École doctorale IAEM Lorraine

Probabilistic Models of Partial Order Enforcement in Distributed Systems

THÈSE

présentée et soutenue publiquement le 12 juin 2017

pour l'obtention du

Doctorat de l'Université de Lorraine

(mention Informatique)

par

Jordi Martori Adrian

Composition du jury

Rapporteurs :	Maria POTOP BUTUCARU Weihai YU
Examinateurs :	Hala SKAF Marine MINIER
Directeur de thèse :	François CHAROY
Co-Directeur de thèse :	Pascal URSO

Laboratoire Lorrain de Recherche en Informatique et ses Applications — UMR 7503

Mis en page avec la classe thesul.

"The point isn't to win?" I asked.
"The point," Bredon said grandly, "is to play a beautiful game." He lifted his hands and shrugged, his face breaking into a beatific smile. "Why would I want to win anything other than a beautiful game?"

The Wise Man's Fear
Patrick Rothfuss

Contents

Acknowledgements

I would like to thank my family for the ongoing support during this PhD, for the video-chats, and visits to Nancy. Also to the friends from Barcelona, London and Singapore, always a Skype away. And to Agatha, that has made this last year much better and much more difficult.

To the coworkers in the team, the ones that finished before and the ones that will finish later, by having lunch, coffee, ping pong, climbing, tak, and maybe even working too.

We should not forget the supervisors, Pascal and Francois, for their constant work and patience when trying to understand something that is not very clear, organized and fuzzy.

Finally, I would like thank the Syncfree project, both for funding this gig and for the opportunities that is has given to collaborate and discuss with researchers from all over Europe.

Abstract

Distributed systems have managed to extend technology to a broader audience, in both terms of location and numbers. However these geo-replicated systems need to be scalable in order to meet the ever growing demands. Moreover, the system has to be able to process messages in an equivalent order that they were created to avoid unwanted side effects. Partial order enforcement provides an ordering of events that all nodes will follow therefore processing the messages in an adequate order. A system that enforces a partial order simplifies the challenge of developing distributed applications, and ensures that the end-user will not observe causality defying behaviors.

In this thesis we present models for different partial order enforcements, using different latency model distributions. While a latency model, which yields the time it takes for a message to go from one node to another, our model builds on it to give the additional time that it takes to enforce a given partial order. We have proposed the following models. First, in a one to many nodes communication, the probability for the message to be delivered in all the nodes before a given time. Second, in a one to many nodes communication from the receivers, the probability that all the other nodes have delivered the message after a given time of him receiving it. Third, in a one to many nodes communication, the probability that the message has arrived to at least a subset of them before a given time. Fourth, applying either FIFO or Causal ordering determining if a message is ready for being delivered, in one node or many. All of this furthers the understanding of how distributed systems with partial orders behave. Furthermore using this knowledge we have built an algorithm that uses the insight of network behavior to provide a reliable causal delivery system.

In order to validate our models, we developed a simulation tool that allows to run scenarios tailored to our needs. We can define the different parameters of the latency model, the number of clients, and the clients workloads. This simulation allows us to compare the randomly generated values for each specific configuration with the predicted outcome from our model.

One of the applications that can take advantage of our model, is a reliable causal delivery algorithm. It uses causal information to detect missing elements and removes the need of message acknowledgment by contacting other replicas only when the message is assumed missing. This information is provided by our model, that defines waiting timers according to the network statistics and resource consumption.

Finally this application has been both tested in the same simulator as the models, with promising results, and then evaluated in a real-life experiment using Amazon EC2 for the platform.

Résumé

Les systèmes distribués ont réussi à étendre la technologie de l'information à un public plus large, en termes d'emplacement et de nombre. Cependant, ces systèmes géo-répliqués doivent être évolutifs afin de répondre aux demandes toujours croissantes. De plus, le système doit pouvoir traiter les messages dans un ordre équivalent à celui de leur création afin d'éviter des effets indésirables. L'exécution suivant des ordres partiels fournit une ordonnancement d'événements que tous les noeuds suivront, ce qui permet donc le traitement des messages dans un ordre adéquat. Un système qui applique un ordre partiel simplifie le développement des applications distribuées et s'assure que l'utilisateur final n'observera pas des comportements défiant la causalité.

Dans cette thèse, nous présentons des modèles statistiques pour différentes contraintes d'ordre partiel, en utilisant différentes distributions de modèles de latence. Etant donné un modèle de latence, qui donne le temps qu'il faut pour qu'un message passe d'un noeud à un autre, notre modèle s'appuie sur lui pour donner le temps supplémentaire qu'il faut pour appliquer un ordre partiel spécifique. Nous avons proposé les modèles suivants. Tout d'abord, dans une communication entre un et plusieurs noeuds, la probabilité que le message soit délivré dans tous les noeuds avant un temps donné. Deuxièmement, après la réception d'un message, la probabilité que tous les autres noeuds aient exécuté ce message avant temps donné. Troisièmement, dans une communication de un à plusieurs nœuds, la probabilité que le message soit arrivé à au moins un sous-ensemble d'entre eux avant un temps donné. Quatrièmement, l'ordre FIFO ou causal qui détermine si un message est prêt à être livré, dans un noeud ou plusieurs. Tout cela favorise la compréhension du comportement des systèmes distribués en présence d'ordres partiels. En outre, en utilisant cette connaissance, nous avons construit un algorithme qui utilise ces modèles de comportement du réseau pour établir un système de livraison causal fiable.

Afin de valider nos modèles, nous avons développé un outil de simulation qui permet d'exécuter des

scénarios adaptés à nos besoins. Nous pouvons définir les différents paramètres du modèle de latence, le nombre de clients et les charges de travail des clients. Cette simulation nous permet de comparer les valeurs générées de façon aléatoire pour chaque configuration spécifique avec les résultats prévus de notre modèle.

Une des applications qui peuvent tirer profit de notre modèle, est un algorithme de livraison causale fiable. Il utilise l'information causale pour détecter les éléments manquants et réduit le besoin d'acquittement de message en contactant d'autres répliques seulement lorsque le message est supposé manquant. Cette information est fournie par notre modèle, qui définit les temporisateurs d'attente en fonction des statistiques du réseau et de la consommation des ressources.

Enfin, cette application a été testée dans le même simulateur que les modèles, avec des résultats prometteurs, puis évaluée dans une expérience réelle utilisant Amazon EC2 comme plate-forme.

Introduction (FR)

Les systèmes actuels comme Google, Facebook ou Linkedin, possèdent des millions de machines interconnectés via Internet. Ils permettent à des milliards d'utilisateurs de partager, de stocker de données et de collaborer entre eux. Ils gèrent des petabytes de données de manière distribuée et doivent supporter une quantité de données en constante augmentation [2].

Depuis le début des systèmes distribués, la quantité de données gérées a considérablement augmenté [40]. Les entreprises se sont adaptées et ont créé de nouvelles techniques pour faire face à cette croissance. Initialement, les infrastructures contenaient quelques puissantes machines localisées. De nos jours, avec l'avènement de l'Internet des objects, nous voyons de plus en plus de dispositifs interconnectés qui vont du coeur du réseau à ses frontières avec des utilisateurs partout dans le monde.

Pour assurer une interaction efficace entre les utilisateurs et les systèmes, il faut que les systèmes sont géorépliqués et, que les données soient répliquées aussi près de l'utilisateur que possible afin de réduire la latence d'accès aux données. La réplication consiste à copier toute ou partie des données dans plusieurs emplacements de stockage. Cela garantit la fiabilité, car il est peu probable que tous les lieux de stockage soient indisponibles en même temps, et un accès plus rapide, car les données sont plus proches de l'utilisateur. Plusieurs noeuds peuvent effectuer la même tâche, ainsi, même si un noeud devient indisponible, le système fonctionne toujours, car les utilisateurs sont redirigés vers un autre noeud. D'autres propriétés utiles, des système distribues actuels, sont la cohérence des données et la tolérance aux partitions. La cohérence peut être présentée sous forme de cohérence forte ou linéarisabilité, de sorte que tous les noeuds aient les mêmes vues successives pour tout objet donné. La tolérance de partition représente la capacité du système à fonctionner même en cas d'une défaillance du réseau.

Cela signifie que le système doit être en mesure d'accéder à l'ensemble des ressources, sans pouvoir communiquer avec l'autre ensemble de serveurs qui existent au-delà de la partition.

En 1999, Eric Brewer a formalisé le théorème CAP, plus tard démontré par Gilbert et al. [31], qui énonce, à partir des trois propriétés souhaitables des systèmes distribués: la cohérence, la disponibilité et la tolérance aux partitions, qu'un système distribué ne peut en respecter que deux à la fois. Les trois types de systèmes qui découlent du théorème CAP sont: (1) les systèmes AP, qui sacrifient la cohérence, (2) les systèmes CP, qui sacrifient la disponibilité, ou (3) les systèmes CA, qui sacrifient la tolérance aux partitions. De nombreux systèmes peuvent passer d'une catégorie à l'autre en fonction de leur configuration. Cependant, la configuration CA dans les systèmes distribués est difficile à gérer, car elle nécessite soit un scénario de réseau impossible qui ne supprime jamais les paquets et ne diminue jamais, soit un seul noeud, ce qui supprime la partie distribuée du problème. De cette façon, dans le théorème CAP, nous avons deux options: nous pouvons sacrifier la cohérence forte ou la disponibilité. Les systèmes tels que Riak [1], Dynamo [27] et Cassandra [42] sont généralement classés comme des systèmes qui sacrifient la cohérence forte pour une haute disponibilité, tant que la configuration de réplication ne force pas une cohérence forte. Les systèmes qui sacrifient la disponibilité, comme Zookeeper [36], ou Dynamo utilisent une configuration de quorum pour obtenir une cohérence forte.

Dans les systèmes où la haute disponibilité est nécessaire, nous devons sacrifier la cohérence sur la disponibilité et la tolérance de partition, aux partitions et se contenter de formes plus faibles de cohérence. Mahajan et al [53], a déclaré et prouvé que le modèle de cohérence le plus fort pouvant être obtenu dans ce cadre est la cohérence causale. Ce type de cohérence, même s'il a un coût en latence plus faible que les solutions de cohérence forte, a un impact sur la latence observée par les utilisateurs. La latence est définie ici comme le temps qu'il faut pour transmettre un message d'un noeud à un autre.

La latence moyenne de retour du LORIA à Nancy (France) à plusieurs datacenters peut être vue dans la première ligne du tableau 1 [1]. En théorie, plus l'écart est important entre les noeuds, plus la latence est élevée. Cependant, en réalité, elle dépend à la fois des charges de travail du réseau en cours et de sa topologie. Rappelons que l'Internet est un ensemble interconnecté de réseaux, qui ressemble plus à une énorme toile, avec de multiples chemins possibles pour relier de A à B.

