Flávio Duarte P. Fernandes

HyPar-Cloud: Partially-Structured Overlay Network for the Cloud

Relatório intermédio para obtenção do Grau de Mestre em

Engenharia Informática

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

June, 2016

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. 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 recovery of the partition.

This work aims to design a datacenter-aware protocol to tolerate network partitions by applying existing approaches and classification techniques that may allow the system to in an efficient way to cope with network partitions without compromising the remaining properties of the overlay networks. Furthermore, we aim at achieving 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. Esse facto obriga as aplicações que suportam a operação destes serviços a escalarem e a minimizarem os custos de manutenção, tanto para evitar a utilização desnecessária de recursos ou como 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, que a maioria realiza através de protocolos de filiação. Estes protocolos funcionam ao nível da aplicação e expõem topologias de redes sobrepostas 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 falhas em componentes independentes, quando utilizados à escala global a vantagem torna-se numa desvantagem. Os nós, a esse nível são incapazes de distinguir entre falhas simultâneas individuais e partições (temporárias) na rede. A ocorrência de partições de rede força o sistema a subdividir-se em dois conjuntos isolados e incapazes de comunicar entre si, mesmo após a reparação da partição.

Este trabalho tem como objetivo conceber um protocolo que infira a localização dos nós de modo a tolerar partições de rede. A inferência é efetuada através de técnicas existentes de classificação que permitem 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

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Context

In recent years, as the World progresses into a more highly connected place, and many global level business opportunities have appeared. These businesses intend to reach as many customers as possible with their services that 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 the Cloud settings must be prepared to handle all kind of network anomalies 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 nodes that allows a high control over data partitioning and load balancing while providing fault-tolerance by tracking and addressing nodes' faults. These membership services many times resort to Peer-to-Peer 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 protocols, designed to provide high reliability for communications and dissemination of information.

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 and multiple propagation paths that emerge from this process.

Protocols such as HyParView[11] 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% node failures, leveraging this protocol it was shown that one could build highly efficient and robust message dissemination mechanisms. 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 action in repairing the overlay.

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 2 network partitions a protocol similar to HyParView will reconfigure itself extremely fast by among individual faults from all remote nodes. This makes the protocol lose its global connectivity and consequently, its reliability, when the network partition holds. This scenario originates from the occurrence of the inevitable network partitions that often affects global-scale applications. Therefore there is a necessity for a membership protocol that takes geo-distributed systems to the next level while providing the fundamental guarantees from previous systems.

1.2 Motivating Problem 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 affect considerably their performance. Moreover if these systems require manual management even for small tasks such as permanently decommissioning a node. Furthermore the error-prone tendencies of humans may comprise the system even more. Additionally, the time necessary for a human to solve a problem combined with its working schedule, and cost 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. A solution that scales requires great amount of resources to monitoring and that does not exclude faulty nodes which affect the performance and correction of the monitoring mechanism. Other solutions track just a partial state and can

efficiently handle dynamic environments by being capable of fast self-recovery, however this kind of solutions are unable of distinguish multiple failures from network partitions. 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 assumes the remaining to be permanently faulty.

The proposed solution attempts to integrate into exiting gossip-based membership protocol location inference techniques. These techniques will exploit the API provided by the Cloud virtualization services or when these are not available, the latency between nodes to classify proximity and the possibility of using the DNS to build a virtual coordination system.

1.3 Expected Contributions

The main expected contributions are:

- Study, develop, and evaluate 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.
- Integration of the presented solutions to design a new membership service for Apache Cassandra, an open-source distribute key-value store.

1.4 Document structure

The remainder of this document is organized as follows:

Chapter 2 presents the related work. 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.

Chapter 3 presents the proposed solution and the work plan by enumerating each phase and the corresponding deadline.

Chapter 2

Related Word

This thesis will focus on exploring a novel design and implementation for a membership protocol that is tolerant to network partitions. Furthermore, we will also study how to integrate such a membership service into distributed storage systems. This chapter discusses fundamental concepts and related technologies, 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 fundamental membership and communication services of the large-scale distributed 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, 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 accumulative computational power and storage space results on a system that has a performance (for parallel programs) that can surpass bleeding edge computers while being cheaper that such solutions. Through cooperation, nodes can attack problems by dividing the workload evenly among them and when facing large-scale problems 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 the communication model, delays while exchanging messages implies delays on the computations and readiness of the data required by individual components.

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, the alternative is to resort to the centralized model, through the use of a coordinator node, that is a single point of failure. Furthermore, the coordination may require up-to-date global information about the system which is not trivial to gather in a large-scale distributed system.

