Flávio Duarte Pacheco Fernandes

LHView: Location Aware Hybrid Partial View

Dissertação para obtenção do Grau de Mestre em

Engenharia Informática

Orientador: João Leitão, Professor Auxiliar, Faculdade de Ciências e Tecnologia
da Universidade Nova de Lisboa

Júri

Presidente:	Doutora Fernanda Maria Barquinha Tavares Vieira Barbosa
Arguente:	Doutor Miguel Ângelo Marques de Matos
Vogal:	Doutor João Carlos Antunes Leitão

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

March, 2017

LHView: Location Aware Hybrid Partial View

Este documento foi gerado utilizando o processador (pdf)LaTeX, com base no template “unlthesis” [1] desenvolvido no Dep. Informática da FCT-NOVA [2].
[1] https://github.com/joaomlourenco/unlthesis [2] http://www.di.fct.unl.pt

Acknowledgements

I want to thank my adviser, Prof Dr. João Leitão, for providing this opportunity that challenged my knowledge about Distributed Systems and the scientific field of Computer Science. Without his detailed feedback, this work would not be possible. Furthermore, I want to thank the university for the infrastructure provided.

Finally, I want to thank my family for everything.

Abstract

The rise of the Cloud creates enormous business opportunities for companies to provide global services, which requires applications supporting the operation of those services to scale while minimizing maintenance costs, either due to unnecessary allocation of resources or due to excessive human supervision and administration. Solutions designed to support such systems have tackled fundamental challenges from individual component failure to transient network partitions. A fundamental aspect that all scalable large systems have to deal with is the membership of the system, i.e, tracking the active components that compose the system. Most systems rely on membership management protocols that operate at the application level, many times exposing the interface of a logical overlay network, that should guarantee high scalability, efficiency, and robustness.

Although these protocols are capable of repairing the overlay in face of large numbers of individual components faults, when scaling to global settings (i.e, geo-distributed scenarios), this robustness is a double edged-sword because it is extremely complex for a node in a system to distinguish between a set of simultaneously node failures and a (transient) network partition. Thus the occurrence of a network partition creates isolated sub-sets of nodes incapable of reconnecting even after the recovery from the partition.

This work address this challenges by proposing a novel datacenter-aware membership protocol to tolerate network partitions by applying existing overlay management techniques and classification techniques that may allow the system to efficiently cope with such events without compromising the remaining properties of the overlay network. Furthermore, we strive to achieve these goals with a solution that requires minimal human intervention.

Keywords: Geo-Distributed Systems, Gossip Protocol, Membership Protocol, Network Partitions, Location Inference.

Resumo

O crescimento de serviços na Cloud criam grandes oportunidades de negócio para as empresas prestarem serviços globais, que requerem aplicações que suportem esses serviços providenciado escalibilidade enquanto minimizam os custos de manutenção, devido à utilização de recursos desnecessária ou devido a excessiva administração e supervisão humana. As soluções existentes têm enfrentado desafios fundamentais, desde a falha de componentes individuais a partições (temporárias) de rede. Uma característica fundamental é que todos os grandes sistemas escaláveis têm de gerir a filiação, i.e, gerir a informação relativa aos componentes que materializam o sistema. A maioria dos sistemas recorre a protocolos de filiação que funcionam ao nível da aplicação e expõem a topologia de redes sobreposta que garantem a alta escalabilidade, a eficiência e a robustez do sistema.

Apesar destes protocolos serem capazes de reparar a topologia de rede na presença de um grande número de faltas em componentes independentes, quando utilizados à escala global (e.g distribuídos geograficamente), esta vantagem é uma lamina de dois gumes porque é extremamente complexo para um nó num sistema distinguir entre um conjunto de falhas e partições (temporárias) na rede. Por isso a ocorrência de partições de rede força o sistema a subdividir-se em dois conjuntos isolados incapazes de comunicar entre si, mesmo após a reparação da partição.

Este trabalho propoe um novo protocolo de gestão de filiação que tolere partições na rede através da aplicação das abordagens existentes de gestão de rede sobreposta e de técnicas de classificação que permitam ao sistema reagir adequadamente ao problema sem comprometer as propriedades da topologia de rede. Por fim, pretende-se alcançar uma solução que utilize o mínimo de intervenção humana.

Palavras-chave: sistemas geo-distribuidos, protocolos epidémicos, protocolos de filiação, tolerância a partições na rede, inferência de localizações.

Contents

1 Introduction **1**

1.1 Context . . . 1

1.2 Motivation and Solution . . . 3

1.3 Contributions . . . 3

1.4 Document structure . . . 4

2 Related Word **5**

2.1 Distributed Systems . . . 5

2.1.1 Essential Management Services . . . 8

2.1.2 Membership . . . 8

2.1.3 Failure detection . . . 9

2.1.4 Distributed Storage Systems . . . 9

2.1.5 Summary . . . 11

2.2 Gossip Protocols . . . 11

2.2.1 Peer-to-Peer . . . 12

2.2.2 Message Dissemination . . . 12

2.2.3 Overlay Network . . . 13

2.2.4 Fundamental properties and metrics . . . 20

2.3 Network Partition Tolerant Distributed Systems . . . 21

2.4 Discussion . . . 22

3 Partition Tolerant Membership Service **23**

3.1 Objective . . . 23

3.2 System Model . . . 24

3.3 Insight . . . 24

3.4 Architecture . . . 25

3.5 Location Oracles . . . 26

3.5.1 Overview . . . 26

3.5.2 Static Label Oracle . . . 27

3.5.3 IP Address Clustering and Latency Oracle . . . 27

3.5.4 Anchor Oracle . . . 28

3.5.5 Dynamic Label Oracle . . . 28

3.5.6 Discussion 29
3.6 Partition Tolerant Membership Service 30
3.6.1 Overview 30
3.6.2 Partial View Management Policies 30
3.6.3 Discussion 32
3.7 Implication on message dissemination 32
3.8 Summary 32

4 Evaluation **35**
4.1 Components implementation 35
4.2 Oracle's classification accuracy 35
4.2.1 IP Address Clustering with Latency Oracle 36
4.2.2 Anchor Oracle 38
4.2.3 Dynamic Label Oracle 38
4.3 LHView's evaluation 39
4.3.1 In-Degree and Out-Degree 40
4.3.2 Crash failure recovery 40
4.3.3 Network partition recovery 41
4.4 Summary 42

5 Conclusion **45**
5.1 Future Work 46

Bibliography **47**

List of Figures

3.1 System Architecture 26
3.2 Anchor Oracle and anchor servers' deployment 28

4.1 Average longest common IP prefix size per scenario. 37
4.2 Percentage of correct classifications in each tested scenario 38
4.3 Number of distinct labels over time per deployment scenario. 39
4.4 Node in and out degree of HyParView and LHView. 41
4.5 Reliability of HyParView and LHView under crash failure scenario. 41
4.6 Reliability of the LHView in face of a network partition. 42

List of Tables

4.1 Deployment configuration for each scenario(rows). 36
4.2 Average round trip time between all datacenters. 37
4.3 Memberships and dissemination parameters. 40

Chapter 1

Introduction

1.1 Context

In recent years, as the World progresses into a more highly connected place, many global level business opportunities have appeared. These businesses intend to have their services reaching as many customers as possible and therefore must be able to operate at global scale. In order to provide global services, organizations resort to geo-distributed systems usually deployed on datacenters around the globe.

Systems designed for large-scale such as Cloud settings, must be prepared to handle all kind of network anomalies, component failures, and software errors by design. Ideally, the system should be able to deal with such faults autonomously, in order to provide a trustworthy, satisfactory, and cost efficient service. Modern systems rely on specialized modules that are responsible for managing the membership of components (e.g nodes) that materialize the system to allow a (high) control over data partitioning and load balancing while providing fault-tolerance by tracking and addressing (i.e, reconfigure the system) nodes' failures. These membership services many times resort to Peer-to-Peer(P2P) techniques that along the years have proved their value when designing distributed protocols that must be highly scalable, efficient, and fault-tolerant. To guarantee that those systems does not compromise themselves, theirs decision must be based on most recent membership's state which on a planetary level is extremely difficult to accomplish, having into account the network's asynchronous message delivery model. A solution consists on using Gossip-based protocols, designed to provide high reliability for communications and dissemination of information.

As an example consider a geo-replicated cloud-based storage system such as Cassandra[16], Dynamo[8], ChainReaction[1], or Cops[23]. These systems require up-to-date membership information to many decisions regarding the processing of application requests, namely to infer the location of each data object maintained by the system. The lack of robust and partition-tolerant membership services has lead all these solutions to rely on human-centered control, which can lead to delays in handling faults while at the same time incurring in additional human cost on highly specialized system operators and administrators which are susceptible to human errors[26][34].

Gossip, or epidemic protocols, use a pattern of messages' dissemination usually on top of a logical network provided by membership services denominated *overlay networks* that resembles the propagation of a biological virus on a population. In the context of the P2P model, a node propagates a message to its neighbors, the nodes to whom it is connected at the overlay level. The receiving nodes will perform the same procedure to their neighbors and so on until all nodes receive the message. The dissemination behaviour explained not only has high scalability because all nodes cooperate on the process and the load is evenly distributed among them but also high fault-tolerance and reliability as a consequence of the high level of redundancy due to the multiple propagation paths that emerge from this process.

Protocols such as HyParView[19] provide a membership management service that guarantees high global connectivity due to its self-healing properties even in cases of catastrophic failures as high as 80% of simultaneous node failures. Leveraging this protocol it was shown that one could build highly efficient and robust message dissemination mechanisms[18]. Following this, the protocol leverages information management strategies that combined with the use of TCP as (unreliable) failure detector, provides a fast failure detection mechanism that allow it to take quick action for repairing the overlay in presence of node failures.

However, the HyParView's impressive self-healing capability is a double-edge sword when scaling to a global scale. The reason for this derives from the protocol's incapability to distinguish simultaneously node failures from (transient) network partitions. In face of network partitions, protocols similar to HyParView will reconfigure themselves extremely fast by replacing all the unresponsive nodes by nodes that are currently responsive while *forgetting* information about nodes that previously were active. This results on the creation of distinct sub-sets of nodes without any information about the others, making the protocol lose its global connectivity and consequently, its reliability. This scenario originates from the occurrence of the inevitable network partitions that often affect global-scale applications[3]. Therefore there is a necessity for a membership protocol that takes geo-distributed systems to the global scale while providing the fundamental guarantees necessaru to ensure the correct operation of these systems while the network is partitioned as well as after the partition is healed.

1.2 Motivation and Solution

Systems deployed in Cloud settings that provide geo-distribute services are composed of thousands of nodes that require precise management by its infrastructure administrators. However, the unpredictable occurrence of failures can affect considerably their performance. Moreover, if these systems require manual management even for small tasks such as permanently decommissioning a node[16][8] the system can operate with a sub-optimal configuration for too long. Furthermore, the error-prone tendencies of humans may comprise the system even more[26][34]. Additionally, the time necessary for a human to solve a problem combined with its working schedule, and cost of these highly specialized administration and operations experts has started to lead organizations to invest on automation. By doing so, organizations aim for autonomously managed systems to achieve a robust base for supporting the operation of their services while reducing maintenance cost.

Current solutions rely on internal components to track service's membership and those capable of tolerating a partition need to maintain at each node the global information about the cluster's state, i.e, the system deployment. A solution that scales requires great amount of resources for monitoring and that does not exclude faulty nodes which affect the performance and correction of the monitoring mechanism. Other solutions focus on tracking just a fraction of the system state at each individual component and can efficiently handle dynamic environments by being capable of fast self-recovery. However, this class of solutions are unable of distinguish multiple failures from network partitions due to the lack of global information that would potentially allow to perform co-relation analysis on detected failures. Thus inaccessible nodes are excluded from the system permanently. However, network partitions divide the system into two or more disconnect components incapable of communicating with each other and so neither will attempt to reconnect because nodes in each component assume the remaining nodes(i.e, nodes in the other components of the partition) to be permanently faulty.

Our solution integrates into exiting gossip-based membership protocols a mechanism for automatic location inference. These mechanism is highly adaptive and can exploit the API provided by the Cloud virtualization services or, when these are not available, monitor environment properties, such as the latency between nodes, to automatically infer the relative location of nodes in a system deployment, offering additional information for the system to deal with network partitions. This empowers the membership service to rejoin connectivity by reconfiguring itself after the healing of a network partition, without human intervention.

1.3 Contributions

The main contributions of this thesis are:

- It proposes novel location inference techniques for nodes in large-scale systems.

- A membership service that leverages epidemic/gossip protocols capable of fast self-regeneration in the presence of faults while being tolerant to network partitions.
- A experimental evaluation of the precision of the location inference mechanisms and their impact on the robustness of the gossip-based membership services.

We present a membership protocol that addresses network partitions, a major challenge for large-scale services without compromising the robustness, scalability, and reliability of the whole system. The LHView achieves such feats by extending the membership protocol HyParView[19] with location inference components, Oracles. The ability to distinguish nodes from different locations allow LHView to apply specific recovery procedures to different categories of failures that previous protocols could not.

The evaluation of our work consisted of evaluating the location-inference accuracy of the oracles proposed in this thesis in three deployment scenarios and on a comparative analysis of LHView with HyParView. The comparison focused on demonstrating that LHView maintains the same guaranties of its predecessor and on LHView equipped with oracles allow it to succeed in network partitions scenarios while its predecessor can not.

1.4 Document structure

The remainder of this document is organized as follows:

Chapter 2 presents the related work. In more detail, section 2.1 describes distributed systems and the technologies that support their essential services. Section 2.2 presents fundamental properties of gossip protocols and the diverse overlay networks that support them. Section2.3 discusses briefly some systems that could benefit from the proposed work in this thesis.

