Paulo Henrique Branco Dias

Degree in Computer Science and Engineering

Tree-based Decentralized and Robust Causal Dissemination

Dissertation plan submitted in partial fulfillment
of the requirements for the degree of

Master of Science in
Computer Science and Engineering

Adviser: João Leitão, Assistant Professor,
NOVA University of Lisbon

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

February, 2019

Abstract

Managing a large-scale distributed system can be a really complex task, that leads engineers to seek a high level of knowledge about the corresponding environment. New challenges and research are often emerging regarding this topic. One of the popular challenges that has gained significant attention recently is related to the pursuit of the maximum level of consistency that is possible to provide while remaining available in light of the CAP theorem. Causal consistency is one of the most common and interesting approaches that has been extensively studied in this context.

One way to build geo-replicated and highly distributed storage services providing causal (or causal+) consistency is to rely on a causal dissemination service that allows replicas to broadcast updates received locally to all replicas, ensuring that these are received in an order that respects causality. Building such dissemination service in a way that is scalable, robust and efficient is however a complex task. In this dissertation we will tackle this challenge by exploring how causal broadcast protocols certify the properties discussed above.

Our solution will have two main aspects: a robust and reliable dissemination protocol and an efficient causal consistency delivery enforcing mechanism. Moreover, we plan to implement our protocol in a realistic context, with large-scale systems that can validate the scalability of our proposed work.

Keywords: Distributed Systems, Causal Dissemination, Causal Consistency, Geo-Replication, Unstructured Overlay Networks

Resumo

Gerir um sistema distribuído de larga escala pode ser uma tarefa complexa, que leva engenheiros a estudar pormenorizadamente o contexto do sistema em questão. Novos desafios e pesquisas relativas a este tema estão a surgir cada vez com maior frequência. Um dos desafios mais populares relacionados com este tópico tem como objetivo alcançar o nível máximo de consistência possível, não comprometendo a disponibilidade, tendo em conta as premissas do teorema CAP. A consistência causal é uma das abordagens mais interessantes e comuns que tem sido estudada ultimamente.

Uma abordagem possível para desenvolver serviços de armazenamento de dados altamente distribuídos e geo-replicados que ofereçam consistência causal é confiar num serviço de disseminação causal que permita que as diversas réplicas possam propagar atualizações recebidas localmente para todas as outras réplicas, assegurando que estas são recebidas numa ordem que respeita a causalidade. Implementar um serviço de disseminação deste género e que seja escalável, robusto e eficiente é, contudo, uma tarefa difícil. Para contornar este desafio, nesta dissertação, o nosso principal objetivo é explorar a forma como os protocolos de disseminação causal certificam as propriedades referidas anteriormente.

A nossa solução terá duas características principais: um protocolo de disseminação robusto e confiável e um mecanismo para contemplar consistência causal de um modo eficiente. Além disso, queremos implementar o nosso protocolo em contexto real, recorrendo a sistemas de larga escala que possam validar a escalabilidade da nossa implementação.

Palavras-chave: Sistemas Distribuídos, Disseminação Causal, Consistência Causal, Geo-Replicação, Redes Sobrepostas Não Estruturadas

Contents

List of Figures

Chapter 1

Introduction

1.1 Context

Nowadays we can find distributed systems everywhere, with wide and heterogeneous structures. The sudden evolution of technology on a global scale highlighted the need of solutions to spread information to different points of the world fast and safely.

One of the main techniques that is used to overcome some performance issues is replication. This is motivated by the fact that most of the systems want to operate at a global scale, which lead to the emergence of the concept of geo-replication. Beyond the possibility of faster responses from the system, with geo-replication it is also possible to fulfill the need of fault-tolerance, since with this approach multiple copies of each data item are kept across different data centers.

Taking into account the existence of several user's operation entry points in this model, it is necessary to propagate the effect of the performed operations across the different replicas. This propagation takes some time to perform and the replicas will receive the indication related to the operation at different moments, so it can be perceived that an operation performed by two different clients at the same instant, can result in two different behaviors, revealing an inconsistent state of the system.

Consistency is a crucial property in many contexts, but is not the only that have to be considered. The CAP theorem [3, 9] has famously established that it is only possible to assure two of the following three properties simultaneously: 1) strong consistency, 2) availability, and 3) partition tolerance.

Since network errors are unpredictable and they will eventually occur, the partition tolerance property is mandatory. As a result of this assumption, one is left with two options: strong consistency or availability.

Despite the relevance of consistency in this kind of systems, availability represents a

necessary requirement when the main focus of the service is usability, or in other words, when human users interact directly with the system.

There are many different consistency models studied in the context of distributed systems, although usually we classify them among two main categories: weak or strong consistency.

Strong consistency models ensure that every process of a system will return the same result to a certain request, at any given moment by ensuring that all replicas execute in a strongly synchronized way. But this guarantee has a high cost in performance, compromising availability. Weak consistency models do not assure this, but allow to considerably improve the performance of these operations.

In order to maintain availability, many weak consistency models were designed and implemented. One of the most popular models is causal consistency, that has been proven as one of the strongest weak consistency models that can be implemented .

The majority of the services that are provided currently focus heavily on user experience. Due to this fact, weak consistency models, namely causal consistency, become very attractive to implement and provide in these contexts.

To maintain a service that is required in many places of the planet, it is necessary to find a reliable way to exchange data between the different locations, considering that the machines that will support this service may be subjected to failures. In order to tackle this, we need to find a reliable approach to disseminate messages.

1.2 Problem statement

The main goal of the thesis is to develop a novel design for robust causal dissemination on dynamic large-scale distributed systems.

A service that relies on a geo-replication scheme is a perfect example of a dynamic large-scale distributed system. Due to that fact our solution will aim at being suitable to integrate in that class of distributed systems.

The main goal of our solution is to provide causal consistency, with high efficiency and robustness. Currently there are many approaches that try to accomplish this, but they differ in several ways. In order to build a solution that behaves in the most efficient and robust way, we will study in detail these techniques, try to extract the best features of each one, and merge them into a more balanced and general solution.

