. Experiments show how this new hybrid structure obtains the best performance in traffic distribution and it a good settlement for the overall performance on the requirements desired in the modern data systems.

This thesis starts with Chapter 2 on P2P Networks, with an application studied initially on autonomic security. Chapter 3 describes the basic version of G-Grid and its features. Chapter 4 presents the evolution of G-Grid using the organised overlay Chord to improve load balancing. Finally in Chapter 5 we describe the last improvement of the G-Grid structure produced by the merging with the Small World network principles. At the end of each chapter there are experiments that show the concrete performances of the new introduced improvements.

In closing, the Appendix A is described the work developed during the PhD at the cole Polytechnique Fdrale de Lausanne, Switzerland, as collaboration with the Distributed Information Systems Laboratory (LSIR) directed by Prof. Karl Aberer.

Chapter 2

Peer-to-Peer Networks

Peer-to-Peer (P2P) networks are virtual communities on the Internet where participants directly connect with one another or where they use an intermediate service to directly connect with one another. Usually, in this distributed application architectures all the participants, called peers, are equally privileged and have the same capabilities and responsibilities. Unlike the client-server model, in which the client makes a service request and the server fulfils the request, the P2P network model allows each peer to function as both client and server. Each peer makes a portion of their resources, such as processing power, disk storage, network bandwidth or simply data, directly available to other network peers, without the need for central organization by servers or stable hosts [117].

P2P systems can be used to provide anonymized routing of network traffic, massive parallel computing environments, distributed storage and other functions. Typically, P2P applications allow users to control many parameters of operation: how many member connections to seek or allow at one time; whose systems to connect to or avoid; what services to offer; and how many system resources to devote to the network. Some simply connect to some subset of active nodes in the network with little user control, however.

Although uses for the P2P networking topologies have been explored since the days of ARPANET, the advantages of the P2P communications model did not become obvious to the general public until the late 1990s, when music-sharing P2P applications like Napster appeared. Napster and its successors, like Gnutella, and more recently, BitTorrent, cut into music and movie industry profits and changed how people thought about acquiring and consuming media. Most P2P programs indeed are focused on media sharing and P2P is

therefore often associated with software piracy and copyright violation.

Today, there is a huge amount of P2P systems. Their growing success is owed to the spread broadband and ADSL flat increasingly accessible.

2.1 P2P Architectures

P2P networks are designed around the notion of equal peer nodes simultaneously functioning as both *clients* and *servers* to the other nodes on the network. P2P networks generally implement some form of virtual overlay network on top of the physical network topology, where the nodes in the overlay form a subset of the nodes in the physical network. Data is still exchanged directly over the underlying TCP/IP network, but at the application layer peers are able to communicate with each other directly, via the logical overlay links, each of which corresponds to a path through the underlying physical network. Overlays are used for indexing and peer discovery, and make the P2P system independent from the physical network topology. Based on how the nodes are linked to each other within the overlay network, and how resources are indexed and located, P2P networks come in three flavors [5, 57, 151]:

- Unstructured networks
- Structured networks
- Semi-structured or hybrid networks

We see specifically what are these categories' features.

2.1.1 Unstructured Networks

Unstructured P2P networks do not impose a particular structure on the overlay network by design, but rather are formed by nodes that randomly form connections to each other [35, 103]. Examples of unstructured P2P protocols are the following:

- **Gnutella**: the first open and decentralized P2P network for file sharing created by Justin Frankel of Nullsoft [113];

- **Gossip protocols**: style of computer-to-computer communication protocol inspired by the form of gossip seen in social networks. The term epidemic protocol is sometimes used as a synonym for a gossip protocol, because gossip spreads information in a manner similar to the spread of a virus in a biological community [30];

- **Kazaa**: P2P file sharing application using the FastTrack protocol licensed by Joltid Ltd. and operated as Kazaa by Sharman Networks [74, 36].

Because there is no structure globally imposed upon them, unstructured networks are easy to build and allow for localized optimizations to different regions of the overlay [22]. Also, because the role of all peers in the network is the same, unstructured networks are highly robust in the face of high rates of "churn" – that is, when large numbers of peers are frequently joining and leaving the network [55, 119].

However the main limitations of unstructured networks also arise from their lack of structure. In fact, when a peer wants to find a desired information in the network, the search query must be flooded through the network to find as many peers as possible that share the data. Flooding causes a very high amount of traffic in the network, uses more peer resources – like CPU/memory by requiring every peer to process all search queries –, and does not ensure that the desired information will always be found and delivered. Furthermore, since there is no correlation between a peer and the content managed by it, there is no guarantee that flooding will find a peer that has the desired information. Popular content is likely to be available at several peers and any peer searching for it is likely to find the same thing. But if a peer is looking for rare data shared by only a few other peers, then it is highly unlikely that search will be successful [119].

2.1.2 Structured Networks

In structured P2P networks the overlay is organized into a specific topology. This makes the search for a resource in the network extremely efficient – typically approximating O(log N), where N is the number of nodes in the network –, even if the resource is rare.

Usually the structured P2P networks employ a Distributed Hash Table (DHT) [110, 129], in which a variant of consistent hashing is used to assign

ownership of each resource to a particular peer [60, 28]. This enables peers to search for resources on the network using a hash table: that is, (key, value) pairs are stored in the DHT, and any peer can retrieve the value associated with a given key [93, 81]. Other design structures are overlay rings and d-Torus [12].

In order to route traffic efficiently through the network, nodes in a structured overlay must maintain lists of neighbors that satisfy specific criteria. This makes them less robust in networks where nodes frequently join and leave the overlay [79, 71]. More recent evaluation of P2P resource discovery solutions under real workloads have pointed out several issues in DHT-based solutions such as high cost of advertising/discovering resources and static and dynamic load imbalance [11].

However DHTs are used in several notable distributed networks such as:

- **BitTorrent**: protocol for P2P file transfer [24] that makes many small data requests over different IP connections to different machines, instead of a single TCP connection to a single machine. BitTorrent downloads in a random or in a "rarest-first" [69] approach that ensures high availability with respect to sequential downloads. This protocol provides no way to index torrent files;

- **Kad network**: P2P network which implements the Kademlia overlay protocol [82], that specifies the structure of the network and the exchange of information through node lookups, using UDP;

- **Storm botnet**: famous P2P remotely controlled network of Microsoft Windows computer [54] used a modified version of the eDonkey/Overnet communications protocol [123];

- **YaCy**: free P2P search engine without main server with indices and where each Linux-server with an installed YaCy separate downloads, indexes the Web and processes user queries to search for documents throw other servers in the YaCy network. YaCy uses DHT for defined, simple and effective allocation documents between nodes: nodes calculate (key; value) pairs for all documents in the network and then use these pairs for looking for and getting required file by participating in required DHT [95];

- **Coral Content Distribution Network (CoralCDN)**: P2P content distribution network that offers high performance and meets huge de-

mand in accessing web contents. CoralCDN automatically replicate content as a side effect of users accessing it and avoid creating hot spots. It achieves this through Coral, a latency-optimized hierarchical indexing infrastructure based on a distributed "sloppy" hash table [37].

Some prominent research projects in structured P2P networks include:

- **Chord project**: distributed lookup protocol that maps a key onto a node that stores a particular data item, characterized by that key [125]. Chord uses a variant of consistent hashing [59] to assign keys to nodes. Chord is routing a key through a sequence of $\mathcal{O}(\log N)$ other nodes toward the destination, where N is the nodes in the network;

- **PAST storage utility**: large-scale, distributed, persistent storage system based on Pastry P2P overlay network [115, 116, 32]. Pastry implements a DHT where the (key; value) pairs are stored in a redundant P2P network;

- **P-Grid**: self-organized and emerging overlay network which can accommodate arbitrary key distributions, providing storage load-balancing and efficient search by using randomized routing [3, 2];

- **CoopNet**: content distribution system for off-loading serving to peers who have recently downloaded the file. It assigns peers to other peers who are locally close to its neighbors – same IP prefix range. If multiple peers are found with the same file the system chooses the fastest node of that peer's neighbors [101].

DHT-based networks have also been widely utilized for accomplishing efficient resource discovery for grid computing systems, as it aids in resource management and scheduling of applications [109, 108].

2.1.3 Hybrid Networks

Hybrid systems, like Spotify [64], combine P2P and client-server principles [29]. A common hybrid model is to have a central server that helps peers find each other. There are a variety of hybrid models, all of which make trade-offs between the centralized functionality provided by a structured server/client network and the node equality afforded by the pure P2P unstructured networks. Tipically, this systems distribute their peers into two

groups: clients nodes and overlay nodes. Each peer is able to act according the momentary need of the network and can become part of the overlay used to coordinate the P2P structure, such as in the Gnutella protocol.

Currently, hybrid models have better performance than either pure unstructured networks or pure structured networks because certain functions, such as searching, do require a centralized functionality but benefit from the decentralized aggregation of nodes provided by unstructured networks [143].

2.2 P2P Applications

The most popular application of the P2P principle is the file-sharing, that made P2P popular. The use of P2P file-sharing software, such as BitTorrent clients, is responsible for the bulk of P2P internet traffic. From 2004 on, such networks form the largest contributor of global network traffic on the Internet.

In P2P networks, clients both provide and use resources. This means that, potentially, the content serving capacity of P2P networks can actually increase as more users begin to access the content, especially with protocols that require users to share, such as BitTorrent. This property is one of the major advantages of using P2P networks because it makes the setup and running costs very small for the original content distributor [72, 127]. Sharing computational power is mainly used for problem solving or complex calculations. Some examples are SETI@Home [**?**], the Great Internet Mersenne Prime Search (GIMPS) [138], Distributed.net [1].

Some multimedia applications use P2P protocols, like P2PTV and PDTP [131], for the diffusion of high data streams generated in real time. Usign trasmission bandwidth of individual peers there are not required huge server performances, but that peers are provided with high bandwidth connections in both reception and trasmission.

Other P2P applications are P2P-based digital cryptocurrencies, like for examples Bitcoin [92] and alternatives such as Peercoin [122], and so on.

2.3 Security and trust in P2P networks

P2P systems pose some challenges also from a computer security perspective. What makes this particularly dangerous, however, is that P2P applications

act as servers as well as clients, meaning that they can be more vulnerable to remote exploits [133], like routing attacks and handling corrupted data and malware.

Since each node plays a role in routing traffic through the network, malicious users can perform a variety of routing attacks or denial of service attacks. Examples of common routing attacks include:

- **incorrect lookup routing**: whereby malicious nodes deliberately forward requests incorrectly or return false results,
- **incorrect routing updates**: where malicious nodes corrupt the routing tables of neighboring nodes by sending them false information,
- **incorrect routing network partition**: where when new nodes are joining they bootstrap via a malicious node, which places the new node in a partition of the network that is populated by other malicious nodes.

On the other hand, the prevalence of malware varies between different P2P protocols. Studies analyzing the spread of malware on P2P networks found, for example, that 63% of the answered download requests on the Limewire network contained some form of malware, whereas only 3% of the content on OpenFT contained malware. Another study analyzing traffic on the Kazaa network found that 15% of the 500,000 file sample taken were infected by one or more of the 365 different computer viruses that were tested for [46].

Corrupted data can also be distributed on P2P networks by modifying files that are already being shared on the network. For example, on the FastTrack network, the Recording Industry Association of America (RIAA) managed to introduce malicious code into downloads and downloaded files in order to deter illegal file sharing [121]. Consequently, the P2P networks of today have seen an enormous increase of their security and file verification mechanisms. Modern hashing, chunk verification and different encryption methods have made most networks resistant to almost any type of attack, even when major parts of the respective network have been replaced by faked or nonfunctional hosts [120].

Some P2P networks (e.g. Freenet) place a heavy emphasis on privacy and anonymity – that is, ensuring that the contents of communications are hidden from eavesdroppers, and that the identities/locations of the participants are concealed. Public key cryptography can be used to provide encryption,

data validation, authorization, and authentication for data/messages. Onion routing and other mix network protocols (e.g. Tarzan) can be used to provide anonymity [133].

2.3.1 Intellectual Property and Network Neutrality

Although P2P networks can be used for legitimate purposes, rights holders have targeted P2P over the involvement with sharing copyrighted material. Companies developing P2P applications have been involved in numerous legal cases, primarily in the United States, primarily over issues surrounding copyright law [45]. Two major cases are Grokster vs RIAA and MGM Studios, Inc. v. Grokster, Ltd.[1]. In both of the cases the file sharing technology was ruled to be legal as long as the developers had no ability to prevent the sharing of the copyrighted material. Moreover, controversies have developed over the concern of illegitimate use of P2P networks regarding public safety and national security. When a file is downloaded through a P2P network, it is impossible to know who created the file or what users are connected to the network at a given time.

P2P applications present one of the core issues in the *network neutrality* controversy. Internet service providers (ISPs) have been known to throttle P2P file-sharing traffic due to its high-bandwidth usage [120]. Compared to Web browsing, e-mail or many other uses of the internet, where data is only transferred in short intervals and relative small quantities, P2P file-sharing often consists of relatively heavy bandwidth usage due to ongoing file transfers and swarm/network coordination packets. ISPs rationale was that P2P is mostly used to share illegal content, and their infrastructure is not designed for continuous, high-bandwidth traffic. On the other hand, critics point out that P2P networking has legitimate legal uses, and that this is another way that large providers are trying to control use and content on the Internet, and direct people towards a client-server-based application architecture. The client-server model provides financial barriers-to-entry to small publishers and individuals, and can be less efficient for sharing large files. As a reaction to this bandwidth throttling, several P2P applications started implementing protocol obfuscation, such as the BitTorrent protocol encryption [53]. Techniques for achieving "protocol obfuscation" involves removing otherwise easily identifiable properties of protocols, such as deter-

[1] `http://www.copyright.gov/docs/mgm/`

ministic byte sequences and packet sizes, by making the data look as if it were random. The ISP's solution to the high bandwidth is P2P caching, where an ISP stores the part of files most accessed by P2P clients in order to save access to the Internet.

The promotion of network neutrality is no different than the challenge of promoting fair evolutionary competition in any privately owned environment, whether a telephone network, operating system, or even a retail store. Government regulation in such contexts invariably tries to help ensure that the short-term interests of the owner do not prevent the best products or applications becoming available to end-users. The same interest animates the promotion of network neutrality: preserving a Darwinian competition among every conceivable use of the Internet so that the only the best survive [139].

2.4 An application to Autonomic Security

At first, to study the properties of some structured P2P networks, we developed a simulator applied to the topic of network security, working with the telecommunications engineering research team of the Prof. Callegati[2] (Alma Mater Studiorum – University of Bologna). This work was partially supported by the Italian MIUR Project *Autonomous Security* in the PRIN 2008 Programme.

In this work [20] we proposed new distributed data mining algorithms to recognize network attacks against a set of devices from statistic data generated locally by each device according to the standard Simple Network Management Protocol (SNMP) available in each modern operating systems. The idea is to place an autonomous mining resource in each network node that cooperates with its neighbors in a P2P fashion in order to reciprocally improve their detection capabilities. Differently from existing security solutions, which are based on centralized databases of attack signatures and transmissions of huge amount of raw traffic data, in this solution the network nodes exchange local knowledge models of few hundred bytes. The approach efficacy has been validated performing experiments with several types of attacks, with different network topologies and distributions of attacks so as to also test the node capability of detecting unknown attacks.

[2] `http://www.unibo.it/Faculty/default.aspx?UPN=franco.callegati%40unibo.it`

2.4.1 P2P Data Mining Classifiers for Decentralized Detection of Network Attacks

Data mining aims to automatically discover new knowledge from huge amount of data useful to explain or predict unknown phenomenon. Most of the current techniques have been developed for centralized systems where all the available data is collected in a single site. However, the growing need to apply these techniques to large data sets distributed over the network, has led to the deployment of distributed data mining algorithms [78, 62, 26, 77, 84, 85, 86, 90]. Recently several distributed data mining algorithms have been introduced, however the best approaches are inadequate to work in decentralized systems where nodes are in general autonomous and may arbitrarily leave and join the system, for instance because belong to different organizations or people.

We introduced two novel distributed classification algorithms taking inspiration from a well-known centralized algorithm called AdaBoost [39]. AdaBoost has been used for distributed analysis and in parallel processing so far. The previous approaches deal with performance improvements aspects and data aggregation. In this work, the new AdaBoost algorithms are used to generate and share knowledge models across network of autonomous resources.

We then show how these algorithms have been applied to network security in which an attacker attacks a single or a group of host. In particular, we have investigated a collaborative behavior between network entities in which each one does not share huge amount of raw data, as it happens in decentralized systems, but rather sharing only knowledge models. The shared knowledge models, which consist only of few hundred bytes, are locally generated from local data according to the Simple Network Monitoring Protocol (SNMP), available in every operating systems. [19] achieved optimal results generating knowledge models according to an unsupervised solution, in which, differently from this new contribution, we shared SNMP data among peers.

The cooperation among peers benefits are mainly in the exchange of knowledge models. In the first place, this produces a strong decrease of the traffic amount in the network, for example with respect to the exchange of raw records. The latter would probably achieve the same result or better, but the records of which each peer derives its knowledge models with the mining algorithm can be huge and very dynamic – constantly changing. When the network have been distributed knowledge models generated by

centralized data, these may no longer be valid because the environmental situation changes rapidly. This is true as far as the network is extensive and real. Exchanging the models the system exchange the same knowledge, because the mining algorithm is the same for each node, but in a higher level and more convenient. This is an advantage for the scalability of the system: if the network is very large, leading all records at each node or in a central node to be analyzed is not convenient, also in terms of consistency, in modern dynamic scenarios. The knowledge models are useful to nodes that receive them because they represent guidelines to judge – in this example, classify – their raw data – in particular in this example data generated by the SNMP – according mining models derived from other data, that carriers new information. We want to reach the results obtained grouping into a single node all raw data for a centralized analysis and handing out the global knowledge models. This in many practical situations it is not possible for obvious reasons: dynamic environments, huge amount of raw data that changes quickly and continuously, very large networks, etc.

2.4.2 Literature on Distributed Data Mining

Extracting information from very large distributed data-bases is a big challenge for data mining. Many databases are too big to be collected at a single site, and centralized training could be very slow in these specific cases. Today some databases are inherently distributed and cannot be unified for a long list of reasons.

For example, in a real-time environment, large amounts of data could be collected so often and in different patterns, which in the time required to make them consistent in a single structure would have lost their validity. As well, in distributed environments, it may happen that the effectiveness of certain information is limited to a single environment, so if you collected them all in a single database, some data may lose consistency.

Therefore it is very difficult to be able to design a single classification algorithm under these constraints. For example, the classification model construction to detect credit card frauds needs a huge strictly distributed data set. In addition, there are some databases having a large amounts of new data, available periodically or in real time. Re-training a new model on an entire data set it is both inefficient and expensive since each at time there is a significant increase or a change of data.

Some researchers have proposed the use of classification techniques to

solve this problem. In general, standard algorithms have been developed, such as adding incremental induction on training of decision trees [130] and implementing an incremental ruled-based learner [23]. Another idea proposed is to parallelize learning algorithms – such as the *parallel* rule induction [105], the construction of decision trees [16] and association rules [4, 50].

An alternative way is to combine multiple classifiers: this approach is also known as "the meta-learning". [21] proposed to train classifiers on different training set partitions. [15] used a statistical method to combine classifiers with voting, trained on a part of the original training set. The combination advantage is so the system can use many learning algorithms but every single algorithm is independent of the other one. [42] present a parallelization of AdaBoost [39] balancing workload on the "master-worker" strategy. [34] studied two new techniques using AdaBoost to combine classifiers. In the first case, they only choose samples from the complete weighted training set to create classifiers are expected to be "weaker" than one trained from the complete training set. However, boosting can still increase the overall accuracy of the voted ensemble after many rounds. In the second case, they regard the AdaBoost weight updating formula as a way of assigning weights to classifiers in a weighted voting ensemble. With this approach, they have an opportunity to reuse old classifiers on new data sets and learn a new classifier to concentrate on portions of the new data where the old classifiers perform poorly.

These techniques use distributed and online learning techniques, and have been developed on JAM [126], a framework of data mining Agent-based systems. In general, the characteristics of software agents, such as autonomy, adaptability and decision-making, match very well to distributed system requirements, and also to distributed data mining.

2.4.3 AdaBoostM1 Algorithm

AdaBoostM1 [40] is a widely used method, designed for classification. It is a meta-algorithm which uses different classification models according to a learning technique called *boosting* [137].

Let assume that the learning algorithm is able to handle instances with a weight, represented by a positive number (we will review this assumption later). The weighted instances change the way it calculates the classifier error: in this case, error is the sum of the weights of misclassified instances, divided by the total weight of all the instances, instead of the fraction of

misclassified instances. At each iteration, the learning algorithm focus on a particular set of instances, which has the highest weight. These records are important because there is a greater incentive to classify them properly. The C4.5 algorithm [106, 107, 63] is a learning method that can handle weighted instances.

The algorithm starts assigning the same weight to every instance of the training set, then calls the learning algorithm, which builds a classifier and assigns a new weight to each instance, based on the outcome of his analysis: the weight of instances correctly classified will be decreased and the weight of misclassified instances is increased. This produces a subset of "easy" instances, with low weight, and a subset of "hard" instances, with higher weight. In successive iterations, the generated classifiers focus their evaluation on "hard" instances and up to date the weights. It is possible to get different situations, for example, instances could become easier, or otherwise continuously increase their weight. After each iteration, the weights reflect how many times each instance has been misclassified by the classifiers produced up to that point. Maintaining a measure of the "difficulty" in each instance, this procedure provides an effective way to generate complementary classifiers.

How much weight should be changed after each iteration? It depends on the error of the overall classification. In particular, if e (a fraction between zero and one) denotes the error of a classifier with weighted data, then the weights of correctly classified instances will be updated as follows:

$$weight_{i+1} \leftarrow weight_i \times \frac{e}{1-e}$$

while misclassified instances will remain unchanged. This obviously does not increase the weight of misclassified instances, as stated earlier. However, after all weights have been updated, they are normalized so the weight of each misclassified instance increases and that of each correctly classified instance decreases. Whenever the error on training data is weighted equal to or greater than 0.5, the current classifier is deleted and the algorithm stops. The same thing happens when the error is zero, because otherwise all the weights of the instances would be cancelled.

After the training session, we obtain a set of classifiers. In order to evaluate them, there is a voting system. A classifier that performs well on the training set (e close to zero) receives a high mark, while a classifier that performs bad (e close to 0.5) receives a low mark. In particular, the following

equation is used for the assignment of votes:

$$vote = -\log \frac{e}{1-e},$$

which always returns a positive number. This formula also explains why the perfect classifiers on the training set must be eliminated: in fact, when e is equal to zero the weight of the vote is not defined. To classify a new instance you have to add the votes of all the classifiers and the class obtains the highest score is assigned to the instance.

At the beginning, we assumed that the learning algorithm is able to handle weighted instances. If not, however, it is possible to generate a set of unweighted instances by the weighted ones through resampling. Instead of changing the learning algorithm, it creates a new set replicating instances, proportional to their weight. As a result, high weight instances will be replicated frequently, while low weight instances could be not sampled. Once new set of data becomes large as the original, it replaces the method of learning.

Algorithm 1 AdaBoostM1 pseudocode.

Input: $D_1(i) = 1/m,\ \forall\ i$ instances.

for $t = 1 \rightarrow T$ **do**
  Call a *learner* using distribution D_t
  Get back a classifier $c_t : X \rightarrow Y$
  Calculate *error* of c_t, $e_t = \sum_{i:c_t(x_i) \neq y_i} D_t(i)$
  if $e_t = 0 \wedge e_t > 0.5$ **then**
    $T = t - 1$;
  end if
  Set a *vote* of c_t as $v_t = e_t/(1 - e_t)$
  Set $D_{t+1} = \frac{D_t(i)}{N_t} \times \begin{cases} v_t & \text{if } c_t(x_i) = y_i \\ 1 & \text{otherwise} \end{cases}$
  where N_t is a normalization constant.
end for

Output: the *predicted class*
arg $c_{fin}(x) = \arg\max_{y \in Y} \sum_{t:c_t(x_i)=y_i} \log(1/v_t)$

A disadvantage of this procedure is given by the loss of information resulting from the repeal of some low weight instances from the data set. However,

this can be turned into an advantage. When the learning algorithm generates a classifier whose error is greater than 0.5, the process of boosting must end if we use weighted data directly, but if you use a resampled data set, you could still produce a classifier with an error less of 0.5 generating a new resampled data set, maybe with a different *seed*. Resampling can be performed also when using the original version of the algorithm with weighted instances.

2.4.4 Distributed AdaBoostM1-MultiModel

In a distributed environment, mining algorithms, which are placed on every monitor node, create models of knowledge based on their training data. Exchanging models between neighbors they increase their knowledge.

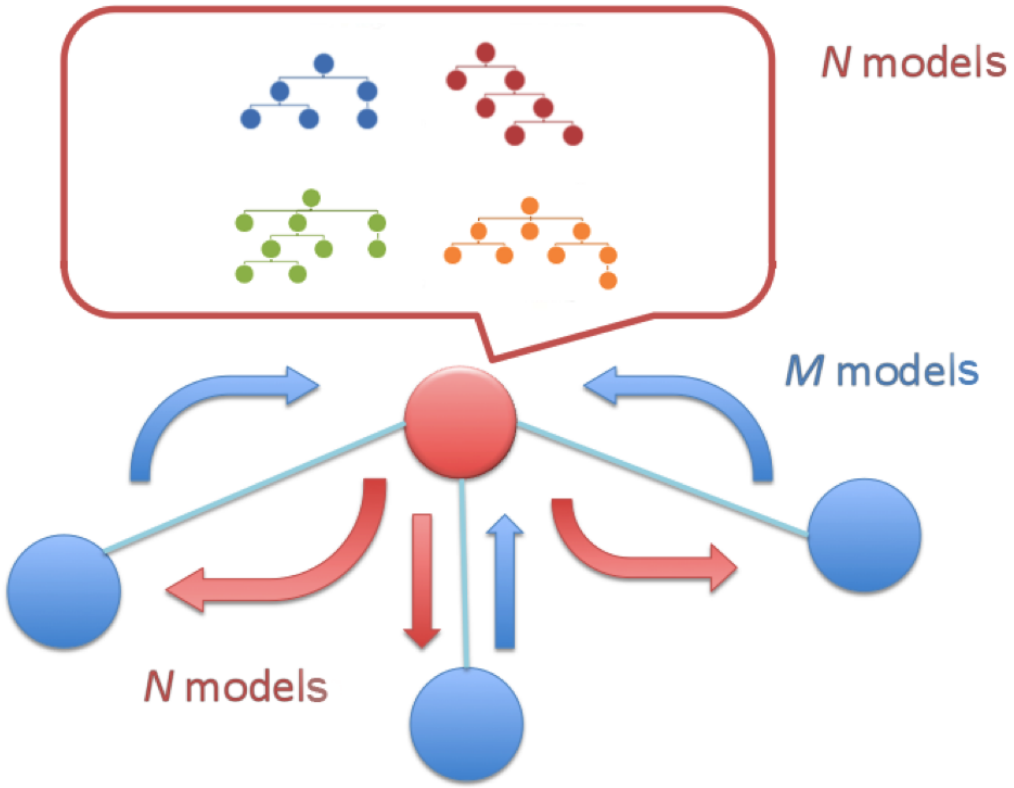


Figure 2.1: Each (red) node runs Distributed AdaBoostM1-MultiModel algorithm spreading all the models to its (blue) neighbors.

During the first iteration of the algorithm, each monitor node runs the AdaBoostM1 algorithm on its training set. The result is a series of classification models – i.e., decision trees (assuming to use C4.5 mining algorithm). In this first phase, each mining engine does not care about their neighbors.

Once all nodes have generated models based on their local data, the algorithm continues with the next phase: knowledge sharing. Each node obtain the result of the previous iteration of every neighbors, the knowledge models previously built according to local data now are going to be changed according to the new neighbors data. In this way, in the global system, there is an exchange of information not in the form of data, but of classifiers, which offer an higher level of abstraction and less network traffic - since a model is smaller than a data set. In this context, however, we do not examine issues related to network communication between nodes, because the execution of the algorithm is simulated in a static way.

After that, classifiers collected from neighbors are added to classifiers generated on the local data to extend the knowledge of each monitor node. This knowledge is evaluated on a the test set of instances. Each test instance receives the class label that gets the most votes from all the available classifiers in the monitor node.

2.4.5 Distributed AdaBoostM1-SingleModel

This section describes a variant of the above algorithm which avoid the multiplication of the classification models on every monitor node. In scenarios where too much knowledge models are shared the Distributed AdaBoostM1-MultiModel algorithm might have issues such as: slowing down operations and a decreasing the accuracy of the results. The number of models in each node increases depending on the number of neighbors, causing a slowdown during the test to evaluate each record. The accuracy decreases as each model focuses on only a few attacks, received during the training of each node, and then those models do not know a particular attack issue a wrong result. When these models are the majority then the node can not detect the attack because good grades are the minority.

In order to fix the described issues we developed an algorithm which shares only a single model to every neighbor. The algorithm shares the best model rather than all the generated ones. The models generation remains unchanged, it follows the classical AdaBoostM1 algorithm. What changes is the sharing phase: when a node asks new models from its neighbors,

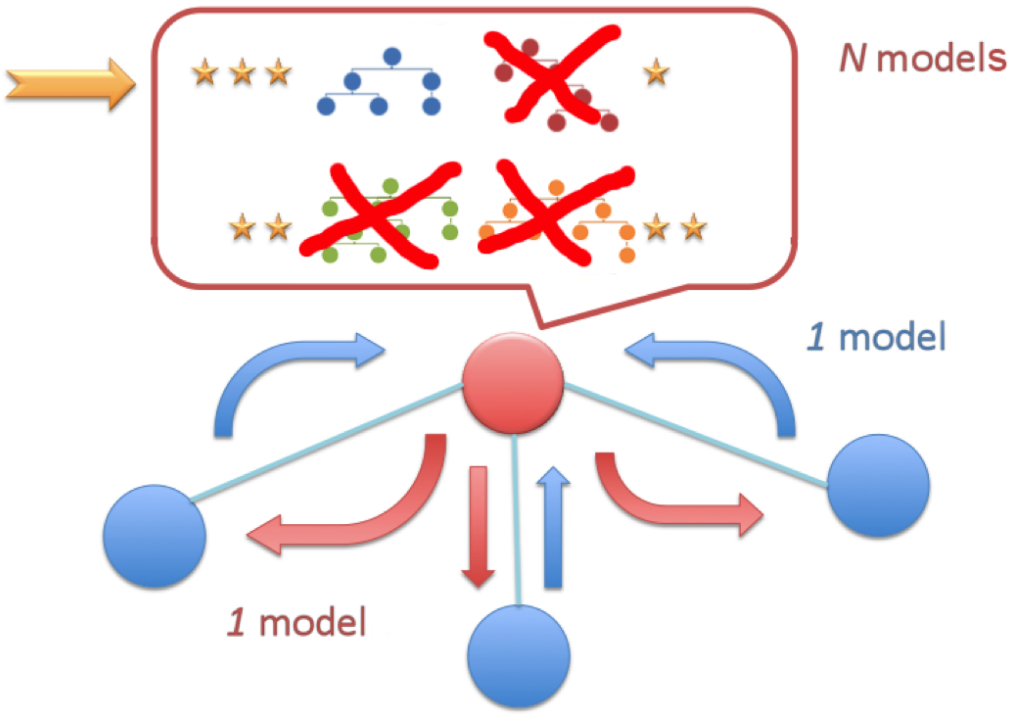


Figure 2.2: Each (red) node runs Distributed AdaBoostM1-SingleModel algorithm spreading only the best model to its (blue) neighbors.

the neighbors share only one model, the one who obtained the best score during the boosting phase. The monitor node itself, during the new instance evaluation, does not consider all the models generated locally, but it chooses the best one among the local ones, and the best classifiers of its neighbors. The evaluation of new instances is done on a shorter list of models, and this reflects an improved efficiency in terms of transmission models to neighbors and during the evaluation of new instances.

2.4.6 Experiments Setup

This section describes the experiments and the obtained results by applying the AdaBoostM1 to SNMP data gathered during attacks and normal network traffic. From fourteen SNMP parameters collected in a given time and during both attacks and normal network traffic, we want to be able to realize

Algorithm 2 Distributed algorithm pseudocode.

Input: m = maximum number of models generated by each node.

```
for each node do
    Generate m knowledge models on local training set with AdaBoostM1.
    for each neighbor do
        Get m' neighbor models and add to its own knowledge base (1 ≤ m' ≤ m).
    end for
    Evaluate test set on its own knowledge models.
end for
```

for the Single-Model variant of the algorithm: $m' = 1$.

whether or not attacks are happening and possibly we want to distinguish between them. We considered a scenario in which a set of monitoring stations, each of which collects raw data on network traffic, control a set of protected machines (i.e. servers, workstations...). The gathered information is provided through SNMP. Each monitor machine has a SNMP agent running to collect SNMP data on a protected machine. The collected SNMP informations are organized into categories in a tree structure, known as MIB (Management Information Base): in this case, we consider the data related to TCP and IP network protocols, such as inbound and outbound packet counts. These MIBs represent the current state of the TCP/IP stack protected machine.