Chapter 3 describes our solution and its components in detail.

Chapter 4 presents the evaluation of each component and the integrated membership system.

Chapter 5 presents our conclusions and proposes future work directions on this domain.

Chapter 2

Related Word

This thesis focus on proposing a novel design and implementation for a membership protocol that is tolerant to transient network partitions. Such solution can be integrated in multiple systems that are now-a-days highly used in practice. This chapter discusses fundamental concepts and related technologies that inspired the solution and systems that could benefit from it, in particular:

Section 2.1 overviews distributing computing and essential services and techniques that are employed in the design of distributed systems.

Section 2.2 describes gossip/epidemic protocols and overlay networks used to support the application-level membership and communication services of large-scale distributed systems.

Section 2.3 discusses concrete systems that could benefit from partition tolerant, highly robust membership systems as the one proposed in this thesis while being also able to leverage robust and scalable dissemination protocols that can be built on top of these systems.

2.1 Distributed Systems

A distributed system is a set of networked computers coordinated through the exchange and sharing of information, typically messages (message passing model). Organizations build, on top of these systems products and/or services that exploit the Internet to reach clients across the globe. As the number of clients increases, systems must be able to scale accordingly and handle the occurrence of unpredictable network anomalies. In order to handle the workload without investing on bleeding edge equipment, organizations many times resort to architectures where the load of the system is (approximately) evenly spread across nodes/computers that materialize a system. Therefore, nodes contribute and share their computational resources to cooperate on solving a problem or providing a concrete service, such that the nodes can be perceived as a single entity that has an amount of computational resources that corresponds to the aggregation of every individual computer resource.

The resulting cumulative computational power and storage space on a system as the described above allow to achieve a performance (for parallel programs) that can surpass bleeding edge super computers while being cheaper, more scalable, and robust that those solutions. Through cooperation, nodes can address problems by dividing the workload evenly among them and when facing large-scale challenges the systems can be scaled by augmenting the number of nodes that materialize the system (we will call such set of nodes, a *cluster*). Note that each cluster member is fundamentally similar to the other elements of the cluster, which also enables one to address the failure of any node by simply replacing it with a new one.

Although these models offer better load-balancing and fault-tolerance, they have to handle several challenges that have a high impact on their performance:

Network latency: As a consequence of delay introduced by communications there might exist delays on the computations and readiness of the data required by individual components. This can translate into delays for providing responses to end users, which has been shown to have a direct translation to loss of revenue for service operations[31].

Coordination: To efficiently leverage the available resources, nodes have to synchronize fractions of their executions, which on a decentralized model may require excessive exchange of messages and have an excessive amount of time. The alternative is to resort to the centralized model, through the use of a coordinator node, which inherently is a single point of failure (and potential bottleneck). Furthermore, the coordination may require up-to-date global information about the system which is not trivial to gather in a timely and efficient fashion in a large-scale distributed system.

Resources Management: Control over the entities in the system that can manipulate and use resources of the system. This can increase the complexity of the system and lead to the introduction of errors in its behaviour.

Failures: Components and network links may fail independently and arbitrary. The occurrence of failures should not cause the complete shutdown or unavailability of the system, i.e, compromise system availability, nor compromise the data/state maintained in it, i.e, compromise the system correctness.

Manual management: As the number of nodes increases, the management of the cluster will become significantly more difficult due to the increase in individual reuse of the components and possible interface between them. Distributed system should support autonomously node and resources management in order to reduce the management cost and avoid human errors.

Large-scale distributed systems have been used to process and/or store large volumes of data, therefore it is necessary that these systems cope with the previously mentioned challenges. These system address a fraction of those issues by incorporating a membership service that track the status of the nodes of these systems. The class of distributed systems branch into the following categories:

Distributed Data Processing: Used to process large volumes of data using a cluster infrastructure in order to achieve a particular goal, such as infer data patterns or co-relate large amounts of data with different provenances. In these systems, the problem is divided into smaller tasks which will be distributed among the nodes by a coordinator which plays a crucial role of scheduling the computations across the available resources. Furthermore, intermediate and final computation results are stored on a distributed file system to guarantee the data's availability. Examples of implementations of systems for distributed data processing include Google MapReduce[7], Apache Hadoop[2] and Apache Spark[38], among others. Such systems need to keep track of available resources in the cluster, which can be seen as a membership related challenge while at the same time they are required to track available resources, which is a form of monitoring that can be achieved through a reliable metadata dissemination system.

Distributed Storage: Used to store great volumes of data, and usually designed to scale as the workload increases by sacrificing strong consistency or availability. Unlike traditional storage system, the query model of many of this systems is simpler exposing an interface that is similar to a key-value store. An example is the storage system Dynamo[8] that uses data partitioning scheme combined with replication techniques in order to provide availability and efficiency. Spanner[6] goes even further by relying on centralized management schemes per geographical region to provide strong consistency on client access to the stored data objects. This sort of system needs to maintain control information about the nodes of the deployment that are available at each time, as to make decisions regarding routing of requests and to manage the life-cycle of replicas. This again can be achieved by using a robust membership system based on partial views, and then use the data dissemination mechanism to build complete membership information. This approach is used by the Riak-KV[29] distributed storage system.

Distributed Cluster Resources Management Used to manage the sharing of resources, maximizing the cluster utilization. Most distributed systems don't use all the resources available on every computational task, thus the free resources may be allocated to computations from others systems or clients (Multi-tenancy model).

YARN[36] and Mesos[11] are management systems that simplify achieving these goals. These systems to provide a high quality service, efficiently share resource among multiple tenant, which, similar to distributed data processing system, could benefit from robust and fault-tolerant membership services combined with a data dissemination mechanism that is both robust and efficient, enabling all nodes to build a common and global view of the whole system state.

2.1.1 Essential Management Services

As the scale of a system increases in terms of number of components and distributed locations where these components are hosted, the complexity of managing that distributed system also increases. To deal with such increasing complexity many distributed systems rely on some of the following essential management services:

Coordination: Controls the access to shared state without breaking consistency invariants. Examples of instances of such service are Zookepeer[13] and Chubby[5], which use the consensus algorithm Paxos[17] to coordinate the execution of operations in replicated system.

Resource Management: Provides the discovery, selection, scheduling tasks, executing jobs, and monitoring to a distributed system in a similar way to what the operating system offers in the context of a single machine. This service usually requires a global view of the system and strives to optimize the distributed system resources usage, while at the same time it uses that information to ensure the completion of concrete tasks in useful time despite node failures[38].

Monitoring and Control: Provides instrumentation that offers administrators ways to diagnosis the state of a distributed system at a local and global level, which allows to assess the overall correctness of the system. The control component usually empowers an administrator to reconfigure parts of the system if the system deviates from a correct and efficient operation/configuration.

2.1.2 Membership

Many distributed systems require the most up-to-date information about its aggregated computational resource's state in order to manage and use them efficiently and correctly. The information about the active nodes on the system is provided by a membership service. While such services can be materialized by a centralized component, many large-scale systems resort to membership services that leverage on peer-to-peer technology to share the nodes' state and trace membership events through epidemic/gossip protocols. We discuss this class of decentralized services in more detail in section 2.2.

2.1.3 Failure detection

The occurrence of network anomalies and component failure are unpredictable, thus distributed system must be prepared to handle these events by design. This can be achieved by employing failure detection[27] and recovery mechanisms. The membership component is typically used as a failure detector[19] by classifying nodes' unresponsiveness as failures by performing periodic monitoring tasks and through the execution of automatic recovery procedures to exclude faulty nodes and move their tasks to other connected nodes, when required.

FALCON[22] is a system that relies on a different approach to achieve such goal. It relies on specialized complementary components, named spy modules that monitor and report the status of every software layer at each node of the system. The FALCON approach is limited to one datacenter and it adds spy modules on at least on four layers: Application, Operative System, Virtual Machine, and Network Switch. The spy modules can communicate with each other and with the component layer that they monitor. The components are arranged in a chain to maximize failure detection surface and avoid disruption such that the lowest layer L component can query the $L+1$ spy module, which allow the system to restart just the compromised components. To judge the status of a component spies require that each component implements a custom procedure that evaluates it according to its functionality.

Although FALCON approach provides fine-grained control, it is still subject to issues such as incorrect timeout parametrization. Moreover, the deployment is not trivially replicable and the solution cannot make adequate decisions on the occurrence of network partitions. Furthermore, this is a platform-specific solution, which has a non-negligible cost to integrate in any particular system, i.e, it cannot be used in systems deployed in public clouds such as Amazon EC2 as the virtualization and network components cannot be (directly) monitored.

2.1.4 Distributed Storage Systems

Many large-scale distributed storage systems are designed to scale as the workload increases by sacrificing either data consistency or availability. Unlike traditional storage system such as relational databases, the data model is simpler and follows the idea of a key-value store that exposes GET and PUT operations. Data is attributed to nodes through a data partitioning scheme that enables any cluster node to answer if it is responsible over the data or forward the request to an appropriated node. A method that increases the overall availability of the system while spreading the load of the system among the multiple nodes that compose it. As other distributed systems it is necessary to track the active nodes in order to avoid incorrect decisions, handle unpredictable network anomalies, and rebalance/replicate data. To achieve the required control, storage systems combine autonomous membership management components and failure detection mechanisms that allow the system to recover from failures and reconfigure itself when external events occur, such as nodes being added to the system and (permanently) decommissioned.

Bellow we discuss in more detail some relevant examples of distributed storage systems, that require information about the system's membership, and hence suffer from the problems being addressed in the context of this thesis.

Dynamo[8] uses a ring-based overlay to handle the system membership where each node has an unique identifier that results from the use of a modified version of the consistency hashing combined with the use of virtual nodes, where each physical machine hosts a set of virtual (independent) nodes with different identifiers. Data is partitioned across the cluster by attributing the responsibility over requests to virtual nodes whose identifier's hash match or is the closest to the data object key. The number of virtual nodes are distributed per physical node according to each node capacity. Furthermore, nodes join the system by contacting special contact nodes called Seeds, which can be defined through a configuration file or provided by an external independent service.

Dynamo's membership management lacks automation, additions and removals are explicit and performed by human operators or system administrators and upon change the modification is propagated through all nodes using a dissemination protocol. Additionally, each node cooperates with others in an exchanging their view of the system to guarantee that it is up-to-date and because every node will eventually exchange its view of the system with a seed node, which avoid logical partitions. The occurrence of node failures have no consequence on the membership because every modification is explicit but Dynamo forwards request to back up nodes until the failed node recovers.

Cassandra[16] is a storage system inspired on Dynamo however, Cassandra uses a hash function that takes into consideration the load of cluster node. To this end the identifier of a node is assigned by a key coordinator. This allows the system to move lightly loaded nodes on the ring to alleviate heavily loaded nodes, which provides a deterministic load. The membership is based on Scuttlebut[28], an anti-entropy protocol that offers efficient CPU utilization by analysing gossip updates and the local resources available. Similar to Dynamo's membership, addition and removal of nodes are explicit, thus no re-balance of partitions or repair procedure for unreachable nodes is automatically performed by the system.

Cassandra handles failures following the same guidelines defined by Dynamo, but it relies on an additional failure detection mechanism that outputs a nodes's suspicious of failure degree and upon failure detection Cassandra resorts to a hinted handoff strategy. While performing the hinted handoff strategy requests are forwarded to a backup node that will periodically check if the faulty node has recovered. When a node recovers, all messages stored by the backup node are sent to it.

Riak [29] is another storage system inspired by Dynamo that inherited most of its design decisions related to the membership management, the failure detection and handling. It differs on the key distributed scheme by using a pre-calculated partition list instead of a variant of consistent hashing. The failure detection and handling follows the same guidelines that Dynamo combined with the hinted handoff strategy employed on Cassandra. For message dissemination, Riak uses a latency-optimized version of the PlumTree protocol[18](whose details can be found in section 2.2.3.5). This message dissemination mechanism is used to build a robust and efficient metadata dissemination system that is used to propagate control information(including membership information) throughout all nodes of a deployment. We follow this design choice to get the motivation for building a membership system based on partial information that can support a highly robust and efficient metadata dissemination service.

2.1.5 Summary

Many businesses that try to reach a extremely high number of customers have to deploy geo-distributed services across the planet, a scale that brings management and maintenance challenges. Those services must handle, by design, component failures, network anomalies, and provide high level of automation without increasing organizations' maintenance costs unreasonably. To achieve these goals, distributed systems rely on essential services that monitor, coordinate, and manage resources in the infrastructure. However, these essential services require knowledge about nodes' status to track the membership and efficiently disseminate information (namely relevant control and membership metadata) throughout the system. To support such membership and dissemination mechanism, systems can leverage very efficient and scalable gossip protocols that provide fault-tolerance and reliability. However, such (well known) approaches still lack the adequate level of automation that is necessary for large-scale dynamic environments and that can cope adequately with network partitions. In particular, we have shown that most NoSQL distributed storage systems still employ a very simple membership service that was first introduced in Dynamo. Due to the limitations discussed above, in the thesis we plan to push forward the design of such membership services enabling the operation of efficient and robust metadata dissemination services on top of them.

2.2 Gossip Protocols

Gossip or epidemic protocols are used, among other purposes to disseminate information across large number of participants in distributed systems. The message propagation pattern resembles the propagation of a biological virus on a population. In the context of Peer-to-Peer (P2P) architectures, a node propagates each message to a predefined number of its neighbors (a parameter usually named fanout, t, typically with a value logarithmic with the total number of nodes in the system). The nodes to which each node propagates messages are selected among its neighbors in the overlay network. The receiving nodes will perform the same procedure to their neighbors and so on until all nodes receive the message. Nodes only forward messages that they receive for the first time, which implies that each node maintains information about messages previously propagated in the systems. This transmission pattern allows the system to distribute the load evenly among nodes and leads to the emergence of multiple independent dissemination paths (depending on the employed fanout), which is crucial to provide the necessary redundancy which is essential to provide failure tolerance and high message delivery reliability.

