Tiago Miguel Ferreira da Costa

Licenciado

Access Control in Weakly Consistent Distributed Systems

Relatório intermédio para obtenção do Grau de Mestre em

Engenharia Informática

Orientadores: João Carlos Antunes Leitão, Professor Auxiliar Convidado,
NOVA University of Lisbon

Nuno Manuel Ribeiro Preguiça, Professor Associado,
NOVA University of Lisbon

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

February, 2016

Abstract

With the increased popularity of web application that run totally or partially in the client machines, new technologies have been developed based on the model of eventual consistency (also denoted weak consistency) to give users availability and low latency. This comes at the cost of possible temporary inconsistencies among replicas. In this context control access policies are themselves maintained with weak consistency and hence the system behavior might be hard to predict in some scenarios.

The objective of this work is to integrate access control to distributed storage systems that offer weak consistency, which are commonly used to support web applications, and to further extend this work for systems operating under a new hybrid model that brings together the cloud storage replication model with peer-to-peer communication on the client side, to provide lower latency and impose lower load on the centralized components.

Due to the use of eventual consistency, there is the need to study and define target access control semantics, and define algorithms to be able to implement the enforcement of such access control policies in these environments.

Keywords: Eventual Consistency; Access control; Replication; Cloud computing; Peer-to-Peer.

Resumo

Com o crescimento de popularidade de aplicações web que executam total ou parcialmente nas máquinas dos cliente, novas tecnologias tem sido desenvolvidas baseadas no modelo de coerência eventual (também chamado de coerência fraca) para providenciar aos utilizadores disponibilidade e menor latência. Este uso de coerência eventual tem um custo, sendo que desta forma existe a possibilidade de os dados estarem temporariamente inconsistentes entre replicas. Neste contexto, também as políticas de controlo de acesso são mantidas sob coerência eventual, o que torna difícil prever o comportamento do sistema em algumas situações.

O objetivo deste trabalho é de integrar controlo de acessos em sistemas distribuídos que oferecem fraca coerência, que são normalmente usados para suportar aplicações web, e ir mais longe e estender este trabalho para ambientes que incluem também comunicações peer-to-peer do lado do cliente e replicas adicionais nos clientes, que visam oferecer uma menor latência e sobrecarregar menos as componentes centralizadas.

Devido ao facto de ser usada coerência eventual, existe a necessidade de estudar e definir uma semântica de controlo de acessos, sendo depois também necessário desenvolver algoritmos que sejam capazes de fazer os sistemas respeitarem e garantirem as semânticas de controlo de acesso nesses ambientes.

Palavras-chave: Consistência eventual; Controlo de acessos; Replicação; Cloud computing; Peer-to-peer.

Contents

1 Introduction **1**

1.1 Motivation . . . 3
1.2 Problem Description . . . 4
1.3 Document Organization . . . 5

2 Related Work **7**

2.1 Access Control . . . 7
2.1.1 Access Control List (ACL) . . . 8
2.1.2 Role-Based Access Control (RBAC) . . . 8
2.1.3 Attribute-Based Access Control (ABAC) . . . 9
2.1.4 Divergence Issues for Weakly Consistent Replication . . . 9
2.2 Data Storage Systems . . . 11
2.2.1 Consistency . . . 11
2.2.2 Conflict Resolution techniques . . . 12
2.2.3 Examples Data Storage . . . 14
2.3 Peer-to-Peer . . . 16
2.3.1 Overlay Networks . . . 17
2.3.2 Example peer-to-peer overlay networks . . . 19
2.3.3 Access Control Basic requirements on a P2P system . . . 19
2.4 Summary . . . 20

3 Proposed Work **21**

3.1 Base Technologies . . . 21
3.1.1 Legion . . . 21
3.2 Proposed solution . . . 23
3.3 Work Plan . . . 24

Bibliography **27**

Chapter 1

Introduction

In recent years there has been an increase in the popularity of web applications. Taking a closer look, a large number of these applications are centered on users. By user centered, we mean applications in which users are both the major producers and consumers of content, and where the major role of the application is to mediate interactions between them. Facebook and Twitter are among some of the most popular examples, while other examples can be found in collaborative editing tools, such as Google Docs or the online Office 365, games, and chat systems such as the Facebook chat service.

With the incresed popularity of web applications, there has been an significant effort to improve browsers that run on client devices, as to allow the possibility of creating more powerful and interesting web applications. The improvements on web browsers made it possible for developers to go one step further, using features that until recently were only available to desktop applications, like multi-threading and direct access to local storage.

However, while these improvements allow to design richer web applications, these still rely on centralized components to provide access to information and to support all interactions between clients. This implies that even though when clients might be on close proximity and using the application to communicate between them, their interactions still need to go through the centralized component, which might be at a remote with high latency location, instead of having clients exchange information directly among them.

It has been demonstrated that it is essential to provide low latency for web applications, as failing to do so has a direct impact on the revenue of applications providers [8]. When there is a large number of users using a web service, which are scattered across the World, there will be latency issues for those that are further away from the centralized component, one simple example is a user from Europe trying to access a service in the United States. The path taken to reduce this issue was to resort to geo-replication, meaning that the service provider has its data distributed (by replicating or partitioning

data) across multiple data-centers, scattered throughout the world, so that content can be served to all users with the lowest possible latency. This implies that data is distributed through servers that are connected by high latency connections that migh not be realiable, meaning that, for instance, a service provider with clients in United States and Europe, would have servers in the United States and in Europe, as to lower latency for all users, independently of their locations. This ensures low latency and fault tolerance (as users can always failover to the other data-center if their local data-center becomes unavailable). There is however a new challenge that arises in this environment, formally captured by the CAP Theorem [16] , which states that it is impossible for a ditributed computer system to simultaneously provide simultaneously strong concistency (all nodes see the same data at the same time) and Availability (every request receives a response about wether it suceeded or not) in an environment where network partitions can happen tolerance (the system continues to operate despite arbitrary message loss or partial failure of the system or unavailability due to network partitions). Due to how the Internet works, partitions due to node failures have to be considered as they will eventually happen, so there is the question of which to sacrifice between strong consistency or availability. The common practice nowadays is to privilege availability, so most practical systems offer only some form of weak consistency, and in this work we tackle additional challenges that arise in this context.

A common used model of weak consistency is eventual consistency. This model guarantees that, if no updates are made to a given item, eventually all accesses to that item will return the last updated value [20] . In this model there may be times of inconsistency among the replicas of a data object, As it does not provide safety guarantees, an eventually consistent system can return any value before it converges. This allows these systems to, even during network partitions, to always serve operations over data. Eventual consistency may not be enough, consider the following example: user A writes to a page and user B answers, due to network latencies user C sees B's answer before A's initial post. This shows that while the consistency model is not violated, it can lead to unexpected behaviour from the standpoint of the user. On the other hand, this model provides high availability (every request receives a response about whether it succeeded or not).

Though many applications already use geo-replication to better serve data to its clients with low-latency, there is still a latency cost between nearby clients, as a round-trip to the server is still required.