The idea is to analyze the given data through mining techniques to obtain classification models letting us understanding whether or not network attacks are happening. Since each of the monitoring stations collect data from a limited group of machines, it might be useful to gather such a data through a P2P network and use distributed data mining techniques: in this way each station can use the knowledge from its neighbors to expand and complete its own.

We considered 6758 observations as our data set, made with SNMP on a single machine. Each observation consists of the values of the 14 attributes listed in Table 2.1 [19]. The collection of these observations has been divided into 6 sessions of different network traffic conditions, listed in Table 2.2. Data are classified according to the session during which they were collected, so that only one session corresponds to regular traffic and each other to different type of attack. These are well known attacks, recognized also by current IDS,

Table 2.1: Relevant Variables Considered from SNMP Data

Features Derived from SNMP Data
Number of processes in TCP listen state
Number of open TCP connections (any possible TCP state)
Number of TCP connections in time-wait state
Number of TCP connections in established state
Number of TCP connections in SYN-received state
Number of TCP connections in FIN-wait state
Number of different remote IP addresses with an open TCP connection
Remote IP address with the highest number of TCP connections
Remote IP address with the second highest number of TCP connections
Remote IP address with the third number of TCP connections
Local TCP port with the highest number of connections
Number of connections to the preceding TCP port
Local TCP port with the second highest number of connections
Number of TCP RST segments sent out

so the results can be compared with them to evaluate the level of accuracy.

To obtain accurate information from the experiments, it was necessary to perform simulations with different parameters. In particular, we tried as many as possible combinations of network topologies, data distributions, and mining algorithms, to see the trend of results. We considered these groups of parameters: those related to the algorithm, those related to the network topology, and those related to the data distribution. For each simulation, the main considered outcome was the accuracy of classification, calculated as the average percentage of test data, which are correctly classified by each node.

The algorithms used in the experiments are Distributed AdaBoostM1-MultiModel and Distributed AdaBoostM1-SingleModel. As stated above, they are the same mining algorithm (AdaBoostM1) that changes the exchanging knowledge with neighbors. In the first case, all the models generated locally are shared with each neighbor, in the last case, there is only one shared model for each monitor-node, that one got the lowest generalization error on the training set, that is the most accurate model.

Parameters of both algorithms are the following:

- as basic classifier to generate the models we used C4.5 [106, 107, 63], a decision tree algorithm. The confidence threshold for pruning was set at 0.3, and the minimum number of instances in the leaf of the trees was 2;

- the maximum number of models generated during boosting is 10;

- a single iteration was made for the exchange of models with the neighbors. This means that the shared knowledge is limited only to monitor-nodes that are just a "hop" from the current one. However, it's possible to increase the iterations of the algorithm to grow the knowledge base to second level neighbors and above. The results of these experiments will be studied in future work;

- the data distributor allocates to each node all instances of a number of randomly picked classes. Both training and test set are distributed according to this logic and independently from each other: so each node may have one or more classes in both training and test set.

The centralized case, bringing all of the training data set on a single site, with all types of attacks shown in Table 2.2, leads to an accuracy greater than 99%. This case, however, involves considerable cost in terms of network traffic, time and also requires a wide and general knowledge of the network, very difficult in modern systems.

We analyzed instead a case where the training sets are distributed in the network. Each node is trained only on certain attacks, not all, and then tested on different attacks to show how the accuracy changes, betweens knowledge

Table 2.2: Simulated Traffic Sessions

Number	**Description**
0	Normal traffic
1	Denial of Service
2	Distributed Denial of Service
3	TCP Port Scanning using different techniques: FIN, SYN, ACK, WINDOW, NULL, XMAS
4	SSH Denial of Service
5	SSH Brute Force

models sharing cases, according to the two algorithms presented in this work, and without models sharing. In the first case, the improvement is evident in all the graphics, since the case in which nodes are trained only on the half of the attack classes and tested on the other half. The unknown attacks are detected thanks to the neighbors models received.

2.4.7 Simulation Results

Now we show some significant results that have been identified among the many performed simulations. In particular, the x-axis are arranged in order of increasing the number of training sessions and, in order of decreasing, the number of test sessions (6 - x). Training sessions as well as test ones are randomly selected from those available, with no repeats, but it's possible some sessions are in both sets, for each node.

The random number generator used to distribute the sessions at the nodes is controlled by a *seed* set as a parameter, which has been changed ten times, with values ranging from 100 to 109, for each simulation. Each value shown in the following graphs is therefore the average of ten simulations.

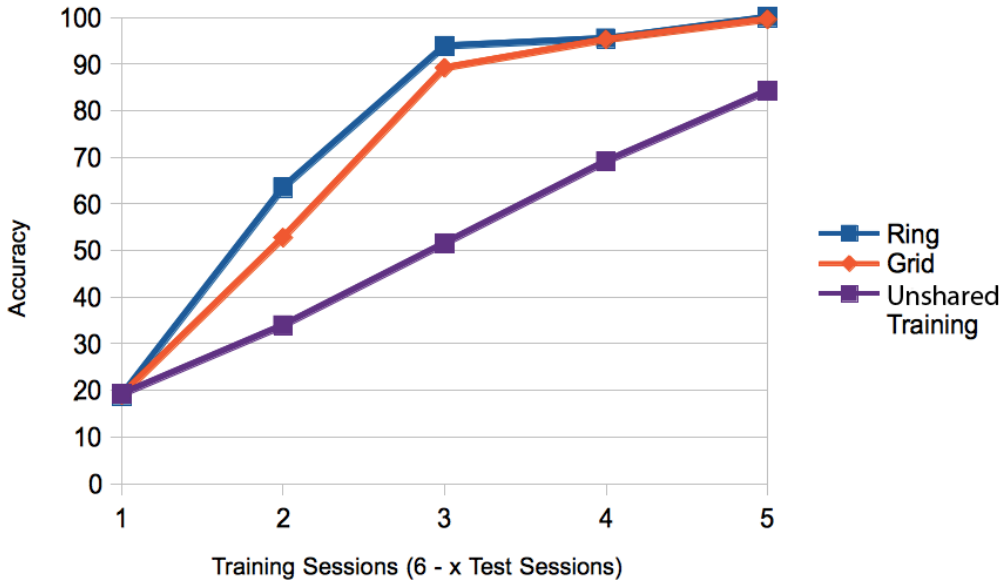


Figure 2.3: Accuracy of Distributed AdaBoostM1-MultiModel algorithm in a Ring Network and a Grid Network both of 64 nodes.

Figure 2.3 represents the trend of the accuracy of the Distributed AdaBoostM1-MultiModel algorithm on a ring network (with two neighbors per node) and a grid network (with four neighbors per node), as the number of training sessions. When the training sessions of each node grow, so the accuracy significantly improves. In particular, when the number of training sessions is three, half of those available, the accuracy is already around 90%. In this scenario, this means it's enough each node knows the half of the attacks to get a good accuracy with the help of neighbors, that is the exchange of knowledge models. However, a slightly better result is obtained in the ring network than in the grid network, so a higher number of neighbors can negatively influence the accuracy of nodes. This depends on the knowledge models shared by neighbors: in fact, some neighbors who don't know certain kinds of attacks provide models that do not work well. Hence, as a future work, it is necessary to find a way to evaluate the goodness of the models coming from the neighbors.

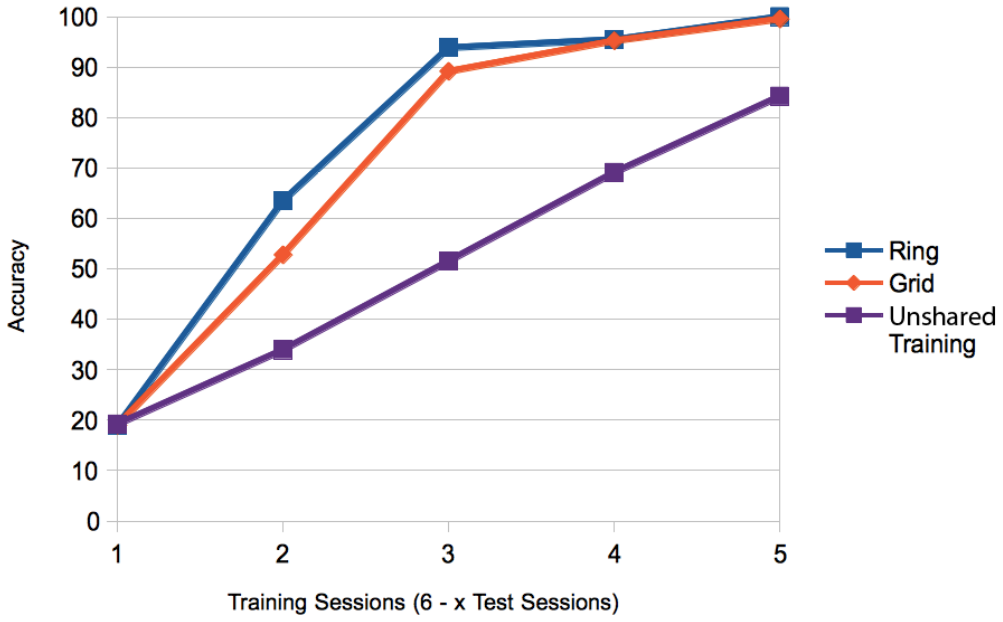


Figure 2.4: Accuracy of Distributed AdaBoostM1-SingleModel algorithm in a Ring Network and a Grid Network both of 64 nodes.

In Figure 2.4 there lines represents the same simulations but with the Distributed AdaBoostM1-SingleModel algorithm: the results are very similar

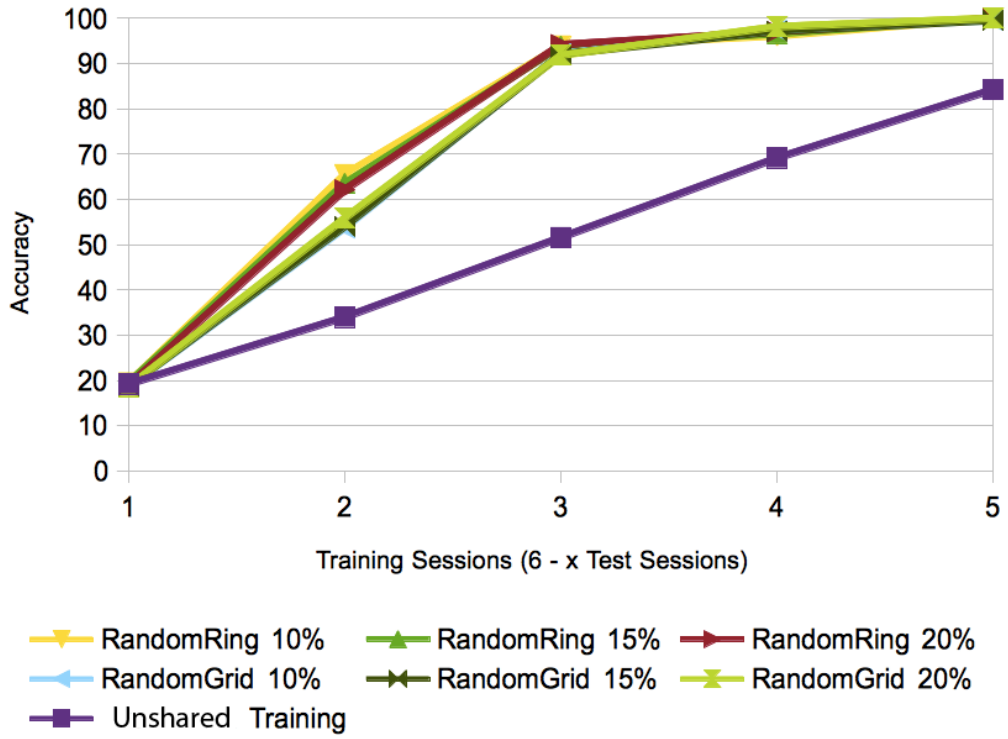


Figure 2.5: Accuracy of Distributed AdaBoostM1-SingleModel algorithm in two types of regular networks (Ring and Grid, with 64 nodes) adding in each one different percentages of random links, respectively 10%, 15% and 20%.

to those of the previous algorithm, but this one have better performance, as less network traffic, because exchanges only one model instead of all, and as faster evaluation of new instances, because each node collects a number of models equal to the number of its neighbors plus one (the best model generated on local data set): *O(log n)* instead of *O(m log n)*, where n is the maximum depth of decision trees and m is the number of models generated in boosting process in every node of network.

Figure 2.5 shows the Distributed AdaBoostM1-SingleModel algorithm dealing with six irregular networks, derived from the previous two regular topologies, namely the ring and the grid. The first three networks come from the same ring network of the previous simulations with the addition of 10%, 15% and 20% of random links between nodes. Added random links are created between nodes previously not connected and to increase the degree

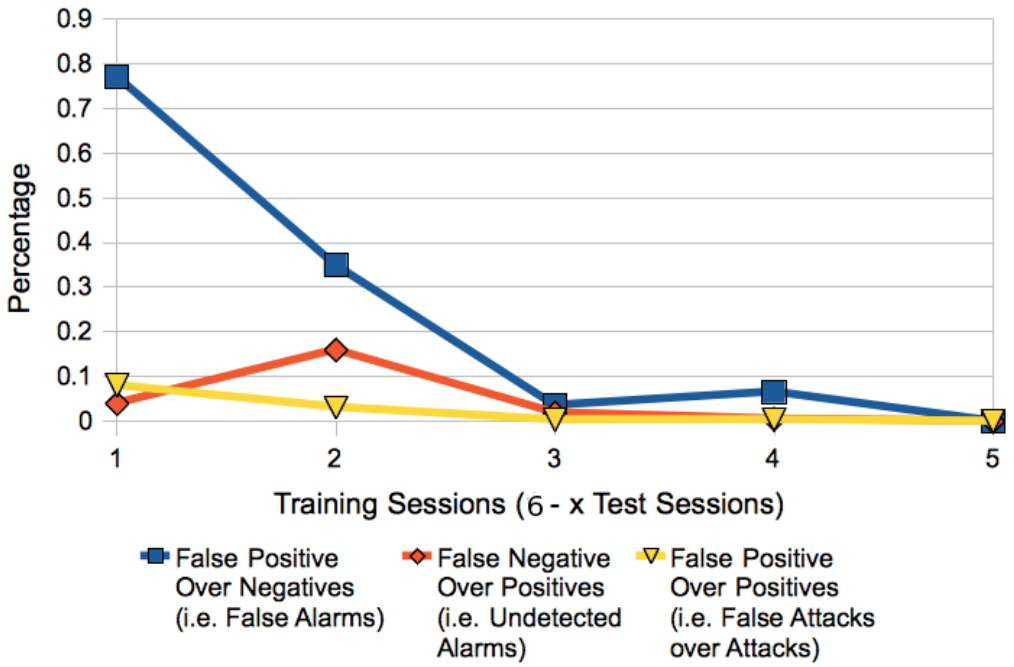


Figure 2.6: False Positive Rates between Normal Traffic and Attacks with Distributed AdaBoostM1-SingleModel algorithm in a Ring Network of 64 nodes.

of network connectivity as the indicated percentage. The result shows the increase of links don't significantly affect the accuracy of the recognition of the attacks.

Finally, Figure 2.6 and Figure 2.7 show the false positive rates derived from the simulations of Figure 2.4 with the Distributed AdaBoostM1-SingleModel algorithm, respectively for the ring network and the grid network. The line of points identified with squares represents the percentage of false alarms, that is normal traffic recognized as attack by the system. When training sessions are few (one or two) there are quite high peaks, especially in the grid network, the peak is close to 4%, due to the fact that many nodes may have never seen the session of the normal traffic, because sessions are distributed casually by the data distributor, and therefore may not be able to recognize it, with the help of neighbors too. On the contrary, as shown by the line of diamonds, the unidentified attacks always remain around 0.1%. The last line of points, identified by triangles, represents the false attacks compared

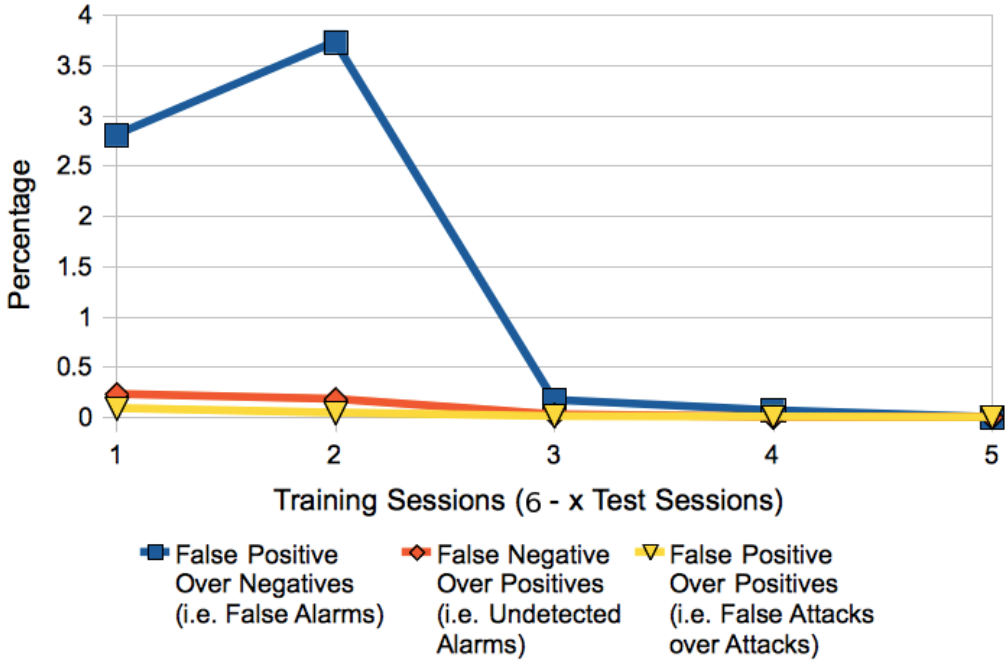


Figure 2.7: False Positive Rates between Normal Traffic and Attacks with Distributed AdaBoostM1-SingleModel algorithm in a Grid Network of 64 nodes.

to attacks detected. If it were high it would means the system is not able to detect enough attacks, therefore is bad. In this case, this line is usually below 0.1%, so false alarms detected by the system are a very small part compared to all correctly detected attack.

Figure 2.8 shows two samples of decision trees generated by the algorithm C4.5 (`weka.classifiers.trees.J48`) implemented in WEKA with the same parameters used in the simulations. The first and largest tree is generated with all the six attack classes, the smaller one instead is generated with three attack classes, in particular "0","2" and "5". We calculate the amount of network traffic generated by our distributed algorithms in the worst case. We consider the size of the decision tree generated by all classes of attack: about 1 Kbyte. The average size of the models however is much smaller because in general it is generated by a smaller number of classes. In the grid network there are 64 nodes, each node has 4 neighbors, and the Distributed AdaBoostM1-MultiModel algorithm shares up to 10 models (decision trees),

so the total amount of bytes flowing in our system, in worst case, is:

$$10 \times 4 \times 64 = 2560 Kbyte$$

We consider also a centralized situation, where all the data are reached by a single node to analyze them and after distribute the generated model to the entire network. We consider the best case to highlight the benefits of our approach. Each node has accordingly the data related to only one attack class: about 1100 of the 6758 data set observations, corresponding in general to 100 Kbytes. To reach a node from any point of a ring network (simpler than a grid network), the data takes a average of $N/4$ hops ($64/4 = 16$). The network has 64 nodes as above, so:

$$100 \times 16 \times 64 = 102400 Kbyte$$

At this number, which is already two orders of magnitude larger than our worst case, we must add the traffic produced by the distribution of the generated knowledge model (1 Kbyte) to every node of the network.

$$1 \times 16 \times 64 = 1024 Kbyte$$

This simple estimate shows the benefit of our work in network traffic, compared to a centralized solution.

2.4.8 Conclusions

We introduced two distributed data mining algorithms called Distributed AdaBoosM1-MultiModel and Distributed AdaBoostM1-SingleModel. Both algorithms and the underlaying framework for the generation of several networks and attack scenarios have been fully implemented as new components of WEKA[3]. Both algorithms work in purely decentralized scenarios where nodes exchange only local knowledge models of few bytes rather than huge amount of network traffic as performed by most of existing solutions. In particular each network node extracts and shares knowledge models from local SNMP data.

The models produced by a node, against a certain type of attack become useful for another node not previously aware of that attack. This allows

[3]Data Mining open source tool developed by the University of Waikato (NZ); http://www.cs.waikato.ac.nz/ml/weka/

nodes to be able to recognize unknown attacks and even to prevent them. The knowledge sharing allow each node to build a more complete knowledge base, compared to that produced using only their local data. The experimental results show that both algorithms can provide accurate classifications, even in case of unknown attacks, moreover the best performance are not far from the ideal solution where all data are centralized in a single machine, which for this reason cannot scale as the network dimension increase.

The algorithms can be extended in several directions that are beyond the scope of this work, such as dealing also with malicious nodes interested in exchanging bad knowledge to reduce the global accuracy or improving the knowledge in order to identify bot-nets as well.

Finally, this work, as well as bringing new contributions in the topic of distributed data mining applied to autonomic security, has been very useful to study various properties of P2P networks, in order to develop the overlays explained in the next chapters.

```
Scheme:       weka.classifiers.trees.J48 -C 0.3 -M 2
Instances:    6758
Attributes:   16

J48 pruned tree
------------------

C_tcpConnectionState_established <= 2
|   C_tcpConnectionState_synReceived <= 2
|   |   tcpOutRsts <= 1: 0 (591.66)
|   |   tcpOutRsts > 1
|   |   |   C_tcpConnectionState_established <= 0
|   |   |   |   C_tcpListenerProcess <= 0: 0 (8.34)
|   |   |   |   C_tcpListenerProcess > 0: 3 (12.0)
|   |   |   C_tcpConnectionState_established > 0: 3 (1040.0)
|   C_tcpConnectionState_synReceived > 2
|   |   2IP_Connections <= 2
|   |   |   1IP_Connections <= 35
|   |   |   |   C_tcpConnectionState <= 110: 1 (24.0)
|   |   |   |   C_tcpConnectionState > 110: 2 (3.0)
|   |   |   1IP_Connections > 35: 1 (1685.0)
|   |   2IP_Connections > 2: 2 (1380.0)
C_tcpConnectionState_established > 2
|   C_tcpConnectionState_timeWait <= 0: 4 (1003.0)
|   C_tcpConnectionState_timeWait > 0: 5 (1011.0)

Number of Leaves  :     10

Size of the tree :      804 byte

Time taken to build model: 0.37 seconds


############################################################


Scheme:       weka.classifiers.trees.J48 -C 0.3 -M 2
Instances:    2994
Attributes:   16

J48 pruned tree
------------------

C_tcpConnectionState_established <= 2
|   C_tcpConnectionState_synReceived <= 0: 0 (600.0)
|   C_tcpConnectionState_synReceived > 0: 2 (1383.0)
C_tcpConnectionState_established > 2: 5 (1011.0)

Number of Leaves  :     3

Size of the tree :      197 byte

Time taken to build model: 0.26 seconds
```

Figure 2.8: Decision tree samples generated by WEKA using C4.5 algorithm (`weka.classifiers.trees.J48`).

Chapter 3

Multi-dimensional Data Indexing for Efficient Routing and Content Delivery in P2P Networks

P2P networks emerged as a computing paradigm for locating and managing contents distributed over a large number of autonomous peers. Autonomy implies that peers are not subject to central coordination. Each peer plays at least three roles, either as (i) a server of data and services, (ii) a client of data and services; and/or (iii) a router to manage network messages. P2P systems realize several of the desirable properties of emergent systems, including self-organization, which provides the ability to self-administer, scalability, which enables support large number of users and resources without performance degradation, and to support robustness, which makes the system fault-tolerant in the event of peer failures or leaving [89].

Moreover, because of the peers autonomy, P2P networks have a comparable behavior to complex dynamic organisms. For example, local changes in the molecular structure of a chemical compound may aggregate to yield altogether a new compound: a global property emerges generally from a series of simple local interactions. The same phenomenon may also occurs in the content distribution as autonomous peers interact independently with each other in a P2P system. Thus, in addition to scalability, our goal is to seek structures for P2P systems which exhibit emergence and self-organization properties characteristic of complex systems, where local interactions and

peers autonomy lead to a global organizational structure with excellent performance features.

In most of the actual P2P structures, the multi-dimensional range query are executed by attribute aggregation. For example, we have a table of song records with three attributes: author, year and genre. The aggregation of these attributes produces a unique key, as "Madonna-1994-R&B". If we follow the order of the attributes in the key for our queries – i.e. we want search all the songs of Madonna or, all the songs of Madonna published in 1994 –, it works well. If we want instead obtain all the R&B songs of Madonna, the key becomes useless and we will check one by one all the keys. This is not acceptable for distributed databases with a huge amount of data.

We present G-Grid as a foundational structure devised to build multi-dimensional indexes even in a decentralized context, and able to support algorithms for data-mining functions, such as clustering, or distributed databases for P2P networks or, in business distributed environments, such as server farms for grid, cloud computing and big data domains.

G-Grid lets us execute efficiently multi-dimensional range query. Furthermore, in G-Grid data can be distributed easily on various peers and with a good degree of storage resources distribution. The structure evolves in a totally independent way, so we can use G-Grid for P2P applications development because each peer does not need a global knowledge of the network. The shape is not imposed a priori and the interactions among peers increase the peers knowledge on the topology of the network. The performance does not deteriorate with increasing number of peers, so its scalability helps to build large data sources. Since G-Grid is a flexible and suitable overlay for P2P environments, we aim to create a large distributed database in a dynamic context.

Today, most databases are centralized and placed on powerful servers. In a context where the input/output of peers is not frequent, for example the server farm of a company, we can deliver on some small computers the company's database via G-Grid peers. This could reduce the required number of centralized and dedicated powerful servers, which do not use most of the computational capacity. The new concepts introduced by G-Grid lend themselves well to the growing use of virtualization techniques. We could not spend or invest in centralized powerful servers to handle a whole database and deliver it directly on many client computers to reduce the resources dedicated entirely to a single database. In fact, all these machines provide a storage capacity much larger than a main-frame and also do not waste their

resources, but rather they will use the most of their entire computational capacity.

3.1 Related works on P2P Data Structures

Content to be shared in P2P systems can be conceptually represented as a single relational table, with multiple data attributes, horizontally partitioned (distributed) among peers. Differently from traditional distributed databases, in P2P systems there is no entity which is aware of the global distribution scheme. This means that P2P systems' actors must efficiently cooperate any time data must be edited (e.g. inserted/updated/deleted) or queried. The efficiency of P2P system is evaluated according to the following parameters: complexity in terms of hops per data editing/query, complexity in terms of number of links per single node.

First generation P2P systems, also named *unstructured P2P systems*, such as Gnutella [56, 96] and its descendants (e.g. Kazaa and Morpheous) provided a valued service for many users but their routing mechanism, which was based on message flooding and a time-to-live parameter, could easily congest the network in case of data intensive applications, due to their exponential costs in terms of routing hops. This problem, together with the fact that query completeness was not guaranteed, allowed hybrid systems such as WinMX and Emule to take the scene. In those systems P2P only takes place during file download, while several servers are in charge to manage and update a catalog of the shared content. The presence of centralized servers made it possible to shut down WinMX for copyright issues.

Researches in the field of distributed systems brought to the development of structured systems which allow query completeness at logarithmic costs. Building on previous work in uni-dimensional distributed data structures, such as RP* [75], LH* [76] and ADST [31], several new approaches were proposed. The most famous and cited are Chord [124], Tapestry [150, 149], Pastry [114] and P-Grid [2]. These new systems did indeed improve performance and extended the flexibility of search by allowing querying by content. All these systems require $\mathrm{O}\big(\log n\big)$ hops and $\mathrm{O}\big(\log n\big)$ links per node. Viceroy [80], FissionE [70], SONAR [118], SKY [148] and Moore [49] achieve $\mathrm{O}\big(\log n\big)$ hops with $\mathrm{O}\big(1\big)$ links but have restricted or absent load balancing capability.

Most of the cited systems are based on Distributed Hash Table (DHT),

where keys and data are stored in the nodes of the network using a hash function. The use of a hash function limits those systems to single-attribute queries, restricting thus the range of possible of applications in a P2P environment. Besides, the lack of locality brought by hash prevents those systems to efficiently support range queries. In Family Tree, [144] solve the problem of designing an ordered, distributed structure with $\mathrm{O}(1)$ links and $\mathrm{O}(\log n)$ performance for search and update operations without using hash, and therefore allowing efficient range querying. However the resulting structure usage is limited to single dimension spaces.

Recently decentralized routing and data management problems in ad hoc networks have been deeply investigated. Some authors have contributed to the development in sensor networks of approaches typical of database systems, such as solutions based on multi-dimensional distributed indexing like DIM [73], PRDS [140], and also with distributed hashing like DIFS [47] that facilitates range searches on a single key. These solutions share some peculiarities, in fact they organize the sensed data in the network according to structures typical of database systems, in particular using structures capable of indexing multi-dimensional or multi-attribute data. In DIM and PRDS, each sensor is an index node, thus the number of index nodes in the sensor network depends on the size of the network. P2P data structures evolved in ad-hoc, sensors, wireless, smart cities networks. However, these applications differ by the aims of G-Grid because they have constraints of physical proximity between nodes, and a limited availability of energy (e.g. devices with rechargeable batteries) and computing (e.g. processors smaller and less powerful than those of data centers). In addition, these networks must first build the transport layer by physical routing because there is already not a P2P existing network, as for example TCP/IP.

In the real world many P2P applications require richer query semantics, involving several attributes, comparable to those available in centralized relational DBMSs. Multi-dimensional structures have been extensively investigated over the last 20 years where the main goal is to support efficiently complex range queries over multiple attributes. A literature survey in this area is available in [41], while two specific structures, IBGF and NIBGF, have been presented by [97, 99]. These structures have been designed for environments where both control and data are centralized, and significant performance improvements have been achieved for both partial and complete range queries. In centralized systems, range queries over a set of attributes may be processed using single-attribute structures with acceptable performance,

despite the large number of local accesses to disk. The same queries in P2P systems turn out to be singularly cost-prohibitive, unless a multi-dimensional structure is available, as each local data access will now give rise to several network messages. Dynamic pure P2P networks will naturally amplify the severity of the costs because of continuous changes in the content, its distribution, and the underlying network topology. G-Grid starts from IBGF to allow distribution of multi-dimensional data among peers in a network. Moreover we develop the managing of the peers autonomy and the dynamic evolution of the overlay, i.e. join and leaving of peers.

RAQ [94] can handle range-queries in multi-dimensional space, supporting such queries in at most $\mathrm{O}(\log n)$ hops and requiring $\mathrm{O}(\log n)$ links per node. Like us, it works by splitting the data space into regions and maps those regions in a binary tree. The main differences from G-Grid are that in their solution each partition (region) holds a single data and that nodes are only responsible for leaf regions (there is no nesting at regions).

Skiptree [6] is a scalable distributed data structure that allows storage of keys in multi-dimensional spaces and the execution of both exact match and range queries. It uses a distributed partition tree as well, partitioning the data space into regions and assigning leaf regions to network nodes. Differently from similar tree-based solutions (like ours or RAQ) the partition tree in Skiptree is only used to define an order relationship which is then used for the routing mechanism and link maintenance. Routing and links are therefore independent from the shape (and the possible unbalanced shape) of the partition tree. Skiptree maintains $\mathrm{O}(\log n)$ links at each node and guarantees an upper bound of $\mathrm{O}(\log n)$ messages for point and range queries. This result is also get by [147].

Among the most recent literature [145, 134, 48, 33], SkipCluster [142, 141] is a hierarchical P2P overlay network. SkipCluster is derived from Skip graphs [8] and SkipNet [52], but it has a two-tier hierarchical architecture. The basic idea is to group the peers in the cluster, according to two levels: in the low level there are peers in a cluster, seen as a single entity, while in high-level clusters are considered as atomic entity that are connected to other clusters. A cluster is a set of peers which have in common the most significant part of the peer identifier. Each cluster has a manager peer called "super-peer". The super-peer is responsible for the connections between its cluster and the others, and for maintaining the routing table of the peers inside its own cluster. Therefore, the super-peer is the only access to the cluster outside. In this overlay, the average number of hop grows logarithmically

with respect to the number of network peers.

We compare our architecture with SkipCluster because it is capable of supporting both exact-match and multi-dimensional range queries, as G-Grid.

3.2 G-Grid

G-Grid is a distributed multi-dimensional data structure. The earlier proposals for P2P structure, such as Chord[124], Pastry[114] and P-Grid[2], handle only one dimensional data, but this limits the query expressiveness. This is even more evident in unstructured P2P implementations, which predominantly are able to perform only exact-match queries.