2.2.1 Peer-to-Peer

Peer-to-peer (P2P) systems have arisen as an alternative model that offers better scalability, fault-tolerance, and load balance properties than the classical Client-Server model. The Client-Server model follows a centralized architecture that attributes different roles to each host. This model, in order to scale, requires to increase the number of servers where sub-sets of those are designed for specific tasks, however the new equipment has an undesirable added cost and servers' specialization might lead to unbalanced load distribution. The load distribution problem results from the tasks' frequency as well as the amount of resources required to execute each task. Consequently, some servers might consume all their resources while others might be mostly idle. Additionally, some servers might be stalled while waiting for the completion of sub-tasks in overloaded servers. This leads to sub-optimal resource utilization. Additionally, servers specialization introduces points of failure in the system because a service, or ever the whole system, can be compromised if a sub-set of specialized servers fail simultaneously. Although some systems apply a fail-over strategy that requires additional equipment, it results on additional maintenance costs and potentially leading to an increase of idle hardware resources.

A system that follows the P2P model assigns to each node both roles of Client and Server. Thus each node contributes to the system progress by sharing resources and cooperating on task execution. As every member is equal from the system perspective, it is easier to distribute tasks across the available resources and on a failure occurrence, the system can replace the node by any other without additional equipment or significative management overhead. The described behaviour allows this kind of system to be resilient, robust, and to scale due to its use of a decentralized architecture unlike the Client-Server model.

2.2.2 Message Dissemination

An essential aspect of distributed systems, that was already mentioned previously, is the exchange of information among nodes, and in particular the dissemination of data to large number, or even all, components of a system.

Message dissemination strategies affect the delivery reliability, the redundancy of messages (i.e, the overhead associated with network usage), and when the information being disseminated is membership control information it might also have a direct impact on the overlay regeneration speed. The fundamental communication mechanism that are employed to design gossip protocols can be classified as:

Eager push Nodes send new messages to their neighbors as soon as they received it for the first time. This strategy allow a fast message dissemination but a high number of selected neighbors(fanout) for the procedure may increase the message redundancy considerably and increase the overhead imposed on the communication links.

Pull Nodes request their neighbors for information about new messages periodically and if they have any, the node requests its payload explicitly. Although it reduces the amount of bytes exchanged on the network (for messages with large payloads), the rounds' frequency may cause nodes to congest the network with unnecessary messages if the interval between rounds is too small or slow down the propagation otherwise.

Lazy push it's similar to the pull approach however nodes, similar to the push strategy, send the identifier of a message to its neighbors as soon as it receives that message payload. Receivers can then explicitly request the message payload if required (for the firsts time). It has the advantage of reducing the amount of information exchanged in the network.

Hybrid while there are multiple possible combinations of the strategies discussed above, one of the most common is to combine the eager push with the lazy push. It uses the first as the primary message dissemination strategy for a fast delivery and complements it with the second strategy for failure recovery.

2.2.3 Overlay Network

Connectivity relationships among P2P nodes form overlays that may be used to disseminate messages related to the system's service or its self management. These overlays are networks composed by logical links that abstract the characteristics of the underlying network, which may be another overlay or the physical network topology. An overlay can be categorized concerning its topological properties[21] as: Structured, Non-Structured, and Partially-Structured. In the following sections, we discuss the properties of each overlay type and provide the description of relevant examples found in the literature.

2.2.3.1 Structured Overlays

Structured overlays are networks where the connections among nodes follow a predefined structure. P2P protocols force structure on the overlay to provide better routing and resources localization primitives. The most relevant protocols of this kind are Chord[33], Pastry[30], and Tapestry[4]. Although structuring the overlay improves search primitives, it decreases the flexibility to handle churn[1] scenarios because it requires nodes to be routed to a specific logical position in the network when they join the system and conversely when a node is detected as failed, only a small subset of nodes in the system can be used to replace its previous logical partition in the overlay. Therefore structured overlays aren't as resilient as unstructured overlays because of the lack of flexibility inherent to the operation of healing mechanisms.

Chord[33] is a protocol that builds a structured overlay using Consistent Hashing to provide efficient lookup primitives. Nodes are organized in a ring considering the relative order of their identifiers (the identifiers space wrap around to allow this). Additionally the membership maintains also a small list of node identifiers that provide shortcuts in order to reduce the number of hops required to transverse the ring. The protocol correctness relies on every node maintaining the correct successor. The incoherence of the remaining overlay links only affects the routing performance. Concurrent additions and removals to the membership may break the overlay. Thus, Chord applies periodically a stabilization protocol at each node to correct the maintained overlay links. However this mechanism is unable to repair an already partitioned overlays. Furthermore, all its fault-tolerance is based on the stabilization protocol and it is not able to tolerate intense-failure scenarios (i.e, scenarios where large amounts of nodes fail simultaneously).

Pastry[30] is a self-healing protocol that offers efficient location and wide-area routing primitives by exploiting nodes' locality. Similar to other protocols, it assigns random identifiers to nodes and maintains a table that is used to support routing between nodes (in the identifier space). This table uses substrings of the node identifier to select which nodes to maintain links to. Additionally each node has 2 additional structures that provide back-up nodes to handle failures. The routing algorithm for a node starts by consulting the structure with its closest neighbors and if it can't find the target id it resorts to the table to forward the message to a node with the identifier closest to the target of the message. Nodes join the system by contacting a contact node that will notify the new node closest neighbors of its arrival and to finalize the process they exchange their routing structures content. Nodes leave silently and Pastry handles them in a similar fashion to the way it handles failures, by replacing failed nodes with nodes maintained in its additional local structures. If the failure was detected on discovery, the message is forwarded to another entry of the same level, otherwise it exchanges the structures content with the closest node to the faulty one to update them.

Tapestry[4] offers location-independent routing that leverages locality while offering self-organizing and fault-tolerant properties. Data is partitioned by assigning objects to multiple nodes, which will be the object coordinators. If a request cannot be routed

[1]Churn is caracterized as a period of time when the frequency of arrival and departure of nodes in the system is extremely high.

to the adequate node, the node that is unable to further forward the message becomes responsible for the object contained in it. To route messages nodes maintain a routing map where each level points to nodes that match the route's owner id suffix to a certain degree. Closer nodes will have more id bits in common. Data is published by sending a message to the root informing that it has a new object and along the way intermediate nodes also store it. Queries return a set of nodes containing the data to which is applies a selector operator to filter against locality metrics. As a self-healing procedure the protocol periodically sends heartbeats to nodes that point to it to detect faulty nodes and corrupted tables. To any faulty node, it is given a chance for recovering within a period of time and if it fails to do so, it is simply removed from all routing structures. Additionally, every entry on the table has two potential backup nodes as alternative paths to reduce the delay upon the need to recover from a node failure.

2.2.3.2 Summary

Protocols that build structured overlay network support efficient routing primitives, though the structured approach reduces their capacity to adapt to dynamic environments. Furthermore, if the protocol operates over a global view of the system at all times then it is difficult to enforce that all nodes have the most up to date information of the system membership. Otherwise, nodes only require information about a few others which scales better but it is also difficult to enforce the constraints on the topology that must be maintained among nodes. Although protocols for building and maintaining structured overlays may provide fault-tolerance and self-healing properties, the synchronization overhead and the amount of information to track don't allow these types of systems to scale to thousands of nodes easily.

2.2.3.3 Unstructured Overlays

Unstructured overlays don't impose significant restrictions on how links between nodes are established. Nodes join the system through an external mechanism that provides a contact node, upon receiving a request to join, the contact node may establish a connection with the new node and notifies some of the other members by flooding the overlay or by using a set of random walks through the current overlay topology. Thus the resulting overlay won't have any specific structure having a topology that is essentially random in nature, which provides high flexibility to churn, high resilience, and robustness. Although these properties are fundamental to provide fault-tolerance, the randomness is not ideal to construct routing and localization primitives that many file-sharing or data storage systems require. However, these overlays are enough to manage memberships, which may operate with partial information of the system. This concept is typically exposed to applications as a *peer sampling service* that provides other modules a set of nodes that might be used on their tasks. Those nodes come from the materialization of the partial information in a set of node identifiers maintained at each node by the membership, named *partial view*. Examples of systems that build unstructured overlays are Scamp[10], Cyclon[37] and HyParView[19].

SCAMP[10] is a simple peer sampling service and self-organizing protocol that builds a randomized overlay which relies on partial views that contain slightly more nodes than $c * log(n)$ where c is a constant related with the desired fault-tolerance and n is the maximum number of nodes expected to join the system. The protocol was designed to ensure high reliability and robustness. The authors affirm that using the mentioned threshold the probability of all correct nodes receiving a message is close to 100%. Initially, a node joins the system by sending a subscription message to an arbitrary member that will add it and forward the message with the new node information to c randomly chosen nodes from its view. A node that receives a forwarded copy of this message will add the new node to its view according to some probability p, which depends on its local partial view size. Otherwise, it will forward the message to a random node among its neighbors. Departing nodes send an unsubscription message to their neighbors, which will remove the node (if it was contained in their partial views). To avoid a logic partition which may happen if all its neighbors fail or unsubscribe, each node will track the time between messages' exchange and when it surpasses a predefined limit triggers a resubscription to the system.

Cyclon[37] offers better connectivity, average path length, average clustering coefficient with high resilience, high failure tolerance, and fast self-healing compared with Scamp by applying periodic shuffling of membership information among nodes of the system. Unlike other protocols, Cyclon node's identifier contains an additional value representing the number of shuffle rounds that have passed since the creation of that reference, which is called the age of the identifier. The shuffle starts by incrementing the age of all its neighbor's identifiers and then selecting the oldest, to which it will send a message containing a subset of its partial view after inserting in this message a new identifier for itself with an age of zero. The neighbor upon receiving it, responds with a subset of its view and then, both nodes proceed to execute the merge procedure where they integrate the remotely received sample in their own partial views of the system. The participating nodes will add nodes until their views are full and then replace the entries that they sent by entries received remotely. The procedure offers good resilience and failure tolerance because eventually all neighbors of each node are tested by establishing a connection during the shuffle procedure however, failure detection in this fashion is only recovered in the following successful round. Even on the occurrence of a massive failure, the surviving nodes have a high probability of having others correct nodes on their views, thus at most n cycles are necessary to discard all dead links, being n the size of the view.

HyParView[19] is a peer sampling service that offers high resilience and high delivery reliability even in cases of extreme failure, which allows to achieve 100% reliability for message dissemination when 80% of nodes failed simultaneously. To accomplish this, it relies on a hybrid approach that uses two partial view of different sizes and each with different maintenance strategies. The principal view called active view represents the active nodes (or neighbors) and it uses a reactive strategy that reacts to join/leave/failure events. The other, named passive view, contains back-up identifiers to handle failures and applies a cyclic strategy that periodically triggers an information exchange procedure on each node. HyParView uses TCP as an unreliable failure detection mechanism that it is used to transmit messages, thus testing at each connection if the neighbor in the active view has failed. The hybrid approach applied allows the system to recover from failures in fewer rounds that previous approaches and to guarantee the global overlay connectivity, the reachability of all nodes and low clustering coefficient, which benefits the latency and efficiency of message dissemination.

2.2.3.4 Summary

Unstructured overlays offer self-healing and reliable broadcast primitives capable of supporting dynamic environments. Furthermore these require smaller partial views of the system (at each node) and cooperate with low fanout, thus generating less redundancy while guaranteeing atomic broadcast even when a high number of individual nodes fail simultaneously (in the HyParView's case). However, the lack of structure may cause the overlay to mismatch the underlying topology, thus blocking the protocol capability to leverage the full potential of the network. However to disseminate information in a reliable and efficient way, unstructured approaches are better suited and hence in this work we focus on membership services that resort to an unstructured design.

2.2.3.5 Partial Structured Overlays

Partial structured overlays are obtained mostly from unstructured overlays in order to leverage their fault-tolerance and adaptability guarantees. Typically over the networks generated by unstructured overlay maintenance algorithms one applies an optimization procedure. The optimization consists mostly on link's exchange between nodes that benefits the protocol according to a pre-defined metric, enabling for instance, the overlay network topology to benefit from knowledge regarding the underlying network topology to provide more efficient delivery mechanisms than would be possible with a completely random unstructured overlay. Examples of such protocol are presented bellow.

T-MAN[14] uses a peer sampling service that provides random nodes and improves the random overlay by applying a ranking function to the nodes. The function takes as input node identifiers that unlike other protocols contain additional node's profile data. T-Man optimization operates periodically, each node selects a candidate to exchange local information and then creates a collection with its identifier, its view's content, and a list of identifiers from nodes provided by the peer sampling service. The candidate after receiving the message executes the same steps as the initiator of the process and after both nodes have received each other lists, they merge them with their view and finally they select the best nodes for their local view by applying a specialized ranking function to the resulting merge list. The usage of an underlying peer sampling service provides robustness to the protocol, however the optimization does not maintain the nodes degree which may cause unbalanced load distribution and even graph connectivity issues, as shown in [20].

Araneola[24] optimizes the overlay by biasing the number of identifiers in a node's partial view in function to a parameterizable threshold. The overlay construction and maintenance rely on three tasks that handle joins, leaves, and failures, which verify if the partial view size oversteps the threshold and react accordingly. To guarantee that the partial view has the most recent information each node exchanges random identifiers from their views through piggyback periodically. The author propose an additionally version that exploits the network proximity and bandwidth heterogeneity by adding links to nearby nodes, which requires a specialized component to evaluate the links and a task to establish these additional connections. On Araneola, message dissemination is performed in gossip rounds, where each node floods the new message identifiers though the overlay. Those messages may piggyback additional requests for instance, the payload of the messages that the sender is aware that exist but have not yet received. By packing messages and disseminate through gossip rounds the protocol reduces the bandwidth overhead and it allows also a better control on messages delay by tuning the frequency of these gossip rounds. The fine-grained control over the view size allows Araneola to offer high delivery reliability and load balancing when parameterized correctly.