Resources Management: Control over the entities in the system that can manipulate resources.

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

Manual management: As the number of nodes increase, the management of the cluster will become increasingly 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 and can be branched 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. In these systems, the problem is divided in smaller tasks which will be distributed among the nodes by a coordinator which plays a crucial role, because there might be a high level of dependencies

between intermediate computations. Furthermore, intermediate and final computation results are stored on a distributed file system to guarantee the data's availability. Examples of implementations of this systems include Google MapReduce[4], Apache Hadoop[1] and Apache Spark[23].

Distributed Storage: Used to store great volumes of data, design to scale as the workload increases by sacrificing strong consistency or availability. Unlike traditional storage system, the query model is simpler exposing an interface that is similar to a key-value store. An example is the storage system Dynamo[5] that uses data partitioning scheme combined with replication techniques in order to provide availability and efficiency. Spanner[3] goes even further by relying on centralized management schemes per geographical region to provide strong consistency on client access to the stored data objects.

2.1.1 Essential Services

As the scale of a system increases the complexity of managing distributed system also increases. To minimize that 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 is the consensus algorithm which is often used 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.

Monitoring and Control: Provides instrumentation that offer administrators ways to diagnosis the state of a distributed system at a local and global level, which allows to access 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.

2.1.2 Membership

Distributed system to efficiently execute computations 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 this can be materialized by a centralized component, many large-scale systems resort to membership services that rely on peer-to-peer

technology to share the nodes' state and membership events through epidemic/gossip protocols. See section 2.2 for more details.

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[15] and recovery mechanisms. The membership component is typically used as a failure detector[11] by classifying nodes' unresponsiveness as failures when performing periodic tasks and through the execute of a recovery procedures to exclude the faulty nodes and move their tasks to other connected nodes, when required.

FALCON[13] uses a different approach, it relies on specialized complementary components that monitor and report the status of every software layer. The FALCON approach is limited to one datacenter and it adds spy modules at least on four layers: Application, Operative System, Virtual Machine, and Network Switch. The spy modules can communicate with each other and the component layer that they monitor. The components are arranged in a chain to maximize failure detection cover and avoid disruption such that the lowest layer L component can query the $L+1$ spy module, which allow a fine-grained control over the component's restart. 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, it is a platform-specific solution.

2.1.4 Distributed Storage Systems

Distributed storage systems are designed to scale as the workload increases by sacrificing either data consistency or availability. Unlike traditional storage system such as relation 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 the appropriated node. A method that increases the overall availability of the system while distributed 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 required.

Bellow we discuss in more detail some relevant examples of distributed storage systems.

Dynamo[5] uses a ring 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 is expected to store 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 and 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 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[9] is a storage system inspired on Dynamo, however Cassandra uses a hash function that has into consideration the load of the cluster nodes. 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 balance. The membership is based on Scuttlebut[16], 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 removals of nodes are explicit, thus no re-balance of partitions or reparation of unreachable nodes is done.

Cassandra handle failures by the same guidelines defined by Dynamo but it uses additionally a failure detection mechanism that outputs a nodes's suspicious of failure degree and upon failure detection Cassandra resorts to a hinted handoff strategy. On the hinted handoff strategy requests are forwarded to a backup node that will periodically check if the faulty node recovered. When a node recovers, all messages stored by the backup node are sent to it.

Riak [17] is another storage system inspired on 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. The failure detection and handling follows the same guidelines that Dynamo combined with hinted handoff strategy used on Cassandra. For message dissemination, Riak uses a

latency-optimized version of the PlumTree protocol[10], (which details can be found at section 2.2.3.5).

2.1.5 Summary

Many businesses that try to reach a maximum number of customers have to employ geo-distributed services across the planet, a scale that brings management and maintenance problems. Those services must handle by design component failures, network anomalies, and provide high level of automation without increasing organizations' spending 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 throughout the system. To support the membership and dissemination mechanism the system, one can employ very efficient and scalable gossip protocols that provide fault-tolerance and reliability but that still lack the automation necessary for large-scale dynamic environments and are vulnerable to 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.