It is also our intention to explore the possibility of transforming our replication structure into a tree topology, a standard type of graphs that can lead to the possibility of using very efficient algorithms, although really sensible to failures. We will study how to enrich such solutions to make them fault-tolerant without sacrificing performance.

1.3 Document structure

The present document is structured in the following chapters:

- **Chapter 2: Related Work**

  This chapter presents the main concepts related to the work to be conducted in the thesis and discusses existing approaches that has been developed to address similar problems.

- **Chapter 3: Planned Work**

  Covers all the aspects of the planned work, in order to implement a solution to the problem of building a robust, efficient and scalable (i.e. decentralized) causal broadcast solution, namely a description of the direction we will follow to build our solution and a work plan.

Chapter 2

Related Work

The purpose of this chapter is to expose the state-of-the-art on the topics related to our planned research.

Firstly, we provide an overview of the most relevant consistency models and how causality fits in these models (Section 2.1). We proceed by describing some of the main overlay networks (Section 2.2) and the meaningful information about data dissemination through a network (Section 2.3), since this will help us in addressing some of the properties of the dissemination protocol to be developed.

Moreover, we will detail data replication strategies (Section 2.4), such as geo-replication, and indicate some of the pertinent geo-replicated storage systems and describe how they can inspire us in the design of our solution (Section 2.5).

Finally, we will present a few conclusions (Section 2.6) and a summary of this chapter (Section 2.7).

2.1 Consistency Models & Causality

There is a significant body of theoretical content (including many theorems and formal proofs) that describe some relevant restrictions on the domain of distributed systems, namely ACID, BASE and CAP.

The CAP theorem states that it is impossible for a distributed system to guarantee all of the following properties simultaneously:

- **Strong Consistency:** All the interactions performed by any client are in accordance with the most recent write;
- **Availability:** Every request must be able to complete and provide an answer to the client;

- **Partition tolerance:** The system works correctly (i.e. continuously provides all guarantees) even in the event of a network partition.

Consistency is a concept that can lead to misleading interpretations because it has a different meaning to the consistency concept in the context of ACID properties, which are defined considering traditional database systems. In that context, in order to accomplish consistency, it is required to continuously enforce all the rules and constraints associated with the data model, between each transaction.

Due to the arbitrary behavior of the network that links every machine of a distributed system, it is infeasible to maintain a system that does not guarantee partition tolerance. So in practice we have to effectively select between strong consistency and availability when designing a distributed system.

With this in mind, it is necessary to deeply understand the requirements of our environment, in order to establish the correct balance between consistency and availability. Currently most large-scale Internet services aim to offer the best usability to their users. Consequently, they have to attain high availability and hence, discard strong consistency.

2.1.1 Strong Consistency

A system that enforces strong consistency, has to provide to the user the perception that the system is evolving through a single sequence of states. With this guarantee, whenever a pair of users perform the same read operation, at the same time, the result should be equal for both of them.

To give this perception to the user, we can adopt some approaches, such as guaranteying linearizability or serializability, which are particular consistency models in the family of strong consistency.

Linearizability guarantees that for a single object, whenever a operation is performed, its effects become visible instantaneously. Therefore, once a read returns a certain value, all subsequent reads should return the same value or another corresponding to a more recent write (considering the wall-clock time).

Serializability is similar to linearizability, but it is usually applied on transactional systems, instead of single operations. A transaction is a group of operations, commonly, containing write and read operations. However, it does not impose any kind of strict order, but requires an execution that is equivalent to a serial execution of all transactions, even if this order does not respect the real time at which transactions were submitted to the system.

Strong consistency allows to easily reason about the evolution of the state of a system, which would be a perfect background to build applications. However, this is an unrealistic scenario in many cases, because it requires to contact a majority of the replicas in

every operation, causing a huge overhead in the network. In scenarios where network partition can happen, it might be impractical to achieve strong consistency, since it becomes impossible to contact a majority of the replicas.

2.1.2 Weak Consistency

Considering that strong consistency models are not much attractive for specific user-centric applications, since they are not able to ensure low latency and always available operations, the interest in weak consistency models has increased significantly in the recent past.

Weak (or available) consistency models provide high availability, although they forsake strong consistency. In certain circumstances, we need a considerable level of consistency in order to ensure some kind of arrangement between the operations performed by the clients. For instance, in a system with shared data items and access control based on roles, preserving some sort of temporal agreement between operations is crucial, to enforce the desired semantics of the system.

Regarding this previous example, consider a simple system with a certain data item i and two clients $c1$, $c2$ that have privileges to access i. One example of an execution that can be problematic is: 1) revoke of the access privileges of $c1$ on i, 2) write operation performed by $c2$ on i, and 3) read operation performed by $c1$ on i. If we consider an execution without consistency guarantees and since the two actions that were performed are not atomic, it is possible that $c1$ obtains the current state of i, even after his privileges to i are no longer available, because it can read an outdated version of the access control list.

One of the more popular weak consistency models is eventual consistency, and it is an example of a model that is not able to avoid the issue illustrated above.

Eventual Consistency

Currently several systems adopt eventual consistency in their architecture, since it provides them with high availability and very low latency. As we can deduce from the name of the model, the only guarantee that it provides is that eventually, when no more write or update operations are made, all replicas of the system will converge to the same state. Consequently, it is possible to observe some obsolete values in portions of the application state (sometimes referred as reading stale data).

Naturally, during the execution of a system providing eventual consistency, different replicas might observe different write operations, leading to their state to diverge. In these cases some mechanism to reconcile such divergent states must be employed. The most popular techniques are:

Last Writer Wins assures that for each set of concurrent operations, the most recent one

is chosen to define the final state of the system. This technique can be difficult to implement using clocks, since different replicas can have their clocks desynchronized. Usually logical time is used for the purpose of the last issue.

Merge Procedure entrusts on the system programmer the responsibility to solve the conflicts. In practice, this is verified by furnishing the replicas with some deterministic heuristic that leads replicas to authentically merge divergent states.

CRDTs [4] which stands for Conflict-free Replicated Data Types, are replicated data types that can be integrated into a distributed system. They work as an isolated module that guarantees eventual consistency by internally handling divergent states and reconciling them, discharging that responsibility from the rest of the system.