Une plus grande disponibilité peut être obtenue en géo-répliquant les systèmes dans le monde entier. Comme la géo-réplication est une réplication, elle offre les mêmes avantages et les mêmes problèmes, mais à plus grande échelle. Les noeuds sont répartis sur différents continents, augmentant la latence de base observée dans le système et augmentant le coût de la messagerie car chaque aller-retour est très coûteux, parfois en temps et en argent. Le fait de travailler avec des modèles de cohérence

[1] Valeur moyenne de 20 messages ICMP émis avec l'outil Ping.

	Frankfurt	Ireland	Ohio	Singapore	Sao Paolo	Sidney
LORIA (ms)	16.5	22.6	93.6	234.2	258.2	317.4

Table 1: Example de RTT Latence entre LORIA et datacentres d'AWS.

plus faibles complexifie les changements sur les objets afin de ne pas rencontrer d'incongruités de cohérence comme le fait de voir un fichier supprimé (sans avoir reçu d'erreur).

Pour éviter ces incohérences, nous pouvons utiliser des modèles de cohérence relachés. Cela offre au système la flexibilité nécessaire pour fonctionner et rendre le système disponible. Le rôle d'un ordre partiel est de permettre de s'assurer que tous les noeuds traiteront les messages dans un ordre équivalent. Un message est associé à un ensemble de dépendances. Si ces dépendances sont remplies, le message peut être livré, sinon il doit être retardé. Nous distinguons la réception d'un message, quand un noeud reçoit un message, et la livraison, quand le noeud peut utiliser un message reçu. Deux messages sont simultanés si aucun n'est dépendant de l'autre.

Dans cette thèse, nous allons travailler sur l'ordre FIFO, où les dépendances ne se produisent que dans le même processus, et l'ordonnancement causal, dont les dépendances sont tout événement qui a été vu avant que l'opération ait été émise. Il existe de nombreux cas d'utilisation dans lesquels, si les opérations ne sont pas livrées dans l'ordre attendu, le système peut tomber en panne. Nous allons présenter trois de ces cas. Le premier exemple concerne les conversations en ligne (chat), où les messages envoyés par un utilisateur doivent être livrés dans l'ordre. Sinon, la conversation peut ne pas avoir de sens. Les messages d'un utilisateur doivent être livrés dans le même ordre à l'autre utilisateur, ainsi que les réponses pour maintenir la conversation dans un ordre adéquat.

Le deuxième exemple concerne les systèmes de fichiers, où les opérations de fichiers doivent être vues dans le même ordre pour éviter des comportements inattendus. Si un document est créé puis supprimé, le comportement attendu est que le fichier ne devrait pas exister. Cependant, si les opérations sont traitées dans le mauvais ordre, nous aurions toujours un fichier et un état conflictuel dans le système.

Enfin, un troisième exemple concerne les autorisations d'accès aux réseaux sociaux. Si nous révoquons les autorisations d'un utilisateur pour voir nos photos et que nous téléchargeons ensuite une nouvelle photo, nous nous attendons à ce que cet utilisateur n'ait pas accès à cette nouvelle photo. Si les opérations ne sont pas traitées dans l'ordre dans lequel elles ont été générées, il se pourrait que la nouvelle photo s'affiche avant que les autorisations ne soient révoquées.

Tous ces problèmes surviennent lorsque les opérations sont générées et traitées dans des ordres différents. Ainsi, l'ordre entre les opérations doit être maintenu afin d'éviter les incohérences potentielles. L'application des ordonnancements est une solution à ce problème.

Pour assurer un ordonnancement, les opérations qui arrivent en avance sont retardées jusqu'à ce que leurs dépendances soient satisfaites. Ce réordonnancement ajoute une latence supplémentaire au message. Mais si une opération manquante n'est jamais récupérée, toutes les nouvelles opérations seront suspendues, ce qui mettra fin à la progression du système. Ceci explique pourquoi la fiabilité d'un système est plus que la fiabilité de ses parties [16]. Un système sur et fiable assure qu'une fois qu'un message est reçu dans un noeud, un tel message sera également reçu par tous les autres noeuds [17].

La fiabilité est coûteuse car elle nécessite des ressources supplémentaires pour contrôler les messages manquants. Ces ressources peuvent être sous la forme de champs supplémentaires dans les en-têtes des messages, des messages aller-retour supplémentaires entre les noeuds pour assurer la réception, ou comme nous l'avons vu, en latence supplémentaire à la livraison du message. Nous proposons un mécanisme qui, tout en fournissant la fiabilité, n'utilise des accusés de réception que lorsque cela est nécessaire, car il fournit une solution efficace et fiable. Pour ce faire, nous devons savoir quand un message est manquant pour demander une retransmission. Pour cela, nous utilisons les informations nécessaires au respect des ordres partiels.

Nous proposons un modèle de la latence de livraison sous le respect d'ordre partiel Nous modélisons les ordres partiels FIFO et causal, et nous utilisons une distribution exponentielle, Pareto et une distribution de mélange comme modèles de latence. FIFO et causal sont les ordres partiels les plus utilisés dans les systèmes distribués [26, 44], FIFO tel qu'il est utilisé dans les communications TCP, et l'ordre Causal car il fournit le plus fort ordre de cohérence faible. De plus, les distributions exponentielles et de Pareto peuvent être utilisées pour modéliser la latence des communications de systèmes distribués entre pairs comme cela a été montré sur PBS par Bailis et al. [11]. En plus de cela, nous généralisons le modèle pour la charge de travail sous des distributions normales, uniformes ou exponentielles. Cette généralisation permet de passer de l'ensemble des probabilités d'une situation spécifique à un cas plus général pour le même scénario. Nos modèles donnent la probabilité qu'un message soit prêt à être livré dans un scénario homogène avec un nombre fixe de nœuds. En outre, nous avons développé les modèles pour les messages envoyés d'un noeud à l'autre, dans une communication pair-à-pair; d'un noeud à un autre, dans une communication de diffusion; et suivant l'ordre FIFO, avec un sous-ensemble de récepteurs. Nous obtenons également la probabilité qu'un troisième noeud ait reçu un message qu'un nœud local possède déjà. Tous ces modèles sont des outils pour exprimer la latence de la transmission des messages de différents points de vue.

Ces modèles nous permettent de créer une récupération d'erreur réactive qui utilise des acquittements négatifs pour demander à l'expéditeur la retransmission d'un message perdu. Nous avons besoin des modèles pour prédire quand un message arriverait et ainsi définir un temps d'attente limitant les retransmissions inutiles en fonction de la charge de travail et du modèle de latence. Nous créons donc un mécanisme de récupération des erreurs réactives qui réduit l'utilisation d'accusés de réception des messages tout en fournissant un canal fiable et ordonné.

Pour valider ces travaux, nous avons développé un outil de simulation utilisant R qui génère et traite des messages. R est un langage statistique bien connu qui fournit les outils et les bibliothèques nécessaires pour le traitement des données et le traçage des résultats. Nous simulons la communication entre plusieurs nœuds distribués. En utilisant ce simulateur, nous avons validé les prédictions de notre modèle, ainsi que les propriétés du mécanisme de récupération en utilisant des modèles de latence obtenus à partir de rééls scénarios de production. En même temps, le simulateur permet de construire des scénarios plus complexes que ceux modélisés par les équations qui pourraient être utilisées pour correspondre à des configurations de scénarios spécifiques. Le simulateur est plus souple que les modèles, lors de la définition d'une charge de travail et peut fournir des charges de travail différentes pour les différents nœuds, ainsi que la flexibilité pour les systèmes basés sur les quorums.

Enfin, nous avons implémenté le mécanisme de récupération des erreurs réactives à l'aide de python, et avons déployé une expérience dans Amazon Web Services EC2, testant ses performances à la fois dans un datacenter et dans plusieurs datacenters de différents continents. Avec cet ensemble d'expériences, nous avons pu observer le comportement prédit à partir des simulations, la consommation de la bande passante et comparer différents modèles de latence et différentes configuration. De plus, nous avons obtenu la fiabilité sans avoir besoin de remerciements de message en couplant le mécanisme de récupération d'erreur avec l'exécution de commande partielle des systèmes.

Chapter 1

Introduction

Current systems like Google, Facebook, and Linkedin, contain from tens to millions of nodes interconnected through the Internet. They allow billions of users to share, store and collaborate. They manage petabytes of data in a distributed manner and have to support a continuously growing amount of data [2]. Since the beginning of distributed systems, the amount of data managed has grown enormously [40]. Researchers have adapted and created new techniques to cope with this growth. Initially, the infrastructures contained a few powerful machines tightly clustered. Nowadays, with the advent of the Internet of Things, we are seeing more and more interconnected devices that range from the network core to the very edge with the users, all over the world.

To interact effectively with users across the globe, systems are geo-replicated and, data is replicated as closely to the user as possible, as this reduces the access latency to the data. Replication consists of copying all, or parts, of the data into multiple storage places. This provides reliability, as it is more unlikely that all the storage places are unavailable, and faster access, as the data is closer to the user. Several nodes can perform the same task. So, even if one node becomes unavailable, the system is still up, as the users can contact the other node. Other properties are Consistency and Partition tolerance. Consistency is presented as linearizability consistency, so all the nodes have the same view for any given object. Partition tolerance is the ability of the system to continue operating when network failures arise. This means that the system may be able to access part of the resources and some clients, without being able to communicate with the other set of servers that exist past the partition.

In 1999 Eric Brewer formalized the CAP theorem, later proved by Gilbert et al. [31], which states that from the three desirable properties of distributed systems, Consistency, Availability and Partition tolerance, a distributed system could only obtain two at all times. The three types of systems that appear from the CAP theorem are, (1) AP systems, that sacrifice Consistency, (2) CP systems, that sacrifice Availability, or (3) CA systems, that sacrifice Partition tolerance. Many systems can shift from one category to another depending on how they are configured. However, the CA configuration in distributed systems is difficult to manage, as it requires to either have an impossible network scenario that never drops packages and never goes down, or just one node, which removes the distributed part of the problem. This way in the CAP we have two options, we can sacrifice either strong consistency or availability. Systems like Riak [1], Dynamo [27], and Cassandra [42] are usually categorized as systems which sacrifice strong consistency for availability, as long as the replicating configuration does not force Strong consistency. Systems that sacrifice availability, like Zookeeper [36] or Dynamo-like systems with a strong consistency quorum configuration.