2.2 Gossip Protocols

Gossip or epidemic protocols are consequently used to disseminate information across a large number of participants. The message propagation pattern resembles the propagation of a biological virus on a population. In the context of the Peer-to-Peer (P2P) model, a node propagates a message to fanout (t) of its neighbors, the nodes to whom it is connected through an overlay network or the nodes known. The receiving nodes will perform the same procedure to their neighbors and so on until all nodes receive the message. This transmission pattern allows the system to distribute the load evenly among nodes and though 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 the some servers might consume all their resources while others might be idle. Additionally, some servers might be stalled while waiting for the completion of sub-tasks in overloaded servers. Additionally servers specialization introduces point of failure on the system because a service or the whole system can be compromised if a sub-set of specialized server sub-set fail simultaneously. Although some systems apply a fail-over strategy that requires more equipment, it results on additional maintenance costs and potentially leading to an increase in idle resources.

A system that follows the P2P model assigns both roles, Client and Server, to each host. 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 cost. 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

Message dissemination strategies affect the delivery reliability, the redundancy of messages, and when the information being disseminated is membership control information it might also affect the overlay regeneration speed. Those strategies can be classified as:

Eager push Nodes send the new message to their neighbors as soon as they received it. This strategy allow a fast message dissemination but a high number of selected neighbors 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. 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 and similar to the push strategy, nodes send the identifier of a message to its neighbors as soon as it receives that message payload. It has the advantage of reducing the amount of information exchanged.

Hybrid combines 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 relations among P2P nodes form overlays that may be used to disseminate messages related to the system's service or its management. The overlays are network composed by logical links that abstract the characteristics of the underlying network, which may be other overlay or the physical network topology. An overlay can be categorized in: 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[19], Pastry[18], and Tapestry[2]. Although structuring the overlay improves search primitives, it decreases the flexibility to handle the churn[1] because it requires that all nodes contain the most up-to-date information, which adds expensive synchronization overhead in order to guarantee the system correctness. Therefore structured overlays aren't as resilient the unstructured overlays because of the lack of flexibility which is necessary to support efficient healing mechanisms.

Chord[19] 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 overlay. 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[18] is a self-healing protocol that offers an 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 node 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

[1] Churn is the frequency of arrival and departure of nodes in the system

the structure with its closest neighbors and if 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 theirs routing structures content. Nodes leave silently and Pastry handles them as it does with failures, by replacing failed nodes by nodes in its additional structures. If the failure was detected on discovery, it forwards to other entry of the same level but if the failure was detected on the others structure, it exchanges the structures content with the closest to the faulty to update them.

Tapestry[2] 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 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 contain 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 fault.

2.2.3.2 Summary

Protocols that build structured overlay network support efficient routing primitives, though it 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. Otherwise, nodes only require information about a few others which scales better but it is also difficult enforce the constraints on the topology that must be maintained among nodes. Although protocols over 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, which provides high flexibility to churn, high resilient, 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 system require. However, these overlays are enough to manage memberships, which may operate with partial information of the system. These concept are typically employed 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[6], Cyclon[21] and HyParView[11].

SCAMP[6] 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 $c * log(n)$. 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 the view size. Otherwise, it will forward the message to a random node among its neighbors. Leaving nodes send an unsubscription message to their neighbors, which will propagate it and any receiver will remove the node if contained. To avoid the 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[21] offers better connectivity, average path length, average clustering coefficient with high resilience, high failure tolerance, and fast self-healing by applying periodic shuffling of information. 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, the age. 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 view after inserting an entry of itself with an initialized 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 but it is only repaired on the next round. Even on the occurrence of a massive failure, the remaining 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[11] is a peer sampling service that offers high resilience and high delivery reliability even in cases of extreme failure, such as 100% reliability when 80% of nodes failed simultaneously. To accomplish this, it relies on a hybrid approach that uses two 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 gossiped join/leave/failure messages. 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 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.

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 under high failure occurrence (in 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.

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 algorithm one applies an optimization procedure. The optimization consists on link's exchange between nodes that benefits the protocol according to a pre-defined metric, enabling for instance, the network to benefit from knowledge regarding the underlying network topology to provide more efficient delivery mechanisms that unstructured overlays couldn't. Examples of such protocol are presented bellow.

T-MAN[7] 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 a new view by apply a specialized ranking function to the resulting 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 [12].

Araneola[14] 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 exchange random identifiers from their views through piggyback periodically. The paper has 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 ids 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 the gossip rounds. The fine-grained control over the view size allows Araneola to offer high delivery reliability and load balancing when parameterized correctly.