It has already been recognized that maintaining high availability allied with a satisfactory level of consistency is not a trivial task. With this in mind, the need for increasing the level of consistency in this type of models became increasingly relevant, leading to the appearance of stronger available consistency models.

2.1.3 Enforcing Causality

Remembering the system example that we referred previously, we were not able to guarantee that after the cancellation of the corresponding privileges, the solicitation of *c1* to access *i* would be denied when considering an operation that does not provide any consistency guarantees (offer than eventual convergence). What we should aspire here is to define a logical relation between the two operations that would prevent such an execution to happen.

Leslie Lamport devoted some of his research regarding on this particular topic and introduced key concepts [13] that will guide our path towards the model we are pursuing.

Happened-Before Relation ($\rightarrow$)

Given two events *e1* and *e2*, then we can define that *e1* happened before *e2* (symbolized by $e1 \rightarrow e2$), if we verify at least one of the following conditions: i) *e1* and *e2* are events in the same process and *e1* preceded *e2* ii) a certain message *m* was sent by a process *p1* to other process *p2* then *e1* corresponds to the send event on *p1* and *e2* to the receive event on *p2* iii) given some additional event *e3*, such that $e1 \rightarrow e3$ and $e3 \rightarrow e2$. Therefore, if $e1 \rightarrow e2$, we can also state that *e1* causally affects *e2*.

Causal order

A system is said to ensure causal order, if all the happened-before relations are preserved across the execution of operations on all replicas of the system.

Causal consistency

Causal consistency is known as one of the strongest weak consistency models and can be defined as the consistency model that enforces that clients observe the evolution of the system in a way that respects the causal order among all write operations. It can be also described as a replicated system that enforces the four session guarantees:

1. **Read your Writes:** A client must always be able to observe the effects of all previous writes issued (and for which it got a reply from the system) by itself;
2. **Monotonic Reads:** Subsequent reads issued by the same client should observe either the same state or an inflation of the system state;
3. **Monotonic Writes:** The effects of multiple write operations issued by a given client, must be observed respecting that order by every other client.
4. **Writes follows Reads:** If a client observes the effects of a write operation in its session, then any subsequent write operation issued by that client must be ordered after the previously observed write.

Causal+ consistency

Causal+ consistency [19] is obtained from the combination of causal consistency with eventual consistency, or in other words, it is causal consistency with guarantees of convergence of the state across the whole network, at some point in time after no more update operations are issued to the system.

2.1.4 Discussion

As we have been discussing throughout this section, large-scale Internet services pursue to have the best usability as possible, which implies both continuous availability and low latency. Consequently, it is necessary to adopt weak consistency models in order to provide high availability.

Our research will target this kind of systems and hence, we will focus on solutions that provide causal+ consistency, which is the strongest approach that is employed currently when considering weak consistency models. In practice, we just need to focus on causal consistency dissemination primitives, since we are aiming at designing broadcast solutions and these already guarantee that every node will converge to the same state, if the broadcast process is reliable.

2.2 Overlay Networks

An overlay network can be defined as a logical network, that is built on top of another network. Each process has a set of neighbors and each neighboring relationship is represented by a logical link. In order for an overlay network to be considered as correct, it has

to accomplish the following requirements: 1) for each pair of two correct processes, there must be a sequence of links that results on a path connecting them, and 2) eventually, when a process fails, every other process that had a link to that failed process, will remove it.

The concept of overlay network has an ambiguous meaning, since any logical network implemented on top of another network fits this definition. For instance, peer-to-peer networks are built on top of the Internet, the most common and general network that exists.

In the context of this research, our purpose is to focus on overlay networks that leveraged to provide some kind of service. In particular, we plan to build our dissemination scheme on top of overlay networks. Hence, we can look at overlay networks as an additional level of abstraction that is implemented over a network, with the intention of increasing the performance and usability of the underlying network.

Furthermore, overlay networks can be categorized by the way they structure their processes and manage the logical links that effectively form the network.

2.2.1 Unstructured Overlays

In unstructured overlay networks, as the name suggests, there is no evident structure or meaningful topology that can be identified. Moreover, the neighbors that are linked to a certain node are usually selected by an arbitrary procedure. The neighborhood of each process is usually captured through their local partial view. There are many implementations of this type of overlay and is precisely in this previous point that they mainly differ, that is in the way they maintain and update their partial views.

Scamp [8] is distinguished by its partial view managing mechanism. The partial views are updated when a node joins or leaves the network. Periodically, every node in the network sends an heartbeat message to all the nodes of its partial view. If a long period of time was elapsed since the last heartbeat message arrival in a certain node, then this node rejoins the network, as it assures to be isolated from the overlay.

Cyclon [26] has a very different mechanism from the one that Scamp provides. This mechanism relies on a cyclic behavior of exchanging neighbors between each node and his oldest neighbor. In order to perform this exchange, both nodes select a few neighbors from his partial view to send to the collaborating node. Symmetrically, each node collects the selected node information and uses it to update the contents of its own partial view.

HyParView [16] is a sort of a hybrid solution of the two above, extracting the best features of both and presenting better performance and resilience to failures than the previous two. In HyParView, each node has two partial views: 1) a small active view, that supports the main purpose of the network, namely the dissemination

of messages, and 2) a wider passive view, whose objective is to provide substitute nodes to the active view in case of failures.

The active view management follows a reactive behavior, similar to Scamp. On the other hand, the passive view is maintained in a way that is similar to the one employed by Cyclon, through a cyclic (or periodic) procedure.

2.2.2 Structured Overlays

In this kind of overlays, a structure that shapes the network can be perceived. Generally, each node is enhanced with an identifier that determines the relative position of the node in the network, in order to enable hashing techniques that will lead to efficient ways of locating a specific resource that is stored in a certain node. Some concrete techniques are further detailed in the following algorithm descriptions:

Chord [24] uses *consistent hashing* [11] in order to assign each node an identifier. This procedure leads Chord to provide the functionality of a Distributed Hash Table (DHT). The network topology can be defined as a ring, as each node is linked to the successor, which is the node with the following (i.e. closest) identifier. Hence, the last node's successor turns out to be the first node, consummating the ring.