In systems were high availability is required, we must sacrifice Consistency over Availability and Partition tolerance, and settle for weaker forms of consistency. Mahajan et al [53], stated and proved that the strongest consistency systems can strive for is causal consistency. Which even though has a lower latency footprint than total ordering solutions, causal consistency still has an impact on the latency observed by the users. Latency is defined as the time it takes to transmit a message from one node to another.

The average round-trip latency from LORIA in Nancy (France) to several datacenters can be seen in the first row of Table 1.1 [1]. In theory, the further away that two nodes are the higher the latency is. However in real-life it depends both on current network workloads, and on the topology, the second row of Table 1.1 . Recall that the Internet is an interconnected set of networks, which looks more like a huge web with multiple possible paths to get from A to B.

Higher availability can be achieved by geo-replicating the systems worldwide. As geo-replication is a form replication, it provides the same benefits and problems, but on a larger scale. The nodes are distributed across different continents increasing the baseline latency observed in the system and increasing the cost of messaging as every round-trip is more expensive, sometimes in both time and money. However, working with weaker consistency models complexifies [74, 65] keeping track of the different object changes to avoid experiencing consistency incongruities like deleting a file and still seeing it (without having received any error).

To avoid these inconsistencies, we can use laxer consistency models. This provides the system with the needed flexibility to work under available systems. The task of a partial order is to ensure that

[1] Mean value from 20 ICMP messages issued with the ping tool

	Frankfurt	Ireland	Ohio	Singapore	Sao Paolo	Sidney
LORIA (ms)	16.5	22.6	93.6	234.2	258.2	317.4
LORIA (km)[2]	240	1012	6716	10465	9600	16720
LORIA (km/ms)	14.5	44.7	71.7	44.6	37.1	52.6

Table 1.1: Example of RTT Latencies and distances from LORIA to AWS Datacenters.

all nodes will process the messages in the same order. A message has a set of dependencies attached to it. If those dependencies are fulfilled the message can be delivered, if not it has to be delayed. We differentiate between a message's reception, when a node gets a message; and delivery, when the node can use a received message. Two messages are concurrent if neither is dependent on the other, and so it does not matter in which order they are delivered.

In this thesis, we work with FIFO ordering, where dependencies arise only within the same process, and causal ordering, whose dependencies are any event that has been seen before the operation was issued. There are plenty of use cases in which if operations are not delivered in the expected order the system can break. We present three of such cases. The first example is chat conversations, where the messages sent by one user need to be delivered in order. If not, the conversation may not make sense. Messages from one user need to be delivered in the same order to the other user, as well as the answers to keep the conversation in an adequate order.

The second example is in filesystems, where file operations need to be seen in the same order to avoid unexpected behaviors like the following. A document is created and then deleted. The expected behavior is that file should not exist. However if the operations are processed in the wrong order, then we would still have a file, and a conflictive state in the system.

Finally, a third example is in social networks access permissions. If we revoke the permissions of a user to see our photos and then upload a new photo, we expect such user to not have access to this new photo. If the operations are not processed in the order they were generated, it could happen that the new photo is shown before the permissions are revoked.

All such problems arise when operations are generated and processed in different orders. Thus order among operations needs to be maintained to avoid potential inconsistencies. Order enforcement is a solution to this problem.

To ensure order enforcement, operations that arrive out of order are delayed until their dependencies as satisfied. This reordering adds additional latency to the message. But if a missing operation is not recovered, all new operations will be suspended thus halting the system's progress. Which explains why a system's reliability is more than the reliability of the independent parts [16]. A strong reliable system ensures that once a message is received in one node, such message will also be received by all nodes [17].

Achieving reliability is costly as it requires additional resources to control the missing messages. These resources can be in the form of an extra field in the header of the messages, additional round-trip messages between nodes to ensure reception, or additional latency to the message delivery. We propose a mechanism that while providing reliability does not use acknowledgments except when needed, as it would provide with a more efficient reliable solution. To do this we need to know when a message is missing in order to ask for a retransmission and for that, we can use the information used to enforce partial orders.

We propose a model of, the delivery latency under partial order enforcement with different latency model and workloads. Modeling how systems behave under partial order gives us more insight into the expected behaviors for these systems. When used with realistic latency models, it allows us to tune and compare distributed systems. We model FIFO and causal orderings as partial orders, and we use an exponential, Pareto and mixed of both distributions as latency models. FIFO and causal orders are the most widely used partial orders in distributed systems [26, 44], FIFO as its used in TCP communications, and causal order as it provides the strongest weak consistency ordering. Furthermore, exponential and Pareto distributions can be used to model the latency of distributed systems communications between peers as it was shown on PBS by Bailis et al. [11]. In addition to this, we generalize the model for operation workload under: normal, uniform or exponential distributions. This generalization allows to separate the set of probabilities from a specific situation to a more general case for the same scenario, and not the same set of operations and times. Our models give the probability that a message is ready to be delivered in a homogeneous scenario with a fixed number of nodes. Furthermore, we developed the models for operations sent from one node to another, in a peer-to-peer communication; from one node to many, in a broadcast communication; and under FIFO ordering, in a one to many, with a subset of receivers. We also got the probability that a third node has received a message that a local node already has. All these models are tools to express the situation of message delivery from different points of view.

This allows to create a reactive error recovery that uses negative acknowledgments to signal the sender for a retransmission of a lost message. We need the models to predict when a message should arrive and with it, we can tune the waiting time to a foretold number of unnecessary retransmissions that depends on the workload and latency model. So we create a reactive error recovery mechanism that reduces the usage of message acknowledgment while still providing a reliable channel.

To validate these works we developed a simulation tool using R that generates and processes messages. R is a well-known statistics language that provides the necessary tools and libraries for data processing and result plotting. It simulates the communication between several distributed nodes. Using this simulator we validated the predictions of our model, as well as the recovery properties of the recovery mechanism using latency models obtained from production scenarios. At the same time, the simulator allows building more complex scenarios than the ones modeled by the equations that could be used to match specific scenario configurations. The simulator is more flexible than the models when defining a workload and can provide different workloads for different nodes, as well as flexibility for quorum based systems.

Finally, we implemented the reactive error recovery mechanism using python, and deployed an experiment in Amazon Web Services EC2, testing its performance both within a datacenter and across multiple datacenters in different continents. With this set of experiments we were able to observe the predicted behavior from the simulations, the bandwidth consumption used under different latency models, and mechanism configurations. Furthermore, we obtained reliability without the need of positive message acknowledgments by coupling the error recovery mechanism with the systems partial order enforcement.

To the best of our knowledge we are the first to model the delivery latency of partially ordered systems, and to parametrize a reliability mechanism specifically designed for such systems.

The contributions and experiments of this thesis are the following:

- a model that yields the probability that a message is ready for delivery under different partial orders, causal and FIFO; with different latency models, exponential and Pareto distribution, and a mix of both.
- the generalization of the previous models, with a rate of operations that follows: uniform, exponential or normal distributions.
- a reactive error recovery mechanism that uses the afore mentioned models. It reduces the use of acknowledgment messages, and sends negative acknowledgments to other nodes when operations need to be retransmitted.

- a simulator built using R to validate the models, and the reactive error recovery mechanism. We used production-like latency models for the error recovery mechanism validation.
- an evaluation experiment deployed in Amazon Web Services EC2 that demonstrates the behavior of the reactive error recovery mechanism. And, a comparison of the bandwidth usage with TCP.

Finally, the rest of this thesis is structured as follows:

- Chapter 2 is the State of the Art. We review the background work in this field, as well as key work used in this dissertation. We mainly review two topics, firstly how to obtain reliability in message communication, with either error recovery, message acknowledgment, or consistency management algorithms like: Consensus, Quorum, or CRDTs. Secondly we review different consistency models, from strong to eventual consistency. We also review divergence control systems that are built upon weak consistency models.
- Chapter 3 is the theoretical work. We present our models and explain in detail the different parts of them. We follow the non-generalized models for FIFO and causal ordering. This non-generalization ties the model to specific operations and times. Later we cut the ties and explain the steps it takes to generalize them, as well as the motivation.
- Chapter 4 is the work related to the simulator. We introduce the capabilities of the simulator as well as the parameters. We present the results from such simulations between the values predicted by our models and the observations from the simulations.
- Chapter 5 is the evaluation of the Reactive Error Recovery mechanism using AWS EC2 infrastructure. We present the motivation, setup, and results from those experiments.
- Chapter 6 are the conclusions and further work, where we summarize the objectives from this thesis and with which degree they were achieved. At the same time we propose prospective lines of inquiry that spawn from this work.

Chapter 2

Background

Contents

Communication is one of the main issues in distributed systems, as it is required to connect the different nodes in the system. By providing good communication, the distributed system provides several interesting features like: low latency, with replicas close to the user; availability and fault tolerance, with geo-replication that makes it less likely to have the whole system down, and scalability, that allows the system to adapt to the load requirements, as needed.

We divide this chapter as follows: (1) Reliability, and how to achieve it in hostile environments, (2) Consistency models, explanation of the different levels of consistency, from weak to strong, and (3) Consistency management algorithms, introduces algorithms like quorums, CRDTs and 2PC. Each of these sections solves a part of the communication problem in distributed systems.

2.1 Reliability

Achieving reliability in distributed systems is a key feature. Without reliability, the messages exchanged between nodes can be corrupted, lost, or actively manipulated to change the system's behavior. This section introduces how faults are generally classified and how nodes acknowledge the messages they receive.

2.1.1 Failure Classification

Before defining how we recover from errors, we introduce different types of failures:

- Crash failures: when a process encounters a crash failure, it does not recover from it. We can assume that the resources were lost, and that those nodes will no longer be available.
- Transient failures: when a process encounters a transient failure, it will eventually recover unscathed. Unintended message corruption belongs to this category [47]. Unless the corruption was continuous as a result of hardware malfunction it would then belong categorized as a crash failure. At the same time if the error was caused by active opponent then it would belong to byzantine failures.
- Omission failures: when a process encounters a failure by omission, it means that a task that had to be done was skipped.
- Byzantine failures: these are the most generic type of failures that a process can encounter. A byzantine failure allows for an arbitrary behavior of the process, either malignant, from an attacker or unexpected random behavior.

Data replication solutions like: Quorums, CRDTs, and Master-slave replication, can protect the system against crash failures, as long as the number of failures tolerated is high enough for the chosen solution. Systems that have Quorums, can tolerate f node crashes for every $2f + 1$ nodes in the system. CRDT offers more tolerance in number of failures, and as long as one node is still available, the system is available. Master-slave [36] solutions have the inconvenience that if the master fails, a new master needs to be elected. However, if a slave crashes, the systems recovers much easily.