GoCast[20] was designed to tune/control nodes' degree in order to offer reliable and resiliently properties. Additionally, it bias the overlay in function to the nodes' proximity to offer better latencies while letting partial views have an additional small number of random nodes to provide robustness to the 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 allow to optimize, without risking removing or establish connections to nodes in critical situations, avoiding the emergence of nodes with high or extremely lower popularity. GoCast uses a hybrid dissemination approach that allows to efficiently deliver the messages and recover missing messages that will be 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[12] 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). The X-BOT is built on top of the 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.

Plumtree[10] 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 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 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 to detect slow links or faulty nodes. In that scenario the second node to provide the identifiers takes the place of the faulty node and on the next message dissemination any redundant links created, due to independent decision by all nodes that perceived the failure will be removed. 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 restricting its scalability to large system sizes.

2.2.3.6 Summary

Protocols that build partially-structured overlays provide efficient delivery mechanism but there are several situations to be avoided to ensure that the resulting overlay network maintains important properties of pure unstructured overlay networks. it is important to maintain some unbiased neighbors to provide more robustness and global connectivity as well 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 logical partitions.

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 evaluation metrics:

Connectivity: States that there is a path that connects every node. It guarantees that messages sent from any node is 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, which is equal to the number of its neighbors.

Clustering Coefficient: A metric that evaluates the density of the nodes neighboring relationships redundancy and a chance of a subset 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 fast usually).

Accuracy: The accuracy of a node allows to analyse the reliability of the membership protocol gossip selection targets used by the dissemination protocol.

Reliability: Value that measures the gossip protocol to delivery a message to all active nodes. 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 the gossip protocol. 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 has into account any control messages exchanged during the dissemination procedure.

2.3 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 technology 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 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 network partitions.

Chapter 3

Proposed Work

This chapter presents the exiting technologies that will be the base for the proposed solution, followed by a brief sketch of the proposed solution and the detailed work plan.

3.1 Proposed Solution

The solution aims to propose, implement, integrate, and execute a novel design for a partition tolerant unstructured overlay to support membership services for large scalable systems. The partition tolerance requires a scalable, robust, and self-healing protocol, which are common characteristics of unstructured overlays. Thus the solution will depart from the general design of the HyParView and enrich the component of the protocol for managing the contents of partial views maintained by nodes to avoid the pre-mature exclusion of faulty nodes due to transient network partitions that make impossible for nodes, across distinct geographical areas to communicate (during the duration of the partition).

The proposed solution intuition is the following. Supposing that the HyParView could distinguish between failure of a nearby node from a distant one then it could reestrict the management of replacing of faulty nodes in the partial view by only using nodes located in the same region as the faulty node. However, to successfully replace nodes in function of theirs location, the HyParView protocol needs to have that knowledge, which is not available for a node, given that it operates on a fraction of that knowledge (the partial view). However, the protocol assisted by an Oracle that could infer the location through the analyses of messages exchanged between a node and its neighborhood could enrich the knowledge base of HyParView to make this possible.

The solution explores three inference techniques that will result in three distinct Oracles that follow the intuitions discussed originally in the X-BOT's paper. This techniques

are:

Deployment labels: Cloud virtualization services provide an API[1] for nodes do query information about the machine. This solution will use such APIs to obtain a machine label that will be added to the node identifier. Thus the oracle could process the identifiers that already travel with the messages to provide the output.

Latency Bucket: Without any additional knowledge, the oracle will process the incoming messages latencies and combine the intermediate result with an IP-based clustering algorithm as described in [8] to classify groups of nodes in a scale of proximity. In order to provide more precise results, the output will be disseminate through the network to allow oracles to improve such inference mechanism using partially shared knowledge.

DNS: Exploit external DNS services to obtain the information about the localization of the virtual machine instance. The result is then handled the same way as on the Latency Bucket described above.

Furthermore to increase the robustness and network partition detection ability of the protocol we also plan to enrich the overlay management algorithm by decomposing partial views by applying restrictions over the minimal number of neighbors that a node can have from the same location. Thus decreasing the average path distance and the average clustering coefficient of the whole overlay, and in some measure minimize long links while still ensuring global connectivity.

The proposed solution will be integrated on Cassandra in order to provide more adaptability to dynamic environments and reduce the need of human intervention on membership management tasks. This will be a case study used in the context of this walk to showcase the benefits of the novel membership scheme to be developed.

3.2 Evaluation

The proposed solution will be evaluated as follows:

Location inference precision: It will be analysed through simulations of diverse cloud settings and by deployment of prototypes on cloud virtualization services and composing the results of our location inference oracles with the real deployment.

Communication overhead: It will be measured the quantity of messages exchanged on the recovery process to determine how much overhead is added by the solution.