Besides the successor's identifier, each node also keeps other useful information, namely a *finger table*. As the name suggests, this table presents some *fingers*, that are a sort of shortcuts to convenient zones of the ring, that make the resource location process and application-level routing more efficient and faster.

Pastry [22] is identical to Chord, since it also follows a DHT philosophy and its network is structured as a ring. Pastry differs from Chord in the identifier generation process, which selects a random hexadecimal regarding a predetermined upper bound, and in the data that each node keeps, as it also stores a *neighborhood set* that maintains information about nodes that are nearby, in terms of network proximity (i.e. low latency nodes).

2.2.3 Partially Structured

The genesis of this classification arises from the lack of efficiency of unstructured overlays on providing adequate support for specific use cases and properties, such as exact match resource location and topology adaptability. Therefore, partially structured overlays implement certain optimization techniques on unstructured overlays. Consequently, this withdraws the total randomness that was verified in the original unstructured overlays. The fundamental pursuit of these overlays is to effectively support those particular use cases, maintaining the benefits that unstructured overlays provide, such as low management overhead and robustness.

T-MAN [10] is known for its ability to adapt any overlay topology to another topology that the user requests. The technique that accomplishes this ensures that every node periodically exchanges their partial views with its neighbors. A merging function is applied, enabling each node to find it target position, in order to achieve the predefined topology. However, this ability to modify the topology can compromise some properties of the primordial overlay, such as the in-degree distribution of nodes, and hence, a few problems can emerge, such as unbalanced load distribution and overall connectivity loss.

X-BOT [17] enables unstructured overlays to bias their topologies concerning a target efficiency metric. It introduces the *oracles*, which are components that are responsible to determine the cost of linking two given nodes, according to the predetermined efficiency criteria. In contrast to T-MAN, X-BOT aims to preserve as much as possible the underlying network's (i.e. HyParView) properties, such as low diameter and clustering, and connectivity.

Plumtree [15] combines tree-based broadcast and gossip primitives. It disseminates messages through trees, while promoting the use of the remaining links of the underlying overlay in order to recover trees from failures in a decentralized fashion. As the major part of the tree-based broadcast protocols, Plumtree is not highly resilient to failures, due to the fact that a failure triggers the *tree repair*, which is a slow process. Nevertheless, it behaves faster than traditional tree-based broadcast protocols in this process, but not so fast as regular unstructured overlays do.

Thicket [5] also uses tree-based broadcast and gossip approaches, like Plumtree does. However, it introduces the technique of combining multiple spanning trees on top of the underlying overlay. This method promotes a balanced load, since the messages being disseminated are split among the set of trees. Furthermore, Thicket presents the possibility of building trees with limited weight, that originates the decrease of the overhead imposed on the trees.

2.2.4 Discussion

Regarding overlay networks, we can conclude that structured overlays are not the best suitable for our goals, since they have lower robustness to failures as a result of the limitations that have to be imposed on the neighborhood to enforce the target topology.

Unstructured and partially structured overlays are easier to implement and maintain, leveraging our goal of supplying a highly dynamic and large network.

Taking into consideration our interest on the integration of tree topology into our protocol, Plumtree and Thicket arise as interesting starting points for our research. Both operate on top of HyParView, which can also be leveraged to design our solution.

2.3 Data Dissemination

In the context of distributed systems, a fundamental concern is how to transmit a given message from a sender process to a receiver process. This procedure must be as fast and safe as possible, therefore many approaches have been discussed. One of the most relevant solutions is the Point-to-Point protocol, that is the responsible layer for data link between the two corresponding nodes. Since most of these transmissions are over the Internet, it became crucial to develop a protocol that deals with the transport layer and was with that purpose that TCP appeared.

Concerning the previous solution for the transmission of a message between two points, the next step is to understand the applications that it might fulfill. We now present two of the most general data dissemination methodologies.

Broadcast

Broadcast is the most general communication method. A given message is spread to every device on the network. Despite the simple definition of broadcast, there are several approaches that are employed in order to accomplish this apparently basic task. There are other versions of this technique that only addresses a subset of the whole network, namely multicast, unicast, and anycast. However, we will focus on broadcast, since our goal will focus on this type of primitives.

Publish-Subscribe

This scheme commonly has two types of users: publishers and subscribers. The publisher is responsible to inject new messages into the network, that are categorized into classes. Each subscriber has a list of classes that it is interested and thereby it only receives the messages that are linked to the classes he has explicitly subscribed to. The filtering process of deciding in which class should a message be inserted can have different behaviors and is precisely on this point that the implementations of this method contrasts among different alternatives. This sort of primitive can be implemented in a centralized way. However, our interest lies on decentralized approaches, out of which Scribe is a well known solution.

Scribe [23] is one of the most popular implementations of the primitive described above. It lays on top of Pastry, a structured overlay network which was previously discussed. In this solution, every node can subscribe, unsubscribe, create a topic, or publish a new message into the network. Each topic has an identifier, which is defined during the topic's creation, and every message is related to a topic. Scribe totally relies on Pastry to route the messages through the network, leveraging some of Pastry's properties, such as scalability and fault-resilience.

2.3.1 Properties

Depending on the type of data dissemination that is required, there are some explicit properties that can be provided by a broadcast solution. We proceed with the description of some relevant properties and how they are used in practice.

Best Effort

Services that achieves this property do not provide any guarantee about the delivery of the data. As the name suggests, in a best effort solution the dissemination procedure aims to spend the least amount of resources as possible. For instance, an implementation of the best effort broadcast can be simply described as a single transmission from the sender to each of the receivers, without performing any type of validation afterwards or recovery mechanisms for lost messages. Consequently, there are no guarantees of total coverage of the network.

Reliable

In contrast to best effort, this property is able to provide delivery guarantees regarding the transmitted messages. It can be built on top of best effort solutions, so it has to introduce those guarantees by its own. Techniques like infinite retransmissions, *acks* and reception windows are used to surpass this issue.

An example in a realistic context is the layering of TCP over IP, a combination that is known as TCP/IP. On this example TCP plays the role of reliable property and IP the role of best effort, since TCP develops a reliable delivery protocol on top of IP, which is unreliable.