Message corruption solutions in distributed communications can take the form of message checksums [58], parity bits or their generalization erasure codes like Read-Solomon [66] but not very efficient for long messages or Digital Fountain [23] which are very efficient but have patent and copyright limitations.

Byzantine fault tolerance [46, 45, 6] uses replication to achieve f faults tolerance in a $3f + 1$ nodes system. f nodes can suffer any failure and the overall system with still be consistent.

We will see more about replication techniques in the following sections. For now we have introduced their usage to manage some types of failures. This thesis focuses on reliable communications for systems that enforce a partial order, either FIFO or causal. We will reduce the need of message acknowledgment while ensuring reliability. We will ensure the congruity of the message to lower layers of the communication stack in the form of checksums.
In the next section we will see how mechanisms manage different types of acknowledgment solutions.

2.1.2 Message Acknowledgment

When communicating with nodes across the network, it is difficult to know whether a sent message has been received. Let us imagine a system such that for every received message, it replies with an acknowledgment to inform the sender that the message has been received. In this system if both messages are received, then both sender and receiver know the message. However if the receiver's message, the acknowledgment, is lost then the sender might keep resending the original, and unless the receiver manages the duplicates it could lead into unexpected behaviors. This type of acknowledgment is a variation of the one bit sliding window algorithm and is used in the three-way handshake of TCP, as seen in Figure 2.1. Here we see two connections with a SYN and an acknowledgment message, both start a connection with a SYN and expect the ACK message.
When the network is unreliable it might be preferable to use this mechanism. With reliable networks, messages are not frequently lost, it is more efficient to send negative acknowledgment, or NACK. These messages are issued by the receiver when it realizes that a message that should have been received is missing thereby requesting the sender to resend.

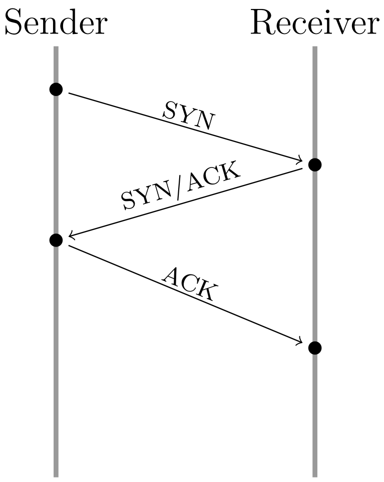


Figure 2.1: Example of communication acknowledgment for every sent message.

Another mechanism to acknowledge messages is selective acknowledgment [52], or SACK. This mechanism does not acknowledge a specific message but instead, a group of send bytes. In TCP this group is called a window. Making the window larger makes it more efficient, as the ratio of acknowledgment bytes by useful diminishes. Errors make the connection less responsive, as it takes longer to realize of the error and recover. Jacobson [37, 51] proposed the algorithm for dynamically changing the window's size that follows an AIMD, additive increase/multiplicative decrease. TCP uses the same mechanism to acknowledge the stream of sent bytes. Figure 2.2 shows this communication, the exchange of data and acknowledgments.

In addition to message acknowledgment, another factor influencing the reliability of inter-nodal communication is the question of how long a sender should wait prior to resending an unacknowledged message. TCP uses a timeout value called Retransmission TimeOut, or RTO. The initial value of RTO, before Round-Trip Time (RTT) values have been computed, is one second [60], and it some conditions it can revert to the old three seconds value [59]. After the first RTT has been acquired the RTO value will change according to Equation 2.5. Then when more RTT have been recorded, Equation 2.9 will be used to update the RTO value. A maximum value bound can be placed on RTO, as long as it is at least 60 seconds, and a minimum RTO value of one second is recommended [8] in order to get a conservative timeout that avoids unnecessary retransmissions. Jacobson et al, [37] suggested α is 0.125 and β is 0.25 in Equation 2.9.

$$RTO_0 = 3 \tag{2.1}$$

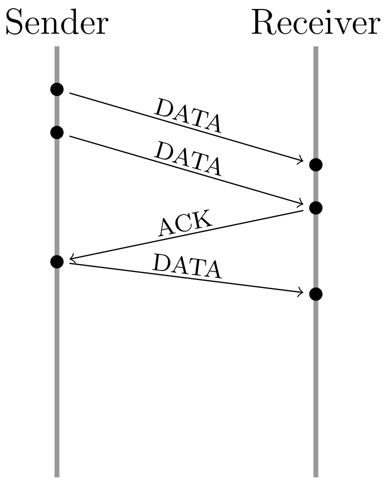


Figure 2.2: Example of communication acknowledgment for every block of bytes sent.

$$SRTT_1 = R_1 \tag{2.2}$$

$$RTTVAR_1 = R_1/2 \tag{2.3}$$

$$RTO_1 = SRTT_1 + max(G, 4 * RTTVAR_1) \tag{2.4}$$

$$\tag{2.5}$$

$$SRTT_t = (1 - \beta) * RTTVAR_{t-1} + \beta * |SRTT_{t-1} - R_t| \tag{2.6}$$

$$RTTVAR_t = (1 - \alpha) * RTTVAR_{t-1} + \beta * |SRTT_{t-1} - R_t| \tag{2.7}$$

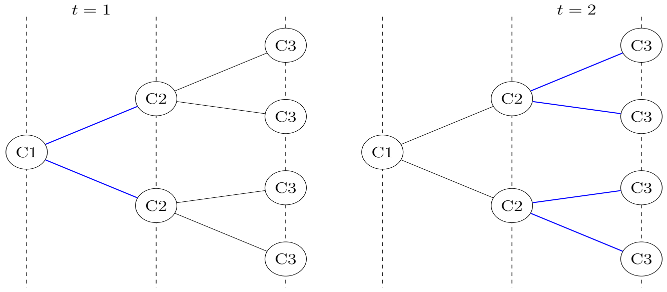
$$RTO_t = SRTT_t + max(G, 4 * RTTVAR_t) \tag{2.8}$$

$$\tag{2.9}$$

Part of the work studied in this thesis, is an alternative method to compute retransmission timers for missing messages in partial order enforcing communication layers. It aims to reduce the amount of network usage in the form of message acknowledgment.

2.1.3 Multicast Messaging

Multicast is a term that describes the communication between one or many nodes to zero or many nodes. There exist different ways a multicast protocol can achieve reliability. In one extreme it can offer atomicity for its messages, which means that either all nodes will receive the message or none will. It can also offer order enforcements like FIFO or causal, which are explained later in this chapter in the partial orders subsection.



(a) $t = 1$ (b) $t = 2$

Figure 2.3: Example of tree-based message dissemination in multicast systems.

Birman et al. [17] proposed *pbcast*, a reliable multicast protocol that could be rigorously quantified and that had stability guarantees. *Pbcast* recovery can cause group partitioning when experiencing delivery failures. As one of the nodes is lagging far behind, due to the delivery failure, the rest of the group will garbage collect the history thus partitioning the group. As this behavior is due to a faulty process, the protocol does not deal with it. Nonetheless Birman proposes two solutions. One of them would be to use Spinglass[1] and vary the length of the buffers history. Some nodes would have different buffer sizes, while most would have a normally sized buffer. This ensures that some members of the group have a far greater log that can keep the history for short-medium time laggings. Thus, these groups would not forget a node due to a short abnormal behavior. Another proposed solution is to use Ensemble [18, 33] with *pbcast*. Combined they work as a recovery mechanism, where nodes can join, leave, and rejoin; either voluntary or as a result of a failure. Other protocols like $LBRM$ [35], Log-Based Reliable Multicast, provide reliability by using heartbeats between the nodes, which ensures that every $MaxIdleTime$ a node will receive news from the other nodes, or they will be assumed down. Some extended work from these two protocols is *rpcast* [68] as it presents a hybrid solution between both *pbcast* and $LBRM$. The former is used as a gossip protocol and the latter is used when the *pbcast* fails providing a deterministic approach with the log-based solution.

Kermarrec et al, [39], provided a theoretical analysis of gossip-based protocols. In their analysis they took into account the relationship between: system size, failure rates, and fanout.

However, receiving all the messages is not enough. Reliable communications, while being a very important part of the distributed systems architecture, are not enough to achieve consistent applications. Let us image a scenario where we sent three messages from one node to another. The messages represent a basic arithmetic operation of $3 - 5$, and each sent message is of the three terms,

[1] Spinglass is a standalone solution introduced in the Bimodal [17] article.

3, $-$, and 5. Without enforcing any order, the three messages can be rearranged, in any of these six combinations: $3-5$, $35-$, -35, $5-3$, $53-$, -53. While out of these six combinations there are two ($35-$ and $53-$) that are not valid statements, the other four are valid, and if no order delivery is enforced, the state of the nodes will diverge. It is for this reason that reliable communication alone is not enough and we must also provide the order in which messages need to be processed.

2.2 Consistency Models

Consistency models provide guarantees about the state of objects in distributed systems. These models define how objects behave under different read and writes operations. There are strong consistency models in which after a write all subsequent read will capture the new value. At the other extreme there are weak consistency models. The weakest is eventual consistency, which ensures that the system will converge, if there are no unrecoverable errors, no new messages, and enough time.

The ACID [32] definition of Consistency is usually called strong consistency. It is a type of consistency model that does not allow divergence among replicas. Of all the consistency models is the one with the highest latency cost. As stated by the CAP theorem defined by Brewer [19] and proven by Gilbert and Lynch [31] if a distributed system ensures strong consistency, it might not be available when errors occur.

This creates three types of distributed systems, depending on which of the three properties, Consistency, Availability and Partition Tolerance, will be dropped when problems arise. Consistency is interpreted with the definition of strong consistency. Systems that sacrifice strong consistency for weaker constraints are called AP systems. Dynamo, Cassandra, and Riak are three examples of systems that are usually considered AP systems. However under specific quorum configuration, they can behave as a strong consistency system. Systems that sacrifice availability are called CP systems, and under specific circumstances they will stall read or write operations until enough nodes are available. Only then they can synchronize changes. Zookeeper is a typical example of a CP system due to its master-slave architecture. Finally, CA systems would be those that choose to sacrifice Partition Tolerance. However, due to the nature of distributed systems being distributed, this could only happen with non-distributed systems. Which makes sacrificing partition tolerance impossible for a distributed system. That said, Spanner has claimed to be in the CA category, but is in fact a CP, as when network partitions have occurred the system chose to sacrifice Availability over Consistency [20].