Recent work has started to address the issue of minimizing the dependency on the centralized component for web applications. In particular, the approach described by [11] achieves this by allowing clients to exchange information directly, meaning that the dependency of a centralized component can be reduced.

This line of work leads to a paradigm shift, where the architecture of web applications changes by incorporating aspects of peer-to-peer systems. Each browser can access and exchange the information it needs from other clients without the need to interact with a server directly.

While these approaches (Geo-Replication and hybrid architectures) address some of the issues in centralized web applications, they also create new challenges, in particular in regard to security and privacy. We don't want applications to lose guarantees on data integrity and data privacy, so there is a need to address these issues in architectures that are becoming increasingly distributed and that operate under weak consistency models. Namely geo-replicated systems and hybrid architectures that combine geo-replication with direct communication between clients.

1.1 Motivation

In classical architectures, the server component is fully responsible for many correctness and security aspects of applications. In particular, a storage system is fully responsible for maintaning all data and ensure the durability of such data. Moreover, the server component mediates operations performed over this data, controlling potencial state divergences that might arise due to concurrent operations issued by users. Access control and integrity are also fully delegated to this centralized component in centralized architectures.

For Geo-Replication the same applies but with multiple centralized components. The data would be replicated between nodes and those nodes still need to be responsible for many correctness and security aspects of the applications. These correcteness aspects have to be maintained individually by each site, but also between them. Given the fact of the use of eventual consistency, state that exists in each replicas, including metadata structures used to perform access control tasks among others, will diverge. This is something that has to be addressed in order to provide access control and integrity of data.

Access control is what defines the permissions that something or someone has on a given object or resource. There are multiple models and implementations of access control, such as the access control list (ACL) in UNIX.

In a typical client-server architecture using with access control, in each request the access control is verified to check the permissions and evaluate if the request is valid. When a revocation of some user access and an attempt to update occurs, they must be handled in a sequential way, meaning that the update will be checked with the new access control changes.

In a Geo-Replicated system with eventual consistency, adding the access control layer, brings issues evaluating permissions. This happens because of the possible temporary inconsistency of metadata structures used to perform access control, since in one server the permissions may have been revoked and in the other the old permissions are still active allowing updates and other operations to be permformed when they should have been disallowed.

The peer-to-peer paradigm mentioned earlier will inherit the issues of the geo-replication since it also uses the multiple centralized components, but also allows for direct communication between clients. This communication will also need to have an access control

mechanism in place among all the clients, adding a new set of challenges to address.

1.2 Problem Description

As mentioned earlier the geo-replication of data already leads to a break up with strong consistency. In the classical client-server model, the updates were made on the same place meaning that multiple users would be updating data in a ordered way, where the first update to reach would be the one that would win, and data would be always consistent.

In a geo-replicated system with eventuall consistency this is no longer a reality, and with this adding a layer of access control becomes more challenging.

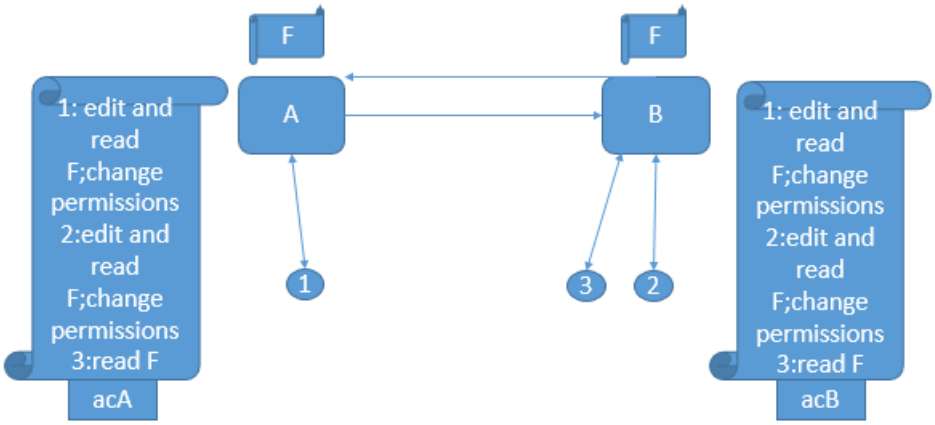

Figure 1.1: Geo-Replicated system with access control

Imagine the example given in Figure 1.1. We have two servers with replicated data and access control (A and B), and three clients. The two servers start with the same file F, being in this way consistent in data and also in the access control (acA, acB). With eventual consistency some issues may appear given the presence of the access control.

If client 1 removes the read permission to client 3, there will be the need for that request to arrive to B, but because of the use of eventual consistency there will be no guaranteee when that revogation will arrive, meaning that client 3 will still be able to see file F given the fact that the access control in B is outdated. This wouldn't happen on a single-server system since the server would take care of the order of actions and apply them.

In the previous case it wouldn't be problematic since 3 would only be able to see a file equal to the one that existed before when he had permissions, since no edit had been made, but it is possible to think of for example user 2 making changes to the document that user 3 shouldn't be able to see but because of the eventual consistency on the access control he will still be able to see.

This leaves us with a set of issues that occcur when there are concurrent writes, reads and changes on the permissions, in a geo-replicated environment with weak consitency. We can also assume malicious behavious, for example, client 3 already knows that his permissions have been revoked but he submits a request with a timestamp with a value such as to be before the actual change in permissions.

Adding communication between clients, the issues from geo-replication will be inherited and some more will be added, since now as data can also be present in the client side and updates shared between them, access control also has to be present in the client side. All of this needs to be maintained using eventual consistency, where updates from a user have to be propagated for other users and centralized components without creating issues related to the access control.

The focus of this thesis is to study how to address these challenges in the context of both geo-replicated systems and in systems that leverage an extention to the new browser-to-browser paradigm.

1.3 Document Organization

The remainder of this document will be organized as it follows

Chapter 2 described the related work. Existing access control models, where it is possible see different kinds os existing access control and see how a weak replicated system can affect the enforcement of access control policies, web based data storage, where is described different consistency models and ways to solve some consistency issues while also giving existing exemples, and also existing Peer-to-Peer tecnhologies where is described how peer-to-peer works.

Chaper 3 describes the work plan for the elaboration of this thesis. It describes a closer approach to the problems that we want to identify.

Chapter 2

Related Work

This thesis addresses the challenges of enforcing access control to eventual consistency distributed data systems, and extending such solutions to distributed data storage systems that leverage on peer-to-peer communication allowing clients to communicate directly. The following sections cover the main aspects of these fields, in particular:

Section 2.1 existing access control models are discussed and compared.

Section 2.2 web based service providers are discussed, in particular addressing the challenge of storing and accessing data, and maintaining that data given the tradeoff between consistency and availability.

Section 2.3 existing peer-to-peer technologies are presented and compared. ...

2.1 Access Control

Access control is a mechanism for a system to know and enforce the execution of a given action by some entity (usually a user or a program acting in the behalf of a user) is allowed. In the context of access control usually one refers a **principal** as someone (or something) that interacts with a system through actions, and a **resource**, as an object that the principal manipulates through the execution of actions.