Total Order

This property is verified only if every process strictly receives the same sequence of messages in the exact same order. We can easily deduce that this is not a trivial task to accomplish, for instance, it is only possible to achieve in synchronous systems, as it has been proved by FLP[1] [6]. Despite that fact, it is possible to circumvent this issue, developing solutions that behave in a synchronous way most of the time, such as Paxos [14].

Causal

As we have previously discussed, causal guarantees have several benefits regarding the levels of consistency on many systems. To reach this property, a network has to assure that all the causal dependencies between events are respected in every node of the network. In this context, the main events are the sending and receiving of the messages and, for

[1] FLP proved that consensus problem and all equivalent problems are only possible to solve in synchronous systems. Since total order broadcast is equivalent to the consensus problem, we can conclude that total order broadcast is also impractical in asynchronous systems.

instance, every receiving event must appear after the corresponding sending event, in order to secure this specific causal relation.

PRC-Broadcast [21] is a causal broadcast implementation that aims to improve the space complexity of the data that forbids multiple delivery. It introduces a new technique that permits to identify the necessary data to maintain, without keeping obsolete data. This technique lays on the determination of the active messages that can be delivered at a given moment and for those messages that were not chosen, it simply discards their data. To determine these obsolete control information, it uses *link memories*. *Link memory* can be defined as a way to guarantee exactly-once delivery while safely removing obsolete data. These *link memories* are the logical relations that are established between the nodes of the network in order to propagate messages among them. Assuming reliable FIFO links to ensure causal order, in a link (a,b), process b remembers among its delivered messages those that it may receive from this specific link and forgets those that it will never receive from it. To accomplish this, in the initialization procedure is necessary to exchange a few control messages with the set of delivered messages of each process. This implementation provides a relevant improvement in the complexity of the data kept in each point of the system.

2.3.2 Techniques

Many specific techniques have been built in order to address the different primitives of data dissemination that are required. Depending on the environment of the network and on the properties that we want to employ, we can develop a distinct technique concerning those characteristics. We now discuss two of the main techniques that we identified as the most relevant in the literature.

Gossip

Gossip, or epidemic, protocols have a generic behavior in order to flood a network: When a node receives a certain message, it verifies if it has already delivered this specific message and if he did not, he send it to t other nodes. With this approach we can, with a huge probability, make sure that the message will eventually be delivered by every process of the network.

Despite the fact that gossip is not able to give total guarantees that the message will be delivered in every process, in practice, it will eventually happen. Therefore, we can even configure this probability, adjusting the parameter t, that is also named *fanout*.

Other adaptable aspect of gossip is the communication mode. The common communication modes are:

- **Eager push:** The sender transmits the entire message in the first interaction with the receiver;

- **Pull:** Every node periodically asks other nodes for new messages, if a node that receives this request has a new message, then it sends the message to the corresponding node;
- **Lazy push:** The sender transmits a message identifier to receiver and if it still does not have that specific message, it asks the sender for that message. Finally, the sender transmits the entire message to the node that asked for the message.

Due to these adjustable features of gossip, it is possible to define different flavors of this method, in order to fulfill the needs of the network environment.

Group Communication

Group communication techniques are employed when a set of processes collaborate to achieve a common goal. Usually, this type of cooperation aims to provide certain services, implement specific algorithms or ensure a given set of relevant properties. We now describe a suitable implementation of this methodology.

Group Communication Service [25] aims to support local workstations, providing an adequate environment to distributed activities that require a group of participants cooperating, such as managing a shared document or interacting with a replicated database. This service focus on three main components: 1) manage the groups memberships, allowing the dynamic creation and reconfiguration of groups, 2) provide efficient support for exchange of information between group members, and 3) supply an execution environment that execute specific algorithms in order to achieve some desirable properties.

2.3.3 Discussion

In this section we examined the state-of-the-art in data dissemination, presenting the relevant properties and techniques related to this topic.

Group communication techniques are not applicable to our target protocol, since this kind of approaches are only useful on specific systems, such as companies' workstations.

Nevertheless, gossip protocols have interesting properties and can be integrated in our solution, namely methods that can ensure an efficient and reliable broadcast implementation.

2.4 Data Replication

Another crucial topic that is essential for the operation of many distributed system is replication. Actually, it is imperative to replicate data among the different nodes of the network, since it addresses some important properties of the system, such as fault-tolerance and load balance.

Nowadays, most distributed systems employ a replication strategy, but there are many different flavors that can be attained. In order to capture some of these, we proceed by presenting some generic categories, in which they are commonly classified.

2.4.1 Full Replication

A system uses full replication if every replica keeps the same entire copy of the system state. When an operation is performed on a certain replica, the effects are propagated to every other replica. Since all the replicas have the same state, read operations might be treated locally, in any of those replicas.

2.4.2 Partial Replication

In opposition to full replication, partial replication solutions have different replicas, containing just a part of the total amount of the system state. Therefore, a strategy has to be outlined in order to conceive replies to the requests that the clients issued, since with this approach, replicas may not have a local copy of a requested data item.

One concrete example of partial replication is caching, since a cache acts like a partial replica, maintaining only data that is often required, but supporting only read operations.

2.4.3 Geo-Replication

Geo-replication is a particular case of replication. It stands for replication schemes that allocate the replicas in different places of the globe. Usually, those locations are chosen wisely, aiming popular areas where clients interacts more frequently with the system.

This kind of strategy is commonly related with dynamic large-scale systems, that are dispersed all over the world. Due to that fact, a large number of replicas might be needed, requiring coordination schemes among the replicas that can overcome the high latency that can be verified on the communication between distant replicas.

2.4.4 Edge Replication & Dynamic Replication

Edge computing [18] is emerging as a very interesting alternative to cloud computing. This tendency brings computing power and memory geographically closer to the user, reducing the need of communicating with remote data centers, which widely increases latency. Edge replication arises in this scope, assigning replicas into near infrastructures, such as regional data centers, routers or mobile devices.