GoCast[35] was designed to tune/control nodes' degree in order to offer reliable and resiliency properties. Additionally, it bias the overlay in function to the nodes' proximity to offer better latency while allowing partial views to hold an additional small number of random nodes to provide robustness in term of global overlay connectivity. Periodically, nodes evaluate their neighbors by comparing the latency between links to these peers and analyses the neighbors' degrees. The analysis of these two factors allows to optimize, without risking to remove or establish connections to nodes in critical situations, avoiding the emergence of nodes with high or extremely lower popularity, i.e, translated into the number of overlay neighbors. GoCast uses a hybrid dissemination approach that allows to efficiently deliver messages and recover missing messages that are propagated through the primary push-based strategy. Additionally, a special node designed root, floods periodically the network with a heartbeat as a failure detection mechanism. GoCast has a strict control over the node's degree to efficiently distribute the load evenly among all participants.

X-BOT[20] is a protocol that optimizes random unstructured overlay networks in function to a predefined criteria. It combines a 4-node optimization technique and a local oracle that outputs the link cost according to the target criteria (e.g. lower latency, high bandwidth, etc). X-BOT was built on top of HyParView and it has the directive of preserving the properties that the initial overlay had except during the optimization procedure, where it attempts to periodically exchange some links in the overlay by better ones, only if the node has a full active view. Furthermore, the protocol does not bias a parameterizable number of neighbors to provide robustness and the passive view contains only unbiased nodes. The 4-node coordination strategy to exchange overlay links, after its conclusion, guarantees that the nodes' degree remain the same, a characteristic that allows the overlay to have low clustering coefficient, balanced load, and be robust in face of failures. Additionally, X-BOT's conditions to perform optimizations make it impose low overhead on the system, by avoiding sudden changes over all neighbors of a node in the context of a single optimization round.

Plumtree[18] results from the use of a deterministic tree-based broadcast with an epidemic protocol, in order to achieve low overhead, while providing high fault-tolerance and high reliability. It uses the hybrid dissemination strategy and the first message propagation is used to build a spanning tree which is reused on future propagations until some node or link fails. Furthermore, each dissemination operates over two distinct neighbor sets that apply the following policy. A node when receives a new message (i.e, payload) adds the sender to the set of the first phase (eager push dissemination) and informs the sender to guarantee the link's symmetry, otherwise the sender is added to the other set (lazy push dissemination). This policy allows the protocol to remove redundant links (from the first phase). Additionally, nodes assign timers for the payload as soon as they receive the message identifier (by lazy push) to detect slow links or faulty nodes. In that scenario the node that provides the identifier of a message whose payload is not received within the timeout takes the place of the faulty node and on the next message dissemination any redundant links created, due to independent decisions taken by all nodes that perceived the failure, will be removed effectively ensuring that the links used for eager push form a tree. Although Plumtree is resilient and has self-healing properties, it is only optimized for the first sender. Alternatively each node can maintain a spanning tree optimized for itself but it imposes additional overhead on the system and restricts the solution scalability to large system sizes.

2.2.3.6 Summary

Protocols that build partially-structured overlays provide efficient delivery mechanisms but there are several situations to be avoided to ensure that the resulting overlay network maintains important properties of completely random unstructured overlays. It is important to maintain some unbiased neighbors to provide additional robustness and global connectivity as well as to maintain the nodes' degree to provide better reachability and throughput. Furthermore, optimizations should only be performed under stable operational conditions and through small modification to avoid accidental logical partitions to occur.

2.2.4 Fundamental properties and metrics

The efficiency of gossip protocols depends on the quality of the underlying overlay on which it operates. A high quality overlay must satisfy several graph's properties which are listed below accompanied with associated evaluation metrics used to verify these and related properties in practical systems:

Connectivity: States that there is a path that connects every pair of node. It guarantees that messages sent from any node are (or at least can be) received by every other node.

Degree Distribution: It states that the nodes' degree (the number of neighbors maintained by each node) must be evenly distributed. The distribution affects the reachability (in-degree), which for a node corresponds to the number of nodes that have it as a neighbor. The out-degree corresponds to the contribution of a node for maintaining the overlay connectivity, which is equal to the number of its outgoing neighbors. Solutions where overlays links are symmetric have similar in-degree and out-degree.

Clustering Coefficient: A metric that evaluates the density of the nodes neighboring relationships redundancy and is related with the probability of a subset of nodes becoming isolated from the rest of the network on the presence of failures. The value corresponds to the average of all nodes' clustering coefficient. For a node, it is number of edges between its neighbors divided by the maximum possible number of edges across its whole neighborhood.

Average Path Length: Value obtained by calculating the average of shortest paths between all pairs of nodes. This metric affects the message dissemination time, thus lower values make the dissemination more efficient (and usually faster).

Accuracy: The accuracy of a node allows to analyse the reliability of the membership protocol in selecting gossip targets used by the dissemination protocol. It is fundamentally defined, for a node, as the fraction of nodes in the partial view of that node that are correct. The overlay accuracy is the average of all node's accuracy values.

Reliability: Value that measures the fraction of correct nodes that can receive a message disseminated by a gossip protocol. Atomic broadcast corresponds to the successful delivery of a message to all active node, which corresponds to a reliability of 100%.

Relative Message Redundancy: A metric that analyses the overhead imposed by the message redundancy of a gossip dissemination protocol, applicable when at least two nodes receive the message. The value is calculated by $(m/n-1)-1$, where for a message the value corresponds to total of messages transmitted m and the total of nodes that received it n. Note that this metric does not take into account any control messages exchanged during the dissemination procedure.

2.3 Network Partition Tolerant Distributed Systems

Different distributed system mentioned on this chapter, depending on their category, rely on different mechanisms to tolerate network failures, namely network partitions.

Distributed Data processing system such as Apache Spark[38] resorts to the recomputation of tasks upon the unresponsiveness of worker nodes. Therefore, unless the network partition affects the distributed file system and thus the data availability, these system can make progress at the expense of duplicated work. Furthermore, these system leverage the Multi-Master strategy that allows the replacement of the master if the current master is affect by the network partition (i.e, becomes unresponsive), unless the distributed coordination system that accompany the system is affected as well, for instance by becoming impossible to gather a majority quorum at this subsystem, which forces the whole system to stop until the partition is healed.

Distributed storage systems deploy data partitioning schemes and data replication strategies that many times require any node to know every other node in the system. The high importance of data loss avoidance require that these systems, in face of failures, do not misclassify unresponsive nodes as failed, hence the cluster topology is manually defined and deployed by a human administrator[16][8][29]. Therefore, these system tolerate the network partition in exchange of non-negligible operational cost, by leveraging human operators. The solution proposed in this work can avoid such costs by automating the recovery process, because, and as we futher detail in the following chapter, our solution does not discard information about faulty nodes due to the location awareness aspect of our membership protocol design.

Mesos[11] is a Distributed Cluster Resources Management system that explicitly terminates all nodes that attempt any communication after missing the health check message transmission, hence all nodes affected by a network partition that are marked as unresponsive by the leader become unable to resume their normal operation without explicitly rejoining the system, which might lead to undesirable overhead[25]. Similar to other types of systems, Mesos relies on coordination systems, such as Zookeeper[13], to elect leaders if the current leader misses its health check. Depending on the distribution of the nodes affected by a network partition, the protocol present in this work can maintain the availability of this system, if both sides of the partition have potential leader nodes. Our solution can be used by these systems to simplify the management of their membership and avoid the premature trigger of failure detection by nodes. Regarding the coordination subsystem, our solution can potentially assist these nodes to detect that previously inaccessible nodes that have again become available.

Coordination systems such as Zookeeper[13] are integrated into diverse distributed system to address coordination issues, thus they are services of extreme importance. These system track the node's membership with periodic health checks and tolerate partitions on the side of the partition with a majority quorum of nodes. The side with the quorum can operate as normal and if the leader was affected the majority of nodes can elect a new leader. In contrast the non-quorum side ceases all operation or just allows read operations until the network partition is fixed and the system state converges[12]. While the protocol presented in this work cannot allow the quorum and non-quorum side to operate as independent systems as to avoid safety violations, it can be used to detect the recovery of the partition and accelerate the merge of the system state, as long these system support data conflict resolution mechanism such as CRDT[32].

2.4 Discussion

To guarantee the correctness of distributed system and to allow reliable and efficient services to be built on top of them requires the use of fundamental mechanisms to, among other, monitor, coordinate, and manage resources in the infrastructure. A frequent way to ensure these systems are scalable it to employ peer-to-peer solutions for instance, to track the system membership. In particular gossip protocols that allow the management of the system's membership and provide efficient information dissemination strategies across all nodes. Furthermore, gossip protocols are a building block for designing and implementing fault-tolerant, reliable, and highly adaptable unstructured overlay networks. However, the lack of structure of these overlays may result in scenarios of topology mismatch that may disrupt the information dissemination efficiency.

Modern distributed system still rely on very primitive structured overlays that still lack some desired autonomous management mechanisms and that still present some limitations in terms of scalability potential. Additionally, existing solutions still own very simple, if any, mechanisms to efficiently and reliably tolerate (transient) network partitions. Our proposed solution tackles these limitations of large-scalable distributed system by enriching unstructured overlays partial view management leveraging techniques to infer the location of nodes to obtain the necessary robustness to handle and recover from network partitions.

Chapter 3

Partition Tolerant Membership Service

This chapter presents and discusses our solution for a distributed membership protocol (based on partial views) that is tolerant to network partitions. We start by explaining the main objectives of this work followed by the assumed system model of the system, the intuition behind our protocol, and the component architecture, on sections 3.1, 3.2, 3.3, and 3.4 respectively.

Section 3.5 introduces location oracles, starting with a general description of an oracle, followed by subsections where we present the details related with the heuristics that allow nodes to infer locations for each of the proposed Location Oracles. Section 3.6 presents the extension for the protocol HyParView and discusses how it leverages the previously mentioned Location Oracles to enable the membership protocol to converge to a correct configuration after the recovery from a partition.

Finally this chapter ends with a discuss on the implications for the message the dissemination mechanism and an overview of the presented contributions, on sections 3.7 and 3.8 respectively.

3.1 Objective

The objective of the work presented in this thesis is to develop a membership service that tolerates network partitions in an automatic and operational cost-free fashion. Network partitions cause distributed system nodes to temporarily become isolated into subgroups that are incapable of exchanging information among each other. In face of such scenario, many current systems, i.g, Cassandra[16], Dynamo[8] and Riak[29], resort to manual membership management, while some resort alternatively to the Master-Slave architecture with multiple nodes serving as backup. These last ones rely on on configured timeouts to suspect on the failure of the current leader, and elect another one. However, these last solutions are typically employed by systems that require strong consistency, whereas in our work we focus on system that operate under weaker forms of consistency, i.e, causal[1][23] or eventual consistency[8][16][29]. However, both of the two alternatives mentioned above have non-negligible costs in terms of operation(e.g, human system administrator) and recovery speed.

Current membership protocols, such as HyParView[19] offers a fast recovery, scalable, and robust membership solution even in face of high rates of failure. These membership services can support very efficient and robust gossib-based dissemination protocols that achieve 100% delivery reliability in scenario with levels of failures (as high as 80% simultaneous node failures). However, HyParView is too eager to classify unresponsive nodes as failures and discards their information because it's not equipped with mechanisms that allow it to distinguish between simultaneous node failures and a transient network partition. Therefore, it is unable to correctly and automatically recover from this type of failures. However, its overlay properties can be leveraged to develop solutions that can cope with network partitions. The work presented in this thesis offers such solution, by extending the functionality of the HyParView protocol and incorporating into it components that assist on decisions related with fault handling, attempting in some high level sense to detect network partitions and distinguish them from simultaneous node failures.

3.2 System Model

Our solution focus on distributed system that aim to provide services globally (i.e, large-scale systems) with decentralized characteristics(many of whom can be found on peer-to-peer systems and in distributed data storage systems) where the load should be (evenly) distributed and faulty nodes easily replaced by the remaining available nodes. The membership protocol developed offers a membership service to be integrated on existing systems, such as data storage systems that have nodes distributed across multiple data-centers.

Our solution, as it will become apparent in the following text, further assumes that nodes behave correctly(i.e, they are not malicious), that faults that modify the communications content are infrequent, and network partition occurs essentially on communication links across data-centers. Therefore, under our model nodes are grouped in clusters(one per data-center) and network partitions will essentially make communication temporarily impossible between nodes in (some of) these groups.

3.3 Insight

The central challenge of tolerating partitions in membership protocols such as HyParView consists on distinguish simultaneous node failures from unresponsive nodes due to a partition under the asynchronous communication model. This however is fundamentally impossible[9]. However, if a system had knowledge about each node location, such information can be leverage to infer that simultaneous nodes failures from a particular location can be consequence of a network partition and trigger the adequate recovery procedures to handle that situation. However, such information may not be available for the system since large-scale deployments should strive to avoid individual node configuration to lower management cost. We can however infer such information by monitoring and collecting metadata related with the system deployment.

Following this insight, our solution incorporates a component, named Oracle, that is responsible for collecting and processing the metadata related to the communications among different nodes of the distributed system, in order to infer different locations of these nodes and assist them on failure related decision making. Oracles allow membership services, in face of partitions, to handle faulty nodes differently and equip membership protocols with better tools for fast recovery procedures to repair system that had been partitioned.