Weaker forms of consistency also allow faster updates or reads operations than systems with strong consistency. Eventual consistency [71, 73] is the weakest form of consistency, and it informally guarantees as stated by Bailis [11] "if no additional updates are made to a given data item, all reads to that item will eventually return the same value". This form of weak consistency can be problematic for users, as the application does not guarantee that if they make an update to an item and then refresh the page, they will see their update. This can cause users to assume that the system failed and upload the same information again, which in turn causes duplicated operations. For example, in a blog comment section, this could cause the same message to be published twice, or in an online shopping cart desired items might be added twice. In this section we will introduce the main different consistency models, as well as partial orders to help understand some specific models.

2.2.1 Strong Consistency

Strong consistency represents the monolithic view of consistency, in which all clients observe the same version of data, and changes are rolled at the same time across all nodes. The system achieves strong consistency when operations are linearizable. Furthermore, in strongly consistent systems, there exists a total ordering of events, and all nodes execute the operations in the same order.

Models such as these, are very costly in (1) synchronization, as all nodes need to communicate to take decisions and inform them, having to stop and wait for the missing ones. (2) Scalability, due to synchronization will be less effective the more nodes are involved in the total ordering. The two strong consistency models are strict consistency and sequential consistency. Given that it is difficult to obtain a global clock in distributed systems it is difficult to implement.

Any other consistency less than sequential is usually regarded as a weak consistency model.

2.2.2 Weak Consistency

In weakly consistent systems, there is a trade-off between the guarantees imposed on the data and the latency and throughput of the system. With the lowest level of weak consistency, the user is not guaranteed to be able to read his own writes. This can cause expected behavior for the user, with the generation of duplicated operations. At the same time given the small number of constraints the system is more efficient at managing these operations and achieves a higher throughput. In this section we will explain several weak consistency models (causal and eventual) and we will introduce the shift to client-centric consistencies with session-guarantees.

Eventual Consistency

Without any other guarantee, eventual consistency [71, 73] ensures that messages will eventually be seen by the all nodes, as long as there are no unrecoverable crashes, or new messages, and that enough time is provided. This definition of eventual consistency does not ensure much except that a message will be in the other nodes at some point in the future. Bailis et al, [11] and Bermbach et al. [15] both propose probabilistic bounds in which this "eventually" could be determined. These studies, while not providing any guarantees, inform us of a systems behavior under eventual consistency, with some probabilistic metrics. In recent years eventual consistency has become very popular due to its lack of scalability limitations.
However the lack of guarantees makes it difficult for developers to build robust applications based upon eventual consistency, as in some cases [22] unexpected behaviors occur.

Session Guarantees

One of the issues with previous consistency models, is that they focus guarantees on the consistency of one object across the whole system. With in strong consistency this makes the system go too slow and on eventual consistency, while quick there are no ordering constraints which can cause missing events from the perspective of a user. Terry [70] shifted this paradigm and to focus on the user's expectations. He proposed a stronger consistency model than eventual consistency in which there are session guarantees. A session is an abstraction of the list of operations that the user is running within the application. The objective of a session is not to behave as an atomic transaction but to provide the user with a consistent view of his operations during that session. Session guarantees can be summarized as:

- **Read Your Writes:** read operations reflect on your writes.
- **Monotonic Reads:** Successive reads reflect a non-decreasing set of writes.
- **Writes Follow Reads:** Writes are propagated after reads on which they depend.
- **Monotonic Writes:** writes are propagated after writes that logically precede them.

Note that if the user ends their session, the system does not guarantee that they will see their operations if they reconnect.

Causal Consistency

The strongest consistency model that tolerates network partition while still being available, is the causal consistency model [53]. It uses the causal partial order to determine the dependencies between operations, as shown in Figure 2.4. A partial order is defined as “a transitive antisymmetric relation among the elements of a set, which does not necessarily apply to each pair of elements”. We present three different partial order wildly used in distributed systems, the first being FIFO or NPRAM, the second is causal [44], and the third is causal+ [50]. FIFO ordering has fewer constraints than causal or causal+, and it has the following notion of potential causality [4, 44]:

- **Thread-of-execution:** $a \to b$ if a happened in the same replica as b and $t_a < t_b$. Where t_i is the time at which event i happened.
- **Transitivity:** if $a \to b$ and $b \to c$ then $a \to c$

For causal ordering we can define similar rules to define the notion of potential causality:

- **Thread-of-execution:** if a happened in the same replica as b and $t_a < t_b$, then $a \to b$.
- **Diff-Replica:** if a is delivered to the replica from which b comes; b has not been issued yet; and a and b come from different replicas, then $a \to b$.
- **Transitivity:** if $a \to b$ and $b \to c$, then $a \to c$.

As defined by Lloyd et al. [50], causal+ consistency is the combination of causal consistency and a convergent mechanism to manage conflicts [2]. While the former has just been defined, we can define the latter as, mechanism f that is applied to handle conflicts, that holds in the following equality: $f(a, f(b, c)) = f(b, f(a, c))$. Different approaches exist in order to generate this mechanism f. By default Causal+ [50] uses the last writer wins [72] approach. However more flexibility can be achieved by designing a custom function. Systems like Bayou [71], and Dynamo [27], allow such functions to obtain a merge that is better suited to the application. This approach provides more flexibility with the cost of being more complex for developers to code. Another approach can be used to deal with more complex conflicts, or conflicts without a handler. This is to flag them for human intervention as does Dropbox [29] or git with merge conflicts.
Another approach to handle conflicts is to use CRDT [48, 63], Conflict-free Replicated Data Type, where each data type has already a set of policies as handlers for conflict solving. They can be used as standalone datatypes [10], or from systems like Riak [1] or Antidote [61].

Logical clocks as defined by Lamport [44] are used to track causal dependencies amongst different distributed events in a system. A clock, C_i, is defined for every replica, R_i. Thus, when a time is assigned to an event, it is just a way to number it. A function clock, C_j , assigns a time to

[2] Conflicts are situations that would cause replicas to diverge forever

an event a, by $C_j(a)$ iif event a belongs to replica R_j. For any events a, b belonging to the same replica, if $a \rightarrow b$ then $C(a) < C(b)$ [3]. Logical clocks can be described with counters. This way there is no actual timing mechanism. Physical clocks can be also used to assign times to events. The main difference between physical time and virtual time, was described by Mattern [55] and Fidge [30].

Vector Clocks have one entry for each replica in the system. Therefore in the vector clock, VC, the replica, R_j, will have in the entry $VC[j]$ the number of events that it has sent.

Vector compaction is a technique that attempts to minimize the size of the vector clock that is transmitted in every event message.

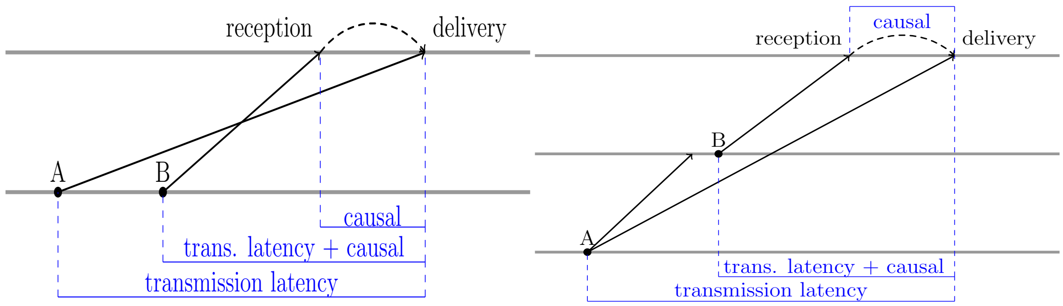
Singhal et al, [65], proposed an efficient implementation of a vector clock in which each replicas will have to use two extra vector clocks, one called LU, for Last Update, and LS, for Last Seen. For example, $LU_i[j]$ contains the value that $VC_i[i]$ had when it updated the entry j. It is a way to remember the state of R_i when it last delivered a message from R_j. $LS_i[j]$ contains the value that $VC_i[i]$ had when it sent its last value to R_j.

This compaction approach improves the scalability of the vector clock protocols, as well as providing a performance improvement in network usage [24].

When enforcing partial orders, we must differentiate between reception time, when a message is received by the communications module of the application, and the delivery time, when the message is seen by the application. In order for a message to be delivered, all its dependencies (messages that happened before it) have to have been delivered already. The extra delay time added by the communications module in order to satisfy these constraints is the partial order latency time. The more restrictive the partial order is, the higher this delay.

In this section we have explored both strong and weak consistency models. We have seen mechanisms that try to leverage the trade-off between strong and weak consistency models with solutions like escrow, consistency rationing and red/blue consistency. At the end of the previous section we noticed that solely delivering operations was not enough to ensure the consistency of an applications data. However now that we have these tools, it is enough depending on each application, to ensure the consistency. Looking back and using the metadata needed for enforcing partial orders we will create a mechanism that provides a reliable communication while reducing the need for acknowledgment and relies on our knowledge of the network, to issue negative acknowledgments when needed.

[3] However the same cannot be said from right to left as it would mean that all concurrent events would have had to happen at the same time.



(a) FIFO

(b) Causal

Figure 2.4: Naming diagram for FIFO and causal partial orders. Example of different events involved in sending messages between replicas and how they are named.

In this next section we will present other mechanisms that offer reliable communication.

2.3 Consistency Management Algorithms

The aim of reliable data replication is to ensure that the same data will be available in all the required nodes, this can only be achieved if one replica has seen a message, then all the other ones will eventually see the same message. These conditions are required in order to elect a leader, to take shared decisions between several nodes, and to set a barrier or synchronize nodes, while ensuring a deadlock free situation [34].

In this section we present several techniques used in distributed system for reliable data replication.

2.3.1 Consensus

A consensus problem is one in which an agreement is required amongst a number of nodes. It can either to elect a new leader, decide a value or decide how to recovery from a conflicting situation.

A consensus protocol must therefore provide the following properties [25].

- **Termination:** Every correct node in the consensus protocol must have a value.
- **Validity:** If all nodes propose the same value, then all correct nodes will chose that value.
- **Integrity:** If a correct node agrees on a value, such value must have previously been proposed by one of the nodes.
- **Agreement:** All correct nodes must agree on the same value.