In dynamic replication schemes, the location and the number of the replicas can change dynamically according to the operation data requirements and system conditions. The environment associated with edge computing perfectly fits this definition, since edge devices are really dynamic and are only able to supply resources in a heterogeneous way.

2.4.5 Discussion

In the work to be conducted in the thesis we aim at a causal dissemination solution for dynamic large-scale systems and hence, we highlight two of the categories that we described: full replication and geo-replication.

These two classifications can be seen as complementary, since we want to propagate messages across every node of a geo-replicated network. Therefore, full replication schemes are better suited in this context than partial replication approaches.

As we have also stated, edge replication is a recent topic that is emerging and has potential to start to be employed in current systems. The systems will have to adopt themselves to operate with a larger number of replicas, which also motivates our work in this context.

2.5 Relevant Geo-Replicated Storage Systems

This section's goal is to introduce the most convenient geo-replicated storage systems, which can be useful to compare results or to extract some of their features in order to integrate them into our protocol.

2.5.1 COPS & Eiger

COPS [19] was the pioneer system providing causal+ consistency guarantees. It is a distributed key-value storage system that is only capable to run across a small amount of data centers. Each data center is represented as a *COPS cluster* that has a complete copy of the stored data. COPS requires that each *cluster* ensures linearizability. However, in order to guarantee causal+ consistency through the whole system, COPS tracks causal dependencies, maintaining for each operation a list of the dependencies that it has to respect. In order to respect these dependencies, an operation, after its local commit, has to be asynchronously replicated to the other clusters. Once this operation arrives at other cluster, it has to wait until all its dependencies are locally executed before being able to execute the received operation.

Eiger [20] is recognized as an evolution of COPS. It strictly follows the COPS structure, but providing some extra features, such as supporting column data models and supplying write-only and read-only transactions.

2.5.2 ChainReaction

ChainReaction [1] is a geo-distributed key-value data store that provides causal+ consistency adopting a slightly different version of chain replication. It has an implementation for systems with a single data center and other for systems with multiple data centers. In the first variant, it uses the same metadata as COPS for tracking causal dependencies, although this metadata is only maintained for write operations whose dependencies are

not yet stable, softly decreasing the amount of metadata. In the second, it is used all the techniques of the first variant and a version vector with one entry per data center, in order to maintain causality across the different data centers.

2.5.3 Saturn

Saturn [2] is a metadata service that provides causal consistency across different geo-replicated data services. The main focus of Saturn is to solve two problems: 1) eliminate the trade-off between throughput and data freshness that other solutions can't address, and 2) fully benefit from partial geo-replication, requiring datacenters to manage only metadata concerning items replicated locally. Saturn pursues an efficient management of metadata, keeping it with small and constant size. To address this goal, Saturn implements a technique to propagate the operations, based in a global dissemination tree interconnecting all data centers.

2.5.4 C^3

C^3 [7] is a replication scheme that offers causal+ consistency in partial geo-replicated scenarios. C^3 explicitly divides itself in two layers: the causality layer and the data store layer. The first layer propagates causality tracking information across the replicas. This information is named as *labels* and exists a *label* for each write operation, containing an unique identifier and the corresponding dependencies. The data store layer executes the operations in the local data center and propagates them to the others data centers. The two layers have to agree when an operation can be executed in order to respect all the causal dependencies.

2.5.5 Cassandra

Cassandra [12] is a popular distributed storage system that ensures high scalability and fault-tolerance. It was used on Facebook's architecture due to the highly available services that it provides. As it can be deduced from the huge amount of data that Facebook processes, its platform has strict operational requirements in terms of performance and efficiency. Cassandra was designed based on the design of Dynamo to offer all this features. However, the price paid to provide this in Cassandra is the lack of consistency guarantees that the system offers, since Cassandra only guarantees eventual consistency.

2.5.6 Discussion

Saturn and C^3 can be perceived as plugins to storage systems, contrasting with COPS, Eiger, and ChainReaction that are authentic storage systems, in addition to provide causal guarantees.

Our protocol can be used as a building block for systems that operate at the same level of Saturn and C^3, since we aim to implement a protocol that can be attached to systems

that do not have any mechanisms to track causality. With this in mind, we will deeply investigate each of the approaches, with special attention to Saturn and C^3.

2.6 Discussion

Throughout this chapter, we have been able to study many essential topics and, for each of them, we identified the most important concepts and designs to further explore in the next phase of this project.

In first section, we verified the usefulness of causality in consistency models, concluding that we will pursue causal consistency. Regarding overlay networks, we discarded structured approaches and targeted more interesting and scalable unstructured overlays. We proceeded by investigating significant properties and techniques of data dissemination and the main outcome was the identification of gossip techniques as a necessary ingredient to our solution.

Finally, we assimilated that full geo-replication has to be our approach and defined that Saturn and C^3 will be our main influences, in understanding how a causal dissemination primitive can be employed.

2.7 Summary

In this chapter we explored all the relevant topics regarding our research, inspiring us to pursue an attractive solution.

Modern distributed systems are increasing in number and size at a breathtaking pace. With this abrupt growth, their performance is depreciating and urgent solutions are required in order to maintain consistency guarantees combined with highly available services. With this in mind, we chose to focus on causal consistency models, since they are able to ally both demands. Therefore, our first concern was to present all the important concepts and definitions related with causality.

Besides causality, it is also crucial to deeply understand how to efficiently disseminate a message across an entire network. Consequently, we presented the generic methodologies regarding this wide topic, such as overlay networks and dissemination primitives.

We continued by describing some pertinent categories of replication in order to perceive what kind of approaches we should adopt, considering this theme. One of the categories we introduced is geo-replication, which is going to be our main target type of distributed systems. Hence, in our last section we presented a few geo-replicated storage systems that we might use to attach our solution and compare performances.

Chapter 3

Planned work

This chapter begins by specifying the crucial requirements that we need to display in the final solution (Section 3.1). We proceed by explaining the initial approach that we are going to follow in order to develop our solution (Section 3.2), and we continue by presenting the techniques that we plan to use to evaluate the solution (Section 3.3).

Lastly, we provide a schedule for the future work that we are planning to conduct (Section 3.4) and a final summary of this chapter (Section 3.5).