G-Grid aims to improve the query expressiveness, but at the same time seeking to rely on a robust system, and providing efficient routing protocols and quick searches. At the moment, it is necessary to make available the major DBMS properties in distributed P2P environments, trying to offer a query expressiveness close to SQL. G-Grid moves towards this direction, introducing a method to index multi-dimensional data and providing the basis to perform more sophisticated queries, such as range-queries in hyperspaces and join.

G-Grid can be used in application domains such as clustering, or distributed databases for P2P networks or, in business distributed environments, such as server farms for grid and cloud computing and big data domains. For example, G-Grid could be useful in a datacenter where each peer is a server that manages hundreds or thousands of regions of the data space. Each peer manages portions of the G-Grid tree, even not contiguous ones, and is linked to the other peers according to the G-Grid overlay network. The G-Grid structure creates a distributed index that leaves data in their nodes and allows distributed search such as exact match and range queries, involving an arbitrary number of attributes (multi-dimensional). All of the operations above are executed with a logarithmic routing cost from one server to others with respect to the number of peer in the overlay network.

3.2.1 Structure and Features

The data space has many dimensions as data attributes. G-Grid splits the space according to the positioning of the data in regions, and the same data

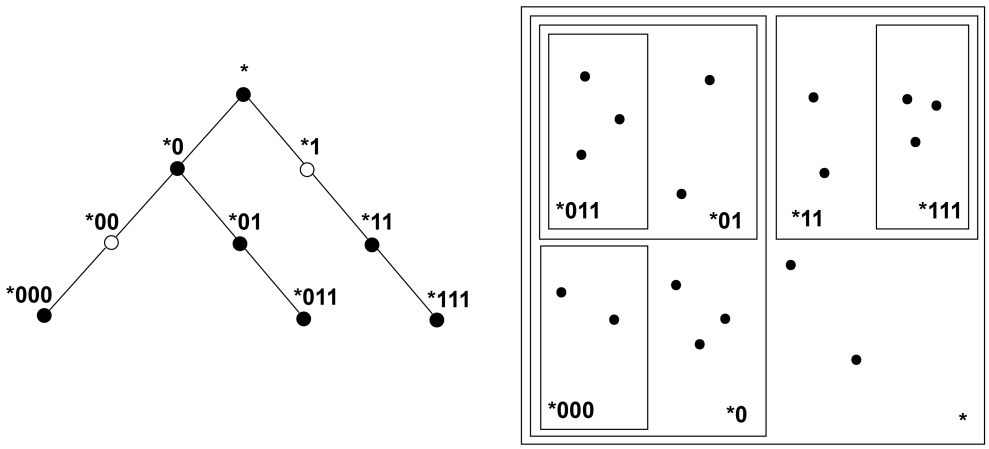


Figure 3.1: A G-Grid partition example in the binary tree representation of the partition and in 2D space.

are placed in a tree structure (see Figure 3.1), where each node represents a region, or a portion of the hyper-space. The tree is binary, each node either is a leaf or has two children. The children represent regions that are subspaces of the parent node region. The root node represents the entire hyper-space and every descendant node is a subspace of it.

The G-Grid binary tree has two main properties:

- **Space property**: all the regions located at the same level of the tree never intersect.
- **Coverage property**: given two regions, if one is included in the other, then they have a family relationship in the tree.

The hyper-rectangle in which the data are placed is limited, particularly for each dimension D_i we have $[minD_0, maxD_0[, ..., [minD_n, maxD_n[$. Each region is bound to have a minimum and maximum quantities of elements in it, in particular the maximum is called *bucket-size*.

When a region exceeds the number of elements, it is halved with a *split* operation, whose final result can be seen in Figure 3.2. On the opposite, if the objects in a region are less then 1/3 of the bucket-size the region collapses into the parent node, but if, after this, the parent node exceeds the bucket-size, then this re-runs the split. At the end of these operations, each region must respect the constraints imposed on the quantity of items. The

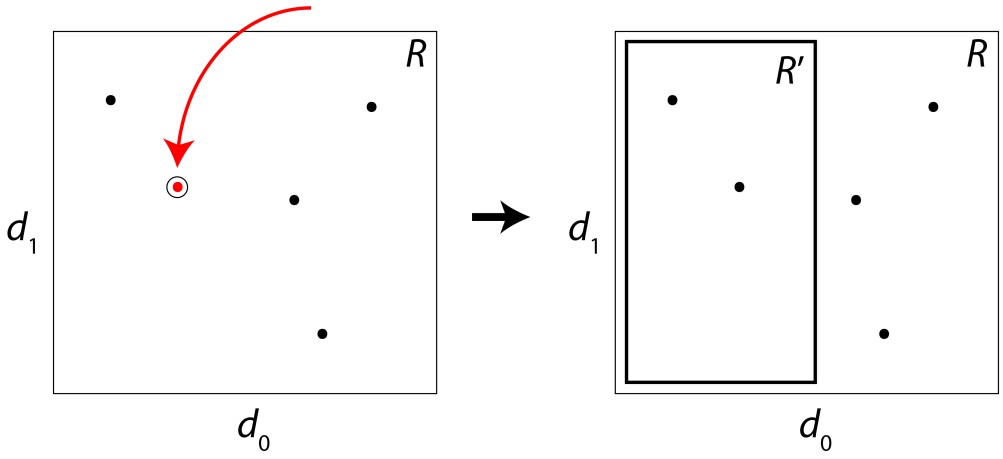


Figure 3.2: Effect of a split operation on a region.

split operation cuts the region by half and this is performed on a dimension selected cyclically from all the dimensions of the data. For example, if we have a three–dimensional space, X, Y and Z, the first split is performed on the X, the second on the Y and the third on the Z, then it starts again from X, and so on. The splits run recursively until the bucket-size constraint is satisfied. Starting from the root node, the regions are increasing or decreasing depending on the input/output of data in G-Grid, so the tree grows or shrinks dynamically.

Similar split mechanisms have been introduced since 1970 with data indexing structures for centralized databases [98] and ensures that in each region of G-Grid there are at least 1/3 of bucket-size data. The experimental data show that regions tend to be occupied on average for about 2/3 of the bucket-size. This result can positively influence the P2P, in fact, if we assign one or more regions to each peer, the data load is distributed in a uniform way and this is a desirable property in P2P systems. This property remains valid regardless of the bucket-size value.

Each region is uniquely identified by a binary string G. $Length(G)$ is the function that returns the number of digits of the binary string, representing the depth of the region in the tree. This number indicates also how many split operations have been executed to create that region. The size of this subspace, relative to the entire data space, is $1/2^{Length(G)}$. G indicates the position of the region in the space as well as the location of the node within

the tree. The binary string G is given by "*", representing the root node, followed by binary digits as many as the depth of the region in the tree. As we already mentioned, the tree is binary then each node, except the leaves, has links to two children. Starting from the root node, we traverse the tree by consulting the binary string G. Starting from the most significant one, each digit provides the way to go inside the tree: more precisely, 0 move to the left child and 1 to the right one. If we analyze the data space, a region that splits is divided into two regions: recalling Figure 3.2, the first region is the original one on which we made the split and represents the parent node in the tree, the second instead is a subspace and the son of the previous node. If the child occupies the left side of the cut it is the left child node and its binary digit is 0, otherwise it is 1.

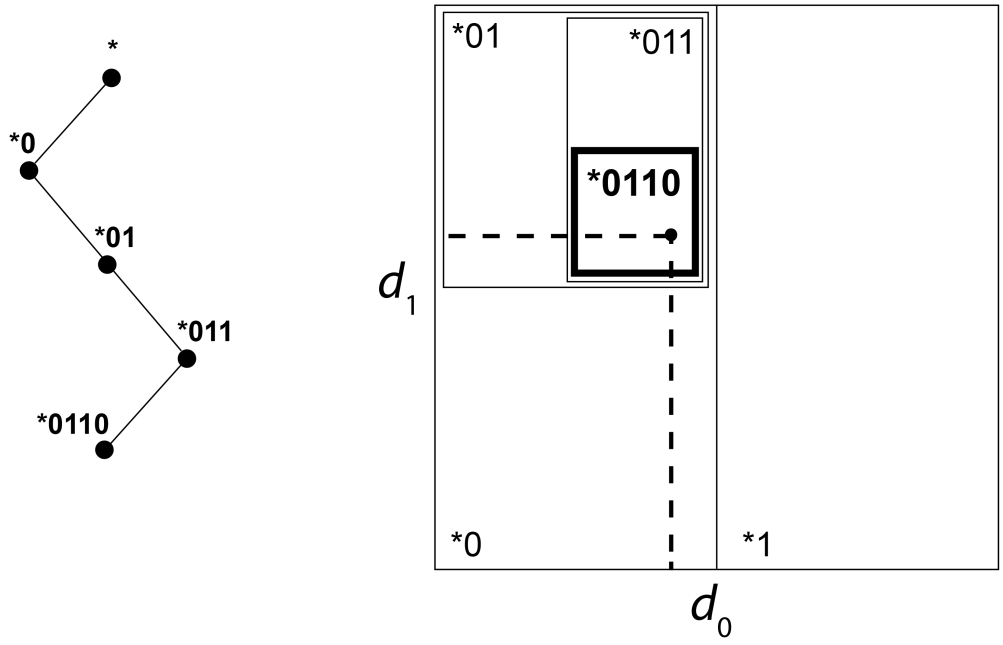


Figure 3.3: Location of a point in the binary tree and in the hyper-space of G-Grid.

To find out where a data record is placed in the hyper-space, we have to create the binary string G of the region in which it is, according to the value of the data record attributes (see Figure 3.3). The candidate region is the one that has the longest common prefix between the region G and

the binary string of the data record. Each region G shows a very specific area within the hyper-space and, in particular, it has a size proportional to $1/2^{Length(G)}$, hence increasing the depth in the tree we increase the granularity of the key location and for $Length(G) \to +\infty$ geometrically we get a point. G is arbitrarily set by the node issuing the query/insertion/elimination of data, then if $Length(G)$ is shorter than the target region depth, the last node that receives the query message, if it has children nodes, recalculates the query region G with the query message attributes and go on. In fact, if we consider a record p on a node X that is not a leaf, the insertion checks whether there is a descendant node of X that might be a better candidate to contain p, perhaps recalculating the G of p with a length value consistent with the depth of the tree explored at that time. The tree is always explored recursively, and if we analyze the scenario in a P2P environment, where each peer is responsible for at least one region, this would have connections with its parent and its sons.

The insert and remove query walk through the tree and each peer along the walking receives the request from its parent, evaluates if the binary string of the data record is in its region, recalculating the binary string since the depth of the tree if it can not know it a priori, and if that fails it forwards the request to both the children. This recursive mechanism shows how this structure works well in P2P environments, because each step requires the interaction of only two entities: parent and son nodes. Regardless of whether working in a distributed or centralized structure, basic operations are performed exploring a binary tree and the only difference is that in distributed environment this tree is split into parts assigned to autonomous nodes.

The advantage of the G-Grid structure is that it allows to perform insertions, deletions, and queries in time proportional to $O(\log N)$ where N is the number of regions. The number of regions in a binary tree of height m is 2^m and the maximum number of hops that a message needs to reach a target region from a source region in this tree is $2m$, so this number is logarithmic with respect to the number of regions in the tree. In addition, with the learning mechanism which will be explained later, this time can be further reduced to become sub-logarithmic. In other words, the efficiency of DBMSs is exported to distributed environments by virtue of the limited number of hops[1] required to complete any operation.

This is a fundamental property that can be found in DBMS and that in

[1]The number of gateways that separate the source host from the destination host.

the field of distributed avoids to make too many hops to process a distributed query, while reducing both the network traffic and the response time.

3.2.2 G-Grid in P2P Environments

After describing the structure from a general point of view, we show the advantages that this structure brings in distributed environments, and in particular focusing on scenarios in which participants are autonomous self-organizing agents, as in P2P. When distributing the binary tree of G-Grid into the peers, each peer is associated with one or more regions, i.e. one or more nodes of the binary tree. Therefore each peer is responsible for several portions of the hyper-space. The assumptions on the environment are the following: a physical network connects peers, each having a unique physical address, and each dimension of the hyper-rectangle of the data space is limited. If a peer decides to participate in the G-Grid overlay and can't contact any other peer within the network, it will take the region of the root node, which controls the entire hyper-space. When a participant agent (peer) is starting, it hasn't any reference to other nodes to communicate with, so we have to provide for the creation of a common boot-strap list with some valid contacts.

Under these assumptions, we ask the following question: why G-Grid is well suited for P2P networks? By analyzing the evolution of regions of the tree, we observe that it expands and contracts dynamically depending on the data inserted and removed, and the only operations taken on the regions are split and collapse. These two operations require the involvement of at most two nodes, just what decentralized self-organizing networks need. P2P networks evolve dynamically over time and this affects the positioning of regions on the nodes, but the movements and interactions always involve no more than two peers. For this property in G-Grid each node does not need a comprehensive knowledge of the network and of how the tree is shaped, but only portions of local information. Moreover, as we already mentioned, G-Grid is able to load the regions for 2/3 of the bucket-size in average and with a low variance, so the amount of resources that the nodes have to assign to data is well distributed among the peer. In G-Grid not all peers hold regions, in particular there are two types of peer:

- **C–Peer** : Client Peer,
- **S–Peer** : Structure Peer.

Both can do the same operations on G-Grid (query, data insert and remove), the difference is that S–Peers routing table is more sophisticated and they implement concretely the services, C–Peers instead perform their operations via S–Peers. Essentially an S–Peer has the same functionality of a C–Peer, but it has the resources and mechanisms that implement the G-Grid distributed data structure. An S–Peer holds one or more tree fragments and nodes of this structure pointing to specific regions, and is responsible for their maintenance. The nodes can be connected to other regions held by S–Peers, in this case we put a node pointer in the tree (see Figure 3.4). The S–Peers must keep the connections to their remote parent and sons up to date and consistent. These structural links, shown in Figure 3.4, build the paths that evolve over time in the P2P network.

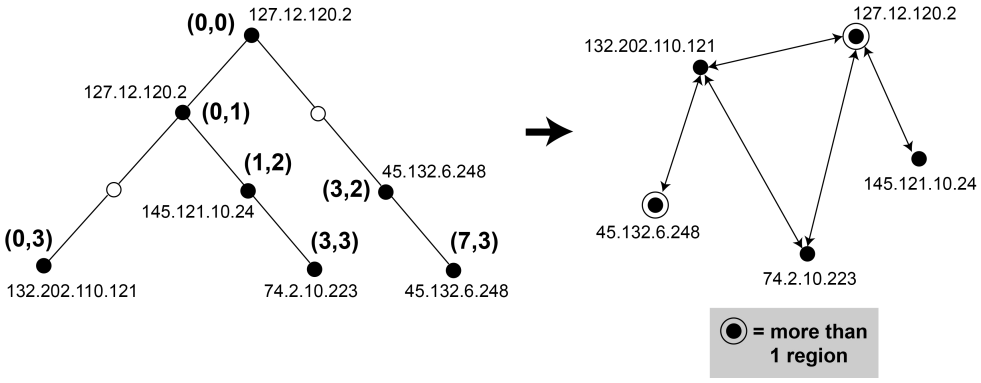


Figure 3.4: Internal link structure (left) and network path (right) of an S–Peer tree

To load the data on the network, each S–Peer has a maximum number of regions that may hold and if it exceeds this threshold, the S–Peer delegates some regions according to the decreasing size, through a C–Peer contact list accumulated over time. When a peer wants to be involved in G-Grid network, it enters as C–Peer, then it can become S–Peer through the delegation of at least one region by a C–Peer or S–Peer that exceeds the threshold. When an S–Peer wants to leave the network, before disconnecting, it delegates all its regions to a C–Peer or to an S–Peer that have relationships with the regions to acquire. This mechanism requires the involvement of only two entities, thus preserving the full autonomy of the peers and the completely decentralized network evolution.

Each peer, regardless of the role it plays in the network, has a routing table, that is a list with references to other peers, pairs region / physical address (G, IP) for the S–Peers, IP only for the C–Peers. The filling of this table occurs when the peer contacts or perceives the existence of other peers. For example, when a peer receives a request the message contains some information about the sender. Every peer doing a request does not know a priori the corresponding node and this is due to the fact that messages are relayed through multiple intermediaries in the network and also if this were not true, the peer should possess a comprehensive knowledge of network that is not pragmatically possible on Internet. For this we can say that each peer feels only the existence of a part of the whole network, but this view often does not fully reflect the reality because the entities may leave the network, or not answer as a result of faults. Therefore, the more requests are performed on the network, the more knowledge about the network is acquired by the peers.

But how can we derive new knowledge at each request? In G-Grid the learning mechanism provides to sender and receiver information about all the intermediate nodes. This works as follows: each intermediary attaches his reference on the message, in this way also the intermediaries increase their knowledge about the network, even if the first node receiving the message learns less than those closest to the receiver. This solution is possible and scalable because in G-Grid the number of hops to deliver a message is sub-logarithmic with respect to the number of nodes so the message will not reach a size large enough to waste the bandwidth of the communication channel.

Analyzing the routing table of a C–Peer, we find IP of C–Peer and S–Peer, the formers are useful in the future if the entity will be transformed into S–Peer and the latter act as an access point to the G-Grid network for services. Why do we find this configuration in S–Peer routing table? Is it not enough that they hold only structural links? These are only a subset of the entire routing table because while nodes are querying there is a learning process in the network. These references may be useful as a form of redundancy but they also have an additional purpose. We stated that in G-Grid the transactions perform at most a logarithmic number of hops, with respect to the number of S–Peer. Let us consider what happens when a node wants to contact the handler of a certain area of the hyper-space, due to its intrinsically limited view of the entire binary tree, it starts scanning. In the worst cases it can start to scan the entire hyper-space, or can forward the request to the root node, who has a vision of the entire hyper-space and if the

required area is not in its region, then the root node will forward the request to the child who has assigned the required region. The child recursively performs the same operation, which will end when the message reaches the required region. This process is made possible by the fact that each node can contact the root node via structural links, going up the tree through links to parent nodes. However, this path can be reduced considerably thanks to the learning mechanism: to perform an operation, which we do not know a priori who will exactly involve, a peer may contact the S–Peer that has a region that is a superset of the requested area. This concept can be exposed as trying to find the S–Peer connection that holds the G with the prefix larger than the G of the request data record. For example: if a peer holds as S–Peer connections the root node * and nodes *010 - *1001 - *01011, if the request is *010110010010111, the peer does not send the request to the root but involves directly the node *01011, which is a superset of the requested region and either it holds that specific area or one of its descendants does. This mechanism reduces the number of hops with the increasing network knowledge owned by the peer. This advantage is opposed by the entry/exit of participants in the G-Grid network, which are seen as disruptive actions on the entire system.

3.2.3 Performance Analysis

Once the network has settled, the number of records stored on average by regions tends to the value $2b/3$, where b is the bucket size. This value was determined empirically. The simulation manages exact match queries and record insertions and incorporates both the region splitting mechanism and the learning capability. Moreover it can be configured with some parameters, such as the region bucket size b and the rate of insertions with respect to queries $\frac{insertions}{queries}$. In the experiments the structure evolves and dynamically grows starting from one peer and by generating operations randomly. In essence, G-Grid is a stochastic system with complex behavior where each state of the system depends on the preceding one, but the set of the possible states evolves dynamically over time growing very quickly and making hard any analysis based on traditional formal methods, such as Markov chains.

However we discuss in some detail the split probability, which can help predicting the growth of the system and finding important theoretical results related to the *average path length* (APL) to deliver any message in the system.

For space reasons we present here only the simplest version which well

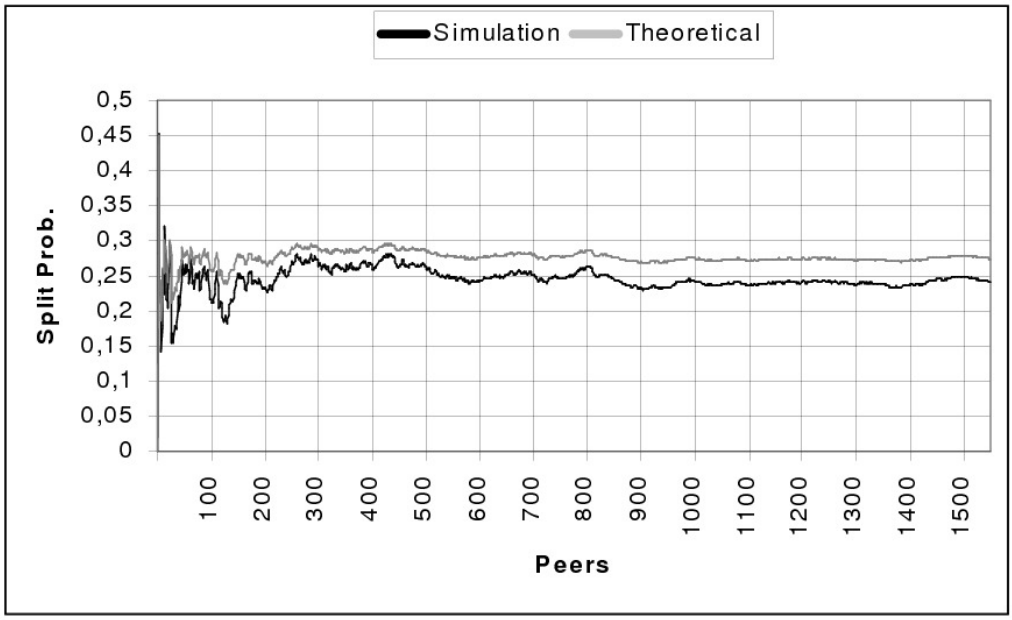


Figure 3.5: The approximated theoretical split probability compared with a simulation

approximates the above-mentioned probability for very low values of the bucket size.

Split Probability Definition: let t be any instant in time in the life of a G-Grid structure G and let us denote the following variables:

- N_t = records at the instant t randomly distributed in G
- M_t = regions/peers in G at the instant t
- b = the region bucket size

then the split probability is the probability that a record insertion at time t happens in a region already full with b records, namely:

$$P_s(N_t, M_t, b) = \frac{1}{f_{max} - f_{min} + 1} \cdot \sum_{i=f_{min}}^{f_{max}} \frac{i}{M_t} \tag{3.1}$$

where f_{min} and f_{max}, which are the minimum and maximum number of full regions respectively, are the following:

$$f_{min} = max(N_t - M_t \cdot (b-1), 0) \qquad f_{max} = floor\left(\frac{N_t - \frac{b}{3} \cdot M_t}{\frac{2}{3} \cdot b}\right) \tag{3.2}$$

Figure 3.5 illustrates a numeric comparison between the formula 3.1 punctually measured and an experiment conducted with $b = 6$ where the system grows up to more than 1500 peers. The empirical curve is simply the ratio of the full regions number and the total regions number in G-Grid. The formula 3.1, as we saw before, only requires the total number of regions. The two split probabilities oscillate until there are less than 50 peers, then both of them stabilize with a difference around 2.5%; in all experiments the stabilization occurs always independently on the rate $\frac{insertions}{queries}$. When new records come in, the split probability increases because the regions begin to saturate. When some regions split, because they have a number of records greater than the bucket size, the split probability decreases. Once the network has settled, the probability of split tends to the value $2b/3$, where b is the bucket size. Experiments have also confirmed an average *storage utilization* per region equals to $2b/3$, that is the number of records stored on average by regions.

The rate $\frac{insertions}{queries}$ instead is determining for the APL in the system, in fact if the rate remains constant the APL tends to grow, but less than logarithmically with respect to the number of peers. This effect is due to the learning capability which reduces the logical distances in the system by creating new links. Finally, we verified experimentally that if the rate $\frac{insertions}{queries}$ changes like $O(\frac{1}{M_t^2})$ then the APL in the system tends quickly to 1 (see Figure 3.6). The insertions cause an increase of APL as the network grows and each peer reduces its global knowledge of the network. The queries do not change the network, but only increase the knowledge mentioned above. If the number of queries grows quadratically with respect to the number of peer insertions then the probability that G-Grid, thanks to learning mechanism, generates a complete graph (e.g. each node is connected with all the others) tends to 1 when the number of nodes tends to infinity. In several realistic scenarios the number of queries is more than quadratic with respect to the number of users/machines, for instance in the World Wide Web each user access may originate in cascade many messages and queries. Also the execution of more traditional queries, such as joins and range queries, can generate that number of requests.

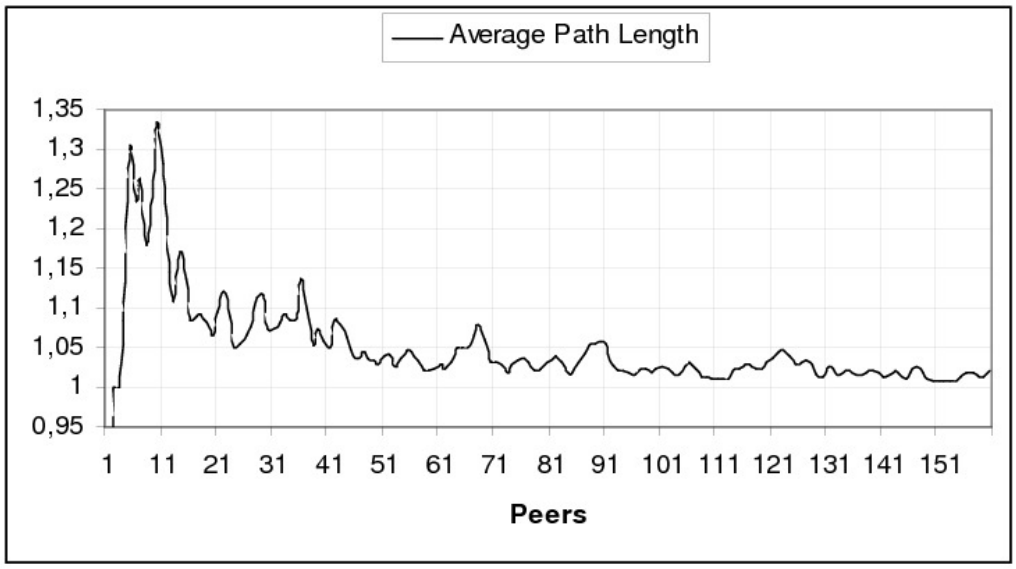


Figure 3.6: When $\frac{insertions}{queries} \approx \frac{1}{M_t^2}$ the average path length to deliver messages tends to 1

3.2.4 Robustness

The G-Grid allows peers to connect and disconnect autonomously from the structure. C-peers can connect and disconnect without an impact on the overall G-Grid structure, except perhaps that responses to their already initiated requests will not have a return address. On the other hand, a non-anticipated disconnection of an S–Peer may make the S–Peer's objects and local routing table inaccessible. Thus, it is important that S–Peer return local information to the system through a non-catastrophic disconnection.

An **orderly disconnection** is one where an S–Peer hands over its content to another peer to preserve data accessibility and routing information. There are two possibilities: (i) merging its local information (both objects and routing table) with its parent S–Peer or with a child S–Peer., or, (ii) requesting a C–Peer as a replacement. The choice is determined based on policies which will enhance the overall performance of the system. These policies must be selected and developed during the implementation of the system.

A **disorderly disconnection** of a S–Peer occurs in catastrophic situations such as computer crashes or physical network problems. Objects in the

S–Peer become inaccessible and routing through the S–Peer is no longer possible. Depending on the network topology, a disorderly disconnection could cause the G-Grid partitioning into two disjoint component. To enhance robustness of the system, one approach is through duplication of information in S–Peers. Besides the associated information integrity problems, duplication does not eliminate the problem of G-Grid partitioning, it only reduces the likelihood of its occurrence. Our approach is to avoid altogether duplication and rely on the learning mechanism of the system, which establishes incrementally links between the various S–Peers as the level of interaction increases, and thus provides multiple routes to get to S–Peers. To what extent does the learning mechanism reduce the likelihood of G-Grid partitioning? Our preliminary experimental results show that the likelihood of partitioning is practically nil. What will be the effect on performance and availability of data in combining both duplication and the learning mechanism? Answers to these questions require an extensive robustness analysis, which we intend to do in the future.

After a disorderly disconnection, an S–Peer may rejoin the G-Grid either as a C–Peer or as an S–Peer. If it chooses the former approach, it will have to issue direct insertion requests for all its objects to the G-Grid system. In the latter, it will have to wait for an interaction with another S–Peer and then integrate its content through the normal partitioning process. The choice depends on who implements the system who should consider values like the time of disconnecting a peer and the rate of input/output peer in the network. In fact, the first choice is better when the network has a low dynamism (aka low rate of input/output peer) and peers stay out for a short time, this means that a S–Peer could easily return to play its role if the network do not have a lot of changes during his absence.

As indicated in 3.2.2, an important concept in the G-Grid is the peers' ability to learn other peers' local routing tables during search operations. When a peer interacts with other peers – e.g. querying –, its local routing table grows and improves its capability to find the most efficient route to its target objects. Clearly, learning content-based routing tables is an emergent property in that the minimum path to a target peer is discovered without having to encode into the system a minimum path algorithm.

3.3 Experiments

In this section we show the performance results of our proposal in four simulated application scenarios. The simulator is used to test the traffic distribution on large-scale networks of 10,000 nodes over PeerSim [87], a testbed for large-scale P2P networks. The traffic distribution parameter is very important because it shows how the working traffic is evenly distributed on network peers. We observed also other important parameters such as the number of hops, the number of messages exchanged to maintain the overlay structure – called system messages –, and the number of connections failed because of busy memory, due to the architecture of the data structures. The basic G-Grid and G-Grid with Learning [100, 88] are two versions of our algorithm implemented in the simulator and tested to obtain performance on commonly accepted benchmarks. As a first element of comparison, we selected the SkipCluster algorithm [142], as proposed and a new version that implements in addition to the overlay HiGLoB [132]. We applied the features of load balancing of HiGLoB to balance the number of peers contained in each cluster.

The experimental environment is a Linux virtual machine with 6 CPUs at 2GHz and 6GB of RAM for the simulations. The program is developed in Java, as the simulator PeerSim. We have defined a class that implements the interface of the node provided by PeerSim, which manages events such as the arrival of messages or internal events defined by the user. In our experiments, for each node, we connected the overlay to be tested, as G-Grid, and another level of transport provided by the simulator natively. We have also created controls that the system runs periodically, according to a configuration file, allowing the generation of query, the connection/disconnection of nodes, and writing a snapshot of the network in a database. Each parameter, nodes and controls, is read from the configuration file. For any other technical details please visit the website of the simulator[2].

In all scenarios, we start from a single node network and connect quickly 10,000 nodes. The nodes create the desired structure by exchanging messages with other peers in the network. When all nodes joined the network we start to run 100.000 random queries. We observe the system state and store it into a database every 10,000 queries. While these parameters are common in every scenarios, the differences among the scenarios are listed below.

[2] `http://peersim.sourceforge.net`

1. The first scenario includes only query executions. The network structure does not change after the last node join. The number of nodes reaches 10,000 linearly and remains the same for all the rest of the simulation.

2. In the second scenario, after the initial configuration of the network, one peer joins the network every 100 queries. This behavior increases the starting network (10,000 nodes) by 10%. This is intended to test the behaviour of a growing network, intuitively, new nodes could introduce longer search paths and increase the number of system messages

3. In the third scenario, after the initial configuration of the network, there is one node connection or disconnection every 100 queries, with 50% probability. This means that, every 100 queries, the simulator randomly adds or removes a peer from the network. The number of nodes fluctuates around the starting value of 10,000. This is intended to give additional stress to the system, because peer removal changes the structure and can impose longer path traversals

4. In the last scenario, every 100 queries, there is a number of peer join equal to the square root of the number of queries run up to that moment. This is in some way similar to scenario 2, but with greater growth.

In every simulation we assume a uniformly distributed load in the network nodes.

3.4 Results

It is worth to observe that, despite the differences among the various scenarios, the results were really similar. For this reason we will comment only the results of the first scenario. We also want to emphasize the improvements introduced by G-Grid, without dwelling on the particulars of the networks, since our experiments show that these improvements are guaranteed in any kind of p2p network.