Access control mechanisms and policies keep track of which principals can access existing resources in the system, for example if I want to access a file on my computer it is the access control that will verify if I am able to access that file and what type of actions can I perform over it (e.g. read, write, execute).

In the domain of access control, one can also think in terms of policies. Policies are statements that will be evaluated to check if the user has access and can vary in complexity, they can either be small and evaluate only one attributed like "user has role X", or more

complex with more attributes, where for example "user has role X, is in local Y, in date Z".

2.1.1 Access Control List (ACL)

An access control list (ACL) is the most simple materialization of access control, it fundamentally relies on a list of permissions attached to each individual resource. This list is a data structure containing entries that specify individual users or groups (Principals) and explicit permissions to that specific object (Resource). It is basically a list with entries that defines the permissions that apply to an object and its properties. This allows a fine-grained access control over the permissions of principals.

When access control lists were first introduced, they were more effective as systems had a low number of principals where each principal had different rights. Modern systems evolved to have a large number of users, leading to a high amount of entries. If there was the need to make changes in various principals one would have to change each one of them individually. Nowadays it is possible to join users in a group and treat that group as a principal, assigning or revoking rights to all elements (i.e. users) of the group at once.

In an ACL implementation it is easy to find the set of all principals who may read a file, but it might be difficult to find the set of all files that a subject may read. This is because the access control list is stored by each resource containing the principals and their rights to that file individually.

2.1.2 Role-Based Access Control (RBAC)

In large scale systems, security policies are dynamic. Access rights need to change as the responsabilities of principals change. This can make management of rights difficult, as when a new user joins the system, the appropriate rights for that user must be established, and when a user changes job functions, some rights should be removed, while others have to be added. In a broad example, we can think of an hospital, where there are multiple doctors, nurses, chief executives, patients, etc... Each doctor should have the same access rights as the others, as well as nurses among them. If a nurse changes functions to doctor, there is the need to change all of her permissions so that they are the same as the remaining doctors. This increases the complexity on changing rights given a multiple number of principles where most of them are in theory in real world function groups where all should have the same permissions.

Role-based access control addresses this problem by changing the underlying principal resource model. In Role-Based Access Control, principals are classified into named roles. A role is a set of actions and responsibilities associated with a particular working activity. Intead of an access control policty being a relation on principals, resources, and rights, a policy is a relation on roles, resources, and rights. For example, the role "nurse" might be assigned the right to see the files of patients. Principals are now assigned to roles, where each subject may be assigned to many roles and each role may be assigned to

many principals. Roles are also hierarchical, for example, the role "doctor" should have all the rights that a "nurse" does, and more.

2.1.3 Attribute-Based Access Control (ABAC)

Attribute-Based Access Control is an access control technique where the principals requests to perform an operation on given resources are granted or denied based on assigned attributes to the principal, assigned attributes of the object, environment conditions, and a set of policies that are specified in terms of those attributes and conditions. This might be a more logical way to perform access control, since there is the evaluation of rules against attributes to check the permissions. In Figure 2.1 it is possible to see a simple ABAC scenario, where the decision to give or deny access to an resource (object) is given by having in consideration the subject (principal) attributes (2b), the object (resource) attributes (2c), the access control policy (2a) and environment conditions (2d).

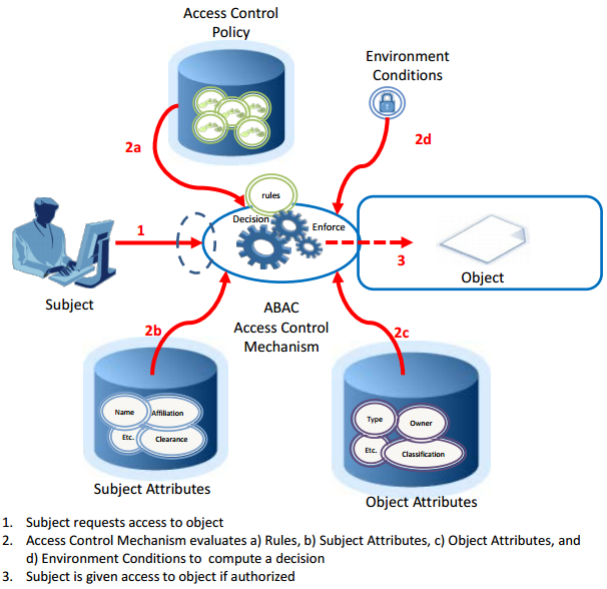

Figure 2.1: ABAC Scenario [5]

It is possible to look at ABAC as a broader access control technique when compared to ACLs and RBAC, since ACLs work on the attribute of "identity" and RBAC on the attribute "role", while ABAC is the concept of policies that express complex rules that can be evaluated over many different attributes.

2.1.4 Divergence Issues for Weakly Consistent Replication

When we are using a system with replicas that relies on weakly consistent replication some problems may arise. If the authorization policy can be temporarily inconsistent, any given operation may be permitted at a particular node and yet denied at another, and without a careful design, permanently divergent state can be a result of such a system.

This issue is more serious when access policies are being modified, as the probability for divergence is higher during such periods.

We can have a system where all the nodes trust each other (for instance they are all controlled by the same entity), in this case the access control policy for allowing an update U can be enforced independently by each node, even though there might be transient variants of the policy in each of the nodes. Because the nodes trust each other to enforce policies, they will never permanently disagree about which operations have been accepted, since once an update is admitted, no further checks are required, although it is possible that the most recent policy is not used for the access control decision of some operations. However one can assume thar such nodes will not disregard the current policy to allow a user to execute some operation that was not allowed.

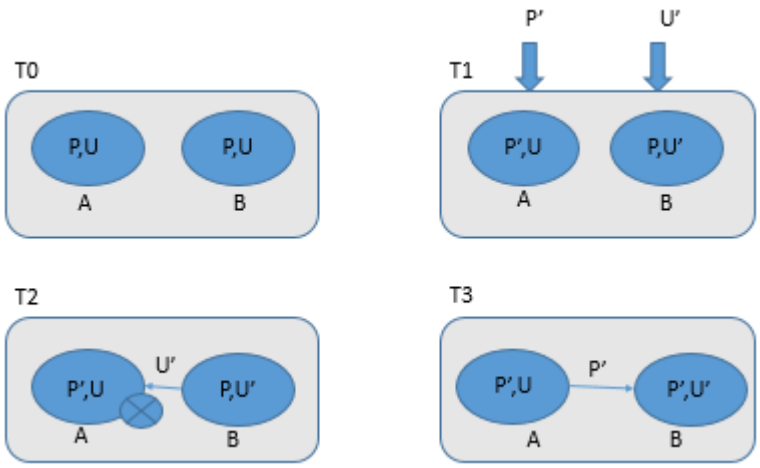

Figure 2.2: Concurrent policy (P') and data (U') updates leading to permanent inconsistency

In a distrustful system (i.e, a system where the indivual components might belong to different administrative domains, and hence might deviate in the execution of a ditributed protocol), there is the need to explicitly address consistency issues, as ilustrated in Figure 2.2. In this example we start at T0 with two replicas (A and B) with the same Access control policy (P) and the same data (U). In T1, two concurrent updates, one of policy (P') in replica A and another of data (U') in B are issued. Assuming that the control policy P allows the update U' and that policy P' doesn't allow it. In T1 it will be possible to change the data to U' in B since the local control policy P allows that operation over that data item, while in A the policy changes from P to P'. In T2 there will be an attempt to propagate data U' to the node A, but since P' doesnt allow the update U,' node A won't acept the update. In T3 A will propagate its control policy to B, meaning that A will stay with (P',U) while B with (P',U'). This leads to a scenario with a permanent state of inconsistency.