Efficiency of recovery: We will measure the time required, using our prototype, to recover the overlay after a partition holds.

[1] The AWS SDK provides the API to query metadata about the deployed services and the location using "public static List<Region> getRegionsForService(String serviceAbbreviation)" from the Region Package.

Fault-Tolerance and Partition Tolerance: We will again resort to simulations to obtain initial results that demonstrate the correctness of the solution, and resort to prototype deployments for a final validation, we will simulate network partitions using the IPTables firewalls on nodes.

3.3 Work Plan

The work plan is divided in two phases. The first focus on designing and evaluating the algorithms and the second the integration of the protocol on Cassandra. Table 3.1 shows the time period predicted for each task, followed by a brief description of each one on sections 3.3.1 and 3.3.2. Figure 3.1 shows how tasks overlap graphically.

Phase	Start Date	End Date
Phase 1	August 1	December 15
Design 1	August 1	September 1
Implementation 1	September 1	September 30
Evaluation 1	October 1	October 10
Implementation 2	October 11	October 26
Evaluation 2	October 26	October 31
Implementation 3	November 1	November 15
Evaluation 3	November 15	December 15
Writing the scientific paper	November 15	December 15
Phase 2	December 16	March 20
Design 2	December 16	December 24
Integration 1	December 24	January 31
Evaluation 4	February 1	February 8
Writing the thesis's document	February 1	March 20

Table 3.1: Schedule

3.3.1 Phase 1: Design and analysis of algorithms

Design 1 This task will focus on the design and integration of the location Oracles in HyParView.

Implementation 1 On this tasks the design of the extended HyParView will be materialized through a prototype to use on a simulator (such as PeerSim) and a running prototype to be deployed in real settings.

Evaluation 1 Period dedicated to evaluate the implementation correctness and benchmark its performance for future overhead comparison with the original HyParView protocol.

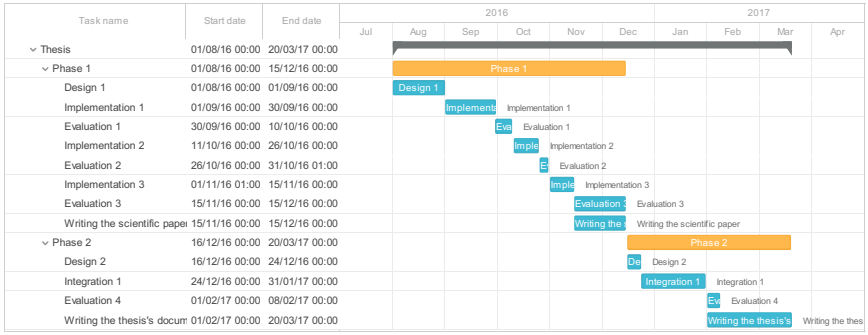

Figure 3.1: Schedule of activities

Implementation 2 Adaptation of the oracle design to address possible limitations detected in the previous evaluation task and implementation.

Evaluation 2 Evaluation of the oracle correctness. It will be performed benchmarks and adjustments to its implementation.

Implementation 3 Period to integrate the two components together with the necessary adaptations if needed (as observed in the previous evaluation phases).

Evaluation 3 Evaluation of the oracle-assisted protocol and analyses of the communications overhead.

Writing the scientific paper To conclude the first phase, we plan to write a paper to an international forum (such as DSN[2] or SRDS[3]) to present the main contributions of the thesis.

3.3.2 Phase 2: Integration with Cassandra

Design 2 This task focus on the adaptation of the design of the proposed membership service into the Cassandra architecture.

Integration 1 On this task it will be integrated the proposed solution into the key-value store Cassandra.

Evaluation 4 Evaluation of the system as a whole using NoSQL typical benchmark tools (such as YCSB[22]).

Writing the thesis's document The writing of the thesis's document.

[2] 2017 IEEE/IFIP International Conference on Dependable Systems and Networks
[3] 2017 IEEE Symposium on Reliable Distributed Systems

3.4 Summary

In this document, we present a proposal for design a novel distributed membership service that requires a fine-grained management of its resources with high levels of automation to reduce maintenance costs while increasing its robustness to faults, particularly to network partitions. Our solution leverages existing gossip-based protocols and location inference techniques to achieve a highly connected, self-healing, and network partition tolerant protocol that will allow distributed system to scale even further, in the future.

Bibliography

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

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

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

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] *Yahoo! Cloud System Benchmark*. https://github.com/brianfrankcooper/YCSB. Accessed: 2016-06-06.

[23] 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.