In the Figures, the horizontal axis represents time, and the marks are the cumulative numbers of queries issued in the system during the experiments. In Figure 3.7 the vertical axis represents the total number of messages issued

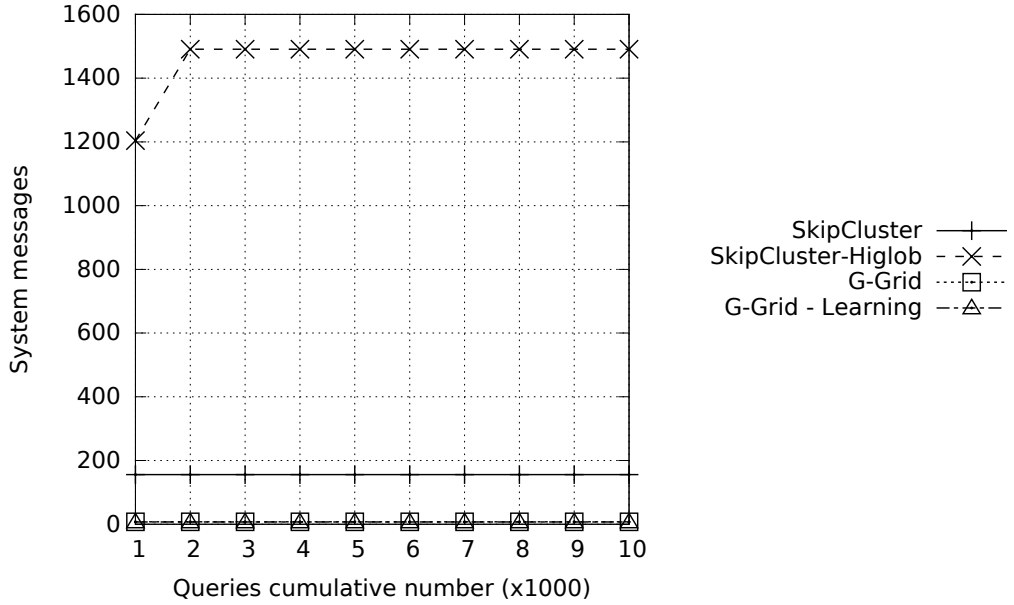


Figure 3.7: Overlay structure messages traffic.

in the system in the unit time, as a result of the querying activity, to maintain the overlay structure. The basic version of SkipCluster requires few messages, less than 200 in a 10,000 nodes network, to create the overlay structure, instead the introduction of HiGLoB increases considerably that value. We observe that the basic version of G-Grid has a low cost, almost zero, due to the binary tree structure on which it is based. The management cost is found in the delegation of the regions leading to the creation of new nodes in the S-peer tree. The cost of G-Grid with Learning and basic G-Grid is the same because the changes due to learning are sent in piggyback on query messages. SkipCluster with HiGLoB requires more overlay messages than basic SkipCluster, because the HiGLoB internal structures keeping/updating the histogram of the clusters requires a heavy exchange of overlay messages.

Figure 3.8 shows the average number of hops per peer. Each peer measures the number of necessary connections to achieve the wanted data. During the simulation starting phase, the number of query messages sent by each peer is very low, so the measured value becomes interesting only after the initial phase of adjustment. When more queries are performed, we get closer to the expected average. We noted, in fact, that the version of G-Grid with

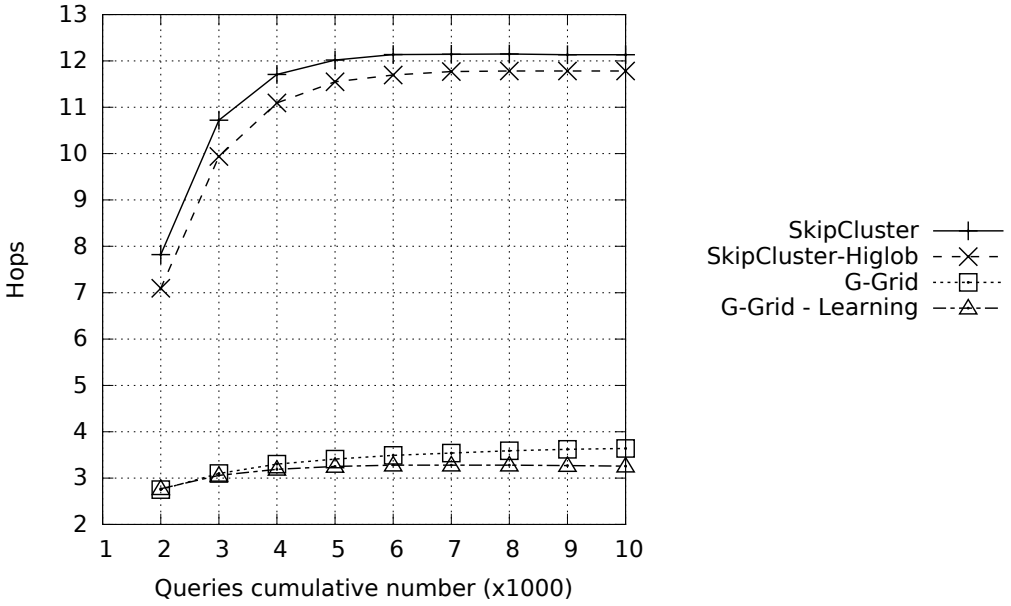


Figure 3.8: Average number of hops per peer.

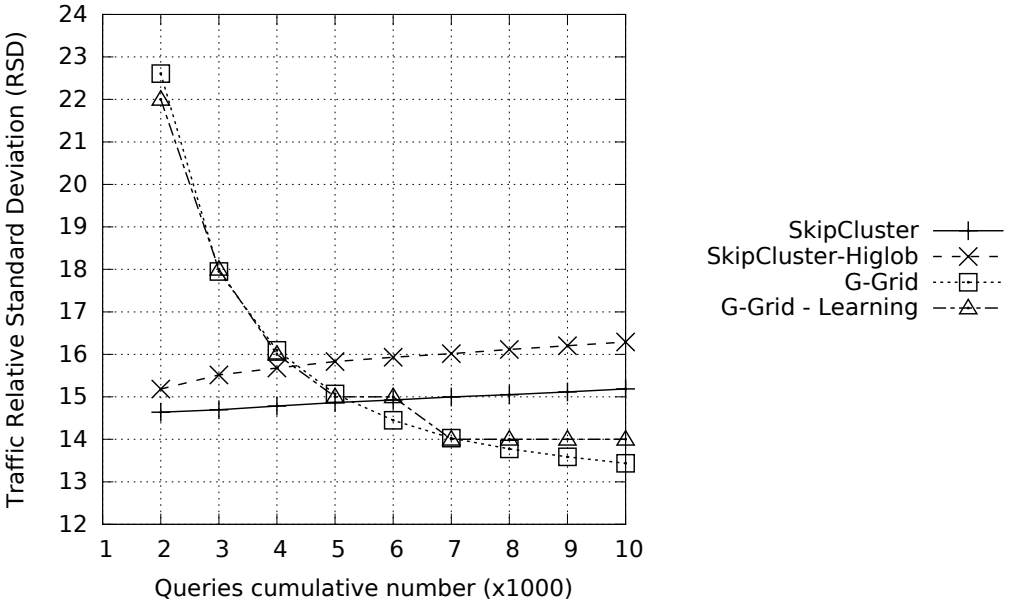


Figure 3.9: Traffic standard deviation for peer.

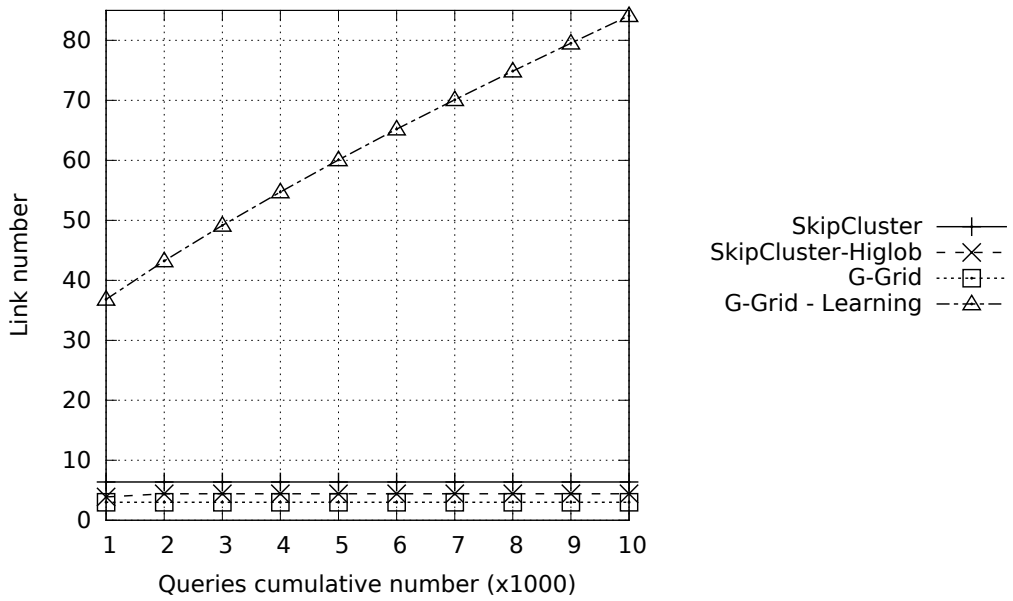


Figure 3.10: Number of links per peer needed to maintain the overlay structure in scenario 1.

Learning, tends to the theoretical value 2. This value is justified by the fact that, since the network does not change its composition, with a large number of queries, each peer has knowledge of the entire network structure, so using only a step (hop) to reach the solution and one for the return.

Even basic G-Grid shows a rather low average number of hops. Both versions of SkipCluster has a hops average of slightly less than the logarithmic value of the network dimension.

In Figure 3.9 we measured the average traffic standard deviation for peer as the ratio between the standard deviation and the average of messages routed from each node of the network. We observe that after an initial phase the traffic standard deviation of both G-Grid algorithms is greater than both SkipCluster algorithms. All overlays have a value that remains on the same order of magnitude, the average is around 15. The version of HiGLoB SkipCluster distinguishes oneself from SkipCluster basic version of about one unit. At least, G-Grid basic version, and even better G-Grid with Learning, offers a lower average traffic standard deviation.

In Figure 3.10 we obtained information on the amount of memory required

for the maintenance of each structure. Large number of links necessarily involves high amount of memory to store them. In every scenario the situation is the same: the worst overlay is represented by G-Grid with Learning, because the number of links increases as the number of executed queries. In fact, for each query, each peer discovers and creates links with the other nodes through which the query is passed. This leads to have a number of links equal to the size of the network, in the steady state. The upward trend of the links will arrive at 10000 after a very large number of queries. The basic version of G-Grid has the least number of links per node due to its binary tree structure. At last we note that the introduction of HiGLoB in SkipCluster decreases the number of links of the structure by one third, close to the value observed for basic G-Grid.

A new scalable self-organising data structure for P2P networks is introduced. This structure enables efficient multi-dimensional search based on partial range queries. We have also illustrated how peers can exploit the properties of these structures to learn dynamically both the distribution of content and the network topology, and thereby, provide algorithms for efficient processing of range queries. In the worst case, search costs for a single object, measured as the number of hops over peers, are less than logarithmic in the number of peers. But, for many realistic workloads of insertions of new objects and retrievals, such as those currently taking place on the web, the average is equal or less than 3 hops, independently on the wideness of the P2P network. We have also sketched out an aspect which is seldom treated in P2P literature, namely the possibility of merging independently constructed data structures in a single unifying structure.

This work summarizes the idea of achieving virtual DBMSs from P2P systems as a set of emergent services, but which abide by the same desirable properties of centralized DBMSs, namely, data integrity and consistence, transaction processing and a complete SQL expressiveness. Now, let see how it is possible to improve some performance results with an hybrid overlay.

Chapter 4

Self-Balancing of Traffic Load in a Hierarchical Multidimensional P2P System

The P2P technologies have begun to attract the interest of the scientific community relatively late compared to their use. In the late 90's many P2P applications could be found on the Internet and these have always enjoyed a high capillarity among worldwide users. Then it was discovered that this technology were very promising for networks efficiency, distribution and sharing of resources. Such networks are dynamic, completely decentralized and characterized by high autonomy.

Today there is a tendency to study networks seen not merely as telematic networks, but as recurrent structures that we can encounter, in some way, in several aspect of daily life, such as: the aggregation of the neurons of the human brain, the connections that form human knowledge within a society of people, the properties and structures of ecosystems, the dynamics of spreading news. This new ideas are defining the "science of networks" [18], a new discipline that studies the structure of the networks in all its different forms, from a general point of view, analyzing their properties and how they affect the daily reality. For its own nature the science of networks involves several scientific disciplines.

The main applications that have affected the world of P2P were primarily oriented to distributed file systems and grid computing. In particular, the early distributed file systems allowed only simple lookups of file directories by file name. In the meanwhile, the scientific literature was proposing to

increase efficiency and operativity by adding some kind of structure.

A P2P network without an organized structure uses the broadcast to allow the comunication among peer. In this way, however, the network becomes overloaded quickly and communication is neither efficient nor scalable. The adoption of organized structures in P2P networks avoids the use of broadcast and decreases network traffic. They build routing tables more efficient, especially in static environments. The advent of a highly dynamicity in networks caused the decline of the efficiency of the structures. However, also in a static environment, the traffic can be unbalanced on the network when there are some peers that are overloaded and others that are instead underloaded. This is typical in tree structure. G-Grid is a distributed data structure for P2P networks, that creates multi-dimensional indexes on P2P networks, making the basic operations research (including range-query) and database editing very efficiently, despite the fact that its structure is not imposed a priori. Its strength makes the structure very flexible, but at the moment its biggest flaw is the computational load balancing of the peer, because its structure is a binary tree.

Now we try to improve drastically load balancing in G-Grid by seamless integrating a new ring-based routing algorithm, as a self-emergent property.

4.1 Reasons for the Improvements

Peer-to-peer networks have attracted the interest of the scientific community because they offer features that other communication paradigms fail to offer.

Current solutions allow to search efficiently, without consuming unnecessary bandwidth with broadcast messages, and even in some proposals also to treat multi-dimensional data. The possibility of treating these networks as distributed DBMS, or systems in which we can issue something similar to SQL queries and get support to consistency checking and transactions, still seems a goal to reach and research is currently going in this direction. The addition of an overlay structure in modern P2P networks is the key to upgrade them. This means that now the location of data on the peer is regulated for content and criteria, whereas previously each node could hold all kinds of data, and this was not conducive to the development of efficient routing algorithms. The research, even if the peer does not have a global knowledge of the topology, is aided by this structure which helps through the collaboration of nodes, to have more precise information on the alloca-

tion of data.

The "datum-node" assignment is performed by DHT [10, 58], and each structured system implements its own DHT as needed [104, 128]. The general idea is to assign a key to each datum by a specific function (usually by hashing), then the set of keys is partitioned and each node is responsible for one of these partitions. The function that links the key-node assignment is surjective and the routing algorithm is closely related to it. In fact, usually every node holds references to its neighbors, which will have subspaces close to that of its competence. The relationship between datum and key is dictated by a bijective function, which must be a common reference point for all nodes. When a node receives a request for a specific key, it checks if this is part of its subspace of competence, in the contrary case it forwards the message to a neighbor, which is selected according to the routing algorithm.

Multi-dimensional structures have been extensively investigated over the last 20 years where the main goal is to support efficiently complex range queries over multiple attributes. A literature survey in this area is available in [146],

In G-Grid (3.2), the tree is always explored recursively, and if we analyze the scenario in a P2P environment, where each peer is responsible for at least one region, this would have connections with its father and its sons. The insert and remove queries walk through the tree and each peer along the walking receives the request from its parent, evaluates if the G of the record is in its region, recalculating the G since the depth of the tree if it can not know it a priori, and if that fails it forwards the request to both the children. This recursive mechanism shows how this structure works well in P2P environments, because each step requires the interaction of only two entities: father and son nodes. Regardless of whether working in a distributed or centralized structure, basic operations are performed exploring a binary tree and the only difference is that in distributed environment this tree is split into parts assigned to autonomous nodes.

The advantage of the G-Grid structure is that it allows to perform insertions, deletions, and queries in time proportional to $\mathcal{O}(\log N)$ where N is the number of regions. In addition, with the learning mechanism which will be explained later, this time can be furtherly reduced to become sub-logarithmic. In other words, the efficiency of DBMSs is exported to distributed environments by virtue of the limited number of hops[1] required to complete any

[1]The number of gateways that separate the source host from the destination host.

operation.

This is a fundamental property that can be found in DBMS and that in the field of distributed avoids to make too many hops to a query, reducing hence the response time.

4.2 *Treelog* Routing Algorithm and Alternative Load Balancing Methods in G-Grid

Until now we showed the advantages of G-Grid over other P2P architectures and analyzed the benefits that it brings to the management of data distributed over a wide network. Unfortunately there are still some open questions to be solved. Each node in G-Grid can determine with a simple computation if a search can be done in its subtree or must be forwarded to its parent node. If we analyze in deep the routing algorithm we note that in G-Grid the data search time is sub-logarithmic on the average, but the peers that are on the top of the tree are heavily overloaded, because tree traversals mostly require to go up towards the root before going down towards the target. The recursive tree exploration mechanism generates a load distribution grows exponentially with decreasing depth, this means that the load of a parent node on the average doubles that of its children. A corresponding data structure has already been proposed by Widmayer [66] with the same problems.

It is reasonable to observe that if the peers have some kind of awareness of the network structure this phenomenon can be mitigated. A possible improvement could be to make available to each node some information about the position in the tree structure of other nodes, not directly descendant or ascendant, and this requires introducing additional links. In this section we introduce this new kind of links, importing ideas from Chord ring, and discuss in deep how it is integrated in the G-Grid structured overlay.

4.2.1 Isomorphism of a Chord Ring on G-Grid

Computational load balancing of different cooperating entities is not a simple task. This problem has many facets and the solution depends on the type of communications forwarded to the various peers. The network traffic and the amount of computation within the nodes are dictated by random time–dependent variables. To simplify the problem we will make some assumptions

on the kind of randomness. The random process of querying peers has three orthogonal aspects: distribution of the peers that run queries, distribution of the peers that hold the queries' response, and distribution of data among the peers. Our first simplification is to assume, according to most of the P2P literature, that each peer in the network has about the same amount of data in its memory and both the query requesting peers and the query responding peers are uniform with respect to the data space. The relaxing of these condition will be a natural evolution of the present research.

When devising our solution to the unbalancing caused by the tree structure, we were inspired by a major advance in P2P research: Chord [125]. The outstanding feature of this proposal is the ring structure, which introduces additional links to obtain a better traffic distribution among peers. In G-Grid [100], the routing explores the binary tree through structural links, which connecting the nodes that have direct connections – e.g. parent-son – among their regions. In Figure 4.1 we can see an example of how a message traverses the network. In this case, for sake of clarity, each node has got a single region. Black circles represent physical nodes that hold regions within data, while the white circles are logical nodes of the data space, that are not part of the structure, because they do not hold any data. These are included and managed by the first physical node met going up the tree, following the links to parent nodes. For example, the node *11, which is beside D and is a white circle, is managed and incorporated by node *1.

Since the query distribution is uniform, it is more likely that the node that runs a query is in the deepest parts of the tree, because these nodes are more numerous than those closer to the root. The same for the target node. We see that the G-Grid operations require a logarithmic number of hops relative to the number of peers. For most requests, query messages start from the lower part of the tree, climb up to the root, or nearby, and climb down again to the recipient. In Figure 4.1, we see what happens when node E is looking for the query region *10101. E sends the query message to A that is its first physical parent node. A routes the query to B, B to D, and D to F, that contains the query region *10101, since it has no child nodes in the tree and its address *101 has the longest prefix in common with the requested data *10101. This behavior overloads exponentially the nodes that are closer to the root. This is also due to the fact that the regions of the upper part of the tree manage a wider subspace (see Figure 4.2), so they are more likely involved during routing.

Our proposal is to connect all peers by a ring structure, building an iso-

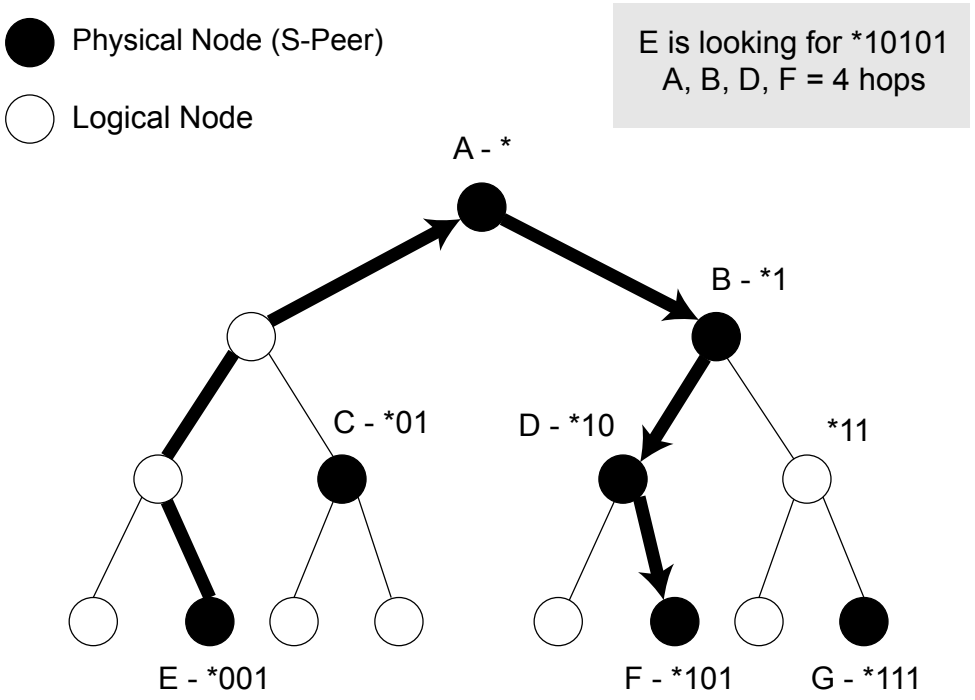


Figure 4.1: Example of a request routing in G-Grid.

morphism of the Chord ring structure within the G-Grid binary tree. Moreover, creating logarithmic links between nodes, in addition to the ring structure, we prevent an imbalanced load distribution over the structural links. The logarithmic links are precisely the link category that makes Chord a network structure in which the load is evenly distributed. At the same time, the number of messages' hops remains logarithmic with respect to the number of nodes.

The construction of a ring on all S–Peers imposes a total order relation on them. The amount of order relations that we can impose is factorial with respect to the number of nodes and is not a trivial choice, since the elements that we want order are independent peers without a global view of the network. Moreover, we want manage an autonomic system where the control is highly decentralized, so we have to find a relationship that joins all peers, regardless of the network shape, which can change dynamically over

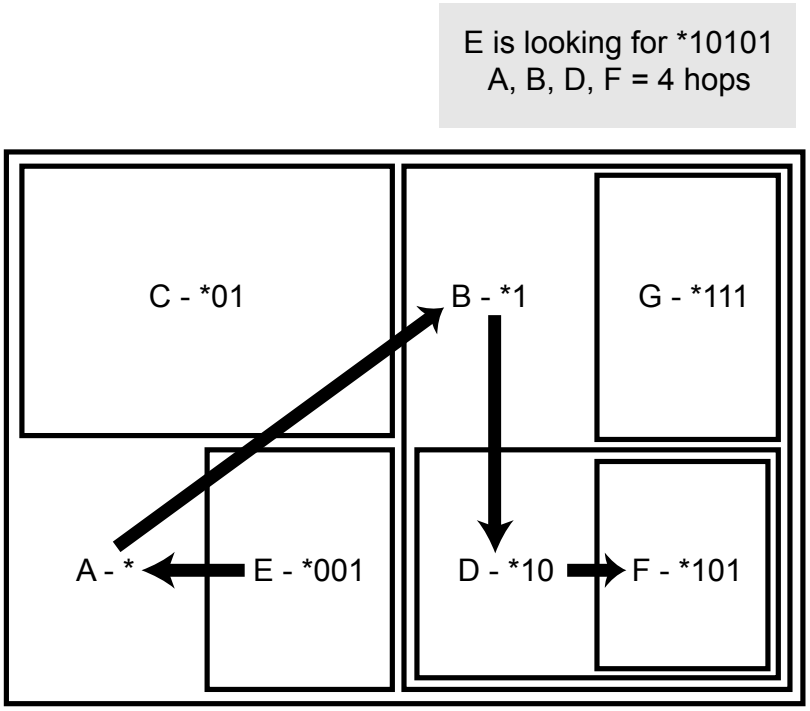


Figure 4.2: Example of the same request routing of Figure 4.1 in G-Grid hyper-space.

time. In particular, the nodes to be ordered within the ring are the only S–Peers, the active part of the G-Grid structure. An S–Peer forwards the messages to their destination and is responsible for a part of the data space.

Each S–Peer has a physical address, therefore we could order them according to this attribute, but this is not appropriate because these addresses are dynamic and change over time on the Internet.

One feature that is common to all S–Peers is the fact that each of them holds at least one region. We can suppose that each S–Peer holds a region R, and, possibly, other regions R_I that can all be included in R, or rather all regions R_I where $R_I \in R : R_I \supset R$. In Figure 4.3, the S–Peer D has a sub-tree formed by S–Peers (black dots, e.g. F), which represent the regions that have a relationship with it, and C–Peers nodes (white dots). In particular, D has a region R_D and manages the node F (owner of the region R_F), which have

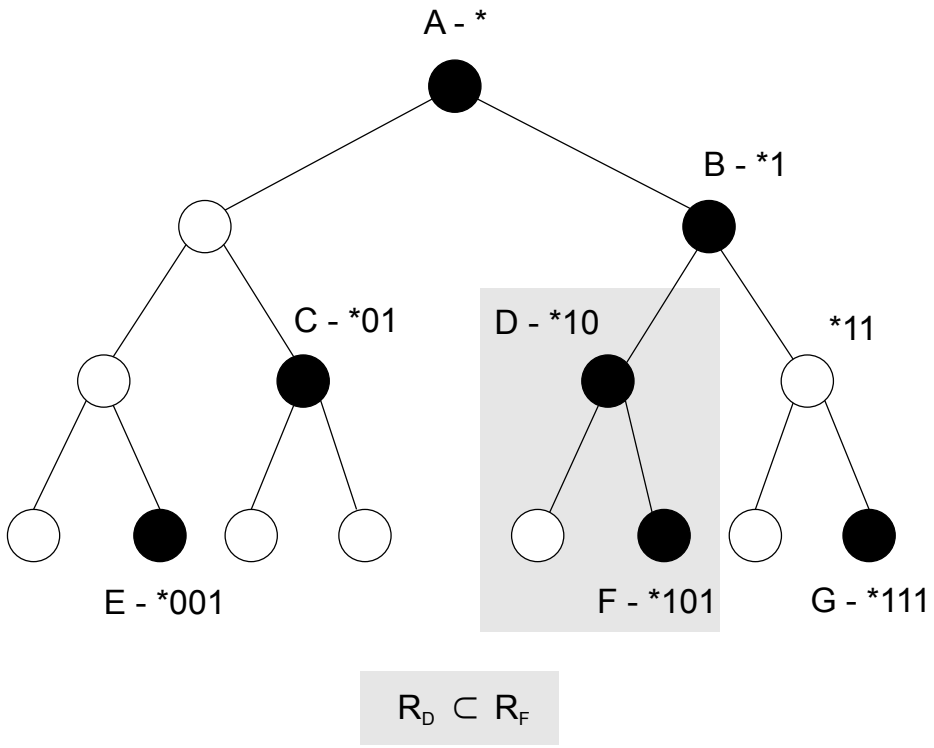


Figure 4.3: Example of a sub-tree owned by S–Peer D in G-Grid tree structure.

a parent-son relationship with the higher level node D. This is the coverage property of G-Grid: we can say that a generic region R_X belong to S–Peer Y, in other words $R_X = R_Y$ of Y or $R_X \in R_Y$ of Y, if and only if the binary string G_Y – e.g. *10 for D – of the region R_Y is the longer prefix, compared to the binary string G_X of R_X, among all the binary strings of the regions R of other peers in the G-Grid tree structure. This means that we just need to know the binary string G_Y of the region R_Y of the peer Y to know if Y may contain a generic region R. In the case in which Y is not the holder of region R, it forwards the request through its parent structural link. Ideally, if we know all the region binary strings G of the peers, we know the entire network structure. However, we state that the more region binary strings we know, the more the probability of finding at the first time the target region owner is high. Moreover, the more the network structure awareness increases, the

more the average number of hops per request decreases.

Under the assumptions made, that every peer has a single sub-tree, we saw that the identifier of the region R of a peer is an important reference for the region querying. Moreover, it guarantees the uniqueness of the attribute assigned to the peer, so we can use it to build a total order among the S–Peers. This means finding a total order among all the possible regions of the data space and it is not trivial because the regions themselves are n-dimensional spaces. The G-Grid binary tree can help us because it is the dual representation of the regions present in the data space, and then the regions' sorting leads to the same result. The total order relations of binary trees are well known [102, 25] and, in this work, we analyze the post-order sorting [112, 135] (see Figure 4.4).

Proposition 1. *If the nodes of a binary tree are sorted according to the total order relation post-order, then if node X descends from node Y, it necessarily holds $X < Y$. The opposite is not necessarily true, that is $X < Y$ does not mean that X is necessarily a descent of Y.*

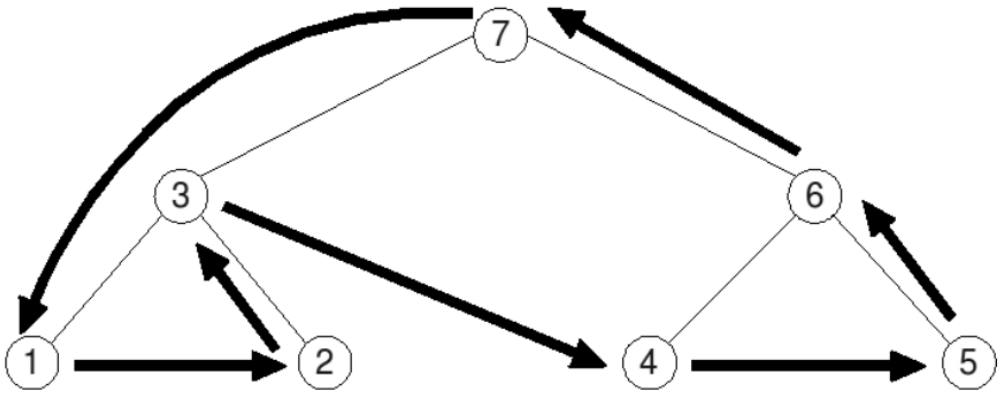
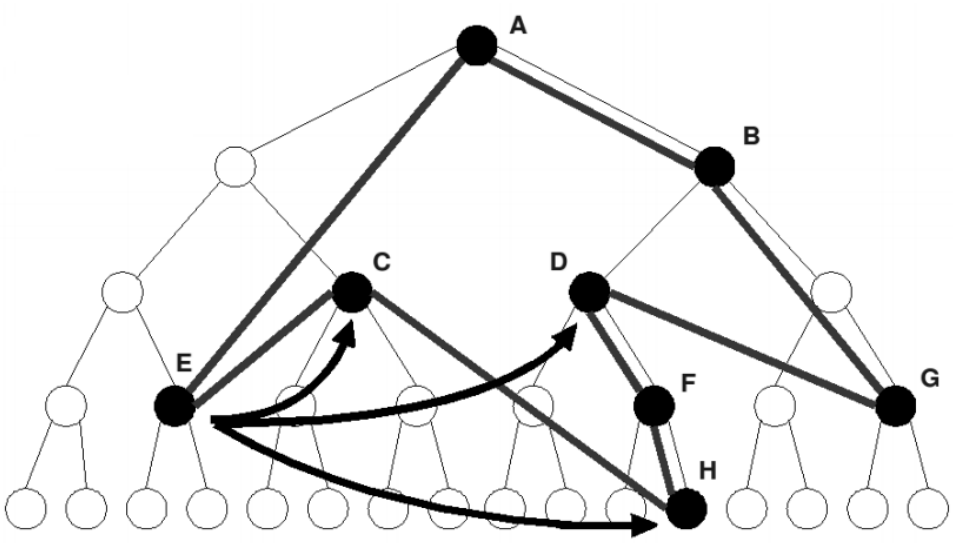


Figure 4.4: Post-order sorting of a binary tree.

We place the peer on the ring following the above mentioned connection, applied to the peers' regions. This connection is useful for region search because of the Proposition 1, derived from both spatial and coverage properties of G-Grid. To allow the reachability from any source node to any target node, it is necessary to close the ring. The total order on a finite set determines there there will be a *top* node and a *bottom* node. Then the ring is

closed linking the top element to the bottom one. See the arch from 7 to 1 in Figure 4.4. After linking the S–Peers ring, we add logarithmic links, as in Chord. Scanning the ring for in the direction of the increasing order, each tree node has a link to its successor in line and to all of those in consecutive positions, that are power of two. Each node has a logarithmic number of links compared to the number of nodes of the ring. Figure 4.5 shows an example, for simplicity of representation in this case every S–Peer holds a single region.



Node	*G*	Log-Link	Node	*G*	Log-Link
A	*	E-C-F	**E**	*001	C-H-D
B	*1	A-E-H	**F**	*101	D-G-A
C	*01	H-F-G	**G**	*111	B-A-C
D	*10	G-B-E	**H**	*1011	F-D-B

Figure 4.5: Isomorphism of a Chord ring on G-Grid tree structure.