2.2 Data Storage Systems

As discussed previously, data storage for web services commonly resorts to geo-replication, as web based services benefit from storing client data on geographically distributed data centers, to provide a lower access latency, improved bandwith, and availability.

There is however a challenge that arises in this environment, which has been captured by the CAP theorem. The CAP theorem, as mentioned in the Introduction of this document, states that it is impossible for a ditributed computer system to simultaneously provide strong consistency and availability in an environment where network partitions can happen. This implies that a geo-distributed system must either sacrifice availability or strong consistency.

2.2.1 Consistency

Strong consistency: In a strongly consistent system, if there is a write operation that terminates, the next sucessfull read of that key is guaranteed to show that write, meaning that a client will never see out-of-date values. The problem with the strong consistency is the trade-off it makes with availability, since a distributed system providing strong consistency may come to a halt if nodes become unavailable due to a fault or a network partition. This may let the service unavailable for the user during a long time, resulting in a bad user experience. Strong consistency however minimizes the inherent challenges related with dealing with consitency of data, since data will be consistent across all the sites.

Eventual consistency: An eventual consistency system is used to achieve a high availability. The system guarantees that if no new updates are made to a given data item, eventually all accesses to that item will return the value of the last update [20]. In this case there may be times of inconsistency among the replicas of a data object, since it does not make safety guarantees, an eventually consistent system can return any value before it converges. This enables these systems to, even during network partitions, always serve read and write operations, following the CAP theorem sacrificing the consistency and promoting availability.

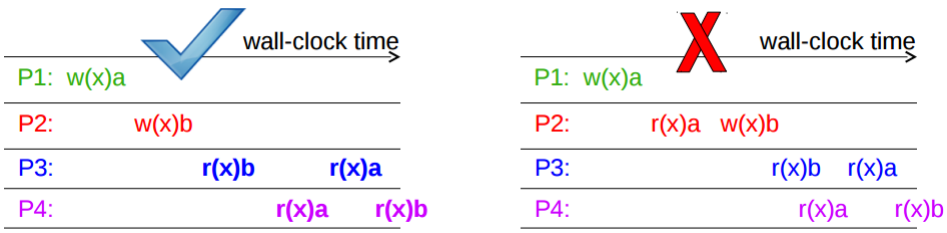

Figure 2.3: Causal consistent vs non causal consistent [10]

Causal consitency: A system provides causal consistency if any execution is the same as if all causally-related read/write ops were executed in an order that reflects their causality. Concurrent operations that are not causally related may be seen in different orders by different nodes. When a node performs a read followed by a write, the first operation is said to be causally ordered before the second, because the value stored by the write may depend on the result of the read operation. Two write operations performed by the same client are also considered causally related in the order they were performed. In Figure 2.3, we have a representation of a causal consistency system on the left, and on the right a system that is not causally consistent since, w(x)b is causally-related on r(x)a on P2, which is causally-related on w(x)a, therefore it must be enforced w(x)a w(x)b ordering, but P3 violates that ordering.

Using eventual and causal consistency comes with the cost of state divergence, since only strong consistency guarantees data consistency (i.e, no divergence) at all times. Since there may be some cases of state divergence, some conflict resolution techniques must be used, such as the ones discussed in section 2.2.2.

2.2.2 Conflict Resolution techniques

As mentioned earlier, applying weaker models of consistency may leave the the system in a state of divergence. In order to ensure replica convergence, a system needs to exchange versions or updates of data between servers (anti-entropy) and choose the appropriate final state when concurrent updates have ocurred (reconciliation). For a system to be able to return to a point of consistency across all replicas, some conflict resolution techniques can be employed, including:

Last Writer Wins: In the last writer wins technique the idea is that the last write based on a node's system clock will override and older one. This is trivial using a single server, since it only needs to check when the writes came and apply them in order, but using multiple nodes where clocks may be out of sync may be an issue. Choosing a write between concurrent writes in this case can lead to lost updates.

Programatic Merge: Programatic merge consists in letting the programmer decide what to do when conflicts arise. This conflict resolution technique requires replicas with to be intrumented with a merge procedure, or to expose diverging states to the client application which then reconciles and writes a new value. With this technique the final write will always be the one decided by the programmer, meaning that the most important data for the programmer will be kept.

Commutative Operations: Commutative operations are as the name hints, operations where changes in the order will not change the final result. If all operations are commutative, conflicts can be easily solved since, independently of the order of when the operations are received (and applied) in each node, the final result (i.e, state) will always be the same. Commonly used techniques based on the commutativity of operations are:

OT, Operational Transformation. OT was originally invented for consistency maintenance and concurrency control in collaborative editing of plain text documents. The idea of OT is to transform parameters of excuted operations so that the outcome is always consistent. Given a text document with a string "abc" replicated at to sites with one user on each site, and two concurrent operations where user 1 makes a request of inserting character "x" at position "0" (O1) and user to makes makes a request for deleting the character "c" at position "2" (O2). This request will reach first the site they are using, and because of latency only after some time be propagated to the other server. Because of this, in site 1 we will have first operation O1 executed, and only after O2, while the opposite will happen on in site 2. With this the result in site 1 will be "xab", while on site 2 the result will be "xac", staying in a state of divergence. Using OT we will be tranforming the operations to solve this problem, the delete is trsnformed to increment one position and the insert can reamin the same. Both outcomes become "xab", independently of the order in which operations are applied.

Operational Transformation has been extensively studied, especially by the concurrent editing community, and many OT algorithms have been proposed. However, it was demonstrated that most OT algorithms proposed for a decentralized OT architecture are incorrect [13]. It is believed that designing data types for commutativity is both cleaner and simpler [15].