3.4 Architecture

The membership is a component implemented bellow the application as shown in Figure 3.1 and it can be further divided into two subcomponents: The data dissemination mechanism and membership service. The membership service manages all the state related to the nodes activity status and provides samples of nodes to the other components for use on their activities. The data dissemination mechanism leverages the information provided by the membership to efficiently disseminate the data (or metadata) for the application across all nodes, typically following a gossip-protocol strategy and using the dissemination strategies mentioned on Section 2.2.2.

The membership component operates over TCP. These connections are managed by the Operating System, ensuring the symmetry of the partial views managed by the membership when these views imply the existence of an active TCP connection. Furthermore, and similar to the original design of the HyParView[19], we rely on TCP as an unreliable failure detector that assists the membership protocol in recovering efficiently and timely from failures.

To assist on the decisions related to recovery from failures, our protocol has an additional component, named Oracle, that is an intermediary between the OS and the membership. It analyses and processes metadata related to the communication among nodes of the system in order to extract useful information to attribute a location label to each node, enabling the local clustering of nodes among different locations(i.e, a node can infer with some precision if two nodes are co-located). It has also the ability to piggyback metadata into messages exchanged by the overlay management protocol or the dissemination protocol as a way to transmit and receive information for oracles on other nodes (if the oracle design requires exchange of information to operate).

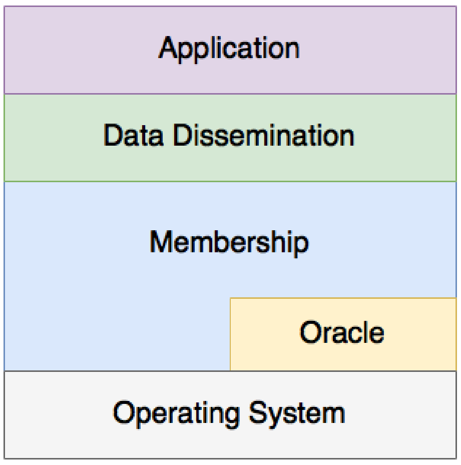


Figure 3.1: System Architecture

3.5 Location Oracles

In this section we present an overview of Location Oracles design and further explain how they are relevant to our solution. Furthermore, we present the design of four different Location Oracles based on four distinct heuristics that enable the inference of a node location or proximity. We conclude this section with a discussion comparing aspects of our different oracle proposals. The three different Location Oracles proposed in the thesis are: Static Label Oracle (SLO), IP Address clustering and latency Oracle (IPCLO), Anchor Oracle (AO), and Dynamic Label oracle (DLO).

3.5.1 Overview

Oracles are inner (in relation to the membership) components that collect and process metadata with the purpose of extracting meaningful information that provide to nodes additional information related with the relative location of nodes in a large-scale deployment. In our context they operate on membership messages exchanged between nodes, with the purpose of inferring the proximity between them. This information is then associated to the identifiers of the nodes as labels, which already contain information about IP and the port which are essential to the correct operation of the membership service.

An oracle exposes the following API:

getLabel(Identifier id): Method that given a node identifier (which is dependent on the membership protocol logic). The oracle, returns the location label for that node according to the metadata collected.

3.5.2 Static Label Oracle

The Static Label Oracle (SLO) uses a label that is configured during the deployment process and is immutable throughout the life of the node. The label configuration can be performed manually, however it increases the burden on the person accountable by the system configuration and deployment since it's prone to misconfiguration. An alternative it is to use external services if these are available. Cloud services providers typically expose an API that allow systems using their infrastructures the functionality to determine the data-centers allocated to them, which reduces the burden on the administrators while offering accurate location inference, under the assumption that the cloud managed services are correct. On the Amazon cloud platform obtaining the location label using the SDK can be done using the *Regions.getCurrentRegion()*[1] API call, which does not require the application to have any administrative permission to access such information.

3.5.3 IP Address Clustering and Latency Oracle

The IP-based Clustering and Latency Oracle (IPCLO) extends the technique presented in [15] that monitors latency communication between nodes to infer if another node is close or distant. Contrary to what is suggested in[15] we aim to be able to distinguish if two distant nodes are themselves in the same location or not. To that end we take into consideration the common IP prefix between nodes. We do this because we expect that nodes on two different locations will have considerably different IP prefixes and latencies making possible to infer that such nodes are not co-located. This technique can be generalized to any number of different locations, even if the latencies are similar across multiple locations.

The technique leverages the longest common IP prefix length (LCPL) and the latency among nodes to generate different location labels for nodes. This is achieved based on a rule that establishes the minimal LCPL and the maximum average latency between two nodes for them to be classified by each other as having the same pre-configured label. For clarity we provide a rule structure and an example:

Rule's structure: (Min number of common bits on the IP, Max average latency) => $Label_i$

(16, 20) => $Label_A$

(8, 60) => $Label_B$

(0, 10000) => $Label_C$

The set of rules allow to classifying nodes, the values in each node need to be tunned to each deployment case to enable distinguishing between different far way locations. Furthermore, among nodes of a cluster there has to be some level of similarity over the

[1]https://aws.amazon.com/blogs/developer/determining-an-applications-current-region/

IP prefix, otherwise if the IP address are attributed randomly this technique will not be able to infer correct locations.

3.5.4 Anchor Oracle

The Anchor Oracle(AO) is similar to the SLO but instead of using the cloud platform API or the manual configuration by the system administrator, it uses an external service that is agnostic to the infrastructure. This service is composed of several servers that accompany every cluster of nodes, named Anchors, as depicted in 3.2. The oracle operates by measuring the latency to each of these servers periodically by sending a ping request. It then selects the anchor with the lowest latency and sets the local node label with the label of the chosen anchor. Alternatively, if the anchors don't have a label pre-defined, its IP prefix can be used because every membership node will converge to the same set of labels. These labels are then integrated into the node identifier used (and propagated) by the membership service, in a similar way to the SLO.

The Anchor Oracle exchanges the configuration burden for the burden of maintaining the additional anchor servers. Furthermore, to avoid misclassification, there has to exist more than one anchor server in each deployment location, otherwise the failure of one of these servers might lead to the incorrect inference of a node location.

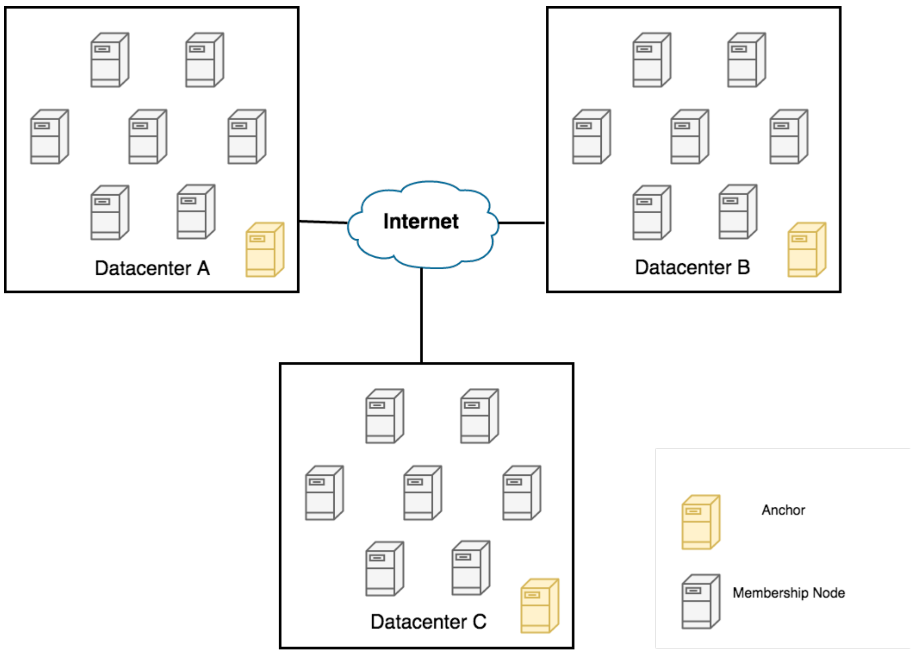


Figure 3.2: Anchor Oracle and anchor servers' deployment

3.5.5 Dynamic Label Oracle

The Dynamic Label Oracle (DLO) is an oracle that leverage communication between nodes to extract information that is used by that node to decide to keep its current location label or to adopt the label currently used by another node. At the beginning, each node oracle generates a unique identifier that is used as his initial location label. Periodically, all nodes ping their neighbors to measure the latency between them, upon which they decide to exchange their label if the latency measured is below a pre-defined threshold value, a parameter that establish that every node whose communications latencies are below that value are in the same exact location. As a consequence of updating a label, the nodes must inform theirs neighbors about the change, in order to allow every node of that particular region to converge to the same label. Additionally, upon detecting that a neighbor of the same region changed its label, a node also adopts the new location label.

As opposed to previous oracle, the DLO has the advantage of not requiring any external service, however, the label convergence is not instantaneous and hence, during initial period there will be incorrect classifications, but the periodic interaction between oracles will ensure convergence guarantees the convergence.

3.5.6 Discussion

The Location Oracles presented have different trade-offs and operate under different assumptions, being therefore unable to operate in every possible settings. The SLO has the advantage of offering perfect classification since the location label is defined at deployment time. It shifts the operational and administration cost to the deployment but it is highly susceptible to misconfiguration, which is the main cause of failure on many systems, if the execution environment does not provide an automatic and robust mechanism to obtain a location label. As alternative, the SLO can be configured to leverage the cloud infrastructure API if available, which will nullify the operation cost mentioned, however this may not be a option for in-house infrastructures(i.e, infrastructures operated by companies themselves).

Alternatively the IPCLO is promising for infrastructures where the administrator has total control over the IP attribution of each machine. However, such solution requires an empiric analysis over the setup in order to define enough and correct rules that partition the nodes across the desired locations.

The AO shifts the manual operation cost to the cost of maintaining the external service (anchor servers) and guarantee its availability. However, message delays may create periods of misclassification due to peak latency values, which is a phenomenon that might also affect negatively the IPCLO.

Finally, the DLO offer the most autonomous management of all oracle because it does not have any external dependency, however, it may have an initial period where it provides wrong classification if the label assignment has not converged yet.

3.6 Partition Tolerant Membership Service

3.6.1 Overview

In this work we employ our Location Oracles as a building block to enrich the HyParView[19] membership protocol. The HyParView protocol is a membership protocol that leverage Peer-to-Peer technology to scale and be fault-tolerant. The protocol support efficient message dissemination offering 100% message delivery reliability for levels of failure as high as 80% of crash failures, even though it only requires every node to maintain a partial view of the system membership with the length of $log(n)$ nodes, being n the (maximum) total number of nodes in the system, named Active View. However, in face of a transient network partition this protocol classifies the unresponsive nodes as crash failures which means that both sides of the partition will discard the nodes on the other side in an attempt to find replacements to the faulty nodes and effectively recover the overlay network defined by the elements of the active view maintained by each individual node. Furthermore, after the network partition is recovered, both sides of the partition become unable to reconnect because neither has the information required to do so, in particular no node will retain information for any node from other partition.

The LHView(which stand for **L**ocation Aware **H**ybrid Partial **View**) is a extension of the HyParView protocol that does not suffers from the mentioned problem. It is a membership protocol than integrates an oracle with the purpose of adding location/proximity inference to the operations of the protocol. The oracle assist the membership in its view management operation and adds extra functionality to manage unresponsive nodes from other locations. The new membership view management policies of the LHVIEW are presented below.

3.6.2 Partial View Management Policies

The LHView bias its active view using the information provided by the oracle and partitions the active view. The protocol attempts to maintain a pre-defined number of node identifiers belonging to the same location as itself and a pre-defined number of identifiers of different(known) locations. However, the X-Bot[20] protocol which bias the network to reduce the latencies among neighbors shows that to maintain the robustness of the overlay it requires that their view maintains at least one identifiers with elevated latencies to avoid the partition of the overlay, but in order reach the overlay stability as soon as possible it uses at least two remote identifiers per location, since the priority of join request is defined by the number of neighbors of the target location.

In order to manipulate the active view according to the location bias and its configuration, the following changes were made on the joins priority, insertion, removal, and replacement procedures of the original HyParView protocol. Furthermore, a new periodic status verification procedure is triggered for faulty nodes for which a suitable replacement was not found, as a way to eventually regain global connectivity after the recovery of a network partition.

Joins Priority The joins request on the HyParView had only into account if the node had other neighbors or not, however, since LHView attempts to bias the partial view in function of the location, the priority associated with a join request should capture this. A join should have high priority if the node partial active view has no other identifier of the same location of the join target identifier, otherwise the node sends a join request with low priority.

Replace its executed every time a communication failure occurs and it replaces the faulty node identifier on the active view by one extracted from the passive view. In the original algorithm the selection of the replacement node is performed uniformly at random. LHView adds one more step which consists on filtering the passive for identifiers of the same location as the faulty node, in order to maintain the connectivity between different locations. However in the presence of a network partition all attempts to find a substitute will fail and in that situation the identifiers of that location are moved to a temporary data structure, which will be used periodically to try and recover communications with that location. This avoids the algorithm to lose information about inaccessible locations during network partitions.

Insertion it is similar to the original mechanism employed by the HyParView protocol with an additional step. If a successful insertion on the active view is performed, it will additionally scan the data structure storing the faulty identifiers and move all from the same location of the identifier inserted to the passive view. This happens because the goal of these is to enable the recovery from isolation among different transient network partitions. The successful addition of a node from a different location implies that the goal was (at least partially) achieved.