3.1 Requirements

Currently, there are some interesting approaches to implement causal dissemination. However, none of them ensure all of the following requirements with an adequate performance and low overhead.

- **Causal Dissemination:** The solution has to perform dissemination in a reliable and efficient way, enforcing causal delivery across the entire system;
- **Support for Highly Dynamic Networks:** Many nodes will join and leave the network constantly and, consequently, it is essential to build an agile protocol that supports these frequent changes on the system membership;
- **Low Metadata Overhead:** A large amount of metadata has to be kept in order to implement causal dissemination primitives, although our solution will seek to minimize this amount of metadata as much as possible.
- **Low Network Overhead:** The solution should strive to lower as much as possible the overheads, both in terms of network usage and CPU consumption, as to minimize the impact on applications operations on top of this protocol.

- **High Scalability:** The network has to be prepared to grow and keep the performance adequate with the number of nodes.

3.2 Planned Solution

From the study of the state-of-the-art, we identified many concepts and protocols that we will further explore. Additionally, we drafted the initial approach that we will chase in order to develop our solution.

We are projecting to design a protocol that combines two methods that were previously introduced: Plumtree and PRC-Broadcast.

Plumtree

Plumtree has several characteristics that are in agreement with the target protocol, namely: 1) gossip primitives, providing strong fault-tolerance and scalability as they have a efficient mechanism to deal with message loss and node failures, and 2) tree-based broadcast, which offers the possibility of having low metadata complexity and high scalability.

However, from the fact that a node failure triggers the *tree repair* process, which is a slow process, we deduced that is crucial to design an efficient solution to this particular issue, since our target network will incorporate a dynamic behavior with several nodes frequently joining and leaving. Additionally, changes in the tree topology disturb causality tracking mechanisms.

PRC-Broadcast

Plumtree already verifies a considerable part of our requirements. However, one of the main concerns would be missing: causal consistency guarantees.

With this in mind, PRC-Broadcast emerges as a viable starting point. It is a recent protocol that brings a novel implementation of causal broadcast, introducing a particular technique based on the ability to know how to identify and then forget the obsolete data that is being kept.

As we have recognized, this implementation significantly decreases the space complexity of the metadata that is kept in order to track causal dependencies, that is an essential aspect concerning the main requirements that we identified, namely low metadata overhead and high scalability. Therefore, integrating this technique into our solution, would be an attractive approach to achieve causal consistency.

3.3 Evaluation

Regarding the methodology that we will employ in order to evaluate our solution, we can divide it in the following phases:

- **Implementation Phase:** We will implement Plumtree, PRC-Broadcast, and a prototype of the solution in similar codebase in order to have impartial comparison criteria. We expect to deploy these implementations in a realistic distributed environment and supply them with a large number of dynamic processes, wrapping up the proposed environment.

- **Validation Phase:** Once the implementations are under the expected conditions, we will start an analytic procedure of tests and evaluations, which will lead us to understand the real potential of our solution. The main metrics that we will employ to judge their performances are: 1) number of active nodes in the network, 2) throughput, that is the number of operations executed per unit of time, and 3) latency, that is the average amount of time required to broadcast messages throughout the system.

As an extra, if we finish our proposed work with time to spare, we aim to attach our protocol to Cassandra. This goal comes from the fact that Cassandra is one of the most popular storage system that does not provide any kind of causal guarantees and hence, would be a perfect use case to test our solution. However, this would imply to integrate Cassandra with our prototype and, finally, allocate one more period for tests and evaluations.

3.4 Work Plan

In order to coordinate the work that we will have to accomplish, we have defined a Gantt chart with a proposed schedule (Figure 3.1) for the necessary tasks. The durations of the tasks were optimistically predicted in order to fit in the task related to the attachment of our solution with Cassandra. Besides this potential task, we defined three main tasks and their respective sub-tasks:

- **Solution Development**
  - Design
  - Implementation
  - Tests
- **Experimental Evaluation**
  - Implement Plumtree
  - Implement PRC-Broadcast
  - Compare performances
- **Integrating with Cassandra**

  - Study Cassandra's source code
  - Integrate our solution with Cassandra
  - Tests and evaluation

- **Writing**
  - Write dissertation
  - Write paper for Inforum

3.5 Summary

In this chapter we described the guidelines that we will follow in order to develop our protocol.

Firstly, we defined the main requirements that we will seek in order to build a robust causal dissemination solution for dynamic large-scale networks and then explained how we are planning to accomplish these requirements, merging two of the previously studied protocols, Plumtree and PRC-Broadcast.

Moreover, we clarified how we are going to judge the performance of our implementation employing two main phases: the implementation phase and the validation phase.

Lastly, we scheduled all the relevant tasks regarding the planned work, identifying three mandatory tasks: solution development, experimental evaluation and writing.