CRDT, Convergent or Commutative Replicated Data Types. CRDTs are replicated data types that follow the eventual consistency model. An example of a CRDT is a replicated counter, which converges because the increment and decrement operations commute naturally. In these data types, there is no need to synchronisation, an update can execute immediatly and return the reply to the client, unaffected by network latency between the replicas of the data object. The replicas of CRDT are guaranteed to converge to a common state that is equivalent to some correct sequential execution by design [15]. CRDTs can typically be divided in two classes:

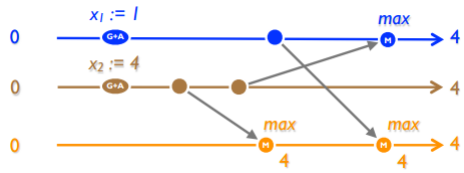

Figure 2.4: CvRDT: integer+max [15]

CvRDT, state-based Convergent Replicated Data Type. In state-based replication, an update occurs entirely at the source, and only after the local execution, there is syncronization between replicas. This syncronization is achieved by transmitting the full state of the CRDT (value and any internally maintained metadata) between replicas. It is possible to see in the example in Figure 2.4, where we have a CvRDT integer with a max fucntion, the update if first made at it's origin and then the whole resulting object (the integer) is sent to the replicas where it will converge, giving us the same final state across all replicas.

CmRDT, operation-based Commutative Replicated Data Type. CmRDT is based on operation commutativity, in this case the operation is executed at the source and after this execution, that operation is propagated to all the remaining replicas. This is possible because in the operation-based class, concurrent operations commute. Operation-based replication requires reliable broadcast communication delivery with a well-defined delivery order, such as causal order between operations. This is important to avoid a replica to evolve to states that shouldn't exist.

In summary we can differentiate CvRDT and CmRDT by the information that is propagated between their replicas. In the case of CvRDT the update is applied and the object that results from executing that function is then sent and merged, while on CmRDT it is the function that is propagated. A simple example could be asking someone to raise their hand, in the CvRDT type we would apply the function of raising someone hand and then send the entire object person (with the hand raised) to merge in the other replicas, while in the CmRDT type we would just send the function telling to raise the hand. ...

2.2.3 Examples Data Storage

Spanner: [2] is a scalable, multi-version, globally-distributed, and synchronously-replicated database. It is a system that provides strong consistency, using the Paxos algorithm as a part of its operation to replicate data across hundreds of data centers. Spanner automatically reshards data across machines as the amount of data or the number

of servers changes, and it automatically migrates data across machines (even across datacenters) to balance load and in response to failures. It makes use of hardware-assisted time synchronization using GPS clocks and atomic clocks to ensure global consistency.

One server replica is elected as the Paxos leader for a replica group, that leader is the entry point for all transactional activity for the group.

All transactions in Spanner are globally ordered as they are assigned a hardware assisted commit timestamp. The timestamps are used to provide multi-versioned consistent reads without the need for taking locks. A global safe timestamp is used to ensure that reads at the timestamp can run at any replica and never block behind running transactions.

As said before, Sapnner has strong consistency and timestamp semantics, providing a scalable, multi-version, globally distributed, and synchronously-replicated database.

Dynamo: [3] Dynamo is a highly available key-value storage system. To provide this level of availability, Dynamo sacrifices consistency under certain cenarios using object versioning and application-assisted conflict resolution, in a manner to expose data consistency issues and reconsiliation logic to the developers. Data is partitioned and replicated using consistent hashing. The consistency among replicas during updates is maintained by a quorum-like technique and a decentralized replica synchronization protocol. Dynamo employs a gossip based distributed failure detection and membership protocol. Dynamo is a completely decentralized system with minimal need for manual administration. Storage nodes can be added and removed from Dynamo without requiring any manual partitioning or redistribution. A write operation in Dynamo also requires a read to be performed for managing the vector timestamp associated with each version of data object. This is can be very limiting in environments where systems need to handle a very high write throughput.

Cassandra: [9] Cassandra is a distributed storage system for managing very large amounts of structured data spread out across many commodity servers, while providing highly available service with no single point of failure. Cassandra can support a

very high update throughput while delivering low latency, not sacrificing read efficiency. Typically a read/write request for a key gets routed to any node in the Cassandra cluster. The node then determines the replicas for this particular key. For writes, the system routes the requests to the replicas and waits for a quorum of replicas to acknowledge the completion of the writes. For reads, based on the consistency guarantees required by the client, the system either routes the requests to the closest replica or routes the requests to all replicas and waits for a quorum of responses. Cassandra partitions data across the cluster using consistent hashing [6] but uses an order pre-serving hash function to do so. Cassandra uses replication to achieve high availability and durability. In Cassandra there is a factor N which is the replication factor configured "per-instance", that will indicate that each data item is replicated at N nodes. Each key, k, is assigned to a coordinator node (described in the previous section). The coordinator is in charge of the replication of the data items that fall within its range. In addition to locally storing each key within its range, the coordinator replicates these keys at the N-1 nodes in the ring. Cluster membership in Cassandra is based on Scuttlebutt [19] , a very efficient anti-entropy Gossip based mechanism. For failure detections, Cassandra uses a modified version of the Accrual Failure Detector [4], where instead of the failure detection model just emiting a Boolean value stating that the node is up or down, it emits a value which represents a suspicioun level for each of monitored nodes. Cassandra is a completely decentralized system.

Riak: [7] is a distributed NoSQL key-value data store that supports high availability by giving the flexibility for applications to run under strong or eventual consistency, using quorum read and write requests and multi-version concurrency control with vector clocks. Eventual consistency in Riak uses CRDTs at its core, including counters, sets, flags, registers, and maps. Partitioning and replication is done via consistent hashing using a masterless approach, thus providing fault-tolerance and scalability. The built-in functions determine how replicas distribute the data evenly, making it easy for the developer to scale out to more nodes.

2.3 Peer-to-Peer

A peer-to-peer system consists in a distribution of tasks among peers (nodes) where tasks are dynamically allocated. A peer-to-peer system has a high degree of decentralization

where peers implement both client and server funcionality to distribute bandwidth, computation, and storage across all the participants and few or none dedicated peers exist in the system that own global state [14].

Once a peer is introduced into the system, there is little or no need for manual configuration. Peers are usually owned and operated by independent individuals who voluntarily join the system and are not controlled by a single organization.

Peer-to-peer also requires little or no infrastucture, usually the cost to deploy a peer-to-peer service is low compared to client-server systems. Peer-to-peer systems also exhibit an organic growth because the resources are contributed by participating nodes, meaning that a peer-to-peer system can grow almost arbitrarily without the need to upgrade the infrastructure, for example, replacing a server for a better one as is common pratice when there is an increase in the number of users in a client-server system. This is because with each new node that joins, the systems increases in the amount of total available resources.

There is also the resilience to faults and attacks, since there are few (if any) nodes dedicated that are critical to the system's operation. To attack or shutdown a P2P system, there is the need to attack a large portion of nodes simultaneously, where with each new node joining the system an attack becomes harder to deploy.

Popular peer-to-peer applications include sharing and distributing files (like eDonkey or BitTorrent [22]), streaming media (like PPLive [21] or Cool Streaming [24]), telephony and volunteer computing.]

2.3.1 Overlay Networks

The network topology of the underlying network has a high impact on the perfomance of peer-to-peer services and applications. Therefore, it is essential to rely on an adequate overlay network for supporting systems in the right way. An overlay network is a logical network of nodes on top of the underlying physical network. It can be thought as a directed graph G = (N,E), where N is the set of participating nodes and E is a set of overlay links.