Removal on the original HyParView protocol, when a node need to remove an identifier from its active view it selects identifiers randomly from its active view, which reduces the network partition robustness, as this can lead a node to lose its only connection to a given location. LHView attempts to have more control over its management by ensuring there is at least one identifier from each different locations on each node partial view at all times. It achieves this degree of manipulation by determining if there is any location that exceeds the limits imposed by the initial configuration and if the case, the node drops a random node of that location. Otherwise it drops a random identifier.

Periodic status verification Additionally, the membership scans the set of identifiers faulty nodes and attempts to reconnect in order to regenerate the overlay after a partition due to a transient network failure.

3.6.3 Discussion

The LHView extends the HyParView functionality while leveraging its scalability, fast-recovery and robustness properties. Our solution strives to not compromise these key properties of the original algorithm. On the other hand, to guarantee strong connectivity it is required to maintain larger active view, $log(n) + (k * d)$ being n the (maximum) total number of nodes in the system, k the number of identifiers from a remote location, and d the number of estimated locations of a deployment, or alternatively it can be a dynamic parameter that captures the number if different location labels observed by a node during its lifetime or a long enough time window.

This solution also requires also that the interval of time between the execution of the periodic activity that attempts to establish connections to the nodes in the faulty nodes set to be tunned adequately. If it is small it will congest the communication channel and cause frequent interruption on the main flow of execution of the system, otherwise it will slow the recovery of the network overlay after a network partition. Alternatively, the interval can have exponential growth per faulty identifier and remove it completely after days of failed attempts to reconnect. Measuring the impact of such technique is however left for future work.

3.7 Implication on message dissemination

Distributed system rely on diverse strategies to disseminate data across nodes, which were previously discussed on Section 2.2.2. Systems that incorporate the protocol proposed in this work to manage the membership can configure the data dissemination strategies to target locations/groups. For instance, distributed storage system can leverage such strategy as a data partitioning scheme and reduce the distance between the data source and its consumer. Distributed cluster resources management systems can extend the characteristics of the resources offers to their clients with location-based offers that would provide lower latencies among those resources. Distributed data processing system are a example that would certainly benefit from this tailored offers.

More importantly, we expect that LHView will have no visible impact on the performance or operation of a gossip-based dissemination scheme configured to propagate data across the whole system. This is useful for systems that require global membership information[8][16][29].

3.8 Summary

This chapter presented an oracle-assisted membership protocol that we named LHView, and its sub-components that contribute to implement a partial-view based membership service that can automatically recover from network partitions. Furthermore, we present four location oracles for different deployment scenarios, each with different trade-offs that must be taken into account in function of the deployment environment characteristics. Finally, we ended the chapter with potential implications that the location inference can have for data dissemination mechanisms and systems that leverage them. In the following chapter we evaluate the proposed membership protocol. We start however, by studying the precision of the classification of each location oracle in different deployment scenarios.

CHAPTER 4

Evaluation

This chapter presents the experimental evaluation of our Location Oracles, where we focus on their precision on assigning location labels according with the real location of nodes (Section 4.2). We then follow with a study on the impact of the use of these Location Oracles in the operation of LHView, the HyParView protocol variant that was introduced in the last chapter that aims at providing self-healing capabilities for network partitions (Section 4.3). Each of these sections begin with the characterization of each experiment and also reports the concrete configuration of the components employed in each experiments.

We start this chapter by discussing our implementations of the Location Oracles, LHView, and our own Java implementation of HyParView, which we made to ensure a fair comparison with LHView (Section 4.3.2).

4.1 Components implementation

All the components presented in this thesis were implemented in Java employing the library New-IO (NIO), which exposes network primitives to handle network operation in a non-blocking way. The Static Label Oracle version that obtains location labels from the Amazon cloud infrastructure required the AWS SDK core.

4.2 Oracle's classification accuracy

In this section we report on our experiment whose goal was to evaluate the capability of each oracle to classify nodes correctly regarding their location on a deployment. We consider a classification to be correct if it matches the node's datacenter label. All oracles results are presented and discussed on the following subsections, except for the Static Label Oracle because its correctness is ensured by the deployment or environment.

The experiments were conducted on the Amazon Cloud Platform using m4-xlarge virtual machine instances using 30 and 300 nodes. In all experiments we ensured that the number of nodes were evenly distributed across datacenters. The experimental scenarios that we explored are summarized in Table 4.1:

Number of datacenters	Europe	USA	Asia	Label
5	2	2	1	5DC
3	1	1	1	3DC-Distant
3	0	3	0	3DC-Near

Table 4.1: Deployment configuration for each scenario(rows).

To conduct these experiments we used the following datacenters in each region. For Europe we used the datacenters in Frankfurt and Ireland. For USA we used the datacenters in California, Virginia, and Oregon, and for Asia we used the datacenter located in Tokyo.

4.2.1 IP Address Clustering with Latency Oracle

We start by reminding the reader that the IP Address Clustering and Latency Oracle depends on two distinct aspects to attribute a location label to a node, the latency measured from the node assigning the label to the node being classified and the longest common IP prefix length(lcpl) between nodes. To configure the oracle appropriately we start by conducting a set of measurements to assert these values for each of our 3 experimental deployment setups.

Table 4.2 shows the results of executing pings between all nodes deployed in datacenters that belong to the three mentioned deployment scenarios. As shown, the round trip time between datacenters allow one to define rules than partition each datacenter except when those datacenters are near each other, for instance, nodes from Tokyo can not clearly distinct nodes of California from nodes of Oregon. Another situation that may induce to misclassification is when a node has the same average latency to two distant datacenters, for example, nodes from Virginia can not distinguish nodes of California from nodes of Ireland.

Complementary to the latency rules, the IP Address Clustering and Latency Oracle, defines rules over the IP prefixes, however as shown in Table 4.1 the average lcpl values among nodes of the same are extremely low and some nodes can not distinguish the other datacenters, e.g, Frankfurt can not distinguish nodes of Tokyo from those of California in scenario 3DC-Distant. The apply for Oregon and Virginia on scenario 3DC-Near. Furthermore, there is a similar case for every datacenter on scenario 5DC.

Overall, even though it is possible to create rules over the latency and lcpl values that partition the nodes per datacenter, these rules will easily generate false positives and false negatives unless there was a significant interval among them. Additionally, one would expect that the LCPL values among nodes of the same datacenter would be more similar, however this is not the case for the Amazon cloud infrastructure. This implies that this oracle is not applicable for this scenarios, as the assumptions node omits design regarding the distinguishing factors of the different locations do not hold.

Datacenter	**Ireland**	**Frankfurt**	**California**	**Virginia**	**Oregon**	**Tokyo**
Ireland	0.667	22.303	148.243	72.087	133.079	206.791
Frankfurt	22.309	0.579	159.216	91.110	171.966	234.148
California	148.602	159.143	0.530	74.873	21.565	109.745
Virginia	72.124	91.130	74.586	0.645	81.573	151.612
Oregon	133.024	171.968	21.427	81.517	0.734	95.059
Tokyo	206.968	234.129	109.429	151.770	95.060	0.621

Table 4.2: Average round trip time between all datacenters.

Datacenter	**Tokyo**	**Frankfurt**	**California**
Tokyo	7	3	6
Frankfurt	3	17	3
California	6	3	10

(a) 3DC-Distant.

Datacenter	**Virginia**	**California**	**Oregon**
Virginia	6	7	6
California	7	10	6
Oregon	6	6	9

(b) 3DC-Near.

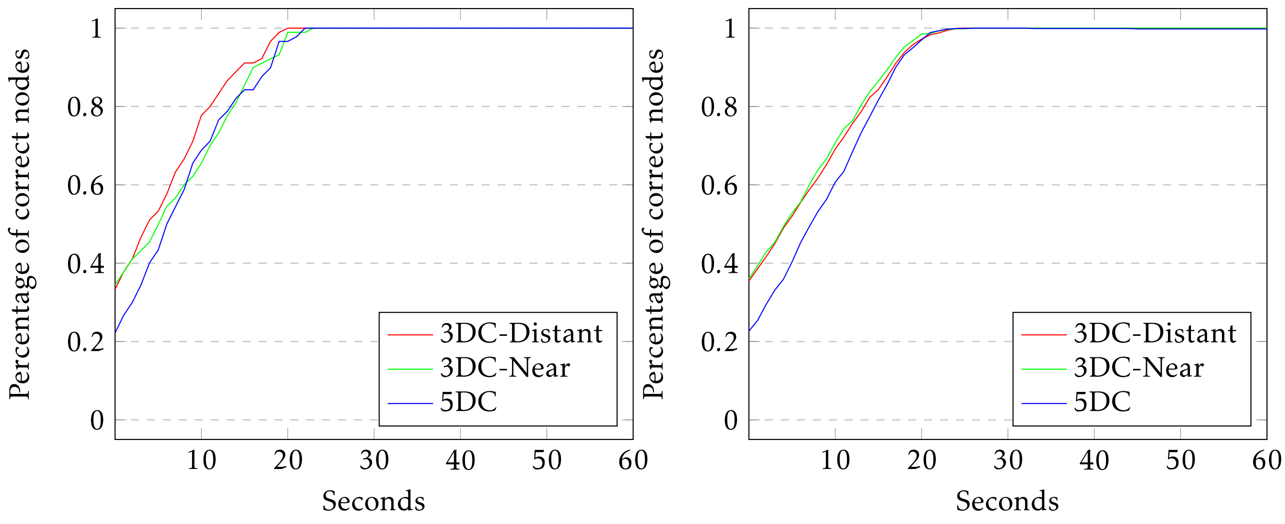
Datacenter	**Tokyo**	**Frankfurt**	**Ireland**	**Virginia**	**California**
Tokyo	7	3	3	5	6
Frankfurt	3	16	7	3	3
Ireland	3	7	15	3	3
Virginia	5	3	3	6	6
California	7	3	3	6	9

(c) 5DC.

Figure 4.1: Average longest common IP prefix size per scenario.

4.2.2 Anchor Oracle

In this experiment the anchor servers required by the Anchor Oracle were deployed across the different datacenters, one for each location. The Figure 4.2 shows the percentage of nodes that correctly classified itself until 60 seconds after the system deployment. On all the scenarios, the oracle dynamic classification nature is impacted by the latency variation as can be observed on the firsts 20 seconds where the accuracy is low. Overall, the oracle performs well for every deployment scenario, achieving 100% accuracy after the firsts 20 seconds.



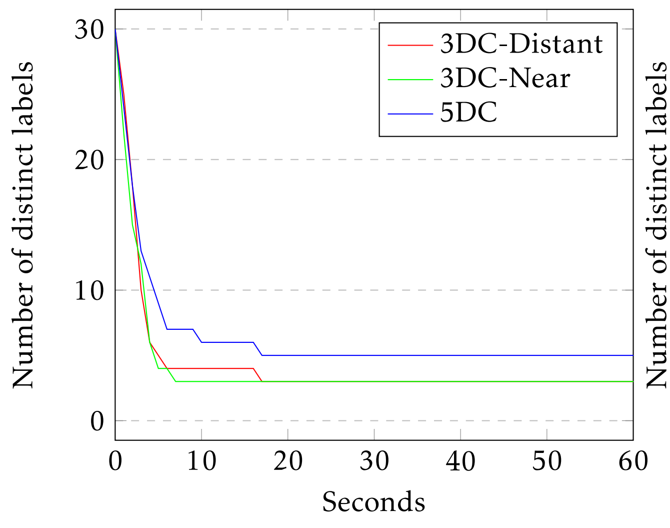
(a) 30 nodes.

(b) 300 nodes.

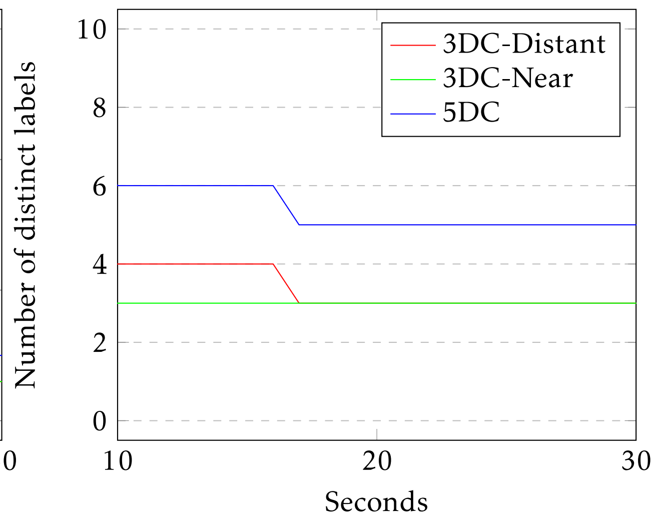
Figure 4.2: Percentage of correct classifications in each tested scenario

4.2.3 Dynamic Label Oracle

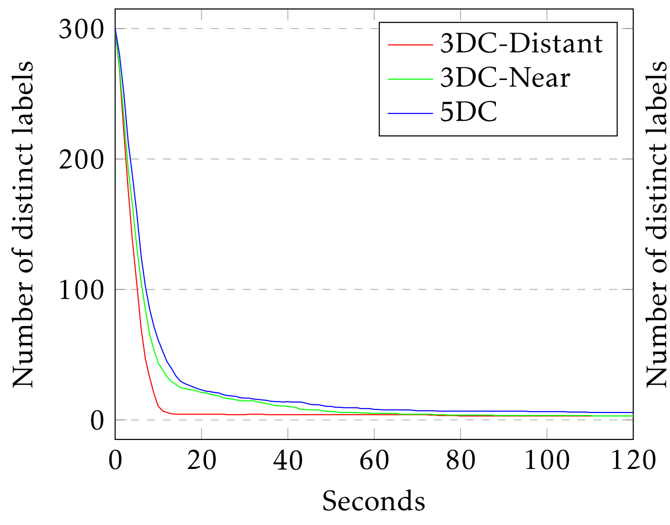
In this experiment we tested the convergence characteristics of the Dynamic Label oracle by determining the number of distinct labels at each second since the system deployment, as shown in Figure 4.3. The required latency threshold was set to 10 milliseconds, which by the values observed on Figure 4.2 is ideal to partition local nodes from remote nodes. As expected, at the beginning the number of labels correspond to the number of nodes but as Figure 4.3 shows, that number rapidly decreases to the number of locations per scenario. However, in some occasions in the 5DC scenario using 300 nodes was observed that some datacenters converged to two labels instead of one, as can be observed in the Figure 4.3 at 120 seconds. Again, this happens due to the concurrent joins during the system initialization but the convergence on the labels is guaranteed since every node periodically pings its neighbors and shuffles information with random nodes.