Each S–Peer manages a sub-tree regions. This condition is not required for the positioning of the regions. The peer is handled as a logical unit that manages a sub-tree, holds its connections, both structural (G-Grid tree) and logarithmic (Chord ring), and has a precise location in the ring.

4.2.2 *Treelog* Routing Algorithm

Each operation carried in G-Grid, regardless of whether it is a query, insertion or removal of data, states that it is required by a message that contains a target attribute, that we call *query region*, which is the binary string G of the region R covered by the requested operation. The S–Peers must forward this message to the owner of this region using the query region, which will run the requested operation and then communicate the results directly to the requester. The physical address of the requester is always present in the messages.

The routing algorithm is a procedure that every S–Peer performs when receives a message, in order to deliver it to the correct target S–Peer. This is successful whether all intermediaries cooperate.

Each region is described by a binary string G, that describes the location of the region in the binary tree. $Length(G)$ gets the depth of the region in the tree, which corresponds geometrically to the granularity of the search area. In addition, $Length(G)$ of the query must be greater than or equal to the $Length(G)$ of the deepest region in the tree. This is because the query must be contained by one region and the $Length(G)$ of the query region can not be greater than the $Length(G)$ of the query itself. Calculating the binary string with a $Length(G)$ of the query region larger than the real one, it does not affect the search mechanism. For example, if a query region with $G_X =$ *100101 corresponds to a sub-space managed by the region $G_Y =$ *100, the query region will be reached even if we considered the $G_Z =$ *100101010110, with $Length(G_Z) = 12$, larger with respect to $Length(G_X) = 6$. The peers do not have a global vision of the network tree structure, then they run requests with large length values of the query region, over-estimating the size of the distributed data structure.

Any peer knows the status of the structure if it changes dynamically. Given a query region and a partial view of that query region, which may not be correct, how do we know where is the query region located in the system structure? Moreover, how do we find out the request target node? We have a set of possible candidates which have a binary string G of R as prefix of the query region. We do not know a priori how many peers are in the system, nor their location. We can not know in advance whether a node is logical or represents a specific region owned by a S–Peer. The possible candidates are all the nodes in the binary tree that have a prefix that fits with the G of the query region. Fortunately, it is a small number – sub-logarithmic, compared

to the number of S–Peers. The candidates are all directly connected by parent-son relationship in the tree.

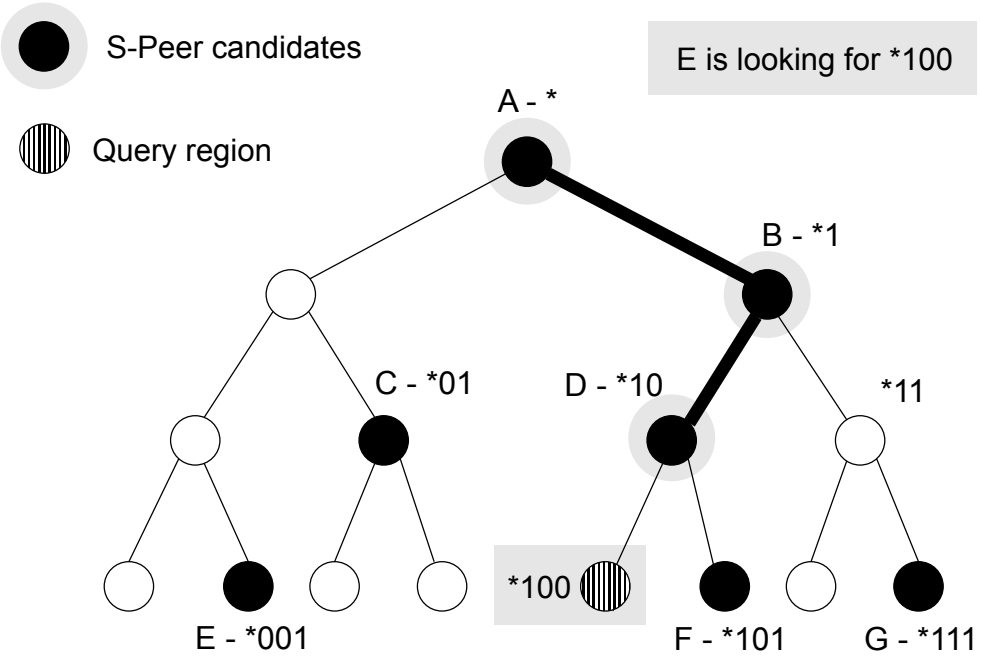


Figure 4.6: S–Peer candidates for a query region.

Figure 4.6 shows an example: node E is looking for the query region $G =$ *100 – which could also be calculated with $Length(G) > 3$, that is a higher value, because the process does not change. If E could know the global binary tree, it sees that the owner of the query region is D, but, in reality, E can not know a priori who is the owner. We can say that the candidates are A, B, D, and even tree node represented by the binary string $G =$ *100, which we do not know in advance whether it is S–Peer or C–Peer. However, E may not be aware of the existence of A, B and D.

In Chord, each peer uses the Definition 1 to search for a query region.

Definition 1. *In Chord, if we take two peers X and Y, with Y successor of X in the ring, we can say that all the k keys such that* $x \prec k \preceq y$*, belong to Y, with x belonging to X and y belonging to Y.*

The isomorphism of the ring in G-Grid has improved the load balancing, as we will show later, but has complicated the query region search, in particular, Definition 1 does not hold anymore. All the nodes of G-Grid represent sub-spaces and they are sorted according to the post-order relationship. Assuming that each peer manages a single sub-tree, the following proposition holds.

Proposition 2. *In G-Grid Chord, given two peers X and Y, with Y successor of X in the ring, we can not say that all the R regions such that* $x \prec r \preceq y$ *belong to Y, but only a non–empty subset of these.*

In other words, while Chord is a pure ring, in G-Grid Chord the ring is used only for message routing, when possible, otherwise the binary tree structure is always available.

The example in Figure 4.5 shows the meaning of Proposition 2: consider the peer D, the peer G and, respectively, their binary strings R_D = *10 and R_G = *111. Along the ring, from D to G, we encounter different logical nodes, which represent the regions in post-order *1100 - *1101 - *110 - *1110 - *1111 - *111. Only *1110 - *1111 - *111, however, belong to the node G, while *1100 - *1101 - *110 are sub-spaces managed by B, which has as a binary string G_B = *1. The distribution of the regions in the nodes is not known a priori, since it depends on the shape of the tree and its evolution. A peer can not know in advance who will be the message last recipient, but can only guess (see S–Peer candidates in Figure 4.5).

Because of these considerations, we developed the *treelog* routing algorithm, which uses the Chord routing mechanisms adapting the search of query regions to the dual structure of G-Grid, according to what Definition 1 and Proposition 2 state. The pseudo-code of the Algorithm 3 describes the actions that each peer executes when receiving a message whose query region is not under its responsibility.

This procedure requires the following input parameters:

- ***selfG*** - The G of the region R of the peer which is currently processing the message and received the message to forward.

- ***links*** - The routing table: a set of links to other S–Peer accumulated over time. Basically it is a two-column table in which the first is the G of the region R of the peer and the second is its physical address. The latter can be retrieved via the function *physicalAddress(G)*.

Algorithm 3 – Find next hop - *Treelog* routing algorithm

Input: msg.

Output: physical address of the peer for the next hop or NO ROUTE error.

```
if selfG = msg.bestCandidate then
    msg.phase ← 2
end if

if msg.phase = 1 then
    bestCandidate ← msg.targetG
    if selfG is prefix of msg.targetG then
        while bestCandidate is not in links and Length(bestCandidate) >
Length(selfG) do
            remove first right digit from bestCandidate
        end while
        if bestCandidate = selfG then
            bestCandidate = "*"
        end if
    else
        while bestCandidate is not in links and Length(bestCandidate) >
1 do
            remove first right digit from bestCandidate
        end while
    end if
    if Length(bestCandidate) > Length(msg.bestCandidate) then
        msg.bestCandidate ← bestCandidate
    end if
    if msg.bestCandidate is not in orderedLogLinks[N] then
return physicalAddress(msg.bestCandidate)
    else if msg.bestCandidate ≺ orderedLogLinks[0] then
return physicalAddress(bestCandidate)
    else
        for i ← 1 to orderedLogLinks.length do
            if msg.bestCandidate ≺ orderedLogLinks[i] then
return physicalAddress(orderedLogLinks[i − 1])
            end if
        end for
    end if

else if msg.phase = 2 then return physicalAddress(selfG parent node)
end if
```

- ***orderdLogLinks[N]*** - A subset of the routing table: it is a vector of N links ordered by the relation of post-order. The first element is *selfG*. Basically they are the Chord logarithmic links of the peer.

- ***msg*** - The received message. If it involves a region that is not the responsibility of this peer, it must therefore be forwarded. The most important message attributes for the routing are:

  msg.targetG query region,

  msg.phase set to 1 upon creation, it allows to know the state of the request,

  msg.bestCandidate indicates what is at present the best candidate found by intermediaries could hold *msg.targetG*.

The output of the algorithm can be:

- the physical address of the next S–Peer to which the message will be forwarded. In the event that the expedition fails, for example, because the recipient is disconnected, the peer updates the routing table and re-runs Algorithm 3.

- a *NOROUTE* error, if the procedure does not succeed in finding a suitable peer, i.e. there are no possible routes to reach the destination through the *treelog* algorithm. In this case the current S–Peer communicates the error to the sender of the message, the one who created it.

The algorithm tries to find the recipient by the logarithmic links. When the recipient, designated in a set of possible candidates, is not correct, then the peer route the message through the G-Grid standard routing. Each intermediate S–Peer that processes the query message executes the *treelog* algorithm. The more S–Peers processes the request and the more likely is that someone knows the recipient, because if a S–Peer has not the exact information on the recipient, might have information that approaches it.

The general algorithm so works in two phases, which indicate the status of the current request with the attribute *msg.phase*: if it has a value of 1 means that the search for the recipient is on the logarithmic links; it is 2 instead if the search is via the G-Grid structural links.

Phase 1

For a generic hop, through logarithmic links, first step aims to get the best candidate that peer intermediaries have found so far. Each peer has a routing table updated over time. In this phase, the peer searches links in the routing table, that may be candidates for *msg.targetG*, that are peers that have a prefix of *msg.targetG* in the binary string G of their region R. This binary string is placed in the attribute *msg.bestCandidate* if and only if it is larger - compared with the $Length(G)$, indicating the depth of the tree - than the current value.

When the *selfG* of the current peer is equal to *msg.bestCandidate*, there are two possibilities:

- the region request is held by the current peer so the final recipient is reached;
- *msg.bestCandidate* is not the correct recipient, but it could still be one of its children peers in the tree. The correct recipient derives hence from the current candidate, so if structural links are consistent, the message arrives at its destination through *msg.bestCandidate* peer.

Furthermore the attribute *msg.bestCandidate* could, at this point, indicating a deeper node of the real owner. This case is solved by the algorithm in the following way: if a peer receives a message that is still in phase 1 and *msg.bestCandidate* is previous than the first logarithmic link of the current S–Peer, then it go to phase 2.

Phase 2

In the event that the peer do not hold any such link, then the root node is considered as the best candidate, because in the worst case it reaches all the nodes. In this case it uses the structural links parent-son among nodes.

The average number of hops is sub-logarithmic respect to the number of peers, because each intermediary transfers a part of its network knowledge on the query message. The mechanism allows a S–Peer to indicate the best candidate. Without the phase 1, this knowledge would not be distributed on the request and the load becomes unbalanced. The *treelog* routing algorithm mixes the Chord routing mechanisms with those of G-Grid: each helps to

improve the defects of the other. In particular, Chord helps to balance the load on the peers, while G-Grid ensures that each message is properly delivered to destination – if structural links are consistent. The more network knowledge is hold by peers, the higher is the probability that intermediaries indicate candidates closer to the recipient.

The learning mechanism, introduced in [100], helps to increase awareness and to discover peers joined the network recently, but does not identify the disconnected peers. In the following sections, we give some indication to identify the inconsistent connections.

Finally, we summarize the properties introduced by the *treelog* algorithm with the following Proposition:

Proposition 3. *Necessary and sufficient condition for the correct routing: each peer must hold consistent structural links (parent-son), which is a small number compared to the number of peers of the network. The average number of hops for queries and load balancing is strongly influenced by the degree of knowledge that peers have on the network. Each peer must then maintain links to the peer that has processed messages and their intermediaries.*

4.2.3 Cooperative Construction of Logarithmic Link

When a node becomes S–Peer has some structural link – certainly a link to the parent node that delegated the regions – and a collection of links to S–Peers. We stated that the *treelog* routing algorithm needs new links, the logarithmic links, which connect the peer with each ring node that is reachable in a logarithmic number of hops respect to the total number of peers in the ring.

Analyzing a single routing table, we can not know its logarithmic links. Each S–Peer should know the global state of the network, moment by moment, to know who is the S–Peer successor on the ring, who the following after 2 positions, 4 positions, 8, and so on. This is not possible because each peer can not have a global view of the network instantly. The received information often may be incorrect due to the dynamics of the network. Each S–Peer knows a portion of the state of the network and to build its logarithmic links, has to cooperate with its neighbors to achieve a fairly consistent network knowledge. Through the exchange of their local information, the S–Peers can compare different views of the network, which differ between them and with the actual state of the network. Deducting reliable information on

targeted portions of the network, S–Peers may discover the location of their logarithmic links.

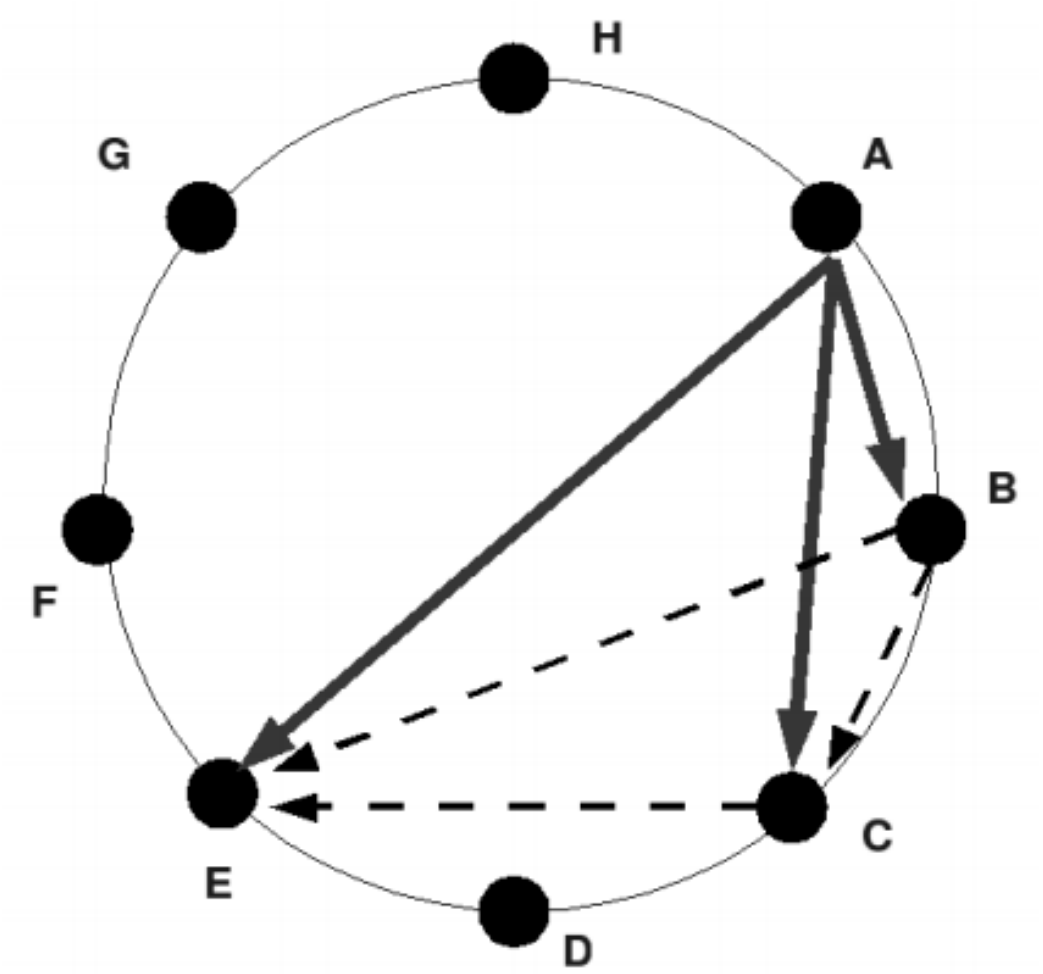


Figure 4.7: Deduction of logarithmic links from S–Peer neighbors.

In Figure 4.7, we see that the logarithmic links of different S–Peers have relationships with each other. In the example there are 8 nodes: analyzing the logarithmic links of node A and assuming that A knows his successor B, A can get all its remaining logarithmic links, which are in order B - C - E. In fact, A has already its first logarithmic link by hypothesis, that is B, obtains C asking to B its first logarithmic link and, once achieved C, asks to C the second logarithmic link that is E. From this reasoning, we can see that there

are defined relationships between the different nodes logarithmic links. The Proposition 4 summerizes that property.

Proposition 4. *If X is an integer that represents the position of an S–Peer inside the ring and $loglink(X, n)$ is a function that returns the n-logarithmic link of the S–Peer that is placed at the X position of the ring, then $loglink(X, n) = loglink(X + 2^{n-2}, n - 1)$, for each $n \in \mathbb{N}$ such that $n \geq 2$.*

Proof.

We introduce the function $loglink : \mathbb{N}^+ \times \mathbb{N}^+ - \{1\} \to \mathbb{N}^+$, defining a generic logarithmic link as:

$$loglink(X, n) \stackrel{\text{def}}{=} (X + 2^{(n-1)}) \bmod P \tag{4.1}$$

where P is the number of the peers in the ring. We prove the following identity for each $n \in \mathbb{N}$ such that $n \geq 2$:

$$loglink(X, n) = loglink(X + 2^{n-2}, n - 1) \tag{4.2}$$

We define the function:

$$SUC(X) \stackrel{\text{def}}{=} loglink(X, 1) = X + 1 \tag{4.3}$$

Base case for n
If we take three *S–Peers*, A, B and C, in sequence on the Chord ring built in G-Grid, and starting from 1, then their positions on the ring are respectively 1, 2 and 3. We can verify that:

$$loglink(1, 2) = loglink(2, 1) = C \tag{4.4}$$

then the position of C is 3.
From the Equations 4.3 and 4.4, we obtain:

$$loglink(1, 2) = SUC(2) = C \tag{4.5}$$

The Equation 4.5 states that 4.2 is true for $n = 2$.
We prove by induction that:

$loglink(X, 2) = loglink(SUC(X), 1) \Rightarrow loglink(X+1, 2) = loglink(X+2, 1)$

Using Definition 4.1:

$$X + 1 + 2^1 = X + 2 + 2^0$$

Recursive case for n
We need to prove that:

$loglink(X, n) = loglink(X+2^{n-2}, n-1) \Rightarrow loglink(X, n+1) = loglink(loglink(X, n), n)$

Using Definition 4.1:

$$\begin{aligned} X + 2^n &= loglink(X + 2^{n-1}, n) \\ &= X + 2^{n-1} + 2^{n-1} \\ &= X + 2^n \end{aligned}$$

This proves by induction principle that Equation 4.2 is true for each $n \in \mathbb{N}$ such that $n \geq 2$. □

Hence an S–Peer can obtain its logarithmic links from its neighbors. Due to the dynamic nature of the network in time, some logarithmic links may become invalid, therefore they provide incorrect information to the neighbors, who rely on them to build their own logarithmic links. The Proposition 5 provides guidelines to solve this problem.

Proposition 5. *Sufficient condition for the correctness of the logarithmic links of a set of S–Peers is that every S–Peer has its own $SUC(X)$ connection correct. If this condition is satisfied, everyone can properly get its logarithmic links from the neighbors with a number of queries $O(log(2^L)) = O(L)$, where L is the tree depth.*

Proof.

Equation 4.1 can be rewritten as:

$$\begin{aligned} loglink(X, N) &= X + 2^{(n-1)} \\ &= X + 1 + 2^{(n-1)} - 1 \end{aligned}$$

the last part is a geometric series, replacing this, and considering Equation 4.3, we obtain the follows identities:

$$\begin{aligned} loglink(X,N) &= X+1+\sum_{k=0}^{n-2} 2^k \\ &= SUC(X)+\sum_{k=0}^{n-2} 2^k \\ &= SUC(X)+1+\sum_{k=1}^{n-2} 2^k \\ &= SUC(SUC(X))+\sum_{k=1}^{n-2} 2^k \\ &= \dots \end{aligned}$$

from this progression we see as each logarithmic link can be expressed by different SUC links and, if these are correct and consistent then higher logarithmic links are consistent too. □

If Y is the successor node of a node X and Z is the successor node of Y, when Y goes offline, X will consider as its new successor Z because this is actually its second logarithmic link, after its successor. A node disconnection causes the rebuilding of all the logarithmic links of each node of the network.

To decide when to stop the logarithmic link building, each peer must check if the last link added to the routing table does not point to a node already connected. A detailed discussion of these aspects is beyond the scope of this thesis. Empirically, if N, the number of peers, is known the ideal number of logarithmic links per peer is $logN$.

Propositions 4 and 5 give the guidelines to develop the Algorithm 4, which derives logarithmic links from neighbors in a logarithmic number of steps with respect to the number of peers in the network. It also follows the guidelines issued by Proposition 5. The Algorithm 4 is based on the function $requestMySUC()$, which returns the successor of the caller, that is the post-order S–Peer successor in the ring. The Algorithm 4 assumes, like Proposition 5, that $requestMySUC()$ returns a reliable value for all callers. The consistency of all the logarithmic links greater than 1 depends on this assumption and on the cooperation of all peers.

Algorithm 4 – Build logarithmic links

requestMySUC() – return caller successor
requestLogLink(X, Y) – request at peer X its Y-th logarithmic link

Initialization
$selfG \leftarrow G$ of the region R of current *S–Peer* that start this procedure
$orderedLogLinks[n] \leftarrow \varnothing$
$orderedLogLinks[0] \leftarrow$ requestMySUC()
$i \leftarrow 0$
$nextLogLink \leftarrow$ requestLogLink($orderedLogLinks[i]$,$i+1$)
while $orderedLogLinks[i] \prec nextLogLink$ **do**
    $orderedLogLinks[i+1] \leftarrow nextLogLink$
    $i \leftarrow i+1$
    $nextLogLink \leftarrow$ requestLogLink($orderedLogLinks[i]$,$i+1$)
end while

The Algorithm 5 implements the $requestMySUC()$: the caller requests its successor to its parent. The $requestMySUC()$ function generates the message msg and sends it to the parent. Any other peer that receives that message must execute the same function. A mechanism of recursive descendant calls is triggered: starting from the latter, each peer, through its structural links, forwards the request to its neighbor peer that promises to be the best successor of the first caller. Each peer performs this mechanism unless it finds itself to be the right successor of the first caller, in that case it communicates its own address directly to the first caller. Figure 4.8 shows an example: (1) the node C asks to its parent, node A, which is its successor; (2-4) node A explores the part of the tree on the right respect to node C and finds the bottom-left node in it, that is node F; (5) node F communicates its own address to node C as its successor. The number of hops is sub-logarithmic with respect to the number of peers in the network. The reliability of the function $requestMySUC()$ is related to the structural links reliability, which is controlled by G-Grid structure algorithm.

Figures 4.13, 4.14 and 4.15 (at the end of this chapter) show an example of how each peer in the network builds its logarithmic links. All peers run the Algorithm 4, but the correct knowledge is not distributed among all peers after the first iteration. Several iterations are needed because each peer at each iteration exchanges information with a small subset of peers,

Algorithm 5 – Find successor - requestMySUC()

Input: msg

Initialization
$selfG \leftarrow G$ of the region R of current *S–Peer* that receive msg
$links \leftarrow$ all links of the routing table

$supSubset \leftarrow$ elements of $links$ that have $G \in]msg.requestG, selfG[$ by using post-order relation
if $supSubset = \varnothing$ **then**
    send a new message to address $msg.senderPhysicalAddress$ notifying that $successor(msg.requestG)$ is $selfG$
else
    order ascending $supSubset$ by post-order relation
    forward msg to physicalAddress(the first element of $supSubset$)
end if

whose cardinality is logarithmic with respect to the total number of network peers. The Algorithm 4 runs at periodic intervals and the overall speed of knowledge growth is influenced by the order of execution of the algorithm in the different peers. The system is completely decentralized, so we can not make any assumption on the order in which the operations are executed by the different peers.

Experimental results show that regardless the execution order of the Algorithm 4 the number of iterations necessary to create the full set of logarithmic links is $O(logN)$, where N is the total number of network peers.

Figures 4.13, 4.14 and 4.15 denote the convergence speed of the system to a valid global structure and give a hint for the reason of the logarithmic behavior. Individual peers can not build all its logarithmic links, but through cooperation, with a low number of interactions – logarithmic with respect to the number of peers – we can create an isomorphism of Chord ring on G-Grid.

In summary, the cooperation is essential to balance the load. Since P2P networks can be highly dynamic, each peer must updated periodically its links. The execution of Algorithm 4 is important not for the individual peers, but for the global network load balancing. The maximum number of hops for successor finding is proportional to the tree depth – bounded above to 2 times the tree depth. The same applies for the maximum number of

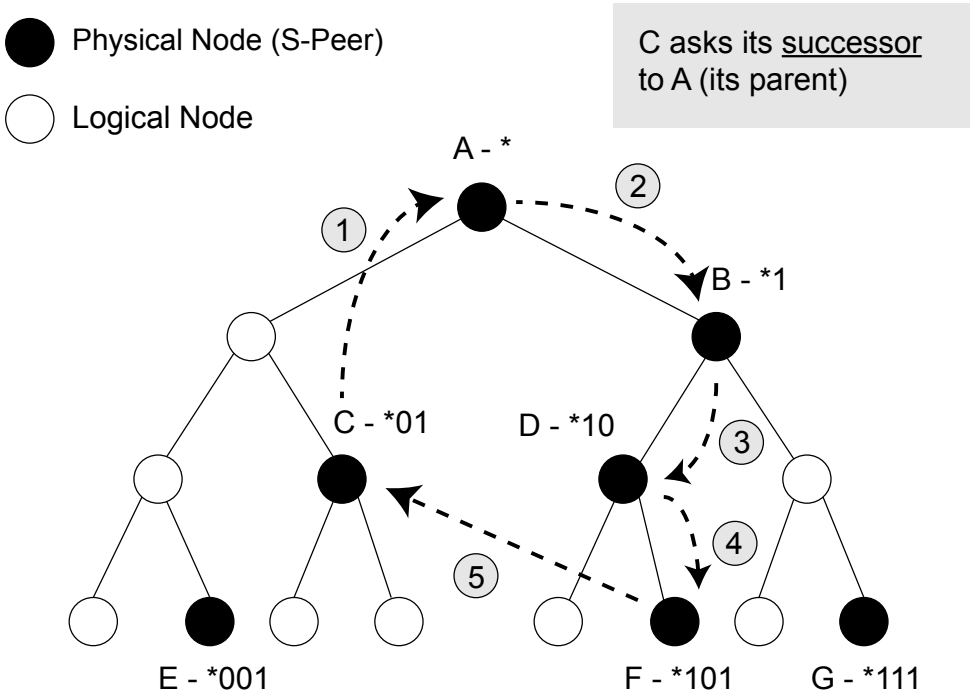


Figure 4.8: Finding the successor node in the ring by G-Grid structural links.

hops to build the logarithmic links.

4.2.4 Peer Joins and Exits Into the Network

The join and exit of peers, as we have seen, require an effort to maintain the system load balancing. The dynamism of the network can also invalidate part of the links and thus increase the number of hops necessary for message routing.

In the original G-Grid definition, the join of a new peer does not require a significant modification of the overlay network structure. If a peer X sends a request to a newly joined peer Y (i.e. before all the links are set for it), the search could stop to an ancestor of Y, because many peers are still not aware of the existence of Y. This fact has a negative impact on performances since it pushes part of the load towards the root.

Exits are even more problematic because they make some links outdated. A peer that exits the network notifies its intention only to their close neighbors in the tree, those that are bound by structural links. Notify the exit by broadcasting would be impractical for scalability. In G-Grid Chord every time that a peer joins or exits the entire Chord-like overlay structure must re-build all the logarithmic links among peers. In G-Grid, the only way to decide the elimination of an outdated link from the routing table of a peer is by direct contact, after the expiring by timeout of several attempts of sending a message.

If G is the binary string representing a region of the G-Grid tree structure, $Left(G, i)$ is its left prefix of size i, with $i \in \mathbb{N}^+$. For example, if (G, L) = *010100110 with L = 9, then Left(G, 3) = *010. Let us suppose that the peer X sends a message to the peer Q to find a datum in its query region G_Q. If X receives the response from Y, instead of Q, this means that Y – that has the query region G_Y with $Length(G_Y) < Length(G_Q)$ – contains the query region G_Q. At this point X must update its routing table. In particular, X must delete all the links to peers that have $Left(G, i)$ equals to $Left(G_Q, i)$ with $i > Length(G_Y)$ and $i < Length(G_Q)$, that are all the nodes directly included between Y and Q.

More information can be extrapolated from the responses to queries and from the algorithm which maintains the logarithmic links. Every time a peer makes a query, that message performs a certain path to the recipient, who processes the response and sends it directly to the sender. The answer message, in addition to the requested data, has two important parameters: the G of the peer that actually responded to the query, and the query region of the request. These attributes reveal the peer that manages effectively the query region investigated. Hence all the links on the routing table of the peer sender have prefix of the query region binary string, with an L greater than that one of the peer responded to the query, these links are outdated because otherwise the answer would come from one of those query regions.

Figures 4.9 shows an example: if the region query is *1011 and *1 answers, then the links to the query regions *10 and *101 in the routing table are outdated. This information is obtained from the queries, which, together with the learning mechanism, are a vehicle for knowledge of the network status. From these statements we can affirm that the greater the traffic in a G-Grid network, more knowledge of the network spreads and enhance the performance of load balancing and reducing the number of hops of requests. However, the greater the hops, the more widespread is the knowledge of the

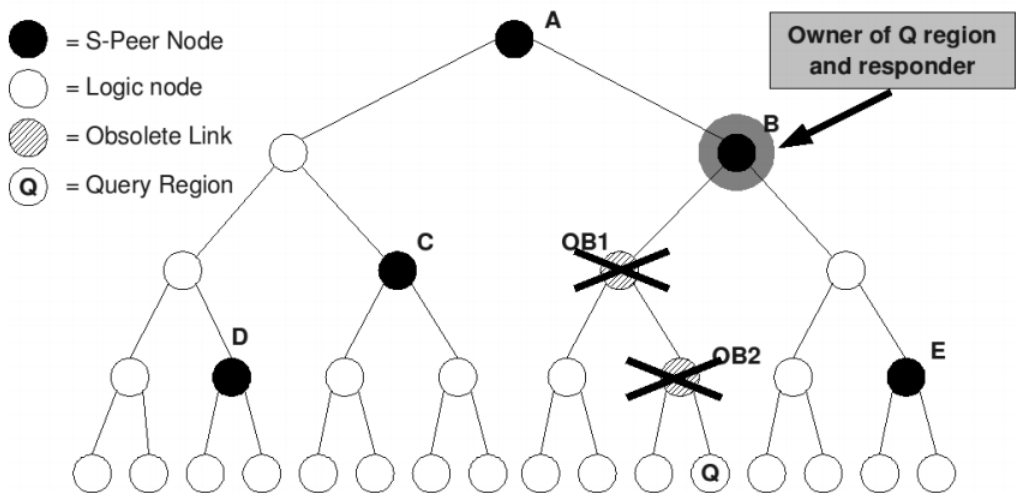


Figure 4.9: Discovery of outdated links from queries.

network status. By the experiments results we see that these two aspects are in balance.

In addition to query answers, we can increase the detection of outdated links using the management algorithm of logarithmic links. The first logarithmic link to a peer is the link to his successor on the ring. We have previously shown that this link is quite reliable, so all the links in the routing table that are between the G of the peer itself and that one of its successor are, with high probability, outdated and thus will be deleted. For example, if *1111 receives as successor *1, then *11 and *111 are considered outdated.

Detecting the outdated links is much more difficult than finding a new peer, so these mechanisms will be reinforced in further work, in order to make network knowledge more consistent for peers. The system would be more robust in scenarios with a high dynamism rate, when the inputs and outputs of the peers are frequent. We have obtained, however, good results with these techniques for scenarios with a sufficiently high dynamism rate. The more the network knowledge for peers is extensive and correct, the lower the rate of outdated links in the system, and consequently improves the message routing.