2.3.2 2PC

2PC stands for two-phase commit protocol [56], an atomic commitment protocol. As the name indicates, the protocol has two phases. A commit request phase or voting phase, and a commit phase or completion phase. During the voting phase, the coordinator sends a message to all the replicas that also have to commit the transaction, and wait for all the responses. Meanwhile, the replicas that receive the message, execute the transaction up to the commit point while logging the operations in a log. At this point every replica that gets to the commit point replies to the coordinator with a success, if all operations in the transaction were properly executed, or an abort if even one of them could not be executed. At this point the first phase is over and the completion phase begins. If all replicas replied a success the coordinator will send the commit message and all replicas will commit the previous operations, sending back an acknowledgment once the commit was successful. Once the coordinator receives all the acknowledgments, then and only then will new transactions be considered. If only one of the replicas replies with an abort message from the first phase (or some timeout expires) then the coordinator will send the abort/rollback message instead. A rollback message will force all the replicas to go back to the state they were before the transaction began. After all the replicas rollback, an acknowledgment is sent to the coordinator that upon receiving all of them will abandon or retry the transaction.

The main disadvantage of this protocol is that if the coordinator fails after some of the replicas have sent their success/abort message from the voting phase, then as they never receive a reply from the coordinator they will be blocked.

2.3.3 Paxos

Paxos [43, 45] is a protocol that solves the consensus problem in an environment with faults. There exist different variations of the algorithms like byzantine-paxos, cheap-paxos and fast-paxos. The basic Paxos protocol contains the following phases:

- **Phase 1a - Prepare/Propose**: A proposal is created with an ID, which needs to be higher than any previously accepted proposal. The node that creates the proposal will be the leader, and it will be in charge to select the nodes (quorum of acceptors) that will accept the proposal. This phase ends when the message's proposal is sent to the quorum of acceptors.

- **Phase 1b - Promise/Agree**: In this phase the acceptors receive the proposal. They can only accept it, if and only if, they have not accepted any proposal with an ID higher than the proposal. Furthermore, if they accept the proposal, they have to ignore all future proposals

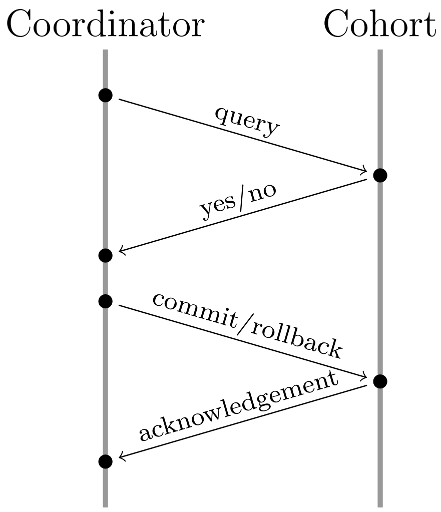


Figure 2.5: 2PC protocol. Order of messages and possible responses.

that have an ID lower than the original proposal ID. If they can and accept the proposal, then the acceptors notify the leader with an acknowledging message.

- **Phase 2a - Accept Request/Commit**: Once the proposer has received enough acknowledgments, a value needs to the defined and accepted. The acceptors will send a value to the proposer, who will chose the highest one. It no value was sent, the proposer is the one that proposes a value. Once the value is decided, the proposer will send an accept request message to all the acceptors with the value that needs to be accepted.

- **Phase 2b - Accepted/Accept**: The acceptors can accept a request, or commit, if the value is the same as in a proposal that was already accepted, and the ID of the proposal if the highest one that has been agreed upon. Multiple proposals can run in parallel, but only the one with the highest ID can be accepted.

With Paxos' robustness, comes a lot of messages used to communicate between the different nodes. This results in a higher cost than most systems can afford. Furthermore, a Paxos system of $2f + 1$ nodes can tolerate f faults, as there always need to be at least half of the nodes up and running to have majority quorum decisions.

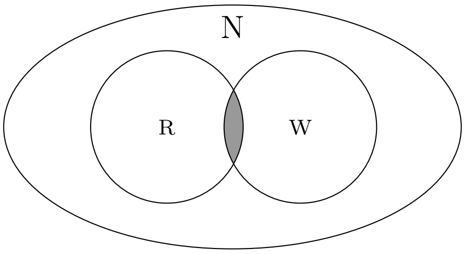


Figure 2.6: Quorum intersection of **R**ead and **W**rite sets.

2.3.4 Quorum

Quorums are a type of data replication technique widely used nowadays by systems like Project Voldemort[67], Cassandra[42], Riak[1]. A systems quorum configuration is defined as N number of nodes, W number of write acknowledgments, and R number of read acknowledgments. When a node receives a write operation, it will send it to all N nodes. However, it will not confirm the write to the client until it has received W acknowledgments. Similarly, for a read operation, it will send to all nodes, and wait for R responses from the nodes. Only then will it forward the response to the client. Different behaviors can be achieved depending on the configuration values of the quorum system (N,W,R).

In a case where $N \leq W + R$, Figure 2.6, the client will always receive the last written value. In which case the quorum is enforcing strong consistency. When $N > W + R$, this is known as partial quorums, and it can happen that the client receives stale data. There are several studies relating this types of quorum trade-offs [13, 7, 3].

2.3.5 CRDT

CRDT stands for Conflict-free Replicated Data Types, and there exist type definitions for counters, registers, sets, maps, trees and graphs. There are databases that implement them, Riak [1], Antidote [61], as well as applications in, collaborative editing, [38], GeoNavigation [28], and real-time Internet chatting [62].

All CRDTs ensure Strong Eventual Consistency(SEC) which states that as soon as two replicas have seen the same events they will have the same outcome, regardless of the order in which the operations arrived. As CRDT are monotonic they ensure that the system will not suffer rollbacks.

There are two main types of CRDT depending on how they replicate their state to the other replicas. State-based CRDTs replicate their state by transmitting their whole state over the network to the other replicas. These other replicas, using a merge function, merge both states.

Operation-based CRDTs only send operations to the other replicas and those replicas have to apply this in order to get the same state.

In order to guarantee SEC, CRDTs have to fulfill the following properties:

- **Monotonicity:** The number of elements or value in a CRDT cannot be reduced over time. Elements cannot be removed from a CRDT.
- **Semi-lattice:** For State-based CRDT, the state of the data type has to be represented with a partially ordered set with a join/LUB[4] operation.
- **Commutativity:** For Operation-based CRDT, concurrent operations have to commute. Therefore if f and h are operations and x is the CRDT, $f(h(x)) = h(f(x))$.

The state-based can be very costly to send, as the state of the CRDT grows without bounds, and applying the merge can also be very costly. However, once the state is ready, it is very fast to read. On the other hand the operation-based is more efficient while sending the operations over the network, but it can be very costly to materialize its state, as all past operations have to be read and assembled together.

There exist solutions where only deltas of the state are sent [9], or the CRDT is only partially replicated, therefore not all parts of the CRDT have to be found in all replicas [21]. On the same page, in operation-based CRDT, operations can be batched together.

2.3.6 Bounding divergence

When sending updates between replicas, there is a time when different replicas have different views of the system. Systems that enforce strong consistency will not experience divergence, however the nodes themselves can. If the number of faults, that cause a divergence, is large enough, the system will cease to be available. However, if the number of fault is small enough, some nodes can experience divergence. In any case, the faulty nodes will have to synchronize to obtain the missing messages, and avoid being considered faulty nodes. Eventually consistent systems tolerate such divergence, and metrics like t-visibility or k-staleness from PBS [11] can be used to describe how much divergence the system is experiencing. Other works [6], propose a relaxed view of staleness in which the system is allowed to return any of the last k values.

In order not to bound the divergence but to predict the probability that a returned value will be

[4] LUB: least upper bound

fresh, [54, 11] models the nodes behavior, network transmission, and errors, to give such probability in different conditions.
Finally, works like Conit [76] and TACT [75] design a system in which three different metrics of divergence are described and the operations are bound to them. These three metrics are:

- **Numerical Error:** Limits the difference between the local image and the final value. In turn, this limits the maximum value that an image can be changed in just one operation.
- **Order Error:** Limits the number of reorderings that an operation can be subjected to.
- **Staleness:** Limits the difference between the operation issue time and the acceptance of it, like a TTL before it gets applied.

There are other mechanisms that trade off the benefits between working in a strongly consistent system, while reducing the cost of synchronization. Some of them are: Escrow and Escrow-like systems, like Exo-leasing [64]; hybrid consistency systems, like red/blue [49] consistency, and consistency rationing [41]. Escrow systems solve the need for constant synchronization without breaking its constraint by providing two functions, split and merge. While these two functions require communication between the two involved nodes, which forces synchronization, they do not require all the nodes to perform the synchronization. The function split reserves a part of the value from the sender node, and the merge function adds the split value to the receiver node's value. This means that as long as the node does not break any application constraints with the split data it can operate without contacting any other node. This allows the node to go off-line and continue progressing, and later come on-line and inform of the state.

Red/Blue operations, belong to hybrid consistency models, that leverage strong consistency with weak consistency. They aim to classify incoming operations into red if the operation can break the applications invariance and blue if it does not. This way, red operations are processed with stronger consistency replications, and will have to contact more nodes to ensure that no constraints are broken, making them safe but more costly to the system. While blue operations, are processed with weaker consistencies as they cannot break the system. This kind of leverage can also be observed in consistency rationing systems. For instance, in a game environment, events that happen closer together are dealt using stronger constraints to avoid unexpected behaviors, while events happening far away are managed with more tolerant delays. These kind of hybrid consistency models manage the trade-off off strong and weak consistencies using the application logic [14].

In this chapter we have seen different approaches for achieving reliable communications, and different consistency levels. We have also seen that stronger consistency requires stronger reliability, as the system is less tolerant of missing messages and experiences greater consequences if a message is missed. We have seen the well-known CAP theorem and the design impact it has in distributed systems. How a system that ensures Availability has to sacrifice strong consistency over weaker levels of consistency. Eventual consistency although providing the weakest level of consistency, also imposes fewer constraints for reliable communication. When imposing stronger consistencies we have FIFO order that ensures that all the messages from the same node are delivered in the order they were issued. TCP offers FIFO ordering in the delivery of its stream of bytes.

Finally, we have seen that causal consistency, is the strongest level of consistency that can achieved, using weak consistency models. It is known that we can achieve reliable causal communication by building a causal partial order enforcement on top of TCP. However, we propose an alternative that reduces the amount of acknowledgments used in TCP, while providing the reliability need for causal consistency systems.

In the following chapters we will present our contributions of this thesis, starting with the models. We will propose the novel reactive error recovery mechanism that uses its knowledge of the network to predict if a message is lost or delayed.

Chapter 3

Theoretical Models

Contents

In this chapter we present the work done in order to model latency delivery of messages. At the same time we propose a reliable message communication mechanism that uses our model and that reduces the use of acknowledgments. We start by defining the system model we base our equations on. Later on we introduce step by step the reactive error recovery mechanism. Along the way we present equations that give the probability that a message is ready to be delivered in all the nodes in the network using two different partial orders, FIFO and Causal. Finally, we introduce the reactive error recovery mechanism that uses the aforementioned equations in order to bound the number of errors that go through the mechanism as false positives[1].