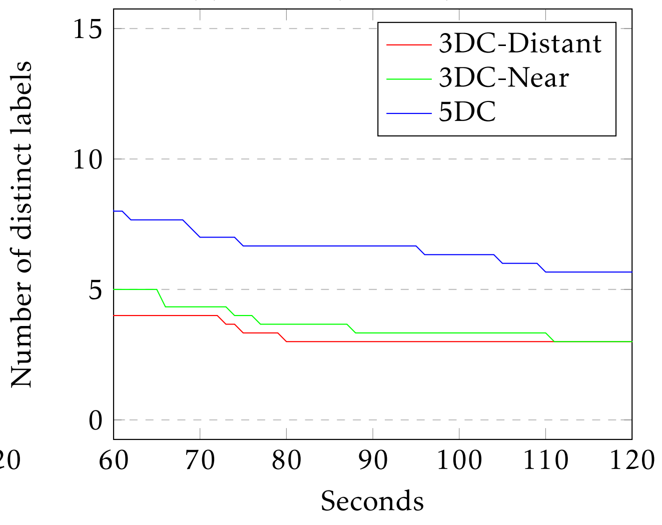
(a) 30 nodes.



(b) 30 nodes(10s - 30s).



(c) 300 nodes.



(d) 300 nodes(60s - 120s).

Figure 4.3: Number of distinct labels over time per deployment scenario.

4.3 LHView's evaluation

In this section, we evaluate the fault-tolerance of our HyParView and LHView implementation. All experiments on this section were deployed on 15 m4-xlarge AWS instances on which 300 nodes were evenly distributed across 3 datacenters (Ireland, Frankfurt, California). Each membership protocol parameterization is presented on Table 4.3.

First, we compare both implementations in relation to their behaviour under difference levels of simultaneous crash-failures and then we study the effects of membership in the reliability of a gossip-based dissemination protocol. Later we compare again both implementation in a scenario where a transient network partition occurs by studying the effects on the reliability of message dissemination using each implementation before, during, and after the network failure.

Component	**Parameter**	**Value**	
HyParView	Active View(AV) Size	5	
	Passive View(PV) Size	30	
	Active Random Walk Length	6	
	Passive Random Walk Length	3	
	Number of Shuffle elements from	AV	3
		PV	4
LHView	Active View(AV) Size	7	
	Passive View(PV) Size	30	
	Local View Size	3	
	Remote View Size	2	
	Active Random Walk Length	6	
	Passive Random Walk Length	3	
	Number of Shuffle elements from	AV	3
		PV	4
Dissemination	Fanout	7	

Table 4.3: Memberships and dissemination parameters.

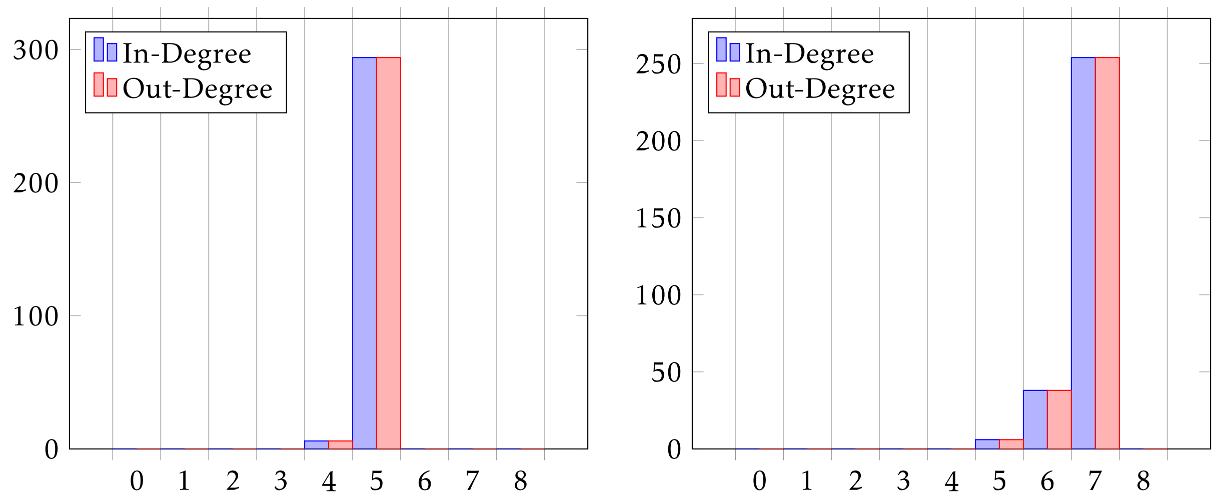
4.3.1 In-Degree and Out-Degree

We start by evaluating the impact on node degree (we measure node out-degree and in-degree to check the correctness of the solution). We remind the reader that in a well balanced and fault-tolerant system, node degree should be similar across the large majority of all nodes. Results are reported in Figure 4.4 for the original HyParView protocol and for our own variant of the protocol using the LHView with Static Label Oracle. The results show that HyParView's essential active view symmetry property holds for our implementation and for LHView because the in and out degree of every node matches. Furthermore, we observe that most HyParView's nodes have their active view full as shown in the original paper, which shows that its overlay is balanced and the same holds for LHView, hence demonstrating that it doesn't break these property of the HyParView.

4.3.2 Crash failure recovery

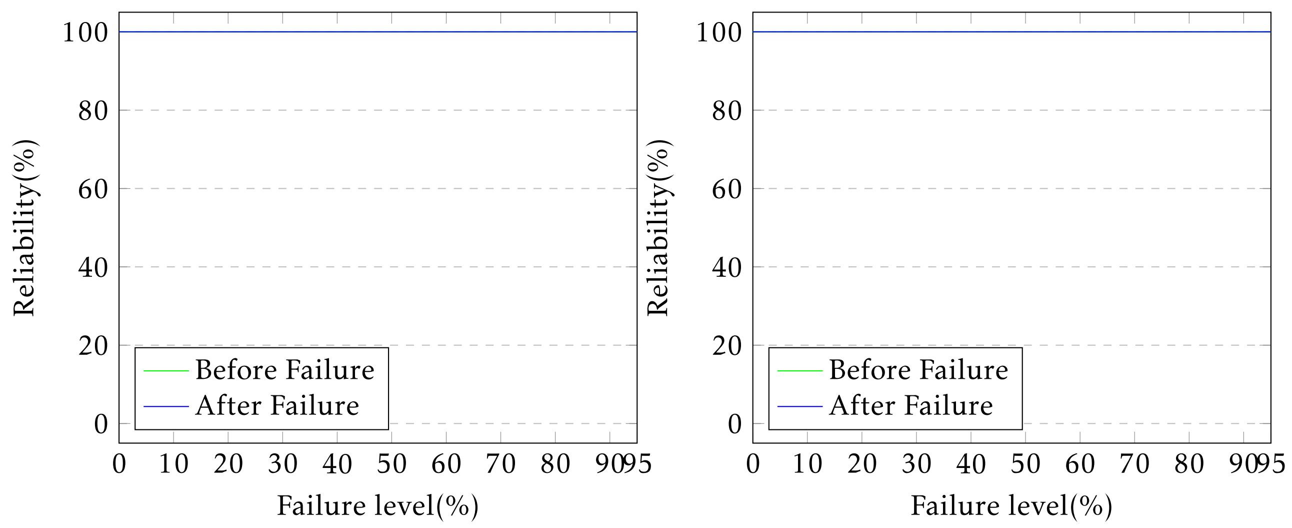
On this experiment we compare the fault-tolerance of our Java implementation of the HyParView protocol, that served as a base for the extended implementation of LHView against the simulation results originally presented in [19]. The experiment consisted on two phases: The creation and stabilization of the network overlay and the injection of crash-failures on nodes.

The results presented on the figure 4.5 show the reliability before and after the failure induction for HyParView and LHView with SLO. As expected the HyParView implementation shows high delivery reliability in face of crash failures, 100% delivery reliability for all levels of failure tested. The LHView reaches the same levels as was expected, since it guarantees the same properties as its predecessor, therefore it is as robust in face of crash failure scenarios as HyParView.



(a) HyParView. (b) LHView with Static Label Oracle.

Figure 4.4: Node in and out degree of HyParView and LHView.



(a) HyParView. (b) LHView.

Figure 4.5: Reliability of HyParView and LHView under crash failure scenario.

4.3.3 Network partition recovery

The test of the extended membership combined with the Location Oracles mentioned on the previous consisted of three phases: The creation and stabilization of the network overlay; Injection of a network partition between each pair of datacenters; Reparation of the network partition after 60 seconds.

For this experiment, LHView used the configurations from the previous test experiment. To simulate a network partition during phases 2 of the experiment, we resorted to the IP Tables configuration of each node, such that nodes became partitioned by datacenter.

The Location Oracles evaluated were the Static Label Oracle(SLO), Anchor Oracle(AO), and Dynamic Label Oracle, which demonstrated the best classification accuracy. Figure 4.6 reports the effect of the partition in the reliability of the dissemination mechanism in different moments of the experiment. On phase 1, that correspond to the creation and stabilization period, every version of the protocol show high levels of delivery reliability, 100%. On phase 2 all protocol version were under a network partition (every datacenter become unable to communicate with any of the other datacenters), and was expected no message reach more than 33% of the nodes.

The last phase, shows that every version was capable of a complete recovery after the network partition healed except the original HyParView protocol. This was expected due to HyParView lacking any location aware procedure that allows it to detect partitions.

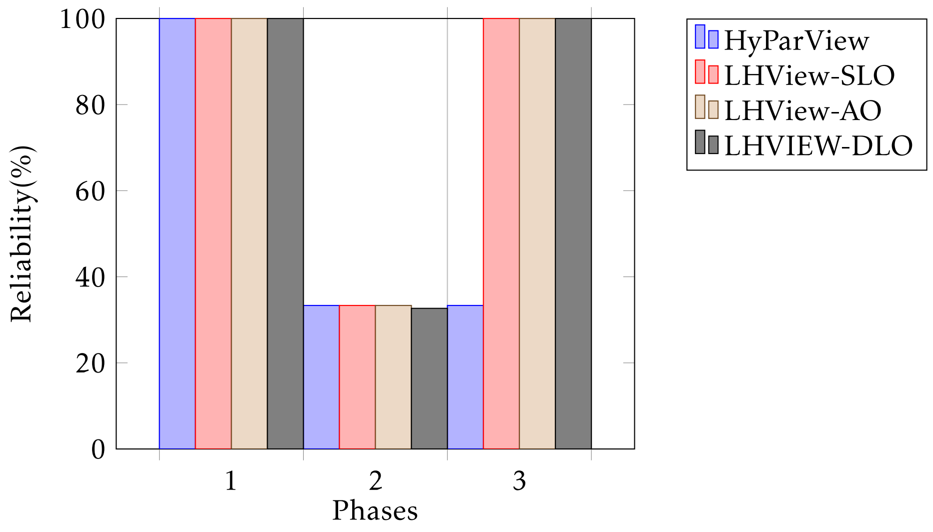


Figure 4.6: Reliability of the LHView in face of a network partition.

4.4 Summary

In this chapter we evaluated the components proposed in the thesis in different deployment scenarios. First we studied the classification capabilities of each Location Oracle that has no knowledge about their locations at deployment time. Our experiments showed how the IP address Clustering and Latency Oracle is unreliable for Amazon Cloud infrastructure, which can be generalized to any scenario where the assumption regarding latency and IP distribution related with locations do not hold. In contrast, the Anchor Oracle and Dynamic Label were promising due to their precision, specially the last, since it does not require any external service which reduces the operation management cost.

Before evaluating the LHView in face of a network partition, we studied its node degree balance, which showed that its partial active view is fully symmetric and that the degree is (slightly) not as balance across nodes of the system, as in the case of its predecessor HyParView. Latter, we tested the perfect oracle, Static Label Oracle for different levels of crash-failure, which allowed us to observe the resiliency of LHView protocol and verify that it holds the properties of its predecessor. Finally, we tested under the main challenge of this thesis, where it performed well. The LHView protocol equipped with the Static Label Oracle, the Anchor Oracle, or the Dynamic Label Oracle, can recover successfully from network partition to the same reliability levels for the dissemination mechanism than before the network partition.

Overall, the LHView guarantees all the overlay properties that the HyParView, such as the global connectivity and robustness, and when equipped with the proposed oracles, it can tolerate transient network failures that the previous partial view based membership protocols could not.

Chapter 5

Conclusion

Many distributed system that support large-scale services have to efficiently manage their resources and disseminate crucial information in order to operate globally. These system rely on membership services that create overlay networks with important properties, such as global connectivity, robustness, scalability, and reliability. The membership service is a sub-component of these systems that provides information about the activity of nodes by tracking their health status, detect failures on nodes or on communication links, and upon failure detection on such components employ recovery mechanism to correct the operation of distributed systems. However, many membership protocols can't tolerate a network partition, which temporarily separates nodes into disconnected components that cannot communicate neither can recover the global connectivity of the system after the recovery of the network failure.