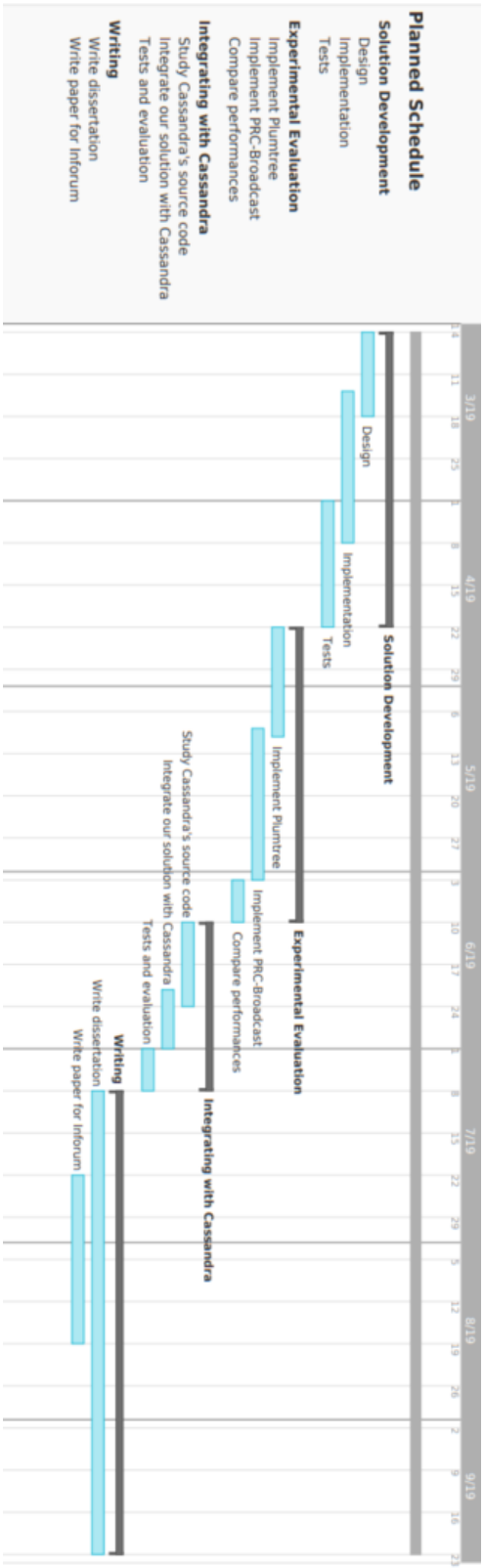

Figure 3.1: Gantt chart with planned schedule

Bibliography

[1] S. Almeida, J. Leitão, and L. Rodrigues. "ChainReaction: a causal+ consistent datastore based on chain replication." In: *Proceedings of the 8th ACM European Conference on Computer Systems*. ACM. 2013, pp. 85–98.

[2] M. Bravo, L. Rodrigues, and P. Van Roy. "Saturn: A distributed metadata service for causal consistency." In: *Proceedings of the Twelfth European Conference on Computer Systems*. ACM. 2017, pp. 111–126.

[3] E. A. Brewer. "Towards robust distributed systems." In: *PODC*. Vol. 7. 2000.

[4] V. Enes, P. S. Almeida, C. Baquero, and J. Leitão. "Efficient Synchronization of State-based CRDTs." In: *arXiv preprint arXiv:1803.02750* (2018).

[5] M. Ferreira, J. Leitao, and L. Rodrigues. "Thicket: A protocol for building and maintaining multiple trees in a p2p overlay." In: *2010 29th IEEE Symposium on Reliable Distributed Systems*. IEEE. 2010, pp. 293–302.

[6] M. J. Fischer, N. A. Lynch, and M. S. Paterson. *Impossibility of distributed consensus with one faulty process.* Tech. rep. MASSACHUSETTS INST OF TECH CAMBRIDGE LAB FOR COMPUTER SCIENCE, 1982.

[7] P. Fouto, J. Leitão, and N. Preguiça. "Practical and Fast Causal Consistent Partial Geo-Replication." In: *2018 IEEE 17th International Symposium on Network Computing and Applications (NCA)*. IEEE. 2018, pp. 1–10.

[8] A. J. Ganesh, A.-M. Kermarrec, and L. Massoulié. "Scamp: Peer-to-peer lightweight membership service for large-scale group communication." In: *International Workshop on Networked Group Communication*. Springer. 2001, pp. 44–55.

[9] S. Gilbert and N. Lynch. "Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services." In: *Acm Sigact News* 33.2 (2002), pp. 51–59.

[10] M. Jelasity and O. Babaoglu. "T-Man: Gossip-based overlay topology management." In: *International Workshop on Engineering Self-Organising Applications*. Springer. 2005, pp. 1–15.

[11] D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine, and D. Lewin. "Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web." In: *Proceedings of the Twenty-ninth Annual ACM Symposium on Theory of Computing. STOC '97*. El Paso, Texas, USA: ACM, 1997, pp. 654–663. ISBN: 0-89791-888-6. DOI: http://doi.acm.org/10.1145/258533.258660.

[12] A. Lakshman and P. Malik. "Cassandra: a decentralized structured storage system." In: *ACM SIGOPS Operating Systems Review* 44.2 (2010), pp. 35–40.

[13] L. Lamport. "Time, clocks, and the ordering of events in a distributed system." In: *Communications of the ACM* 21.7 (1978), pp. 558–565.

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

[15] J. Leitao, J. Pereira, and L. Rodrigues. "Epidemic broadcast trees." In: *2007 26th IEEE International Symposium on Reliable Distributed Systems (SRDS 2007)*. IEEE. 2007, pp. 301–310.

[16] 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)*. IEEE. 2007, pp. 419–429.

[17] J. Leitao, J. P. Marques, J. Pereira, and L. Rodrigues. "X-bot: A protocol for resilient optimization of unstructured overlay networks." In: *IEEE Transactions on Parallel and Distributed Systems* 23.11 (2012), pp. 2175–2188.

[18] J. Leitão, P. Á. Costa, M. C. Gomes, and N. Preguiça. "Towards Enabling Novel Edge-Enabled Applications." In: *arXiv preprint arXiv:1805.06989* (2018).

[19] 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*. ACM. 2011, pp. 401–416.

[20] W. Lloyd, M. J. Freedman, M. Kaminsky, and D. G. Andersen. "Stronger semantics for low-latency geo-replicated storage." In: *Presented as part of the 10th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 13)*. 2013, pp. 313–328.

[21] B. Nédelec, P. Molli, and A. Mostefaoui. "Causal Broadcast: How to Forget?" In: *The 22nd International Conference on Principles of Distributed Systems (OPODIS)*. 2018.

[22] A. Rowstron and P. Druschel. "Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems." In: *IFIP/ACM International Conference on Distributed Systems Platforms and Open Distributed Processing*. Springer. 2001, pp. 329–350.

[23] A. Rowstron, A.-M. Kermarrec, M. Castro, and P. Druschel. "SCRIBE: The design of a large-scale event notification infrastructure." In: *International workshop on networked group communication*. Springer. 2001, pp. 30–43.

[24] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan. "Chord: a scalable peer-to-peer lookup protocol for internet applications." In: *IEEE/ACM Transactions on Networking (TON)* 11.1 (2003), pp. 17–32.

[25] W. Vogels, L. Rodrigues, and P. Veríssimo. "Fast group communication for standard workstations." In: (1992).

[26] 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.