3.1 Notation

We will use the following notation during all this chapter, unless it is specified otherwise. Table 3.1 summarizes such notation.

Name	**Description**
n	Number of Nodes
r	Rate of Messages
λ^{-1}	Exponential Distribution mean value
x_m, α	Pareto Distribution parameters
X	Sending one message to another node. Sender point of view.
R	Sending one message to another node. Receiver point of view.
V	Visibility of one message that was sent to all the nodes.
F	Message enforcing a FIFO Partial Order.
C	Message enforcing a Causal Partial Order.
D	Message being ready for delivery.

Table 3.1: Summary of the notation used in the equations

n determines the number of sent messages in the scenario. A node can either send or receive a message, and after receiving a message if all dependencies are fulfilled then the message can be delivered.

[1] Are those messages that were assumed to be lost and asked for retransmission, but turn out to be just delayed. If untreated, this is a cause of duplicated messages

The rate of messages is defined with r. The rate of messages units are messages issued by the same time unit defined by the latency model. Thus, if the latency model is an exponential distribution with a mean of $50ms$, and the rate of messages is 1, then the rate means that 1 message, in average, is sent every millisecond.

The latency models parameters are defined accordingly to the specified latency model. In our case we propose a mixed distribution of Pareto and exponential distributions as latency model as seen in the PBS work by Bailis et al, [11]. This model is defined by four parameters, (1) λ^{-1} that is the mean value of the exponential distribution, (2) x_m in the scale of the Pareto distribution and also represents the minimum value that the distribution takes, (3) is the shape, a Pareto distribution parameter, and (4) p is the proportion of Pareto distribution in the mixed distribution, thus $1 - p$ will be the proportion of exponential distribution.

We have named the random variables used in every function so that they help represent the probability that they are giving. The random variable X, is used when a message is sent to just another node and the probability is computed from the senders point of view if the message has been received. R is used when a message is sent to two nodes and the receiver of the message asks the probability that the other node has also received such message. V is used for the visibility of a message, from the point of view of the sender. If it is the visibility from the point of view of a receiver, it will be noted as VR.

When the message is enforcing a type of partial order, it will be noted with an F for FIFO or a C for causal, and by default they represent the probability that a sent message can be delivered in another specific node.

Combinations of the letters like VF or FR should be read like, visibility with FIFO partial order enforcement, and FIFO partial order enforcement from the point of view of a receiver of the message.

3.2 System Model

We use the following abstraction model for all the following set of equations. We will be using the same notation for all the equations unless its said otherwise.

We assume a system with (1) n homogenic nodes, where a node is a network object capable of sending and receiving messages, and without storage or memory limitations. (2) All nodes have the same probability of issuing a message, and the system's global rate of messages is r.

(3) Nodes use a broadcast mechanism to transmit a message to the other $n - 1$ nodes. (4) All

communications follow the same latency model, which (5) does not change. (6) The latency issued to each message is independent from one another. While this does not hold for some type of errors (burst errors, link down, node down), it does for a typical behavior. Finally, (7), we assume that messages are not lost. The latency distribution models the loss of messages with arbitrarily large latencies.

3.2.1 Latency Model

Latency models, represent the different latency values that the messages can experience when sent from one node to another. $P(X \leq t)$ gives the probability for a given latency model that the sent message has been received before time t.

The first latency model that we use for our system is an exponential distribution in which the λ^{-1} represents the mean time of communication, as seen in Equation 3.1. Other latency models could be used like Equation 3.2 which we will be using in Section 3.6 to model a production-like latency scenario, in which the Pareto distribution models the bulk latency time and the exponential models the tail.

$$P(X \leq t) = 1 - e^{-\lambda t} \tag{3.1}$$

$$P(X \leq t) = p(1 - e^{-\lambda t}) + (1-p)(\frac{\alpha x_m^\alpha}{t^{\alpha+1}}) \tag{3.2}$$

Equation 3.2 is a mixed distribution between the Pareto distribution; α being the shape, and x_m being the scale, which marks the lower bound in the distribution's range; and the exponential distribution, with λ^{-1} being the mean value, while p is the proportion of exponential distribution. From Figure 3.1, the parameters that generate the CDF[2] for both the exponential and Pareto distribution are such that they have the same mean value, and a relatively close median value. Nonetheless, they are two different distributions with different profiles.

3.2.2 Visibility

Visibility of a message can be defined as when a message has been received by all the nodes it was sent to. Equation 3.3 shows the probability that $n-1$ messages that were sent at the same time,

[2] CDF: Cumulative Distribution Function

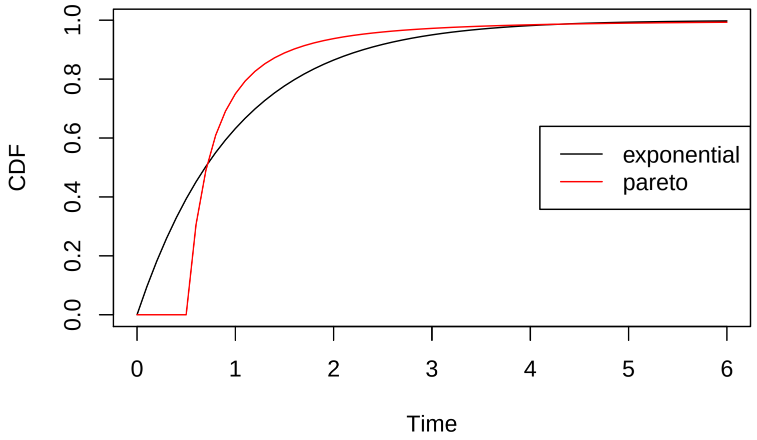


Figure 3.1: CDF of a Pareto and Exponential distributions with parameters $\lambda = 1$, $x_m = 0.5$ and $\alpha = 2$.

arrive to their destination before time t after they were sent. Recall that n is the total number of nodes, thus $n-1$ nodes are the receivers, plus 1 sender.

$$P(V \leq t) = \prod_{i=1}^{n-1} P(X_i \leq t) = P(X \leq t)^{n-1} \tag{3.3}$$

They are equal because each X_i uses the same latency model and t is equal for all $n-1$, and all events are independent from each other.

Figure 3.2 shows graphically what visibility is. $P4$ sends a broadcast message to all other processes and the visibility is when the last one of them has received the message. If one of the messages were to be lost, then there would be no visibility, as one of the processes would not have received the message.

3.3 Equations

In this section we present the equations that give the probability if a sent message has been received. However, we present several variations of the equations depending on:

1. The message is sent in a peer to peer communication.
2. The message is broadcast from one node, to many nodes.

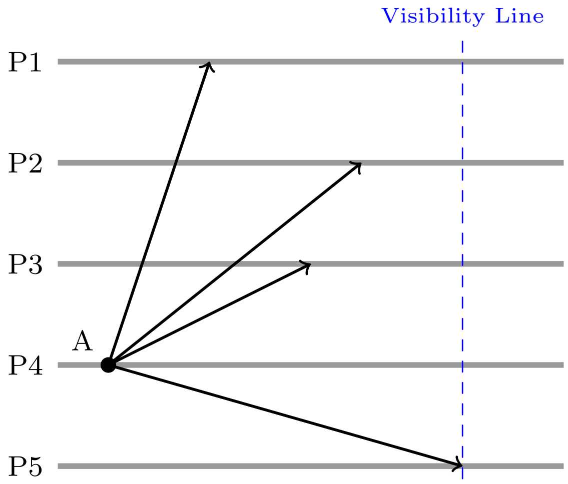


Figure 3.2: Diagram definition of the message communication's visibility.

3. From the point of view of a receiver, about the probability that other receivers got the message, either with one receiver or many.

4. The probability that a message is visible in all the nodes it was sent to. The use of partial orders will be introduced in later section. For this one, we only send one message that has no dependencies, thus reception and delivery coincide.

3.3.1 Sending to 1 or P2P

The first equations that we will see, are the ones to compute the probability that a message has been received by another node before time t. We consider that the time t starts when the message has been sent. Figure 3.3 shows an example of a message being sent from P4 to P2, and the time between sending and reception is considered as the transmission latency time. Considering an exponential distribution latency model, the probability that a message has been received before time t, is equal to the exponential distribution CDF as seen in Equation 3.1.
Figure 3.3 illustrates a message that was broadcast from $P4$, and if it was received by $P2$ before t. This is linked to the probability from Equation 3.1.

3.3.2 Sending to n-1

When sending the message to the all the nodes in the group, the probability that the message has been received by all the nodes, before time t, is equal to $P(max(X_1, X_2, \ldots, X_{n-1}) \leq t)$ where

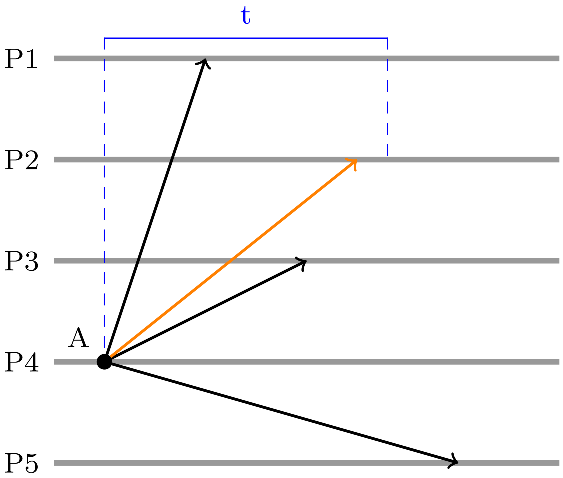


Figure 3.3: Diagram that describes how a node send a message, and how such message has arrived to a specific node before t.

each X_i is a random variable from the same latency distribution. This max can be converted to $P(X \leq t)^{n-1}$. This also represents the probability that a message is visible by all the nodes in the group as seen in Figure 3.4.

If we consider an exponential distribution as the latency model, then Equation 3.4 gives the probability that a sent message is visible, received by all the nodes.

$$P(V \leq t) = (1 - e^{-\lambda t})^{n-1} \tag{3.4}$$

3.3.3 Reception from n-1 (for n = 3)

The aim of this equation is to get the probability of a message being received by a node, if it was sent to more than one replica, and at least one replica has already received it. Figure 3.5 exemplifies this situation.