4.3 Experiments

In this section we show the performance results of our proposal in four simulated application scenarios. The simulator is used to test the traffic distribution on large-scale networks of 10,000 nodes over PeerSim [87], a testbed for large-scale P2P networks. These are the measures that we analized in the experiments:

- total number of system messages, which indicate how the system must work in addition to keep the overlay with respect to the useful traffic;

- number of hops, which tells us how average load generates a query on the network, how heavy is an average of a query in terms of traffic. Decreased by the presence of structured overlay;

- relative standard deviation (RSD), to measure the traffic on each node. To compare different situations due to different systems, we display the RSD, which is the ratio between the standard deviation and the average number of messages routed from each node of the network.

The traffic distribution parameter is very important because it shows how the traffic is evenly distributed on network peers. We observed also other important parameters such as the number of hops, the number of messages exchanged to maintain the overlay structure - called system messages -, due to the architecture of the data structures.

We show the behavior of the new overlay structure we described above and implemented in the simulator, called G-Grid Chord. As a first element of comparison, we selected the recent (2011) SkipCluster algorithm [142], as proposed by Xu et al., and a new version that implements in addition the overlay HiGLoB [132]. We applied the features of load balancing of HiGLoB, described in [132], to balance the number of peers contained in each cluster.

The experimental environment is a Linux virtual machine with 6 CPUs at 2GHz and 6GB of RAM for the simulations. The program is developed in Java, as the simulator PeerSim. We have defined a class that implements the interface of the node provided by PeerSim, which manages events such as the arrival of messages or internal events defined by the user. In our experiments, for each node, we connected the overlay to be tested and another level of transport provided by the simulator natively. We have also created controls that the system runs periodically, according to a configuration file, allowing

the generation of query, the connection / disconnection of nodes, and writing a snapshot of the network in a database. Each parameter, nodes and controls, is read from the configuration file. For any other technical details please visit the website of the simulator http://peersim.sourceforge.net.

In all scenarios, we start from a single node network and connect immediately after 10,000 nodes. The nodes create the desired structure by exchanging messages with other peers in the network. When all nodes joined the network we start to run 100,000 random queries. We observe the system state and store it into a database every 10,000 queries. While these parameters are common in every scenarios, the differences among the scenarios are listed below.

1. The first scenario includes only query executions. The network structure does not change after the last node join. The number of nodes reaches 10,000 linearly and remains the same for all the rest of the simulation.

2. In the second scenario, after the initial configuration of the network, one peer joins the network every 100 queries. This behavior increases the starting network (10,000 nodes) by 10%. This is intended to test the behavior of a growing network, intuitively, new nodes could introduce longer search paths and increase the number of system messages;

3. In the third scenario, after the initial configuration of the network, every 100 queries, the simulator randomly adds or removes a peer with 50% probability. The number of nodes fluctuates around the starting value of 10,000. This is intended to give additional stress to the system, because peer removal changes the structure and can impose longer path traversals;

4. In the last scenario, every 100 queries, there is a number of peer join equal to the square root of the number of queries run up to that moment. This is in some way similar to scenario 2, but with greater growth.

In every simulation we assume a uniformly distributed load in the network nodes.

4.4 Results

It is worth to observe that, despite the differences among the various scenarios, the results were really similar. For this reason we will comment only the results of the first scenario. We also want to emphasize the improvements introduced by G-Grid Chord, without dwelling on the particulars of the networks, since our experiments show that these improvements are guaranteed in a wide range of operating condition.

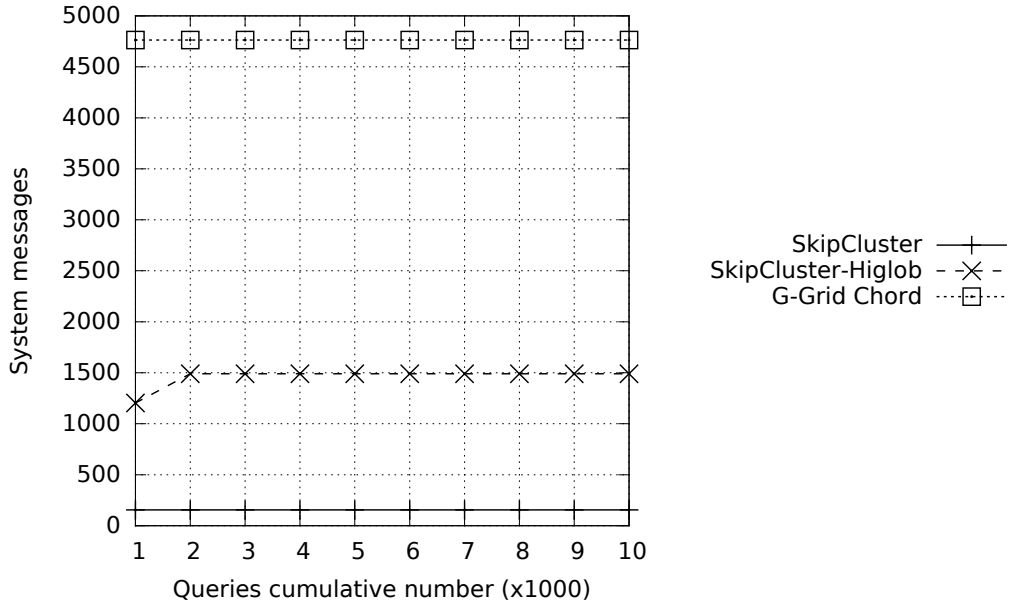


Figure 4.10: Overlay structure messages traffic.

In the Figures, the horizontal axis represents time, and the marks are the cumulative numbers of queries issued in the system during the experiments.

In Figure 4.10 the vertical axis represents the total number of messages issued in the system in the unit time, as a result of the querying activity, to maintain the overlay structure. The basic version of SkipCluster requires few messages, less than 200 in a 10,000 nodes network, to create the overlay structure, while the introduction of HiGLoB increases considerably that value. SkipCluster with HiGLoB requires more overlay messages than basic SkipCluster, because the HiGLoB internal structures keeping/updating the histogram of the clusters requires a heavy exchange of overlay messages. G-

Grid Chord requires a bigger number of system messages (about three times SkipCluster with HiGLoB) because of the large cost of maintaining the ring structure.

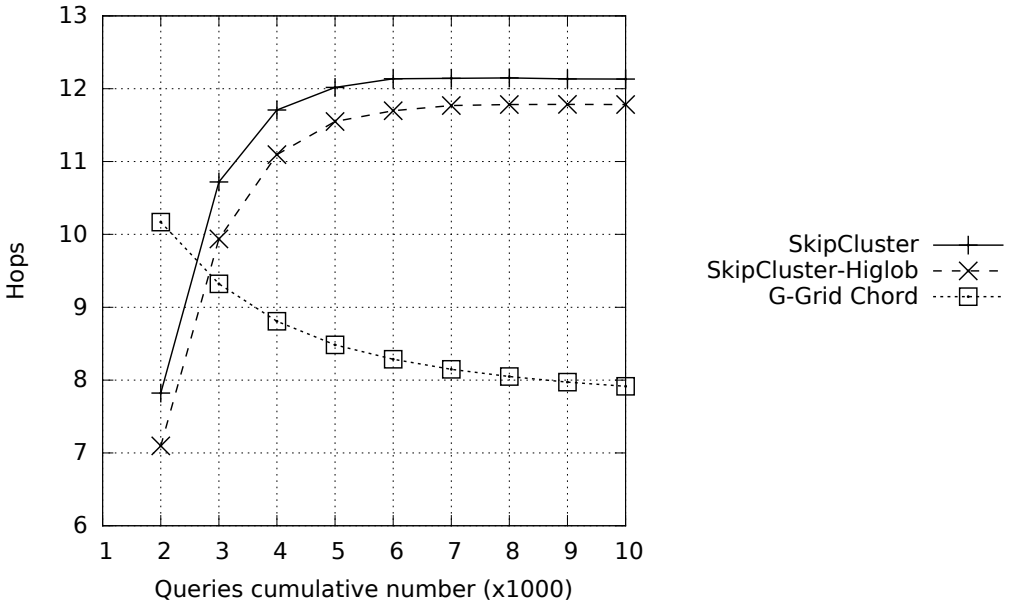


Figure 4.11: Average number of hops per peer.

Figure 5.5 shows the average number of hops per peer. Each peer measures the number of necessary connections to obtain the wanted data. During the simulation starting phase, the number of query messages sent by each peer is very low, so the measured value becomes interesting only after the initial phase of adjustment. When more queries are executed, we reach the standard operating condition. Both versions of SkipCluster have a hops average slightly less than the logarithm of the network size. The important result is that G-Grid Chord improves by one third the already good result of SkipCluster.

Figure 5.6 shows the Relative Standard Deviation (RSD) of the traffic in the peers. We observe that both SkipCluster overlays have the same slowly increasing behavior, around 15 for the base version and 16 for HiGLoB. The G-Grid Chord value is around 2: this indicates that the traffic with G-Grid Chord is much more balanced among nodes. In particular, from the simulation, it comes out that the 95% of nodes has a load less than 160 with

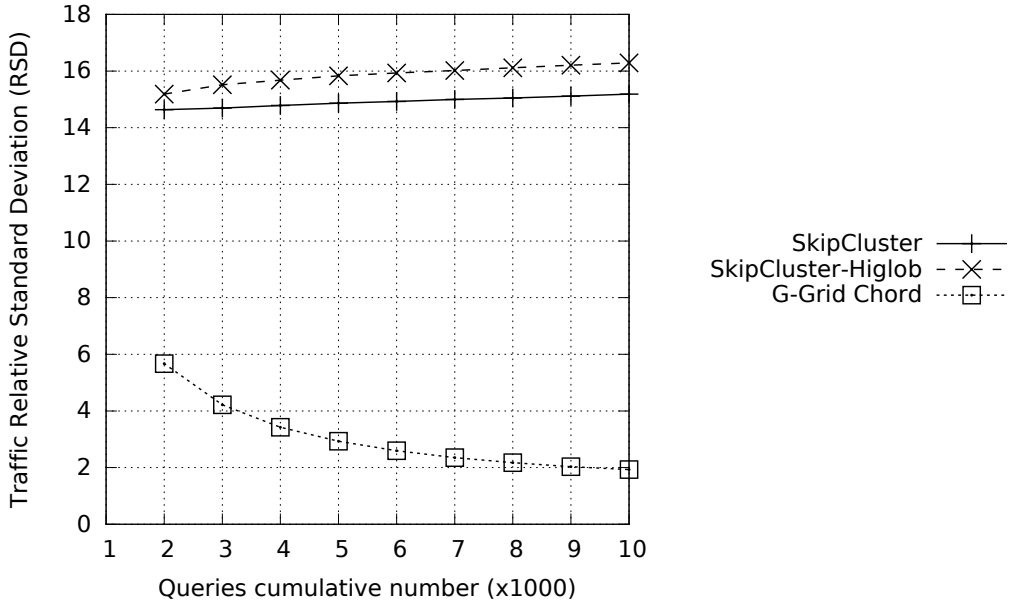


Figure 4.12: Traffic standard deviation for peer.

SkipCluster HiGLoB and 213 with G-Grid Chord. In relative terms, dividing by the respective load average, we find that 95% of nodes in SkipCluster have a load under 1.45. Analogously, the 95% of nodes in G-Grid Chord have a load under 2.73. If I want to ensure an adequate dimensioning of the system I have to size the channels to cover at least 95% of cases, with SkipCluster I have a much greater need to support the traffic because it is not predictable.

From the results obtained we see that the load distribution is much improved compared to other overlays, and tends to a lower value. Moreover, the average number of links, S–Peer and C–Peer networks, which have nodes is greater than the other overlays: this means that each peer has a greater knowledge of the network status.

We have set up an overlay that allows to update the routing tables of each peer, reducing by one third the average number of hops. This overlay system is not bound to the binary tree structure, so it can exploit shortcuts, avoiding the crowding of high levels of the tree, those close to the roots, and prevents load imbalance as saw in RSD results.

Regardless of the model used, we can see that increasing the number of peers, the load distribution improves and this is a good feature in terms of scalability. However there is still work to do in terms of memory because now the peers hold a number of connections that tends to be linear with respect to the number of peers in the network. For these considerations, we can say that this work has laid the foundations for better load distribution in P2P networks and make G-Grid a valid framework for the development of distributed databases and systems.

The next step is to find a way to improve the structure management without losing the load balancing properties. In the next chapter we propose a further evolution of G-Grid that achieves this goal.

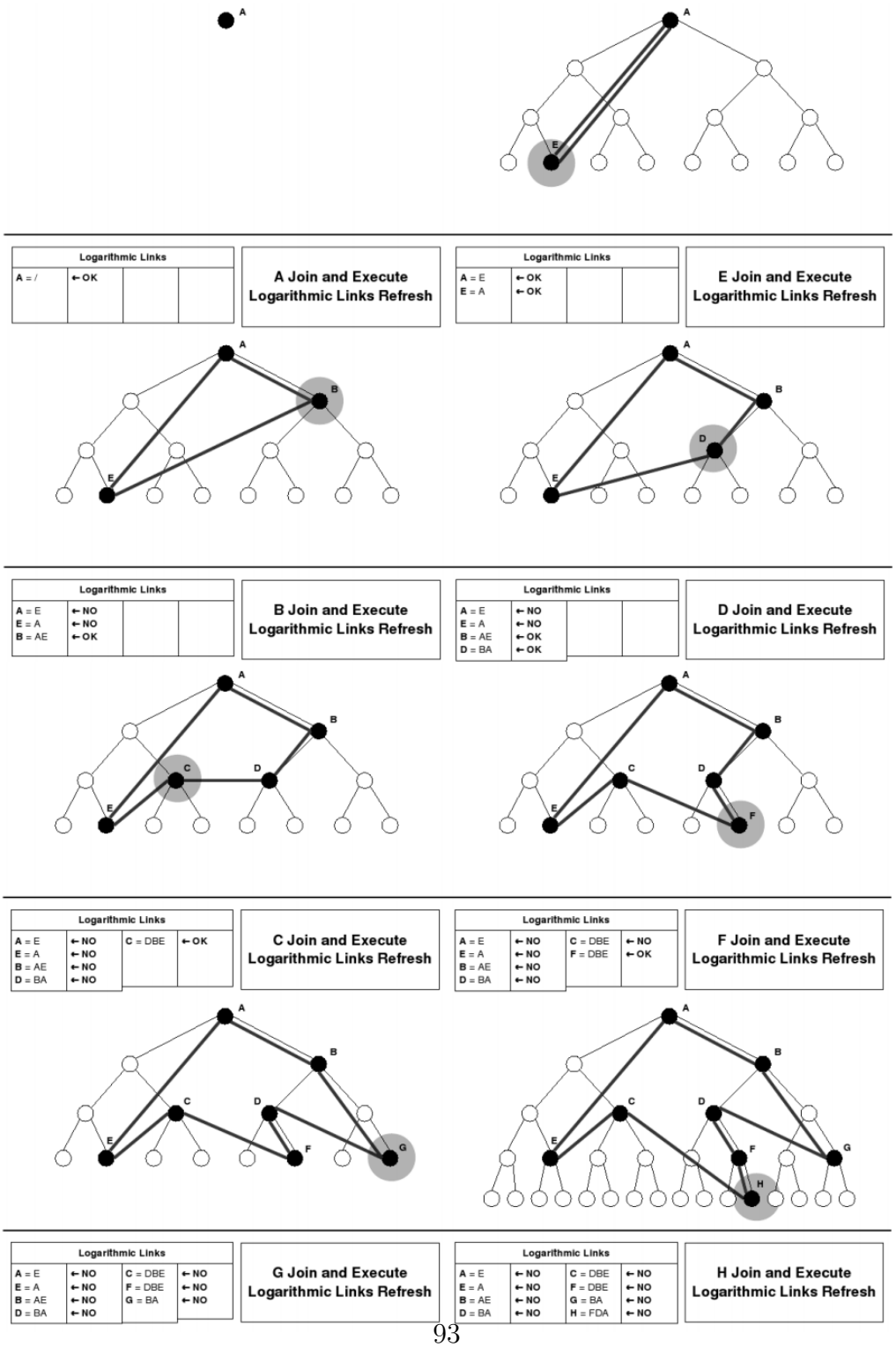


Figure 4.13: Example of logarithmic links construction – incoming peers.

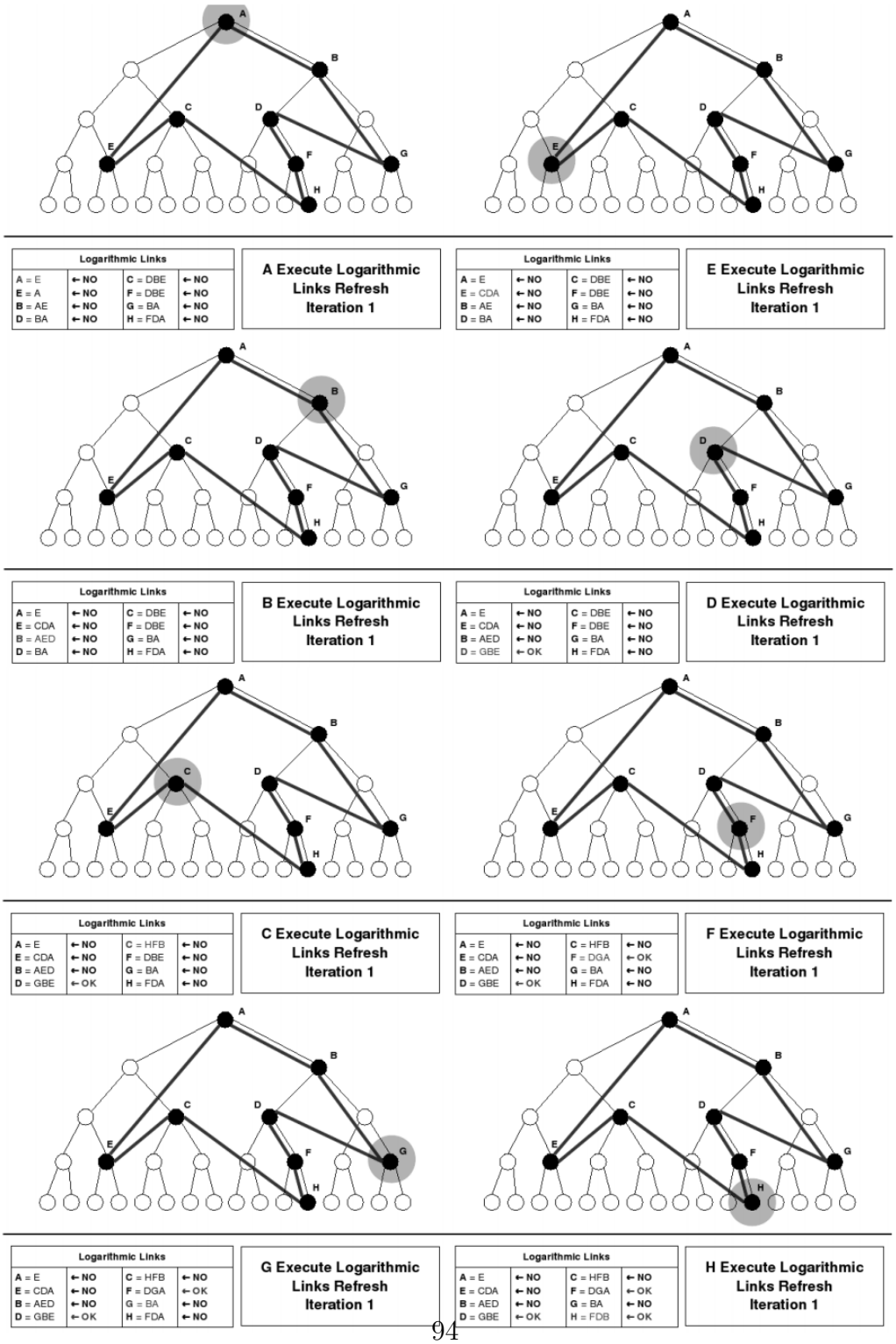


Figure 4.14: Example of logarithmic links construction – iteration 1.

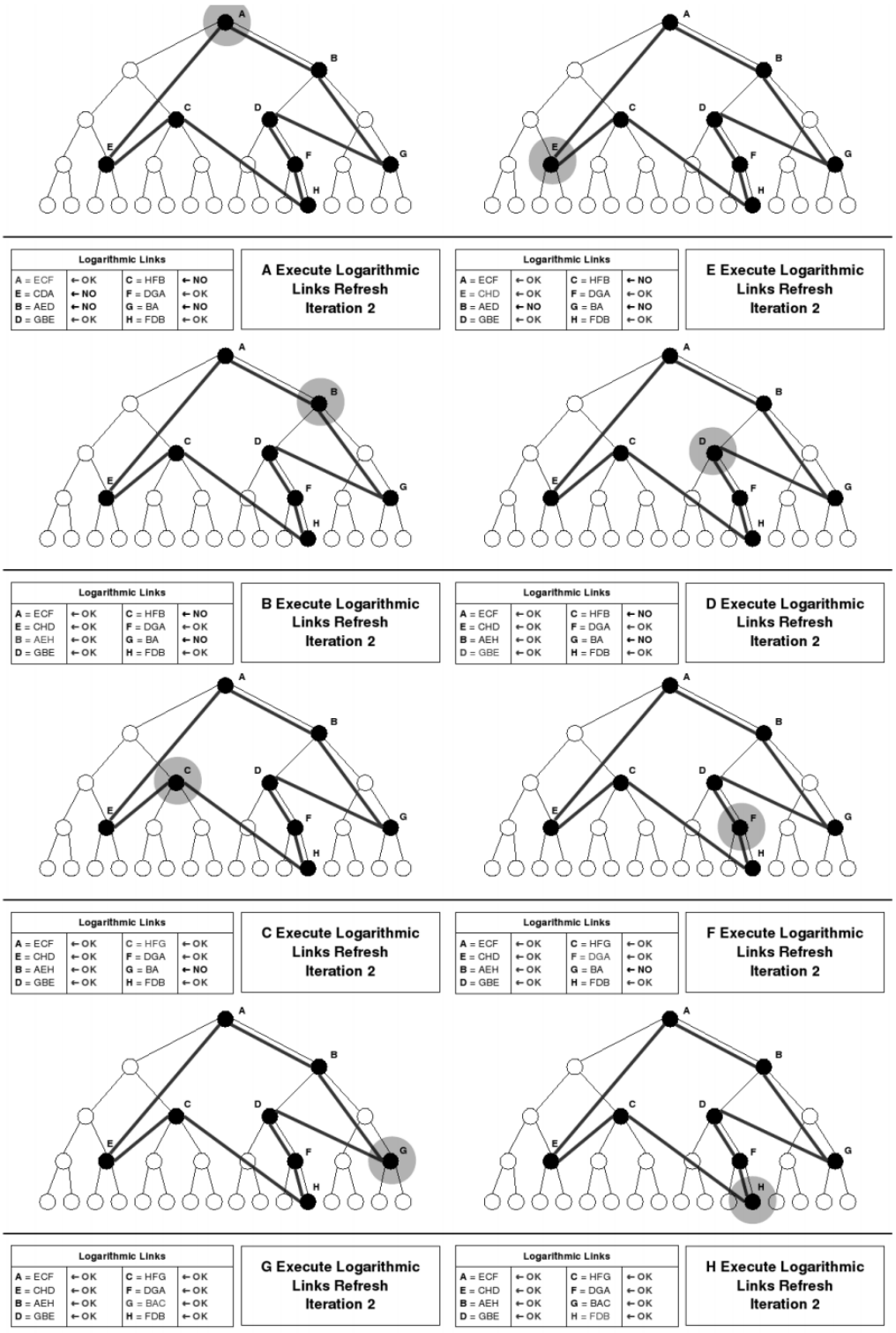


Figure 4.15: Example of logarithmic links construction – iteration 2.

Chapter 5

A Bit of Randomness to Build Distributed Data Indexing Structures With Network Load Balancing

The information request has become a driving force for the global economy and academic research. This has led people to pool data individually owned before, to create huge communication and sharing networks. A network architecture that fit conveniently to these requirements is the P2P architecture: a set of users that are all at the same hierarchical level, unlike client-server systems. In P2P systems, each user can provide its own data, and can leverage the network to search for data shared by other users.

The increasing use of P2P systems has attracted the attention of researchers in computer networks that started to observe their evolution and to make improvements. Users growing, and then information increasing, they observed the main problem is to get data, if there are present in the network, in a finite and short as possible number of operations. When a user requires an information that does not own, she sends a request message for that information along the network. In a modern network, where the information are highly distributed and the devices are a lot, the requests messages creates a traffic load that limits the bandwidth. For this reason, most of the latest P2P architectures are structured and organized in order to avoid broadcast and limit the number of steps (hops) needed to achieve the information into the network. A network structured scheme, that we can call *overlay*, allows

to keep the number of hops in the system approached to the logarithm of the number of network nodes. Maintaining an organized overlay is necessary for the network peers, that are connected users, to get the information in a efficient way. However the structure maintenance needs an additional message exchange over the network, which produces network traffic. In addition, each peer joins into or exits from the network forces an overlay update that can need time and extra traffic load. The overlay messages must be limited despite the peers' dynamism: this feature can be achieved optimizing the number of hops for each message [132, 68].

Another evaluation parameter is the traffic balancing over the network. Any peer could have more information than others, or simply highly requested information, focusing most of the traffic on the network sectors that maintain these data. This real and recurring situations lead to bottlenecks in the network, due to that peers have to satisfy much many requests then others: their own resources would be committed for a long time and slow down the response time and the global performances of the system.

We present an evolution of the work presented in the previous chapter, which leads to the proposal of a new structured overlay, called G-Grid Small World[91], that balances the network traffic load on P2P systems with a number of overlay messages that does not condition the network load. This feature is important for the system scalability. We introduced a randomness level in the overlay structure that facilitates and reduces the maintaining of network links among peers. These improvements make G-Grid architecture an excellent solution to build decentralized self-organizing P2P systems. The applications can be from the media streaming to the sensors networks, from the company intra-organizational networks to the inter-organizational networks. We show some experiments in which we compare our P2P structure with the previous ones and with other literature proposals to demonstrate the improving of load balancing, as well as enhancements in main features, such as the number of hops and the cost of structure maintaining.

5.1 Related Works on P2P Load Balancing

The P2P networks expansion focused researchers' study on their architectures, in order to build more efficient systems improving some technical features. The most advanced overlay structures [115, 7, 65, 100, 90] have different characteristics from each other, some are completely structured, others

less, leading to different evaluation parameters.

Fuzzynet [44, 38] is a semi-organized mono-dimensional overlay network, based on the theory of small world networks [136, 61, 67]. The algorithm used to create the small world network is the Oscar algorithm [43]. Oscar is an algorithm applied to data networks with non-uniform keys distribution. Fuzzynet introduced changes concerning the entry and search of information in the network. The search of information is made via a "Greedy" algorithm that should return the right solution. However, the Greedy algorithm could not find the requested information since, by definition, it could stop the researches with negative results – information not found – if it does not find steps for improvement. The success rate indicated by the developers is 96%. The entry of a datum occurs in two stages. First it determines on which node saving the datum, then it propagates backup copies of the datum in the neighbor nodes. In this way, the requests for that datum must not reach a single specific peer but can return the first found replica. Since this has problems with information consistency in the various replicas, the authors proposed to assign a time-to-live to data. This overlay leads low costs of peer connection – the creation of links – and zero cost of peer disconnection.

Mercury [14] is an organized overlay network supporting multi-attribute range queries. Mercury organizes its data by creating an overlay ring, called "hub", for each attribute. Each data has a number of copies equal to the number of its attributes, each saved in the respective hub. The requests start from the more binding attribute in the generated query. Each node within the hub has links with next and previous ones in the ring and a set of long-range links. When a new peer joins the network chooses randomly a hub to connect itself in the relative ring structure. When a peer wants leave the network have to restore before the ring of its hub. A problem common to all these systems is that the work load is unbalanced [9].

A proposal to solve the problem of load distribution is HiGLoB [132], that stands for Histogram-based Global Load Balance. This framework provides guidelines on how to divide the data in a network to balance the work load. HiGLoB is constituted by two elements: the histogram that shows how the load is divided on the network nodes and the load balancer. Each peer divides the network into many areas as the number of its neighbors, without overlapping. Each network peer creates its own histogram with the zone information gathered from their neighbors, such as data load, traffic distribution, or other parameters that you want to distribute fairly. Then it sends information to the neighbors about its parameters to update their

histograms. Depending on how evolves its own diagram, peers may change zone to balance the network. This framework leads to an increase of the overlay messages exchanged between the peer. To limit this increase the authors proposed the inclusion of a threshold for the cascade histogram update.

In G-Grid, when a peer want to send a message to a peer at the same depth in the tree, the request must go up the tree to the root and then descend down to the recipient peer. This causes more traffic in the nodes close to the root respect to the peers of the tree leaves that are contacted only by messages that have them as recipients.

SkipCluster also presents the same problem. Each super-peer, the only peer capable to forward messages outside the cluster, has to manage more messages than cluster internal peers, that pass only intra-cluster communications. As we can see from the results provided in [142], the traffic distribution depends strongly on the cluster size.

Mercury aims to improve the traffic distribution building long-range links, not at random, but with more unloaded peers. For this, each peer need to know the traffic distribution of the network. It can do that by sending random walkers that gather information on traffic load. More walkers are generated greater is the approximation of the traffic distribution in the entire network. In contrast, if there are too many walkers these increase the system messages number for the maintenance of the overlay. Besides the number also the frequency with which a peer updates its knowledge on the traffic distribution affects the cost of network maintaining.

To make uniform the resource utilization of peers we can analyze the load distribution on individual peers [38, 17]. The idea stems from the observation of bottlenecks due to non-uniformity of elements kept for each peer. For example, when peers divided areas of interest in a rather symmetrical manner, the majority of elements concentrates, according to their key, in a single area. This leads to greatest load on the machine that keeps such data. P2P networks also have stakeholders which can be very different from each others: in current networks the latest generation computers are side by side to a few years ago computers, and there are mobile devices that communicate with stable devices. The bottleneck previously mentioned is accentuated if the peer with the majority of the elements is an old computer with limited processing capabilities. The solutions proposed focus on particular variables, such as the peer storage, the bandwidth and computational capabilities. However it is not possible to unify simultaneously all factors, then in the first instance we try to be homogeneous the load distribution.

Consequently, a uniform load distribution would lead to a uniform traffic distribution in the network. In the above example, if the elements of a network were placed uniformly on the responsibility areas of each peer, the traffic would be balanced.

5.2 G-Grid Improvements

In this section we present the improvements to the G-Grid overlay structure we developed and described in chapters 3 and 4. Moreover, we need to understand how its behavior changes, operating on storage and structure, to improve the weaknesses and to achieve the last overlay version, that is a trade-off between efficiency and effectiveness.

5.2.1 Load Distribution in G-Grid

The critical point of the basic version of G-Grid is the load distribution. In fact, being a tree structure, the nodes closer to the root are in charge of the routing of a large number of messages. To better understand we describe in detail the path of the messages from the leaf nodes, to reach the root.

Each leaf node handles only messages that are addressed to itself and that it creates. The nodes belonging to the level up to the leaves collect messages addressed to them and routing the messages that leave and reach the leaf nodes. As we climb the tree, we see that the traffic increases on each nodes level, because, in addition to managing the messages in which the node itself is the sender or recipient, they must perform the routing of all messages of lower levels. So, the greater the number of underlying layers to a node, the greater the traffic routing that it have to manage. Up to the root node, which manages the messages of all the lower nodes, allowing to reach the opposite sides of the tree structure.

5.2.2 G-Grid with Learning

A first way to improve the performance of G-Grid, is to propose a solution which does not require additional management costs. The version of G-Grid "with Learning" has the ability to acquire the structure of the network itself. We need that every message sent on the network stores a list of the nodes

that it has gone through. In this way, each peer that performs the routing of the message may acquire the nodes contained in the list.

In a stable network, a large number of query messages would bring the entire system structure to all peers. In this way, each peer builds internally a copy of the G-Grid tree and thus has connections with all peers of the network. Moreover, the tree structure requires a few space of storage because a node identifier is made by the identifier of the parent node followed by a single digit, '0' or '1'. Knowing the entire network, each node can send messages directly to the recipient - routing unicast - without requiring routing information to peers intermediates.

Obviously, this situation occurs if there are a high number of queries uniformly distributed in the network. If a node is reached by a few queries is likely that it is not known to all peers in the network. Therefore, the number of hop, ideally equal to 1 (the recipient is contacted in unicast), is slightly higher due to the non-uniformity of the query generated in the system. The increase in the number of hops is not a serious problem, because from 1 to about 2 does not involve large delays in reaching the recipient of the message. The problem, once again, remains the load distribution. In the ideal situation, where all peers know the complete structure of the network, the traffic distribution would not be a problem. In fact, always using the direct connections, the messages is directly sent to the destination node, then, with a uniform distribution of query, we have a uniform traffic distribution in the network.

If a peer does not know the entire network, uses some intermediate nodes to deliver messages. The problem is accentuated during the entry of a new peer in the network. Initially, a peer just entered the network is a C-Peer and is connected only with an S-Peer, then will charge it with every request. This leads to a non-uniform load distribution. In areas in which the network has high dynamics, precisely when the node input frequency in the network is higher compared to the number of queries, the distribution is not uniform at all.