To achieve an efficient and robust delivery of data through a peer-to-peer system there is the need to construct an adequate overlay network. For this, the fundamental Architectural choices are the degree of centralization (partly decentralized vs decentralized) and the topology of the network overlay (structured vs unstructured).

Degree of centralization We can categorize Peer-to-Peer networks architectures by their use of centralized components.

Partly centralized networks resort to some dedicated node or use a central server to perform some special control task such as indexing available resources or to provide a set of contact for nodes to join the system. New nodes can then join the overlay network by connecting to the controller. This overlay starts as a star-shapped because of the

commnunication to the centralized unit by the participants and aditional overlay links are formed dinamically among participants that have been introduced to the controller. With this approach we get some of the downsides of a centralized unit, such as the existence of a single point of failure and bottleneck. Having this in mind, this system is not as reliable and resistent as a fully decentralized architecture. Still it provides organic growth and abundant resources, that are relatively simple to be managed via the single controller. Examples include Napster [12], Skype (and old version) [1], and BitTorrent using trackers.

Decentralized P2P networks do not use any form of dedicated nodes or centralized components that have a critical role in the operation of the system. In this type of overlay network, nodes that are joining the system are expected to obtain, through an outside channel, the network address of one of an already participating node in the system. This makes the decentralized P2P more reliable compared to the partly decentralized system, avoiding the single point of failure and bottleneck issues, and increasing the potential for scalability and resilience. In this type of architecture some nodes may be used as supernodes having increased responsabilities. A node bacomes a supernode if it has a significative amount of resources, high availability, and a publicly routable IP address. Supernodes can increase the efficiency of a P2P system, but may also increase its vulnerability to node failure.

Structured vs Unstructured Overlay: There is also the need of choosing between structured and unstructured overlays. This decision depends mostly on the usefulness of key-based routing algorithms and amount of churn (that is, when large numbers of peers are frequently joining and leaving the network at the same time) that the system is expected to be exposed to under operation.

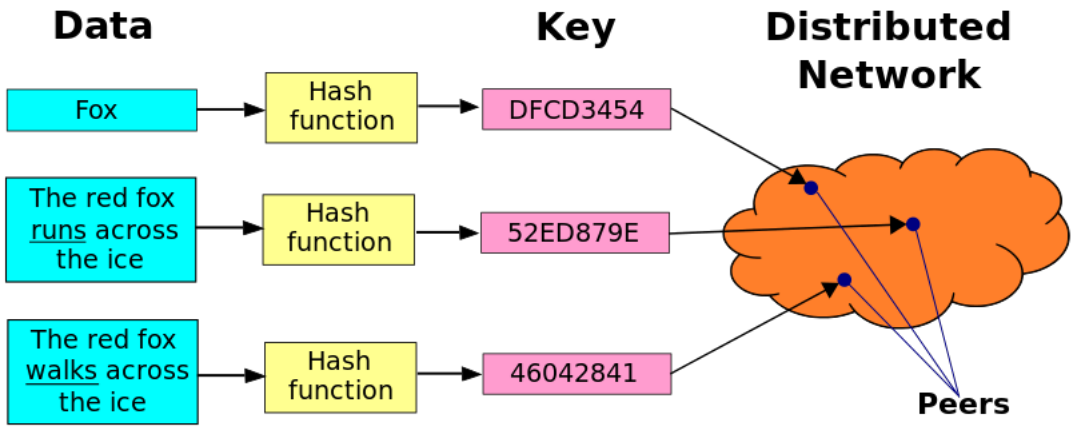

Figure 2.5: Representation of a Distributed Hash Table [23]

Structured overlays: In a structured overlay typically, each node gets an unique identifier in a large numeric key space, where the identifier will determine the position of the node in the overlay structure. Identifiers are chosen in a way that peers will be distributed uniformily at random over the key space. This allows to create a structure called DHT (Distributed Hash Table, Figure 2.5), that provides a lookup service similar to an hash table, where any participating node can retrieve the value associated with a given key (in the same space as the node identifiers). The responsability for maintaning the mapping from keys to values is distributed among the nodes, in such a way that a change in the set of participants causes a minimal amount of disruption. In a structured overlay there is a trade-off, since it gets more efficient queries but at the cost of poor performance when churn is high.

Unstructured overlays: In an unstructured overlay there is no particular topology formed by the network links and queries are usually done by flooding the network. Unstructed overlay networks are formed by establishing random links between the participants of the peer-to-peer system. In an unstructured overlay we get the oposite of what we get in a structured overlay, since we get less efficient queries having to flood the network to find as many peers as possible to locate a particular data object, resulting in the peers needing to process all the search queries, but in return we get a more robust system when the churn is high.

2.3.2 Example peer-to-peer overlay networks

Chord [17] is distributed lookup protocol that made to enable peer-to-peer applications to efficiently locate the node that stores a particular data item. Chord provides support for just one operation: given a key, return the nodes responsible for the key. Keys are distributed over the nodes using consistent hashing and replicated over succeeding nodes. Nodes typically store their successor nodes, forming an ordered ring, making it easy to reason about the overlay structure. For fault-tolerance a list of successor nodes is kept and for efficient lookups a finger table, shortcuts to nodes over the graph, is used to perform lay jumps in the ring topology.

2.3.3 Access Control Basic requirements on a P2P system

A peer-to-peer system is different from other systems where we are more used to see access control being applied. To apply access control on a peer-to-peer some guarantees need to be made. In this case we will describe four main requirements that an access control model for P2P file-sharing networks should support [18]:

No centralized control or support: Traditional access control models, generally rely on central servers for authorization operations. This allows the existence of a single central locationwhere the policies can ve store and evaluated. In a P2P network this

doesnt happen, in fact, a peer has a significant level of autonomy and is in charge of storing and managing its own access control policies.

Encourage sharing files: One of the characteristics of P2P networks is the anonymity of the peers. Unlike client-server systems, peers in P2P systems are typically loosely coupled and provide very little information about their real-world identities. The ineractions is done by peers that are mostly unknown. A P2P access control model must provide a mechanism for a host to classify users and assign each user different access rights, even if the users were previously unknown.

Encourage sharing files: Given the fact we are taling about a file-sharing P2P network, we know that users join the networkd for its availability and richness of files. Implementing an access control system can reduce the chance that users will get their desired files. This way, access control for P2P should attempt to minimize this problem.

Limit spreading of malicious and harmful digital content: The open and unknown characteristics of P2P make it a good everinmoent to malicious spreading and harmful content. A P2P access control system should support mechanisms to limit such malicious spreading and punish the ones responsible for it.

2.4 Summary

This chapter discussed previous work in the areas related to the development of this dissertation.

In the peer-to-peer context the need for an overlay network has been described, explaining that different application requirements can require different types of overlays. Overlays can generally be described by degree of centralization and structured vs unstructured.

In the data-storage context we talked about leveraging strong consistency with high availability. Different consistency models have been explored and, in the case of eventual consistency, several techniques for conflict resolution have been described.