In this thesis we proposed a new membership protocol that extends the hybrid partial view membership protocol, HyParView, while its maintaining its properties. The HyParView is the most fault-tolerant protocol in the class of fully decentralized membershp service based on partial views, however to tolerate network partitions it would need to distinguish simultaneous node failures from network partitions. Our protocol targets this challenge by equipping the HyParView with an oracle that collects and process metadata to offer assistance to the membership services. In this work we propose four different oracle implementation that infer the location of nodes, named Location Oracles, and adapted the HyParView protocol to leverage these sub-components. Thus, this combination resulted on the Location Aware Hybrid Partial View(LHView).

Location Oracles were evaluated in diverse deployment scenarios to study their accuracy and to understand which scenario would benefit or were adequate/incompatible for each one. Furthermore, using the best oracles we analysed how the integrated solution, LHView, would offer better robustness, recovery and enable message delivery reliability properties for gossip-based dissemination protocols operating on top of the membership service. Our observations shown that the Static Label Oracle, Dynamic Label Oracle, and the Anchor Oracle allowed the LHView to recover from the network partition and provide high message reliability after a network partition occurrence which is a significant gain in relation to the original HyParView protocol without compromising its predecessor overlay properties.

In summary the main contribution are:

1. A network partition tolerant membership protocol.
2. Four different implementations of Location Oracles that through the processing of message's metadata infer a node location.
3. A membership service that can be leverage to design gossip-based message dissemination mechanism that target specific locations in a deployment.

5.1 Future Work

We now discuss some potential directions for the future work. Our proposal would benefit from partial view management policies that would maintain HyParView global connectivity. Furthermore, it could be optimized to stabilize faster by adapting neighbour requests to be location aware. Additionally, a code migration of the current implementation to a language without garbage-collection, in order to avoid uncontrollable behaviour even though these are not malevolent can also improve the protocol operation in practice.

Since the IP Address Clustering and Latency Oracle classification accuracy assumptions don't hold for the Amazon cloud infrastructure, it would be interesting to expand the study to other cloud providers and to privately owned infrastructures. Furthermore, communication between datacenters is costly, thus a solution that targets decrease of such values could benefit the solution in a astonish way. The technique proposed on X-Bot[20] can be a good start point for this endeavour.

Finally, membership are designed to assist other distributed system, hence, studying how to integrate the LHView on an existing system such as Cassandra can be beneficial to the distributed systems community.

Bibliography

[1] S. Almeida, J. a. Leitão, and L. Rodrigues. "ChainReaction: A Causal+ Consistent Datastore Based on Chain Replication". In: *Proceedings of the 8th ACM European Conference on Computer Systems*. EuroSys '13. Prague, Czech Republic: ACM, 2013, pp. 85–98. ISBN: 978-1-4503-1994-2. DOI: 10.1145/2465351.2465361. URL: http://doi.acm.org/10.1145/2465351.2465361.

[2] *Apache Hadoop*. https://hadoop.apache.org. Accessed: 2016-06-05.

[3] P. Bailis and K. Kingsbury. "The Network is Reliable". In: *Queue* 12.7 (July 2014), 20:20–20:32. ISSN: 1542-7730. DOI: 10.1145/2639988.2639988. URL: http://doi.acm.org/10.1145/2639988.2639988.

[4] J. K. Ben Y. Zhao and A. D. Joseph. *Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing*. Tech. rep. UCB//CSD-01-1141. U. C. Berkeley, 2001.

[5] M. Burrows. "The Chubby Lock Service for Loosely-coupled Distributed Systems". In: *Proceedings of the 7th Symposium on Operating Systems Design and Implementation*. OSDI '06. Seattle, Washington: USENIX Association, 2006, pp. 335–350. ISBN: 1-931971-47-1. URL: http://dl.acm.org/citation.cfm?id=1298455.1298487.

[6] J. C. Corbett et al. "Spanner: Google&Rsquo;s Globally Distributed Database". In: *ACM Trans. Comput. Syst.* 31.3 (Aug. 2013), 8:1–8:22. ISSN: 0734-2071. DOI: 10.1145/2491245. URL: http://doi.acm.org/10.1145/2491245.

[7] J. Dean and S. Ghemawat. "MapReduce: Simplified Data Processing on Large Clusters". In: *Commun. ACM* 51.1 (Jan. 2008), pp. 107–113. ISSN: 0001-0782. DOI: 10.1145/1327452.1327492. URL: http://doi.acm.org/10.1145/1327452.1327492.

[8] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. "Dynamo: Amazon's Highly Available Key-value Store". In: *SIGOPS Oper. Syst. Rev.* 41.6 (Oct. 2007), pp. 205–220. ISSN: 0163-5980. DOI: 10.1145/1323293.1294281. URL: http://doi.acm.org/10.1145/1323293.1294281.

[9] M. J. Fischer, N. A. Lynch, and M. S. Paterson. "Impossibility of Distributed Consensus with One Faulty Process". In: *J. ACM* 32.2 (Apr. 1985), pp. 374–382. ISSN: 0004-5411. DOI: 10.1145/3149.214121. URL: http://doi.acm.org/10.1145/3149.214121.

[10] A. J. Ganesh, A.-M. Kermarrec, and L. Massoulié. "Scamp: Peer-to-Peer Lightweight Membership Service for Large-Scale Group Communication". In: *Networked Group Communication: Third International COST264 Workshop, NGC 2001 London, UK, November 7–9, 2001 Proceedings*. Ed. by J. Crowcroft and M. Hofmann. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 44–55. ISBN: 978-3-540-45546-2. DOI: 10.1007/3-540-45546-9_4. URL: http://dx.doi.org/10.1007/3-540-45546-9_4.

[11] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. H. Katz, S. Shenker, and I. Stoica. "Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center." In: *NSDI*. Vol. 11. 2011. 2011, pp. 22–22.

[12] *How ZooKeeper Handles Failure Scenarios*. https://wiki.apache.org/hadoop/ZooKeeper/FailureScenarios. Accessed: 2017-03-19.

[13] P. Hunt, M. Konar, F. P. Junqueira, and B. Reed. "ZooKeeper: Wait-free Coordination for Internet-scale Systems." In: *USENIX annual technical conference*. Vol. 8. 2010, p. 9.

[14] M. Jelasity and O. Babaoglu. "T-Man: Gossip-Based Overlay Topology Management". In: *Engineering Self-Organising Systems: Third International Workshop, ESOA 2005, Utrecht, The Netherlands, July 25, 2005, Revised Selected Papers*. Ed. by S. A. Brueckner, G. Di Marzo Serugendo, D. Hales, and F. Zambonelli. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 1–15. ISBN: 978-3-540-33352-4. DOI: 10.1007/11734697_1. URL: http://dx.doi.org/10.1007/11734697_1.

[15] P. Karwaczyński, D. Konieczny, J. Moçnik, and M. Novak. "Dual Proximity Neighbour Selection Method for Peer-to-peer-based Discovery Service". In: *Proceedings of the 2007 ACM Symposium on Applied Computing*. SAC '07. Seoul, Korea: ACM, 2007, pp. 590–591. ISBN: 1-59593-480-4. DOI: 10.1145/1244002.1244137. URL: http://doi.acm.org/10.1145/1244002.1244137.

[16] A. Lakshman and P. Malik. "Cassandra: A Decentralized Structured Storage System". In: *SIGOPS Oper. Syst. Rev.* 44.2 (Apr. 2010), pp. 35–40. ISSN: 0163-5980. DOI: 10.1145/1773912.1773922. URL: http://doi.acm.org/10.1145/1773912.1773922.

[17] L. Lamport et al. "Paxos made simple". In: *ACM Sigact News* 32.4 (2001), pp. 18–25.

[18] J. Leitao, J. Pereira, and L. Rodrigues. "Epidemic Broadcast Trees". In: *Reliable Distributed Systems, 2007. SRDS 2007. 26th IEEE International Symposium on*. 2007, pp. 301–310. DOI: 10.1109/SRDS.2007.27.

[19] J. Leitao, J. Pereira, and L. Rodrigues. "HyParView: A Membership Protocol for Reliable Gossip-Based Broadcast". In: *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07)*. 2007, pp. 419–429. DOI: 10.1109/DSN.2007.56.

[20] J. C. A. Leitao, J. P. d. S. F. M. Marques, J. O. R. N. Pereira, and L. E. T. Rodrigues. "X-BOT: A Protocol for Resilient Optimization of Unstructured Overlays". In: *Reliable Distributed Systems, 2009. SRDS '09. 28th IEEE International Symposium on*. 2009, pp. 236–245. DOI: 10.1109/SRDS.2009.20.

[21] J. Leitao. "Topology Management for Unstructured Overlay Networks". In: *Technical University of Lisbon* (2012).

[22] J. B. Leners, H. Wu, W.-L. Hung, M. K. Aguilera, and M. Walfish. "Detecting Failures in Distributed Systems with the Falcon Spy Network". In: *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*. SOSP '11. Cascais, Portugal: ACM, 2011, pp. 279–294. ISBN: 978-1-4503-0977-6. DOI: 10.1145/2043556.2043583. URL: http://doi.acm.org/10.1145/2043556.2043583.

[23] W. Lloyd, M. J. Freedman, M. Kaminsky, and D. G. Andersen. "Don'T Settle for Eventual: Scalable Causal Consistency for Wide-area Storage with COPS". In: *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*. SOSP '11. Cascais, Portugal: ACM, 2011, pp. 401–416. ISBN: 978-1-4503-0977-6. DOI: 10.1145/2043556.2043593. URL: http://doi.acm.org/10.1145/2043556.2043593.

[24] R. Melamed and I. Keidar. "Araneola: a scalable reliable multicast system for dynamic environments". In: *Network Computing and Applications, 2004. (NCA 2004). Proceedings. Third IEEE International Symposium on*. 2004, pp. 5–14. DOI: 10.1109/NCA.2004.1347755.

[25] *Mesos High-Availability Mode*. http://mesos.apache.org/documentation/latest/high-availability/. Accessed: 2017-03-19.

[26] *Postmortem of database outage of January 31*. https://about.gitlab.com/2017/02/10/postmortem-of-database-outage-of-january-31/. Accessed: 2017-03-16.

[27] R. van Renesse, Y. Minsky, and M. Hayden. "A Gossip-Style Failure Detection Service". In: *Middleware'98: IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing*. Ed. by N. Davies, S. Jochen, and K. Raymond. London: Springer London, 1998, pp. 55–70. ISBN: 978-1-4471-1283-9. DOI: 10.1007/978-1-4471-1283-9_4. URL: http://dx.doi.org/10.1007/978-1-4471-1283-9_4.

[28] R. van Renesse, D. Dumitriu, V. Gough, and C. Thomas. "Efficient Reconciliation and Flow Control for Anti-entropy Protocols". In: *Proceedings of the 2Nd Workshop on Large-Scale Distributed Systems and Middleware*. LADIS '08. Yorktown Heights, New York, USA: ACM, 2008, 6:1–6:7. ISBN: 978-1-60558-296-2. DOI: 10.1145/1529974.1529983. URL: http://doi.acm.org/10.1145/1529974.1529983.

[29] *Riak compared with Dynamo*. https://docs.basho.com/riak/1.1.0/references/dynamo/. Accessed: 2016-05-29.

[30] A. Rowstron and P. Druschel. "Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems". In: *Middleware 2001: IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg, Germany, November 12–16, 2001 Proceedings*. Ed. by R. Guerraoui. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 329–350. ISBN: 978-3-540-45518-9. DOI: 10.1007/3-540-45518-3_18. URL: http://dx.doi.org/10.1007/3-540-45518-3_18.

[31] E. Schurman and J. Brutlag. "Performance related changes and their user impact". In: *velocity web performance and operations conference*. 2009.

[32] M. Shapiro, N. Preguiça, C. Baquero, and M. Zawirski. "Conflict-free replicated data types". In: *Symposium on Self-Stabilizing Systems*. Springer. 2011, pp. 386–400.

[33] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. "Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications". In: *SIGCOMM Comput. Commun. Rev.* 31.4 (Aug. 2001), pp. 149–160. ISSN: 0146-4833. DOI: 10.1145/964723.383071. URL: http://doi.acm.org/10.1145/964723.383071.

[34] *Summary of the Amazon S3 Service Disruption in the Northern Virginia (US-EAST-1) Region*. https://aws.amazon.com/message/41926/. Accessed: 2017-03-18.

[35] C. Tang, R. N. Chang, and C. Ward. "GoCast: gossip-enhanced overlay multicast for fast and dependable group communication". In: *2005 International Conference on Dependable Systems and Networks (DSN'05)*. 2005, pp. 140–149. DOI: 10.1109/DSN.2005.52.

[36] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, B. Saha, C. Curino, O. O'Malley, S. Radia, B. Reed, and E. Baldeschwieler. "Apache Hadoop YARN: Yet Another Resource Negotiator". In: *Proceedings of the 4th Annual Symposium on Cloud Computing*. SOCC '13. Santa Clara, California: ACM, 2013, 5:1–5:16. ISBN: 978-1-4503-2428-1. DOI: 10.1145/2523616.2523633. URL: http://doi.acm.org/10.1145/2523616.2523633.

[37] S. Voulgaris, D. Gavidia, and M. van Steen. "CYCLON: Inexpensive Membership Management for Unstructured P2P Overlays". In: *Journal of Network and Systems Management* 13.2 (2005), pp. 197–217. ISSN: 1573-7705. DOI: 10.1007/s10922-005-4441-x. URL: http://dx.doi.org/10.1007/s10922-005-4441-x.

[38] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica. “Resilient Distributed Datasets: A Fault-tolerant Abstraction for In-memory Cluster Computing”. In: *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*. NSDI’12. San Jose, CA: USENIX Association, 2012, pp. 2–2. URL: http://dl.acm.org/citation.cfm?id=2228298.2228301.