5.2.3 G-Grid with Chord Ring

The improvements made by the G-Grid with Learning decreases the average number of hops of the system, leading, however, to a non-uniform load distribution on the network. Since the uniform distribution of the load is a key point in distributed systems, we step back, starting from the basic version

of G-Grid, to show the improvements that focus on load distribution. We applied primarily an overlay to the basic structure of the G-Grid. We see the merits and defects.

The first overlay applied to G-Grid is Chord [125]. The choice of this overlay was guided by the results of good load distribution of literature [125, 27, 13, 111].

Chord allows you to arrange the nodes on a ring structure, sorted in ascending order according to the node identifier. Each node has a link with each of the peer that are located at an exponential distance from it, on the ring. That is, each node has the following neighbors, as nodes with a direct link: the next node on the ring, the second next node, then the fourth, the eighth, and so on, as seen in the example shown in Figure 5.1.

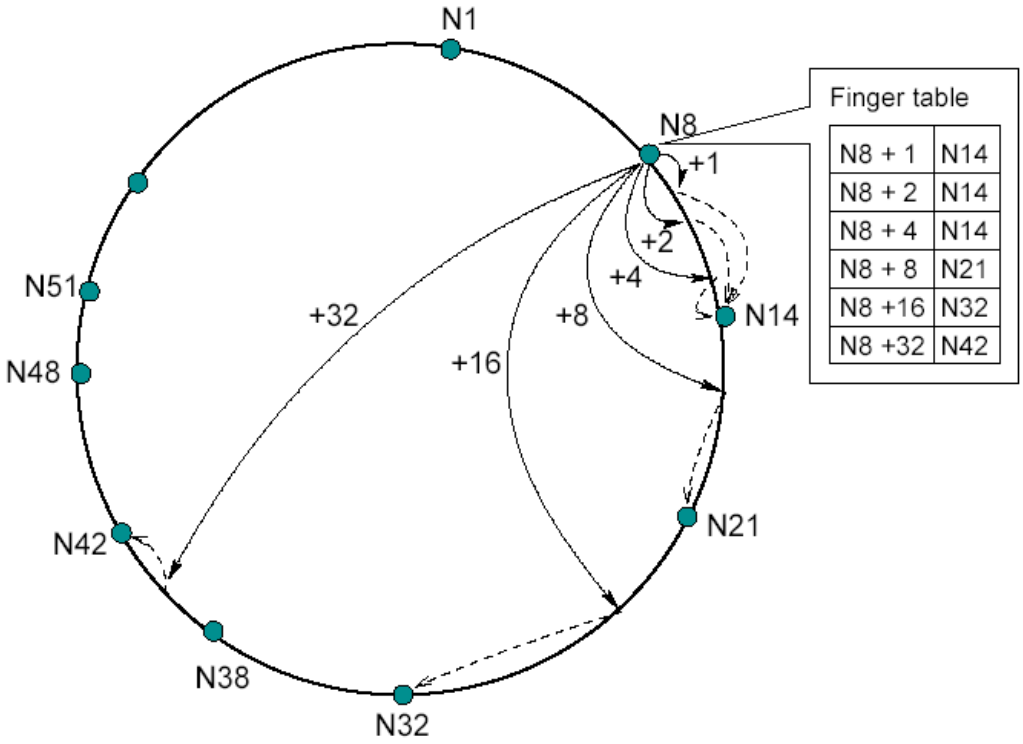


Figure 5.1: Chord network structure example.

In Chord, the routing algorithm allows to quickly reach the area surrounding the recipient of the message. The messages, in fact, are sent to

the farther peer whose identifier does not exceed the goal to be achieved, and so on, until the recipient peer. For implementation details of the Chord architecture the reader is referred to bibliographical notes.

We describe now how we have applied the Chord ring to the structure of G-Grid. First, we organize the peers as a ring. G-Grid has a binary tree basic structure, in order to transform the tree in a ring must define a node order in the tree. The peers that are part of the ring are the only S-peers. The order of the peers is chosen using a graph criterion: we order, before the left son node, then the right one, and then the parent node. The criterion is recursively along the tree, from the root node to the leave nodes. In Figure 5.2 we see an example of sorting the nodes of a G-Grid tree.

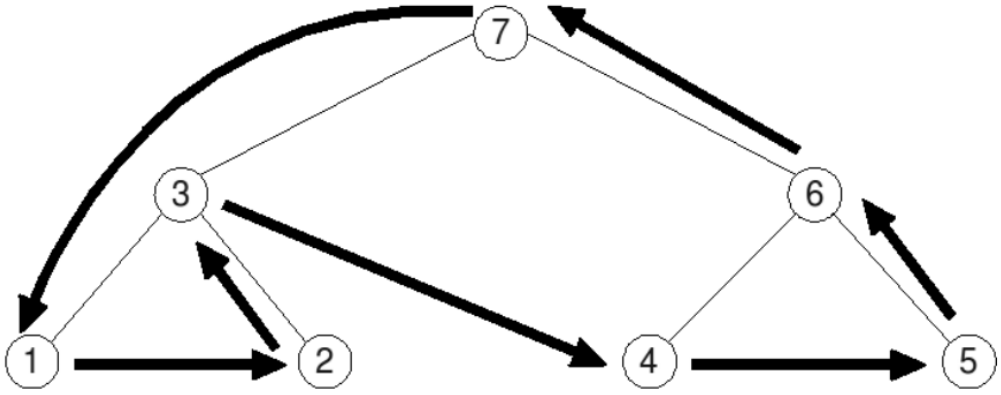


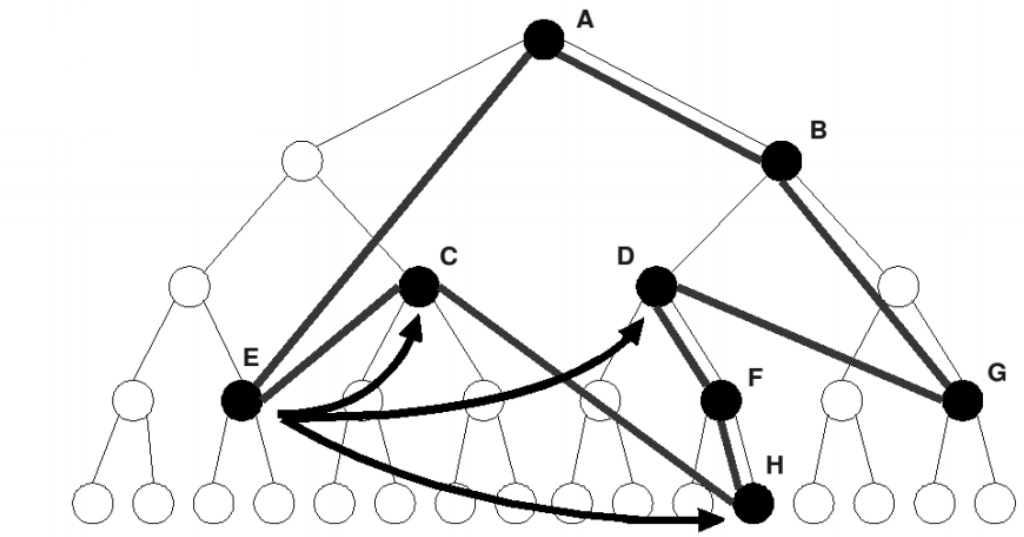
Figure 5.2: Sorting the G-Grid regions tree according the Chord ring.

Once you have defined the peer order is necessary to connect the nodes and create exponential links to create the Chord structure. Each peer, instead of searching one by one all the nodes that will become its neighbors, uses a more efficient mechanism to get this connections. Generally, for each peer, its n exponential neighbor is the $n-1$ exponential link of its $n-1$ exponential neighbor. In this way, when a peer find its first neighbor, to get its second neighbor, asks the first exponential link to its first neighbor. Then asks the second exponential link to its second neighbor to obtain the third neighbor, and so on. In this way the cost of routing table building of each peer that enters the ring has a complexity of $O(\log n)$, where n is the number of nodes of the network.

Thanks to this structure the system obtains an improvement in load dis-

tribution, as can be seen later in the result section. However emerges a critical point: the maintaining cost of the structure. Each time that a peer joins in the structure, this must first find its right place in the ring. Then it have to build the exponential links. The problem is the construction of the exponential links because it is not limited to the peer neighbors links, but to every peer of the network. In fact, the exponential links of the other peers may change after the peer join. All peers so have to check and recalculate their neighbors after the peer join. These operations should be performed even when occurs a peer leave. The maintenance of the structure creates a high traffic on the network. This critical point has led to a further overlay for G-Grid, to maintain a good traffic distribution with lower network maintaining costs, in front of the modern networks dynamism.

In Figure 5.3 is shown an example of G-Grid structure with Chord ring.



Node	*G*	Log-Link	Node	*G*	Log-Link
A	*	E-C-F	**E**	*001	C-H-D
B	*1	A-E-H	**F**	*101	D-G-A
C	*01	H-F-G	**G**	*111	B-A-C
D	*10	G-B-E	**H**	*1011	F-D-B

Figure 5.3: Overlapping of Chord overlay on G-Grid tree.

5.2.4 G-Grid as Small World

In 1967 Milgram [83] performed some experiments concerning the small world networks and their characteristics with respect to the low number of hops needed to connect two elements in a network with a high number of elements. The theory was reconsidered by Watts and Strogatz [136] in the Nineties, they emphasized the fact that the small world networks are very common in nature. The characteristics observed in these studies are mainly the low number of hops and a fair traffic distribution. The model developed for small world networks is perfectly suited to P2P networks, and the features presented is chosen to be developed as an overlay for the G-Grid architecture.

The key feature of small world networks states that every node in the network must be connected to some neighbor nodes and some other randomly chosen nodes across the network. We decided to determine the neighbor nodes as the predecessor and successor nodes of the ring created as for the G-Grid with Chord. The ring structure provides, in fact, the completeness in the network, because there is always at least one link to send a message toward the destination. Once established the neighbors, we add some long-range links to other nodes to create a small world overlay network. We used a number of long-range links equal to the logarithmic value of the number of peer in the network. To get these links each peer performs search queries randomly in the network. When a search query message reaches the destination node, this connects the sender node with itself and vice-versa. An important feature of small world networks is that the long-range links are equally distributed in the network. If the network is stable, this feature is respected since the peers create random queries with uniform distribution.

When peer joins in with the small world network, it has to be placed inside a ring of S-Peers, ordered as to the structure of G-Grid with Chord. Once inserted in the ring, the peer performs random search queries to get its long-range links. In the case of high dynamics of the network, the peer long-range links may not be evenly distributed in the network. In fact, when a peer enters the network, it is unable to be contacted from others peers with a long-range link of peers already into the system. To overcome this problem, the new peer has to create long-range links through query messages generated at the join phase. In this way, the peer creates links which helps to balance and distribute the load in the network.

The advantage of this network topology, compared to the overlay which is based on Chord, is the fact that the peer joins do not invalidate the connec-

tions of all the other peers in the network. In fact, long-range links randomly generated do not have a logical structure that have to be restored at each modification. This feature saves a lot of traffic system, in particular of those messages that were used to reconstruct the exponential links of every network peer. Even the operation of peer leave does not create an huge amount of traffic system. If a peer leaves the network, the remote peers that were connected with it, realized that the peer is no longer there, create randomly another long-range link to replace the lost one. The number of messages is still admissible. Finally we state that this is the best overlay to add to G-Grid because the peer join and leave peer operations do not involve cascade updates, unlike what happened in G-Grid with the Chord ring.

These benefits are visible in the experiments that we show in the next section, comparing all versions of G-Grid listed in this work to SkipCluster, with and without the HiGLoB traffic management.

5.3 Experiments

Among the main overlays of the literature we have chosen two structures for comparison with the G-Grid versions: SkipCluster and HiGLoB. These structures have been implemented in the simulator of P2P networks PeerSim [87]. In our experiments we connected to each node by the overlay specific test, such as G-Grid, and an overlay of transport provided by the simulator in native mode. We have also created the controls, the system runs periodically: for example, every 10 000 query is saved a network snapshot on a database.

SkipCluster was chosen because it is one of organized structures with the best performance among the recent ones in literature. Therefore we added to SkipCluster HiGLoB, an overlay that helps to balance the load distribution in the network, but increases the system maintenance traffic too. We also analyze all the versions of G-Grid, highlight the improvements of this work that led G-Grid Small World to be a good compromise between ordered structure and load distribution.

The key measure that we want to evaluate here is the traffic distribution. As we have already said, this parameter is important because it indicates how the load is evenly distributed on the network peers. We observed also other important measures as the average of:

- number of hops,

- number of messages exchanged to maintain the structure (called system messages)

- number of links of the peers, which indicates how many memory is occupied by the data structures of the overlay.

We tested the overlays in four different scenarios. In all the scenarios we start from a network consisting of a single element and connect quickly 10 000 nodes. The nodes create each structure by exchanging messages with other nodes in the network. At the end of the introduction phase, we begin to execute queries periodically, up to 100 000 queries. The observations of the state of the network are performed each query 10 000, so 10 times.

The first scenario presents only the query execution, without changing the structure of the network. In the second scenario, after the introduction phase, we insert a new node every 100 queries. This increases the size of the network, up to 11 000 nodes. In the third scenario we perform a new connection or disconnection, in a random way – with a probability of 50% – every 100 queries. In this case, the number of nodes oscillates around 10 000, the start number. In the last scenario, we perform a number of connections equal to the square root of the number of queries executed up to that moment.

5.4 Results

It is worth to observe that, despite the differences among the various scenarios, the results were really similar. For this reason we will comment only the results of the first scenario. We also want to emphasize the improvements introduced by G-Grid, without dwelling on the particulars of the networks, since our experiments show that these improvements are guaranteed in any kind of P2P network.

In the Figures, the horizontal axis represents time, and the marks are the cumulative numbers of queries issued in the system during the experiments.

In Figure 5.4 the vertical axis represents the number of messages issued in the system in the unit time, as a result of the querying activity, to maintain the overlay structure. The basic version of SkipCluster requires few messages, less than 200 in a 10,000 nodes network, to create the overlay structure, instead the introduction of HiGLoB increases considerably that value. SkipCluster with HiGLoB requires more overlay messages than basic

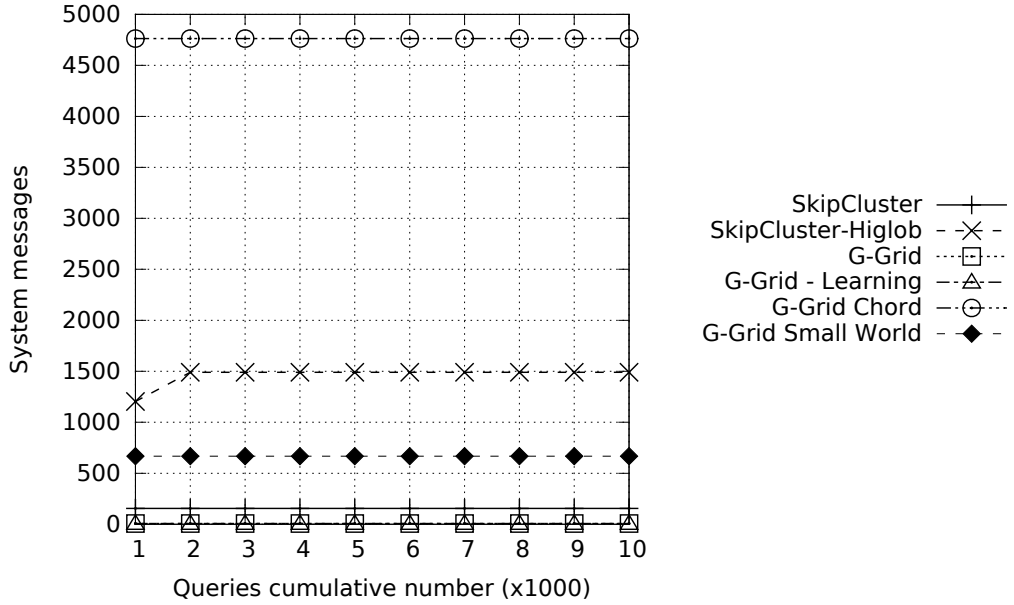


Figure 5.4: Overlay structure messages traffic.

SkipCluster, because the HiGLoB internal structures keeping/updating the histogram of the clusters requires a heavy exchange of overlay messages. We observe that the basic version of G-Grid has a low cost, almost zero – the value is around 3 –, due to the binary tree structure on which it is based. The management cost is found in the split operation of the regions leading to the creation of new S-Peers. The cost of G-Grid with Learning and basic G-Grid is the same because the list of passed nodes is sent in piggyback on query messages. We note that the G-Grid with Chord ring has the worst result in terms of system messages number, resulting the highest. This increases the cost of structure maintenance, because this structure is the most organized. Finally, we observe that G-Grid Small World needs a lower cost than other organized overlay tested, that are G-Grid with Chord and SkipCluster with HiGLoB. This because it has the ring structure and long-range links to maintain, which are easier to restore after network changes.

Figure 5.5 shows the average number of hops per peer. Each peer measures the number of necessary connections to achieve the wanted data. During the simulation starting phase, the number of query messages sent by each peer is very low, so the measured value becomes interesting only after the

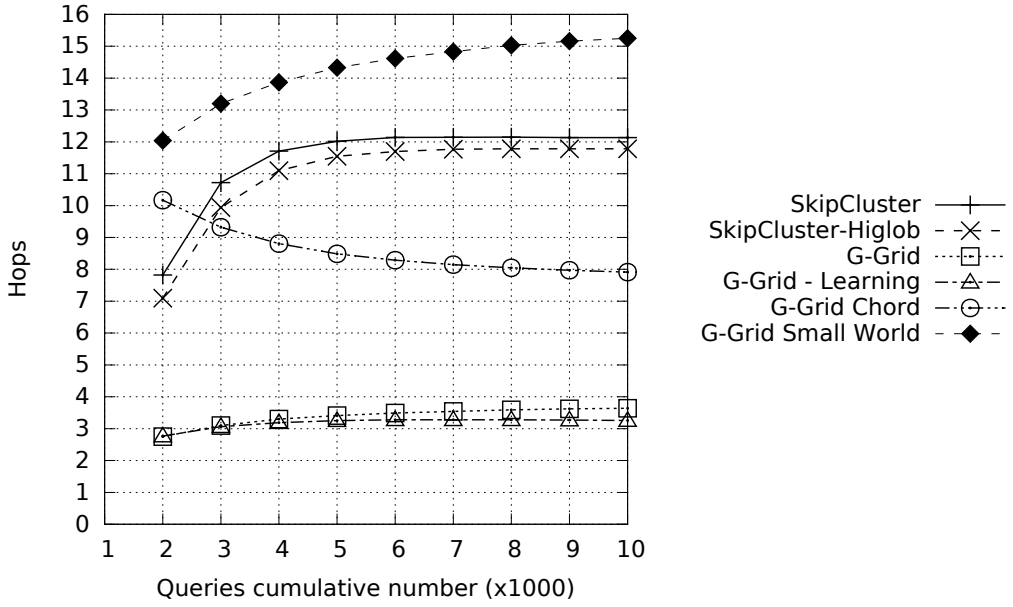


Figure 5.5: Average number of hops per peer.

initial phase of adjustment. When more queries are performed, we get closer to the expected average. We noted, in fact, that G-Grid with Learning, tends to the theoretical value 2. This value is justified by the fact that, since the network does not change its composition, with a large number of queries, each peer has knowledge of the entire network structure, so using only a step (hop) to reach the solution and one for the return.

Even G-Grid shows a rather low average number of hops. Both versions of SkipCluster has a hops average of slightly less than the logarithmic value of the network dimension. The number of hops of G-Grid Chord is greater than that of the other G-Grid versions, but still better, apart in the initial phase, of the other overlays. This is necessary to have a better load distribution, but in any case the data is still sub-logarithmic respect to the number of peers. G-Grid Small World here is the worst, because the randomness of long-range links does not help the routing as for the other overlays. However the difference is not so high to penalize this one.

In Figure 5.6 we measured the average traffic standard deviation for peer as the ratio between the standard deviation and the average of messages routed from each node of the network. We observe that after an initial phase

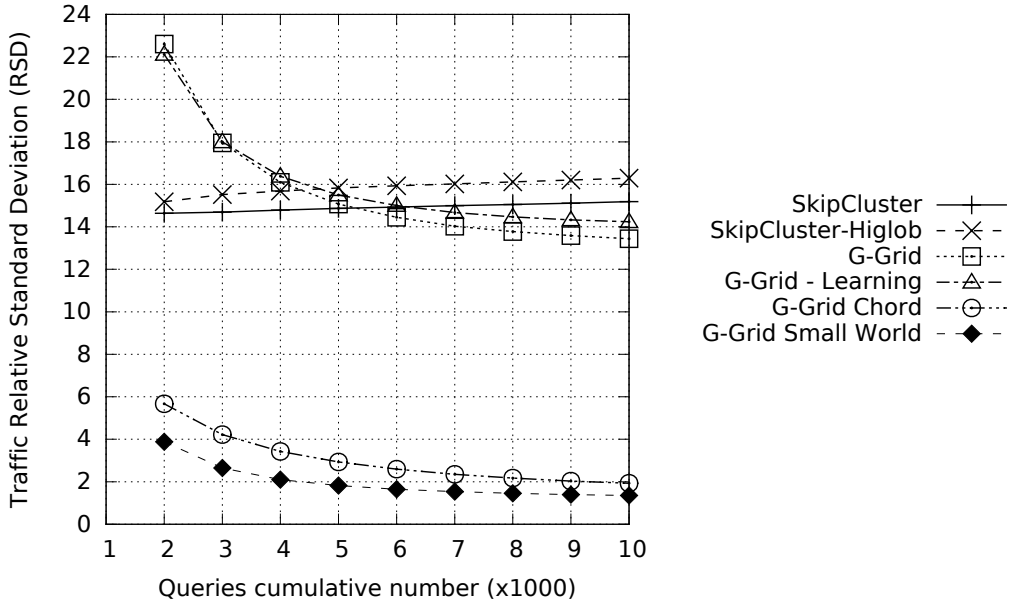


Figure 5.6: Traffic standard deviation per peer.

the traffic standard deviation of both G-Grid algorithms is greater than both SkipCluster algorithms. All overlays have a value that remains on the same order of magnitude, the average is around 15. The SkipCluster with HiGLoB distinguishes oneself from SkipCluster of about one unit. At least, G-Grid, and even better G-Grid with Learning, offers a lower average traffic standard deviation. We note that G-Grid Chord and G-Grid Small World are the overlays that have a better result. The value is about 1: this indicates a nearly uniform distribution of traffic.

In Figure 5.7 we obtained information on the amount of memory required for the maintenance of each structure. Large number of links necessarily involves high amount of memory to store them. In every scenario the situation is the same: G-Grid Random requires a high number of links. The number of links is not logarithmic with respect to the size of the network, because the peers had to store the links with the last entered nodes on the network to distribute more evenly the long-range links. This value is, however, a fixed value.

The worst overlay is represented by G-Grid with Learning, because the number of links increases as the number of executed queries. In fact, for each

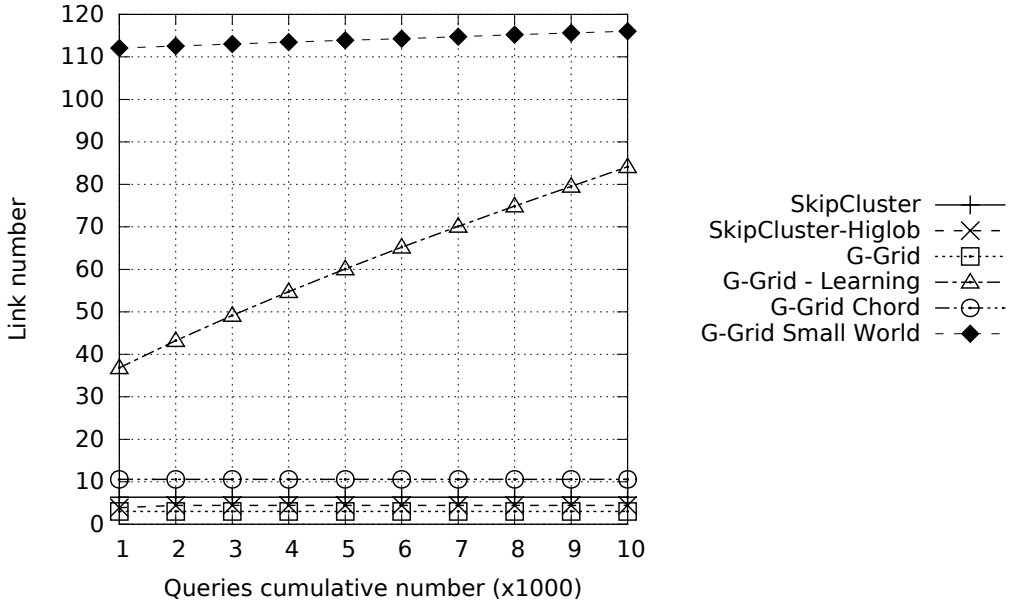


Figure 5.7: Number of links needed to maintain the overlay structure.

query, each peer discovers and creates links with the other nodes through which the query is passed. This leads to have a number of links equal to the size of the network, in the steady state. The upward trend of the links will arrive at 10 000 after a very large number of queries. The basic version of G-Grid has the least number of links per node due to its binary tree structure. At last we note that the introduction of HiGLoB in SkipCluster decreases the number of links of the structure by one third, close to the value observed for basic G-Grid. Moreover, we note that the number of links stored to maintain the structure Chord above G-Grid is approximately equal to the logarithm of the number of nodes of the network, as expected for construction.

Chapter 6

Conclusions

Modern devices and applications manage large data distributed over multiple sites that also can move independently. The P2P tecnologies try to provide efficient distribution, sharing and management of resources over autonomous and heterogeneous peers. Most P2P systems, such as Gnutella, Chord, Tapestry, support only queries over a single data attribute. We presented an evolution of G-Grid, a multidimensional data structure able to efficiently execute range queries in totally decentralized P2P environments. This structure is not imposed a priori over the network. Self-organization is led by local interactions among peers, and the total decentralization of control makes it perfect for grid computing and server clusters. G-Grid is also adaptive regarding dynamic changes in network topology. This robustness makes it suitable for mobile networks. We presented several experiments to compare G-Grid with other recent and efficient P2P structures presented in literature.

Analyzing the results of experiments, we can get some conclusions and suggestions for future works. First, we report the main characteristics of the analyzed overlays. The values obtained are proportional to the size of the network used in the experiments, each overlay changes its performance in relation with the peer number of the network. Starting from SkipCluster, we note that the cost to maintain the system is not high: about 1/50 of the size of the network. This means that the peer joins and leaves in the network involve a relatively low cost. In addition, the structure has an hop average equal to the logarithm of the number of peers connected to the network. The number of links required to maintain the structure is low, approximately equal to half the value of the logarithmic network size. The measure about the traffic load,

that is the ratio of the standard deviation and the average of the traffic load, is a logarithmic value of the network size. This last parameter is a weakness of SkipCluster, despite showing good performance compared to others.

SkipCluster with HiGLoB, shows an average number of links per node less than about 1/3. This value allows to maintain a smaller number of connections and therefore decreases the bandwidth needed. Nevertheless, this overlay has a memory occupancy greater than SkipCluster, because each node stores the histogram of the clusters, and equal to about 1/100 of the network size. Moreover, to update the histogram, this overlay uses a double number of messages with respect to those used by SkipCluster.

We note that the basic version of G-Grid is showing significant advantages over SkipCluster. First, G-Grid has a very low maintaining cost because the peer joins and leaves cause a change in only a node of the tree, that holds up to three links. The number of links is a superiority of G-Grid: this value is very low, due to the binary tree structure. The average number of hop is lower than that of SkipCluster and it is around the logarithmic value of the network size. The traffic distribution in G-Grid, however, gets the same performance of SkipCluster, showing the same problem of load balancing.

G-Grid with Learning shows the same characteristics in terms of system traffic, because the added information travels in piggyback on query messages. The improvement consists in reducing the number of hops, which, in a stable network, tends to one. The disadvantage of Learning is the number of links that grows linearly with the network size. This requires a large amount of memory in the nodes and a high bandwidth to maintain valid all the links. Even in this case the traffic distribution is unbalanced between the various peers of the network.

In G-Grid with Chord ring, we note that this overlay requires a huge number of system messages, approximately equal to half the network size. This brings a hop average approximately equal to half the logarithm of the number of network peers. The number of links per peer is equal, by construction, to the logarithm of the number of nodes. Although this version degrade the performance compared to G-Grid and G-Grid with Learning, we observe that the traffic is distributed almost equally, bringing the relative measure to 2.

Finally, G-Grid Small World was created to get the benefits of both G-Grid and G-Grid with Chord ring. The number of system messages is about 1/15 of the number of network nodes, so it is rather small compared to that required by the version Chord. The number of hops increases, reaching to a

couple of units compared to the logarithmic value of the number of nodes. The number of links also increases: it is 1/100 of the network size. This degradation of performance is offset by the traffic load measure that remains around 1. This indicates an almost uniform network traffic distribution. G-Grid Small World reaches the best traffic distribution with an acceptable decline of other measured performance.

The analyzes and considerations presented in this thesis have shown the features of G-Grid, in various versions, in comparison with one of the latest similar literature overlay, that is SkipCluster. In next works we will evaluate the performance in networks with uneven load distribution on peers, considering the behavior of these overlays in cases of non-uniform load distribution. These simulations will be able to measure the performance of architectures in cases closer to reality. In addition, we will evaluate the possibility to realize the HiGLoB structure also on G-Grid. This work creates many opportunities for the continuation of the study and analysis in realistic P2P environments.

Appendices

Appendix A

Distribution on C3P Framework

Due to the abundance of attractive services available on the cloud, people are placing an increasing amount of their data online on different cloud platforms. However, given the recent large-scale attacks on users data, privacy has become an important issue. Ordinary users cannot be expected to manually specify which of their data is sensitive, or to take appropriate measures to protect such data. Furthermore, usually most people are not aware of the privacy risk that different shared data items can pose. Starting from the study of the C3P framework, in which privacy risk is automatically calculated using the sharing context of data items, we tried propose novel approaches that take into account a P2P environment. To overcome ignorance of privacy risk on the part of most users, C3P framework uses a crowdsourcing based approach. It uses Item Response Theory (IRT) on top of this crowdsourced data to determine the sensitivity of items and diverse attitudes of users towards privacy. In this lightweight mechanism, users can crowdsource their sharing contexts with the server and determine the risk of sharing particular data item(s) privately. This scheme converges quickly and provides accurate privacy risk scores under varying conditions [51].

Our new contribution was the application of this privacy framework in a P2P environment, to improve the privacy guarantees among the network peers and to let them contexts' sensitivity elaboration only upon local interactions, considering the information coming from neighbour peers not always trustful. We paid particular attention to improving privacy conditions as compared to the existing centralised solution. In addition to following the

theory, we implemented a simulator to evaluate the performance of these solutions and obtain results to support our ideas. A manuscript for publication based on this work here is under preparation, which is expected to be submitted this Winter. In the following we describe some ideas on which we worked on, assuming to have this prerequisite: we consider a network of peers in which each peer has almost one link to another one.

The first idea was to split the storage database of Crowdsource Information (CI) from a single server into different peers. Each peer of the network stores some Crowdsourcing Information (CI) in its own memory. The CI comes from the network, when other peers share them, and from the sharing operations of the peer itself. The CI can be stored with a timestamp, so if the memory space is limited the peer can maintain only the most updated CI. When a peer needs to know the sensitivity value of a context item, it uses its own CI storage. It calculates the sensitivity value using the IRT algorithm with the owned CI. Then it decides the right privacy policy to apply to the context item and it shares the context information in the established cloud service with the decided privacy policy. Finally, when the privacy guarantees are satisfied, the peer spread a new CI, arranged by its peer profile, the context and the privacy policy, to the network by a gossip algorithm: it sends the CI to its neighbors, and they do the same until the timestamp is elapsed. When a peer receives a CI, stores it in its own CI storage: this improve the ability to calculate the sensitivity value of context items. If the timestamp of the Ci is not elapsed, the peer has to forward the CI to its neighbors, decreasing the timestamp before the sending. In this scenario, every peer calculates the context sensitivity based only on its local CI database, that is the result of the spreading of all the information by a gossip algorithm. This improve the privacy guarantees, because all the information are not gathered by a single server.