In the collaborative-editing context, various commonly used approaches have been explored, describing how concurrency is handled in real-time editing in each of them.

In the next chapter the work plan for the elaboration of this thesis will be described, starting off with the technological basis over which this work will be developed.

Chapter 3

Proposed Work

This chapter will firstly describe Legion, a framework that will be leveraged for performing part of the work proposed in this document. In the second section a brief oeverviw at the solution is presented, focusing on what are the main objectives. In the final section the work plan for the remainder of this work is described.

3.1 Base Technologies

This section covers some prior work that will serve as a starting point to some of the work in access control to be tackled in the context of this dissertation. The main challenge tackled in this work is the integration of access control on eventually consistent data replication systems, we will build upon systems that already exist. In a first stage we will use a geo-replicated storage system where the replicas are trustfull like Cassandra (expalined in the section 2.2.3). This first stage will allow us to address the main challenges that arise due to eventual consistency. In a second stage: we will extend our mark to a scenario where data object replicas also exist in the client side, to do so we will leverage Legion, a system that uses p2p client communication between clients where those they are not mutually trustfull. This second stage will enable us to tackle additioal challenges that arise both due to the fact that not all nodes that pocess operations are trustful, but also due to the increase in scale, i.e., scenarios with a large number of replicas for te same tem. From completness, we now provide a brief description of Legion.

3.1.1 Legion

As we have seen before, systems that worked as client-server service have evolved to multiple servers distributed, and recently to incorporate peer-to-peer communications, that allowed to provide users with lower latency and enriched availability.

Legion is a hybrid framework, that provides data replication using peer-to-peer communication between clients, alowing to get even lower latency to the client and lowering the load on the centralized components. Given recent thecnological advances, especially in browsers and with the emergence of HTML5, it was possible to develop a framework where direct communication between web clients is supported transparently. The objective of this framework is to reduce as much as possible the interaction that clients need to have with the centralized component, since there is the oportunity for the clients to directly exchange information between them. With this we have web applications that can benefit from lower communication latency and that lowers the load imposed on the centralized component, but also increases the availability of the services, since even if the centralized component becomes temporarily unavailable, clients can still use the service by communicating directly between them, without the need to resort to the centralized component to mediate the interactions between clients.

This is illustrated in Figure 3.1, where it is possible to see the previous client-server model at the left, and the legion hybrid framework on the right that uses centralized components but also leverages on direct peer-to-peer communication between clients. The need for the centralized components still exists, even though if the centralized components are temporarily unavailable the service remains available(to those clients that we already using the service). However, the centralized component is still a key component of the architecture, as it serves as an entry point for clients, but also ensures data persistence (if all the client nodes fail or leave at the same time, data would be lost otherwise). Another relevant aspect of the centralized component, is that it is essencial to allow clients that run in environents that are incompatible with Legion (for instance due to extremely restrictive firewalls) to access the service.

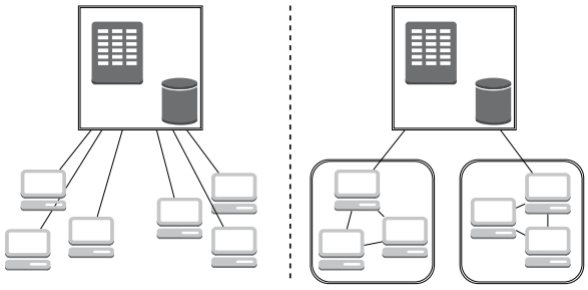

Figure 3.1: Client-server model (on the left) | hybrid model (on the right) [11]

As availability is one of the goals of Legion, the framework was designed by promoting the use of weak consistency instead of strong consistency (this is well aligned with the observations made by the CAP theorem).

In Legion objects are replicated between clients using the appropriated CRDTs for the data type required by the application. CRDTs allow to minimize coordenation between clients when executing operations on their local replicas. Also, these data types allow for

easy convergence of replicas whose local state has diverged. They also allow clients to aggregate various operations (potentially from various neighbours) into a single message to be propagated to the central component, further creating the opportunity to minimize load on this component.

3.2 Proposed solution

As the work focuses on providing access control services to distributed data storage architectures that already exist, and that resort to eventual consistency, the first step is to come up with sensible semantics for the behaviour of the system where a policy is modified concurrently whith and operation affected by the policy change. We then plan to integrate access control mechanisms on multiple systems with weak consistency, departing from an architecture where a level of mutual trust exist among nodes that host data and after this, moving to architectures that follow the design of Legion, where replicas also exist in clients, that might not mutually trust each-others. Note that in this second case it is also expected that the number of replicas for each data object is much greater. In the second case special care has to be taken to avoid adding too much overhead or too much communication with the centralized (trusted) components, since that would lead to lose the benefits provided by the Legion framework.

In summary, the work to be done in the context of the dissertation is organized in three phases:

First phase is to define the semantics that the control access mechanisms should provide. Defining the semantics will lead us to develop algorithms to enforce these semantics across multiple scenarios that can arise in both distributed architectures discussed above. In this case we will have to address the challenges introduced by eventual consistency.

Second phase: of this work, is to integrate algorithms that enforce the semantics defined before in a geo-replicated storage system. In this case we will use a storage system, for example Cassandra (that has been described in section 2.2.3), and modify it to include access control mechanisms. In this context we have to tackle particular challenges, such as, the mobility of users across replicas, since if clients can change servers when the access control policies can be inconsistent, users may end up using that to issue operations when they shouldn't be allowed (and they might be awae of tha already). This is an issue since it is not possible to make it fully impossible for clients to move between servers, as that would render the system unavailable in the face of failures. Ho to address this challenge is still an open research question.

Third phase: In this phase we plan to extend the work of the two previous phases to tackle the new hybrid model that combines the ditributed geo-replicated servers with a peer-to-peer infrastructure between clients which is used to have additional

replicas of data objects, where operations can also be issued. For this we will be using the framework descibed in the beggining of this chapter, Legion. In this system we will have nodes that are not trustfull meaning that it is possible for these nodes to deviate from the probocols being used (for example, changing the time of an update so that it is done before a given new policy starts to be enforced). This makes it hard for a parcial order to exists, since we are dealing with a system that is not mutually trustfull.

The objectives are first of all giving guarantees that the access control works in the way we wanted and that it respects the semantic designed in the first part of this work, and also enforcing such access cntrol semantics with low overhead and minimal interference on the systems operation.

Evalutation: We plan to implement our solution leveraging an existing geo-replicated storage system and leveraging the existing Legion prototype. This will allow us to conduct an experimental evaluation by deploying our prototypes and the original sstems on which our implementation are build upon (for baselines) to measure the overhead (in terms of CPU,communication, and latency) introduced by our access control mechanisms.

Additionally, and to demonstrate that our algorithms really enforce the desired access control semantics n the target systems, we plan to use formal methods. This can be achieved by either writting formal correctness proofs, or resortin to full apecification of our semantics, and solutions to use TLA+ to automatically check correctness.