We investigated a second step to improve the previous solution in terms of network traffic and storage load. The new idea was to do not spread the CI with a gossip algorithm but let to each peer to asks the data useful to their specific sensitivity requests to the other network peers. Each peer of the network stores only the CI of the sharing operations that it requires: the ones generated from itself and the ones expressly asked the network. When a peer needs to know the sensitivity value of a context item, it sends a request over the network, by its neighbors, containing a number of context items, such as to calculate the sensitivity value. This number can not be too high to do not send a query message too large, but the context items have to be enough to

calculate the sensitivity, that is to build an almost full matrix (contexts x peers) for the IRT algorithm. When a peer receives a query message, it has to check in its CI storage if there are almost an available subset of CI that matches (almost a sufficient part of) the context items requested in the query message. If the conditions are satisfied that peer sends the useful CI set to the requester, otherwise not. Anyhow, the peer decreases the timestamp of the query message and if it is not elapsed yet, the peer forwards the query message to its neighbors. If the requester peer receives enough replies from the network, in a specified time period, it can calculate the sensitivity value. When the peer assigns the privacy policy to the context item, evaluating the sensitivity result or not, it creates a new CI record that save into its storage, for the further network queries that it could be receive. Also the CI set received from the network can be cached into the peer's CI storage.

An example:

1. A peer P_1 that had previously shared 3 context items (C_1, C_2, C_3) with respectively 3 privacy policy $(pp1 = 0, pp2 = 1, pp3 = 0)$ wants to share a new context item C_4.

2. P_1 sends a request to its neighbors (es. P_2, P_3, P_4) asking the policies for $N = 4$ context items (C_1, C_3, C_4, C_5), with a timestamp.

3. P_2 replies to P_1 with a list of the privacy policies of M context items, where $M \leq N$ and $M \geq M_{min}$. P_2 forwards the query message to its neighbors, only if the timestamp is not zero.

4. P_3 does not have enough context items, so P_3 does not reply anything to P_1, but it forwards the query message to its neighbors (until the timestamp is not zero).

5. P_1 waits until almost Z peer responses, or a timer elapsed.

6. If P_1 receives almost Z peer responses, P_1 can elaborate the sensitivity of the query set.

7. P_1 stores the privacy policy of C_5 in its sensitivity storage.

The parameters are:

- N: number of context items in the query set, it can be variable;

- $Timestamp$: number of neighbors level in which the query message is spread;
- M: number of context items in the replies;
- M_{min}: minimum number of context items asked in the replies. M_{min} should be close to N;
- Z: minimum number of peers to elaborate sensitivity.

A.1 Requirements of the CI to calculate sensitivity values

To calculate the sensitivity of a certain context item is necessary to have a sufficient number of CI with some requirements. For example, the privacy policies of a context item can not be the same for all the peer profiles that shared it.

Finally, as a future work, we proposed to use some Data Mining techniques to find correlation between context fields and profile policies, to detect the sensitivity of new contexts and in general to support the sensitivity calculation, specially when there are not enough local CI into a peer to calculate a context sensitivity.

Bibliography

[1] Distributed.net. `http://www.distributed.net`, 2006.

[2] K. Aberer, P. Cudrè-Mauroux, A. Datta, Z. Despotovic, M. Hauswirth, M. Punceva, and R. Schmidt. P-Grid: a self-organizing structured P2P system. *SIGMOD Rec.*, 32(3):29–33, 2003.

[3] Karl Aberer. P-grid: A self-organizing access structure for p2p information systems. In *Cooperative Information Systems*, pages 179–194. Springer, 2001.

[4] R. Agrawal, M. Mehta, J. Shafer, R. Srikant, A. Arning, and T. Bollinger. The Quest data mining system. In *Proceedings of the 2nd International Conference on Knowledge Discovery in Databases and Data Mining*, pages 244–249, 1996.

[5] Syed A Ahson and Mohammad Ilyas. *SIP handbook: services, technologies, and security of Session Initiation Protocol*. CRC Press, 2008.

[6] S. Alaei, M. Ghodsi, and M. Toossi. Skiptree: A new scalable distributed data structure on multidimensional data supporting range-queries. *Computer Communications*, 33(1):73–82, 2010.

[7] Emmanuelle Anceaume, R Ludinard, A Ravoaja, and F Brasileiro. Peercube: A hypercube-based p2p overlay robust against collusion and churn. In *Self-Adaptive and Self-Organizing Systems, 2008. SASO'08. Second IEEE International Conference on*, pages 15–24. IEEE, 2008.

[8] J. Aspnes and G. Shah. Skip graphs. *ACM Transactions on Algorithms (TALG)*, 3(4):37, 2007.

[9] James Aspnes, Jonathan Kirsch, and Arvind Krishnamurthy. Load balancing and locality in range-queriable data structures. In *Proceedings of the twenty-third annual ACM symposium on Principles of distributed computing*, pages 115–124. ACM, 2004.

[10] Hari Balakrishnan, M Frans Kaashoek, David Karger, Robert Morris, and Ion Stoica. Looking up data in p2p systems. *Communications of the ACM*, 46(2):43–48, 2003.

[11] HMN Dilum Bandara and Anura P Jayasumana. Evaluation of p2p resource discovery architectures using real-life multi-attribute resource and query characteristics. In *Consumer Communications and Networking Conference (CCNC), 2012 IEEE*, pages 634–639. IEEE, 2012.

[12] HMN Dilum Bandara and Anura P Jayasumana. Collaborative applications over peer-to-peer systems–challenges and solutions. *Peer-to-Peer Networking and Applications*, 6(3):257–276, 2013.

[13] Hakem Beitollahi and Geert Deconinck. Analyzing the chord peer-to-peer network for power grid applications. In *Fourth IEEE Young Researchers Symposium in Electrical Power Engineering*, page 5, 2008.

[14] Ashwin R Bharambe, Mukesh Agrawal, and Srinivasan Seshan. Mercury: supporting scalable multi-attribute range queries. In *ACM SIGCOMM Computer Communication Review*, volume 34, pages 353–366. ACM, 2004.

[15] L. Breiman. Pasting small votes for classification in large databases and on-line. *Machine Learning*, 36(1):85–103, 1999.

[16] L. Breiman and P. Spector. Parallelizing CART using a workstation network. In *Proc Annual American Statistical Association Meeting*, 1994.

[17] André Brinkmann, Yan Gao, Miroslaw Korzeniowski, and Dirk Meister. Request load balancing for highly skewed traffic in p2p networks. In *Networking, Architecture and Storage (NAS), 2011 6th IEEE International Conference on*, pages 53–62. IEEE, 2011.

[18] Mark Buchanan. *Nexus: small worlds and the groundbreaking theory of networks*. WW Norton, 2003.

[19] W. Cerroni, G. Monti, G. Moro, and M. Ramilli. Network Attack Detection Based on Peer-to-Peer Clustering of SNMP Data. *Quality of Service in Heterogeneous Networks*, pages 417–430, 2009.

[20] Walter Cerroni, Gianluca Moro, Tommaso Pirini, and Marco Ramilli. Peer-to-peer data mining classifiers for decentralized detection of network attacks. In *Proceedings of the Twenty-Fourth Australasian Database Conference-Volume 137*, pages 101–107. Australian Computer Society, Inc., 2013.

[21] P.K. Chan. An extensible meta-learning approach for scalable and accurate inductive learning. 1996.

[22] Ann Chervenak and Shishir Bharathi. Peer-to-peer approaches to grid resource discovery. In *Making Grids Work*, pages 59–76. Springer, 2008.

[23] S.H. Clearwater, T.P. Cheng, H. Hirsh, and B.G. Buchanan. Incremental batch learning. In *Proceedings of the sixth international workshop on Machine learning*, pages 366–370. Morgan Kaufmann Publishers Inc., 1989.

[24] Bram Cohen. The bittorrent protocol specification, 2008.

[25] R. Cole. Parallel merge sort. *SIAM Journal on Computing*, 17(4):770–785, 1988.

[26] Josenildo Costa da Silva, Matthias Klusch, Stefano Lodi, and Gianluca Moro. Privacy-preserving agent-based distributed data clustering. *Web Intelligence and Agent Systems*, 4(2):221–238, 2006. `http://iospress.metapress.com/content/1dva3ew1f2emam44/`.

[27] Frank Dabek, Emma Brunskill, M Frans Kaashoek, David Karger, Robert Morris, Ion Stoica, and Hari Balakrishnan. Building peer-to-peer systems with chord, a distributed lookup service. In *Hot Topics in Operating Systems, 2001. Proceedings of the Eighth Workshop on*, pages 81–86. IEEE, 2001.

[28] Frank Dabek, Ben Zhao, Peter Druschel, John Kubiatowicz, and Ion Stoica. Towards a common api for structured peer-to-peer overlays. In *Peer-to-Peer Systems II*, pages 33–44. Springer, 2003.

[29] Vasilios Darlagiannis. 21. hybrid peer-to-peer systems. In *Peer-to-Peer Systems and Applications*, pages 353–366. Springer, 2005.

[30] Alan Demers, Dan Greene, Carl Hauser, Wes Irish, John Larson, Scott Shenker, Howard Sturgis, Dan Swinehart, and Doug Terry. Epidemic algorithms for replicated database maintenance. In *Proceedings of the sixth annual ACM Symposium on Principles of distributed computing*, pages 1–12. ACM, 1987.

[31] A. Di Pasquale and E. Nardelli. Adst: An order preserving scalable distributed data structure with constant access costs. In *SOFSEM 2001: Theory and Practice of Informatics*, pages 211–222. Springer, 2001.

[32] Peter Druschel and Antony Rowstron. Past: A large-scale, persistent peer-to-peer storage utility. In *Hot Topics in Operating Systems, 2001. Proceedings of the Eighth Workshop on*, pages 75–80. IEEE, 2001.

[33] M. El Dick, E. Pacitti, and B. Kemme. Flower-cdn: a hybrid p2p overlay for efficient query processing in cdn. In *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology*, pages 427–438. ACM, 2009.

[34] W. Fan, S.J. Stolfo, and J. Zhang. The application of AdaBoost for distributed, scalable and on-line learning. In *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 362–366. ACM, 1999.

[35] Imen Filali, Francesco Bongiovanni, Fabrice Huet, and Françoise Baude. A survey of structured p2p systems for rdf data storage and retrieval. In *Transactions on large-scale data-and knowledge-centered systems III*, pages 20–55. Springer, 2011.

[36] Cormac Flanagan and Stephen N Freund. Fasttrack: efficient and precise dynamic race detection. In *ACM Sigplan Notices*, volume 44, pages 121–133. ACM, 2009.

[37] Michael J Freedman, Eric Freudenthal, and David Mazieres. Democratizing content publication with coral. In *NSDI*, volume 4, pages 18–18, 2004.

[38] Michael J Freedman, Karthik Lakshminarayanan, Sean Rhea, and Ion Stoica. Non-transitive connectivity and dhts. *USENIX WORLDS 2005*, 2005.

[39] Y. Freund and R. Schapire. A desicion-theoretic generalization of on-line learning and an application to boosting. In *Computational learning theory*, pages 23–37. Springer, 1995.

[40] Y. Freund and R.E. Schapire. Experiments with a new boosting algorithm. In *Machine Learning-International Workshop Then Conference*, pages 148–156. Citeseer, 1996.

[41] V. Gaede and O. Günther. Multidimensional access methods. *ACM Comput. Surv.*, 30(2):170–231, 1998.

[42] V. Galtier, S. Genaud, and S. Vialle. Implementation of the AdaBoost Algorithm for Large Scale Distributed Environments: Comparing JavaSpace and MPJ. In *2009 15th International Conference on Parallel and Distributed Systems*, pages 655–662. IEEE, 2009.

[43] Sarunas Girdzijauskas, Anwitaman Datta, and Karl Aberer. Oscar: Small-world overlay for realistic key distributions. In *Databases, Information Systems, and Peer-to-Peer Computing*, pages 247–258. Springer, 2007.

[44] Sarunas Girdzijauskas, Wojciech Galuba, Vasilios Darlagiannis, Anwitaman Datta, and Karl Aberer. Fuzzynet: Ringless routing in a ring-like structured overlay. *Peer-to-Peer Networking and Applications*, 4(3):259–273, 2011.

[45] Andrea Glorioso, Ugo Pagallo, and Giancarlo Ruffo. The social impact of p2p systems. In *Handbook of peer-to-peer networking*, pages 47–70. Springer, 2010.

[46] Jan Goebel, Thorsten Holz, and Carsten Willems. Measurement and analysis of autonomous spreading malware in a university environment. In *Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 109–128. Springer, 2007.

[47] B. Greenstein, S. Ratnasamy, S. Shenker, R. Govindan, and D. Estrin. Difs: A distributed index for features in sensor networks. *Ad Hoc Networks*, 1(2):333–349, 2003.

[48] F. Gui-Hai, C. Jin-Xiao, and X. Jun-Yuan. P2p super-peer topology construction based on hierarchical quadrant space [j]. *Chinese Journal of Computers*, 6:004, 2010.

[49] D. Guo, J. Wu, H. Chen, and X. Luo. Moore: An extendable peer-to-peer network based on incomplete kautz digraph with constant degree. In *INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE*, pages 821–829. IEEE, 2007.

[50] E.H. Han, G. Karypis, and V. Kumar. Scalable parallel data mining for association rules. In *Proceedings of the 1997 ACM SIGMOD international conference on Management of data*, pages 277–288. ACM, 1997.

[51] Hamza Harkous, Rameez Rahman, and Karl Aberer. C3p: Context-aware crowdsourced cloud privacy. In *14th Privacy Enhancing Technologies Symposium (PETS 2014)*, number EPFL-CONF-198473, 2014.

[52] N. J. A. Harvey, M. B. Jones, S. Saroiu, M. Theimer, and A. Wolman. Skipnet: A scalable overlay network with practical locality properties. In *Proceedings of the 4th conference on USENIX Symposium on Internet Technologies and Systems*, volume 4, pages 9–9, 2003.

[53] Erik Hjelmvik and Wolfgang John. Breaking and improving protocol obfuscation. *Chalmers University of Technology, Tech. Rep*, 123751, 2010.

[54] Thorsten Holz, Moritz Steiner, Frederic Dahl, Ernst Biersack, and Felix C Freiling. Measurements and mitigation of peer-to-peer-based botnets: A case study on storm worm. *LEET*, 8(1):1–9, 2008.

[55] Xing Jin and S-H Gary Chan. Unstructured peer-to-peer network architectures. In *Handbook of Peer-to-Peer Networking*, pages 117–142. Springer, 2010.

[56] M. A. Jovanovic, F. S. Annexstein, and K. A. Berman. Scalability issues in large peer-to-peer networks-a case study of gnutella. Technical report, Technical report, University of Cincinnati, 2001.

[57] Mina Kamel, Caterina Scoglio, and Todd Easton. Optimal topology design for overlay networks. In *NETWORKING 2007. Ad Hoc and Sensor Networks, Wireless Networks, Next Generation Internet*, pages 714–725. Springer, 2007.

[58] Verena Kantere, Spiros Skiadopoulos, and Timos Sellis. Storing and indexing spatial data in p2p systems. *Knowledge and Data Engineering, IEEE Transactions on*, 21(2):287–300, 2009.

[59] David Karger, Eric Lehman, Tom Leighton, Rina Panigrahy, Matthew Levine, and Daniel Lewin. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 654–663. ACM, 1997.

[60] M. Kelaskar, V. Matossian, P. Mehra, D. Paul, and M. Parashar. A study of discovery mechanisms for peer-to-peer applications. In *Proceedings of the 2Nd IEEE/ACM International Symposium on Cluster Computing and the Grid*, CCGRID '02, pages 444–, Washington, DC, USA, 2002. IEEE Computer Society.

[61] Jon M Kleinberg. Navigation in a small world. *Nature*, 406(6798):845–845, 2000.

[62] Matthias Klusch, Stefano Lodi, and Gianluca Moro. Distributed clustering based on sampling local density estimates. In *IJCAI*, pages 485–490, 2003. `http://www-ags.dfki.uni-sb.de/~klusch/papers/ijcai03-KDEC-paper.pdf`.

[63] SB Kotsiantis. Supervised machine learning: A review of classification techniques. *Emerging artificial intelligence applications in computer engineering: real word AI systems with applications in eHealth, HCI, information retrieval and pervasive technologies*, page 3, 2007.

[64] Gunnar Kreitz and Fredrik Niemela. Spotify–large scale, low latency, p2p music-on-demand streaming. In *Peer-to-Peer Computing (P2P),*

2010 IEEE Tenth International Conference on, pages 1–10. IEEE, 2010.

[65] Brigitte Kröll and Peter Widmayer. Distributing a search tree among a growing number of processors. In *ACM SIGMOD Record*, volume 23, pages 265–276. ACM, 1994.

[66] Brigitte Kröll and Peter Widmayer. *Balanced distributed search trees do not exist*. Springer, 1995.

[67] Vito Latora and Massimo Marchiori. Efficient behavior of small-world networks. *Physical review letters*, 87(19):198701, 2001.

[68] Jiunn-Jye Lee, Hann-Huei Chiou, Chia-Chang Hsu, and Chin-Laung Lei. An adaptive sector-based routing model over structured peer-to-peer networks. *Computer Networks*, 57(4):887–896, 2013.

[69] Arnaud Legout, Guillaume Urvoy-Keller, and Pietro Michiardi. Rarest first and choke algorithms are enough. In *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, pages 203–216. ACM, 2006.

[70] D. Li, X. Lu, and J. Wu. Fissione: A scalable constant degree and low congestion dht scheme based on kautz graphs. In *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, volume 3, pages 1677–1688. IEEE, 2005.

[71] Deng Li, Hui Liu, and Athanasios Vasilakos. *An efficient, scalable and robust p2p overlay for autonomic communication*. Springer, 2009.

[72] Jin Li. On peer-to-peer (p2p) content delivery. *Peer-to-Peer Networking and Applications*, 1(1):45–63, 2008.

[73] X. Li, Y. J. Kim, R. Govindan, and W. Hong. Multi-dimensional range queries in sensor networks. In *Proceedings of the 1st international conference on Embedded networked sensor systems*, pages 63–75. ACM, 2003.

[74] Jian Liang, Rakesh Kumar, and Keith W Ross. The fasttrack overlay: A measurement study. *Computer Networks*, 50(6):842–858, 2006.

[75] W. Litwin, M. A. Neimat, and D. Schneider. Rp*: A family of order preserving scalable distributed data structures. In *Proceedings of the International Conference on Very Large Data Bases*, pages 342–342. Institute of Electrical & Electronica Engineers (IEEE), 1994.

[76] W. Litwin, M. A. Neimat, and D. A. Schneider. Lh*a scalable, distributed data structure. *ACM Transactions on Database Systems (TODS)*, 21(4):480–525, 1996.

[77] Stefano Lodi, Gabriele Monti, Gianluca Moro, and Claudio Sartori. Peer-to-peer data clustering in self-organizing sensor networks. In *Intelligent Techniques for Warehousing and Mining Sensor Network Data*, pages 179–211. IGI Global, Information Science Reference, December 2009, Hershey, PA, USA, December 2009. `http://www.igi-global.com/chapter/peer-peer-data-clustering-self/39546`.

[78] Stefano Lodi, Gianluca Moro, and Claudio Sartori. Distributed data clustering in multi-dimensional peer-to-peer networks. In *ADC*, pages 171–178, 2010.

[79] Qin Lv, Sylvia Ratnasamy, and Scott Shenker. Can heterogeneity make gnutella scalable? In *In Proceedings of the first International Workshop on Peer-to-Peer Systems*, pages 94–103, 2002.

[80] D. Malkhi, M. Naor, and D. Ratajczak. Viceroy: A scalable and dynamic emulation of the butterfly. In *Proceedings of the twenty-first annual symposium on Principles of distributed computing*, pages 183–192. ACM, 2002.

[81] Gurmeet Singh Manku. *Dipsea: a modular distributed hash table*. PhD thesis, stanford university, 2004.

[82] Petar Maymounkov and David Mazieres. Kademlia: A peer-to-peer information system based on the xor metric. In *Peer-to-Peer Systems*, pages 53–65. Springer, 2002.

[83] Stanley Milgram. The small world problem. *Psychology today*, 2(1):60–67, 1967.

[84] Gabriele Monti and Gianluca Moro. Multidimensional range query and load balancing in wireless ad hoc and sensor networks. In

Eighth IEEE International Conference on Peer-to-Peer Computing, 2008, pages 205–214. IEEE Computer Society, Sept. 2008. http://doi.ieeecomputersociety.org/10.1109/P2P.2008.27.

[85] Gabriele Monti and Gianluca Moro. Self-organization and local learning methods for improving the applicability and efficiency of data-centric sensor networks. In *Quality of Service in Heterogeneous Networks (QSHINE 2009)*, volume 22 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 627–643. Springer Berlin Heidelberg, 2009. http://dx.doi.org/10.1007/11942634_40.

[86] Gabriele Monti, Gianluca Moro, Giacomo Tufano, and Marco Rosetti. Self-organizing mobile mesh networks with peer-to-peer routing and information search services. In *Proceedings of The Third International Conference on Advances in Mesh Networks (MESH 2010)*, pages 17–22, Venezia, Italy, 2010. IEEE Computer Society. http://doi.ieeecomputersociety.org/10.1109/MESH.2010.14.

[87] A. Montresor and M. Jelasity. Peersim: A scalable p2p simulator. In *Peer-to-Peer Computing, 2009. P2P'09. IEEE Ninth International Conference on*, pages 99–100, 2009.

[88] G. Moro, G. Monti, and A. M. Ouksel. Merging g-grid p2p systems while preserving their autonomy. In *Proceedings of the MobiQuitous*, volume 4, pages 123–137, 2004.

[89] G. Moro, M. A. Ouksel, and C. Sartori. Agents and peer-to-peer computing: a promising combination of paradigms. In *Proceedings of the 1st International Workshop on Agents and Peer-to-Peer Computing, Bologna, Italy, July 2002*, volume 2530, pages 1–14. Springer, 2003.

[90] Gianluca Moro and Gabriele Monti. W-grid: A scalable and efficient self-organizing infrastructure for multi-dimensional data management, querying and routing in wireless data-centric sensor networks. *Journal of Network and Computer Applications*, 35(4):1218–1234, July 2012. http://dx.doi.org/10.1016/j.jnca.2011.05.002.

[91] Gianluca Moro, Tommaso Pirini, and Claudio Sartori. Network traffic load balancing in hierarchical peer-to-peer systems. In *P2P, Paral-*

lel, Grid, Cloud, and Internet Computing (3PGCIC 2015), 2015 10th International Conference on, 2015.

[92] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Consulted*, 1(2012):28, 2008.

[93] Moni Naor and Udi Wieder. Novel architectures for p2p applications: the continuous-discrete approach. *ACM Transactions on Algorithms (TALG)*, 3(3):34, 2007.

[94] H. Nazerzadeh and M. Ghodsi. Raq: a range-queriable distributed data structure. *SOFSEM 2005: Theory and Practice of Computer Science*, pages 269–277, 2005.

[95] University of Karlsruhe. YaCy Project. `http://www.yacy.net`, September 2006.

[96] A. Oram. *Peer-to-peer: harnessing the benefits of a disruptive technology.* " O'Reilly Media, Inc.", 2001.

[97] M. A. Ouksel. The interpolation-based grid file. In *Proceedings of the fourth ACM SIGACT-SIGMOD symposium on Principles of database systems*, pages 20–27. ACM, 1985.

[98] M. A. Ouksel, V. Kumar, and C. Majumdar. Management of concurrency in interpolation based grid file organization and its performance. *Information sciences*, 78(1):129–158, 1994.

[99] M. A. Ouksel and O. Mayer. A robust and efficient spatial data structure: the nested interpolation-based grid file. *Acta Informatica*, 29(4):335–373, 1992.

[100] M. A. Ouksel and G. Moro. G-Grid: A class of scalable and self-organizing data structures for multi-dimensional querying and content routing in P2P networks. In *Agents and Peer-to-Peer Computing*, pages 123–137.

[101] Venkata N Padmanabhan and Kunwadee Sripanidkulchai. The case for cooperative networking. In *Revised Papers from the First International Workshop on Peer-to-Peer Systems IPTPS*, 2001.

[102] C.H. Papadimitriou. *Computational complexity*. John Wiley and Sons Ltd., 2003.

[103] Al-Sakib Khan Pathan, Muhammad Mostafa Monowar, and Zubair Md Fadlullah. *Building Next-generation Converged Networks: Theory and Practice*. CRC Press, 2013.

[104] Theoni Pitoura, Nikos Ntarmos, and Peter Triantafillou. Saturn: range queries, load balancing and fault tolerance in dht data systems. *Knowledge and Data Engineering, IEEE Transactions on*, 24(7):1313–1327, 2012.

[105] F.J. Provost and D.N. Hennessy. Scaling up: Distributed machine learning with cooperation. In *Proceedings of the National Conference on Artificial Intelligence*, pages 74–79. Citeseer, 1996.

[106] J.R. Quinlan. *C4. 5: programs for machine learning*. Morgan Kaufmann, 1993.

[107] JR Quinlan. Improved use of continuous attributes in C4. 5. *Arxiv preprint cs/9603103*, 1996.

[108] Rajiv Ranjan, Lipo Chan, Aaron Harwood, Shanika Karunasekera, and Rajkumar Buyya. Decentralised resource discovery service for large scale federated grids. In *e-Science and Grid Computing, IEEE International Conference on*, pages 379–387. IEEE, 2007.

[109] Rajiv Ranjan, Aaron Harwood, and Rajkumar Buyya. A study on peer-to-peer based discovery of grid resource information. *University of Melbourne, Australia, Technical Report GRIDS-TR-2006-17*, 2006.

[110] Rajiv Ranjan, Aaron Harwood, and Rajkumar Buyya. Peer-to-peer-based resource discovery in global grids: a tutorial. *Communications Surveys & Tutorials, IEEE*, 10(2):6–33, 2008.

[111] Ananth Rao, Karthik Lakshminarayanan, Sonesh Surana, Richard Karp, and Ion Stoica. Load balancing in structured p2p systems. In *Peer-to-Peer Systems II*, pages 68–79. Springer, 2003.

[112] R.G. Reynolds. An introduction to cultural algorithms. In *Proceedings of the Third Annual Conference on Evolutionary Programming*, pages 131–139. World Scientific, 1994.

[113] Matei Ripeanu. Peer-to-peer architecture case study: Gnutella network. In *Peer-to-Peer Computing, 2001. Proceedings. First International Conference on*, pages 99–100. IEEE, 2001.

[114] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. *Lecture Notes in Computer Science*, 2218:329–340, 2001.

[115] Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware 2001*, pages 329–350. Springer, 2001.

[116] Antony Rowstron and Peter Druschel. Storage management and caching in past, a large-scale, persistent peer-to-peer storage utility. In *ACM SIGOPS Operating Systems Review*, volume 35, pages 188–201. ACM, 2001.

[117] Rüdiger Schollmeier. [16] a definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications. In *Peer-to-Peer Computing, IEEE International Conference on*, pages 0101–0101. IEEE Computer Society, 2001.

[118] Thorsten Schütt, Florian Schintke, and Alexander Reinefeld. A structured overlay for multi-dimensional range queries. In *Euro-Par 2007 Parallel Processing*, pages 503–513. Springer, 2007.

[119] Xuemin Shen, Heather Yu, John Buford, and Mursalin Akon. *Handbook of peer-to-peer networking*, volume 1. Springer, 2010.

[120] Vivek Singh, Himani Gupta, and Kaveri Dey. Anonymous file sharing in peer to peer system by random walks. 2012.

[121] Andrew Ross Sorkin. Software bullet is sought to kill musical piracy. *New York Times*, 1:36, 2003.

[122] John Stevenson. *Bitcoins, litecoins, what coins?: A global phenomenon.* John Stevenson, 2013.

[123] Joe Stewart. Storm worm ddos attack. *Internet: http://www. secureworks. com/research/threats/storm-worm*, 2007.

[124] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. Frans Kaashoek, F. Dabek, and H. Balakrishnan. Chord: a scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Trans. Netw.*, 11(1):17–32, 2003.

[125] Ion Stoica, Robert Morris, David Karger, M Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. *ACM SIGCOMM Computer Communication Review*, 31(4):149–160, 2001.

[126] S. Stolfo, A.L. Prodromidis, S. Tselepis, W. Lee, D.W. Fan, and P.K. Chan. JAM: Java agents for meta-learning over distributed databases. In *Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining*, pages 74–81, 1997.

[127] Daniel Stutzbach, Daniel Zappala, and Reza Rejaie. The scalability of swarming peer-to-peer content delivery. In *NETWORKING 2005. Networking Technologies, Services, and Protocols; Performance of Computer and Communication Networks; Mobile and Wireless Communications Systems*, pages 15–26. Springer, 2005.

[128] Yuzhe Tang, Jianliang Xu, Shuigeng Zhou, and Wang-chien Lee. m-light: indexing multi-dimensional data over dhts. In *Distributed Computing Systems, 2009. ICDCS'09. 29th IEEE International Conference on*, pages 191–198. IEEE, 2009.

[129] Paolo Trunfio, Domenico Talia, Harris Papadakis, Paraskevi Fragopoulou, Matteo Mordacchini, Mika Pennanen, Konstantin Popov, Vladimir Vlassov, and Seif Haridi. Peer-to-peer resource discovery in grids: Models and systems. *Future Generation Computer Systems*, 23(7):864–878, 2007.

[130] P. Utgo. An improved algorithm for incremental induction of decision trees. In *Proceedings of the Eleventh International Conference on Machine Learning*, pages 318–325. Citeseer, 1994.

[131] Silvio Valenti, Dario Rossi, Michela Meo, Marco Mellia, and Paola Bermolen. Accurate, fine-grained classification of p2p-tv applications by simply counting packets. In *Traffic Monitoring and Analysis*, pages 84–92. Springer, 2009.

[132] Q. H. Vu, B. C. Ooi, M. Rinard, and K. L. Tan. Histogram-based global load balancing in structured peer-to-peer systems. *Knowledge and Data Engineering, IEEE Transactions on*, 21(4):595–608, 2009.

[133] Quang Hieu Vu, Mihai Lupu, and Beng Chin Ooi. *Peer-to-peer computing: principles and applications*. Springer, 2009.

[134] B. Wang and Q. Shen. Tide: An effective and practical design for hierarchical-structured p2p model. *Computer Communications*, 2012.

[135] M.S. Warren and J.K. Salmon. A parallel hashed oct-tree n-body algorithm. In *Proceedings of the 1993 ACM/IEEE conference on Supercomputing*, pages 12–21. ACM, 1993.

[136] Duncan J Watts and Steven H Strogatz. Collective dynamics of small-worldnetworks. *nature*, 393(6684):440–442, 1998.

[137] I.H. Witten and E. Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann Pub, 2005.

[138] George Woltman and Scott Kurowski. The great internet mersenne prime search, 2004.

[139] Tim Wu. Network neutrality, broadband discrimination. *J. on Telecomm. & High Tech. L.*, 2:141, 2003.

[140] Lin Xiao and Aris M Ouksel. Scalable self-configuring integration of localization and indexing in wireless ad-hoc sensor networks. In *Mobile Data Management, 2006. MDM 2006. 7th International Conference on*, pages 151–151. IEEE, 2006.

[141] M. Xu and G. Liu. Building self-adaptive peer-to-peer overlay networks with dynamic cluster structure. In *Communication Technology (ICCT), 2011 IEEE 13th International Conference on*, pages 520–525. IEEE, 2011.

[142] M. Xu, S. Zhou, and J. Guan. A new and effective hierarchical overlay structure for peer-to-peer networks. *Computer Communications*, 34(7):862–874, 2011.

[143] Beverly Yang and Hector Garcia-Molina. Comparing hybrid peer-to-peer systems. In *Proceedings of the 27th Intl. Conf. on Very Large Data Bases*, 2001.

[144] K.C. Zatloukal and N.J.A. Harvey. Family trees: an ordered dictionary with optimal congestion, locality, degree, and search time. In *Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 308–317. Society for Industrial and Applied Mathematics, 2004.

[145] L. Zeng, L. Li, L. Duan, K. Lu, Z. Shi, M. Wang, W. Wu, and P. Luo. Distributed data mining: a survey. *Information Technology and Management*, pages 1–7, 2012.

[146] Chong Zhang, Weidong Xiao, Daquan Tang, and Jiuyang Tang. P2p-based multidimensional indexing methods: A survey. *Journal of Systems and Software*, 84(12):2348–2362, 2011.

[147] R. Zhang, W. Qian, A. Zhou, and M. Zhou. An efficient peer-to-peer indexing tree structure for multidimensional data. *Future Generation Computer Systems*, 25(1):77–88, 2009.

[148] Y. M. Zhang, X. Lu, and D. S. Li. Sky: efficient peer-to-peer networks based on distributed kautz graphs. *Science in China Series F: Information Sciences*, 52(4):588–601, 2009.

[149] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. D. Kubiatowicz. Tapestry: A resilient global-scale overlay for service deployment. *Selected Areas in Communications, IEEE Journal on*, 22(1):41–53, 2004.

[150] B. Y. Zhao, J. Kubiatowicz, A. D. Joseph, et al. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. 2001.

[151] Ce Zhu, Yuenan Li, and Xiamu Niu. *Streaming media architectures, techniques and applications: recent advances*. IGI Global, 2011.