3.3 Work Plan

In this section, the work plan for the elaboration of the dissertation is descibed. The work plan is to begin with defining the access control semantic, since that will be the basis to our work on the following two phases. Next is the design and implementation of the access control enforcement mechanisms that will be divided in two parts:

The first part dedicated to a geo-replicated storage system such as Cassandra; and a second part dediated to a system enriched with direct client interactions such as Legion.

After these two parts, there will be a final evaluation phase, whose goal is to both demonstrate the correctness of our solution and experimentally evaluate the overhead introduced by our mechanisms. The work is to be done roughly in seven months. Table 3.1 depicts the start and end dates as well as the duration of each task.

Define Semantics. The main task is to define the access control semantics that will be tacled by our work. It is expected that some refinement are to be done when unexpected challenges are encountered (in subsequent phases).

Table 3.1: Calendar

Task	Start Date	End Date	Weeks
Define Semantics	29 February	25 March	4
Design and Implementation	7 March	24 June	16
Phase 1	7 March	6 May	9
Phase 2	18 April	24 June	10
Final Evaluation	27 June	29 July	5
Writing	11 July	23 September	11

Implementation of the system, this task can be divided into two phases, which will partially overlap, as the solutions for one pahse may also be used in the second phase. The phases are as follows:

- Phase one - Design and Integration of access control in a geo-replicated storage system.
- Phase two - Design and Integration of access control in Legion.

Final evaluation. In this task the implemented solutions will be more thoroughly evaluated as a whole. The knowledge obtained from this task will help us decide what improvements have to be made and how to implement additional features.

Writing, the final task, consisting of writing the dissertation.

Figure 3.2 depicts a Gantt chart presenting a summary of the described schedule, better showing how dates overlap.

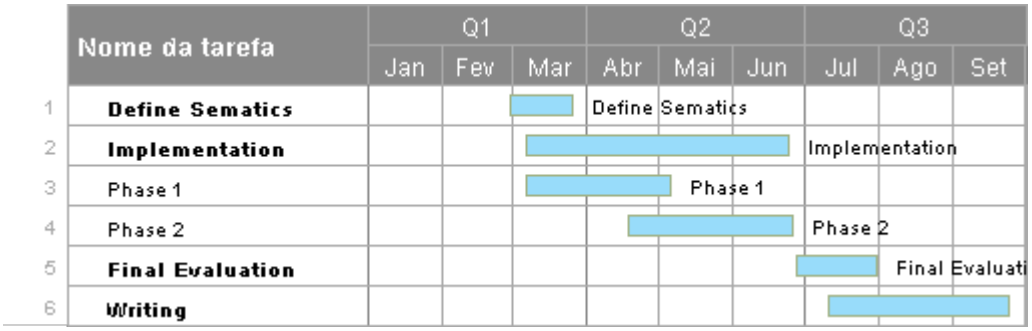

Figure 3.2: Proposed work schedule

Bibliography

[1] S. A. Baset and H. Schulzrinne. "An analysis of the skype peer-to-peer internet telephony protocol". In: *arXiv preprint cs/0412017* (2004).

[2] J. C. Corbett, J. Dean, M. Epstein, A. Fikes, C. Frost, J. J. Furman, S. Ghemawat, A. Gubarev, C. Heiser, P. Hochschild, et al. "Spanner: Google's globally distributed database". In: *ACM Transactions on Computer Systems (TOCS)* 31.3 (2013), p. 8.

[3] 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: *ACM SIGOPS Operating Systems Review*. Vol. 41. 6. ACM. 2007, pp. 205–220.

[4] N. Hayashibara, X. Defago, R. Yared, and T. Katayama. "The φ accrual failure detector". In: *Reliable Distributed Systems, 2004. Proceedings of the 23rd IEEE International Symposium on*. IEEE. 2004, pp. 66–78.

[5] V. C. Hu, D. Ferraiolo, R. Kuhn, A. R. Friedman, A. J. Lang, M. M. Cogdell, A. Schnitzer, K. Sandlin, R. Miller, K. Scarfone, et al. "Guide to attribute based access control (ABAC) definition and considerations (draft)". In: *NIST Special Publication* 800 (2013), p. 162.

[6] 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*. ACM. 1997, pp. 654–663.

[7] R. Klophaus. "Riak core: building distributed applications without shared state". In: *ACM SIGPLAN Commercial Users of Functional Programming*. ACM. 2010, p. 14.

[8] R. Kohavi and R. Longbotham. "Online experiments: Lessons learned". In: *Computer* 40.9 (2007), pp. 103–105.

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

[10] J. Li. *Distributed Systems - Lec 12: Consistency Models – Sequential, Causal, and Eventual Consistency*. URL: http://www.cs.columbia.edu/~roxana/teaching/DistributedSystemsF12/lectures/lec12.pdf.

[11] A. Linde. "Enriching Web Applications with Browser-to-Browser Communication". MA thesis. Faculdade Ciência e Tecnologia, Universidade Nova de Lisboa, July 2015.

[12] L.Naspter. *Napster*. URL: http://www.napster.com.

[13] G. Oster, P. Urso, P. Molli, and A. Imine. "Proving correctness of transformation functions in collaborative editing systems". In: (2005).

[14] R. Rodrigues and P. Druschel. "Peer-to-peer systems". In: *Communications of the ACM* 53.10 (2010), pp. 72–82.

[15] M. Shapiro, N. Preguiça, C. Baquero, and M. Zawirski. "A comprehensive study of convergent and commutative replicated data types". PhD thesis. Inria–Centre Paris-Rocquencourt, 2011.

[16] S. Simon. "Brewer's CAP Theorem". In: ().

[17] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. "Chord: A scalable peer-to-peer lookup service for internet applications". In: *ACM SIGCOMM Computer Communication Review* 31.4 (2001), pp. 149–160.

[18] H. Tran, M. Hitchens, V. Varadharajan, and P. Watters. "A trust based access control framework for P2P file-sharing systems". In: *System Sciences, 2005. HICSS'05. Proceedings of the 38th Annual Hawaii International Conference on*. IEEE. 2005, pp. 302c–302c.

[19] 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*. ACM. 2008, p. 6.

[20] W. Vogels. "Eventually consistent". In: *Communications of the ACM* 52.1 (2009), pp. 40–44.

[21] L. Vu, I. Gupta, J. Liang, and K. Nahrstedt. *Insights into PPLive: A Measurement Study of a Large-Scale P2P IPTV System*. Tech. rep. Technical report, 2005.

[22] Wikipedia. *Bittorrent(protocol)*. URL: http://en.wikipedia.org/wiki/BitTorrent_(protocol)#Adoption.

[23] Wikipedia. *Distributed hash Table*. URL: https://en.wikipedia.org/wiki/Distributed_hash_table.

[24] X. Zhang, J. Liu, B. Li, and T.-S. P. Yum. "CoolStreaming/DONet: a data-driven overlay network for peer-to-peer live media streaming". In: *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*. Vol. 3. IEEE. 2005, pp. 2102–2111.