Computing the reception of a message is a bit trickier from the point of view of the receiver, as the time at which the message was sent is unknown to the node. Equation 3.5 yields such probability, and the proof of such equation can be found in Appendix A.2. The receiver can give such probability, because (1) it knows that the messages were sent at the same time, and (2) it knows the latency

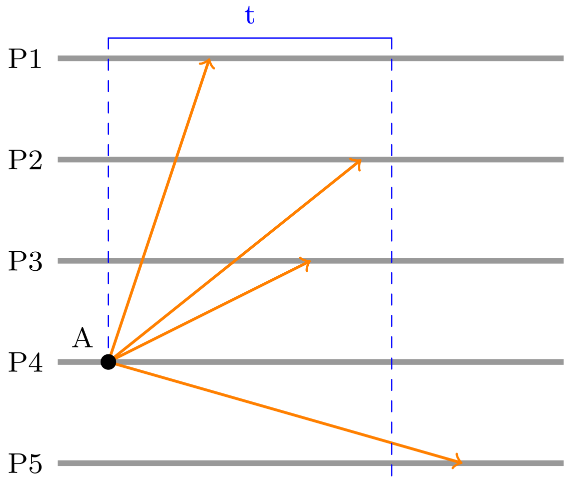


Figure 3.4: Diagram that describes if a message sent from a node is visible after time t.

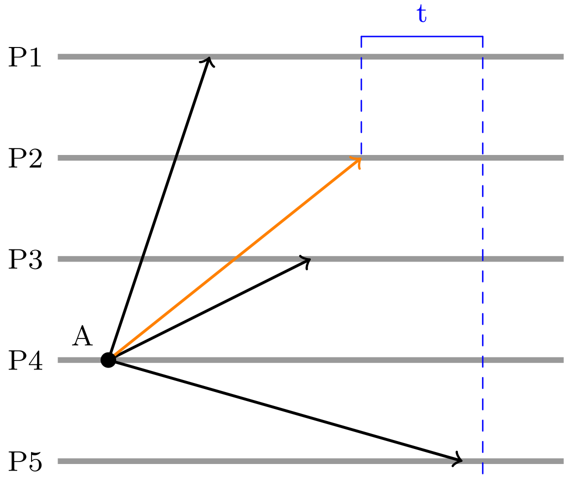


Figure 3.5: Diagram where a node has received a message, that was broadcast, and wonders if another node has received the message after a time t.

models from the sender to the other receiver.

$$P(R \leq t) = 1 - 0.5e^{-\lambda t} \tag{3.5}$$

3.3.4 Reception from n-1

The same way that we saw the equation for visibility from the point of view of the sender, we can also give the probability from the point of view of the receiver of a message. Equation 3.6 yields the probability of being the last receiver of a message that was sent to a group.
In a scenario with n nodes, one of them sends a message to the other $n-1$, and the reception of the message is only with these $n-1$ nodes.

$$P(VR \leq t) = e^{\lambda t}(1 - (1 - e^{-\lambda t})^{n-1})/n - 1 \tag{3.6}$$

Equation 3.6 (Proof A.1) is a generalization of Equation 3.5, and we can see it Appendix A Proof A.5[3] by comparing it with Equation 3.6 with $n = 3$.

In this section we presented two basic equations. One that models the reception from the sender's point of view and another from the reception's point of view. We have extended both of them to include the visibility of the message. Now we can use these equations to model more complex scenarios with partial order enforcement.

3.4 Partial Orders

In this section we use the previously defined equations in order to get the probability of a message being delivered under the enforcement of FIFO or causal partial orders.

3.4.1 FIFO

Recall from Chapter 2.2.2 that messages that are under FIFO ordering constraints can only be delivered if all the previous messages from the replica the message comes from have already been delivered. We now present different equations that compute the probability of a sent message being ready to be delivered to (1) a specific node, (2) all the nodes in the group, and (3) in at least m of the n nodes in the group. We will also present the probabilities from the point of view of message receivers.

[3] Thanks to the ResearchGate community for their help.

Delivery readiness in a specific node of the group

In order for a message to be ready for delivery, all its dependencies need to be fulfilled. Equation 3.7 gives such probability by multiplying the probability that every previous message has been received by the other node. Furthermore, it adds Δ_i, which as can be seen in Equation 3.8, gives the time between message k and its $i - th$ dependency.

$$P(F_k \leq t) = \prod_{i=0}^{k} P(X_i \leq t + \Delta_i) \tag{3.7}$$

$$\Delta_i = t_{s_k} - t_{s_i} \tag{3.8}$$

The probability that a message is ready to be delivered can also be given from the point of view of a replica that received the messages. Equation 3.9 gives the probability of a message being ready for delivery, from such point of view.

$$P(FR_k \leq t) = \prod_{j=0}^{k} P(R_j \leq t + \Delta_j) = \prod_{j=0}^{k} (1 - 0.5e^{-\lambda t + \Delta_j}) \tag{3.9}$$

$$\Delta_j = t_{r_k} - t_{r_j} \tag{3.10}$$

The difference between Δ_i and Δ_j is that the former uses the time of sending the message, while the latter uses the reception time. Both of them use the reference time that they have and could not use the other one as, Δ_i is from the point of view of the sender and ignores the reception time in the other nodes, and the same thing applies to Δ_j that ignores when were the messages sent.

Delivery readiness in all the nodes of the group

In order for a message to be ready for delivery and visible, as seen in Section 3.2.2, all nodes in the group must have received the message. Equation 3.11 gives such probability.

$$P(VF_k \leq t) = \prod_{j=1}^{n} \prod_{i=0}^{k} P(X_{ij} \leq t + \Delta_i) \equiv \prod_{i=0}^{k} P(X_i \leq t + \Delta_i)^{n-1} \tag{3.11}$$

The approach in Equation 3.11 is to multiply the probability that every message from the sending node has arrived to every other node in the group. The first approach would be useful if the probabilities for different nodes were different. However as our variables are i.i.d[4], both statements of Equation 3.11 are equivalent.

[4] i.i.d stands for independent and identically distributed.

As done in the previous section, we can also give the probability that a message is ready to be delivered with visibility, but from the point of view of a receiver of such message. Equation 3.12 gives such probability.

$$P(VFR_k \leq t) = \prod_{i=0}^{k} P(VR_i \leq t + \Delta_j) = \prod_{i=0}^{k} ((n-1)^{-1} \cdot e^{\lambda(t+\Delta_j)} \cdot (1 - (1 - e^{-\lambda(t+\Delta_j)})^{n-1})) \quad (3.12)$$

Delivery readiness in M out of the N nodes of the group

A quorum configuration is defined by three parameters, N, W, R. N is the number of nodes in the quorum, while W is the number of nodes we will wait an acknowledgment from when sending a write message and R is analogous to W but with the read messages. The following equations represent the probability that a message that was sent to N nodes has been received by M of those, which would fit either R or W depending on the message being a write or read message.

To begin with, we will define this probability with a binomial distribution, where the number of experiments is the number of nodes and the number of successes is the number of nodes we want to receive the message from. Equation 3.13 is this binomial construct where the probability of success is defined by Equation 3.7.

$$P(F_k \leq t, N = m) = \binom{n}{m} P(F_k \leq t)^m (1 - P(F_k \leq t))^{n-m} \quad (3.13)$$

A less strict definition of the quorum probability, where the number of success is just the lower bound, would be Equation 3.14. This probability also includes the cases where more than m successes have happened before t time.

$$P(F_k \leq t, N \leq m) = \sum_{i=m}^{n} P(F_k \leq t, N = i) \quad (3.14)$$

An analogous construct can be defined to get the probability from the reception point of view. In such case, we would have to change the probability of success from Equation 3.7 to Equation 3.9.

3.4.2 Causal

Recall from 2.2.2 that messages under order enforcement can only be delivered once all the messages that “happened before” have already been delivered, and that under causal ordering this “happened before” might include messages from other replicas.

We will now introduce a set of equations that give the probability of a message being ready for delivery under two situations. First, ready to be delivered in one replica, and second, in all the replicas.

Delivery readiness in a specific node of the group

We define an equation that gives the probability of a specific message k being ready to be delivered in one replica. In order for the message to be delivered, all the previous messages to k will also need to be received. Furthermore, we differentiate if the i was issued from the same replica the is issuing the k message of if it was issued from another replicas. In the former, we will use Equation 3.1 and in the latter Equation 3.5. In either case, we add a Δ_i time that ensures that older messages have a higher probability of being received, while younger ones have a lower probability. Finally, the product of all the dependent messages from k gives the probability that all past messages were received by another replica, and therefore message k can be delivered. $Deps(k)$ provides the id of all the dependencies of message k.

$$P(C_k \leq t) = \prod_{i \in Deps(k)} \begin{cases} P(X_i \leq t + \Delta_i) & \text{if } i \text{ is local} \\ P(R_i \leq t + \Delta_i) & \end{cases} \tag{3.15}$$

This product grows with t from zero to one, and at time zero, the probability of the message being ready for delivery is zero as the message k cannot be received. This highly depends on the latency model in use and, for example, in a Pareto distribution any time before the scale parameter (x_m), would also be zero due to the distribution's range.

In the case of an exponential distribution latency model, the equation for causality is:

$$P(C_k \leq t) = \prod_{i \in Deps(k)} \begin{cases} 1 - e^{-\lambda(t+\Delta_i)} & \text{if } i \text{ is local} \\ 1 - 0.5e^{-\lambda(t+\Delta_i)} & \end{cases} \tag{3.16}$$

Delivery readiness in all the nodes of the group

To define an equation that gives the probability of a message being ready for delivery in all the nodes of the network, we have to use the notion of visibility, previously introduced in 3.2.2. The equation is very similar than the one previously described, but we have to use the visibility forms for transmission and receptions of messages in Equation 3.3 and Equation 3.6.

$$P(VC_k \leq t) = \prod_{i \in Deps(k)} \begin{cases} (1 - e^{-\lambda(t+\Delta_i)})^{n-1} & \text{if } i \text{ is local} \\ (n-2)^{-1} e^{\lambda(t+\Delta_i)} (1 - (1 - e^{-\lambda(t+\Delta_i)})^{n-2}) & \end{cases} \tag{3.17}$$

In this section we have introduced the equations that define the probability of a message being ready for delivery in both FIFO ordering and causal ordering. This concludes the first of this thesis contributions, a model that yields the probability that a message is ready for delivery under different partial orders, causal and FIFO; with different latency models, exponential and Pareto distribution, and a mix of both. However, these models have two big inconveniences, first as the number of messages grows, and as the value of k becomes bigger, the number of messages that have to be checked also grows, without a bound. Second, the values needed for Δ_i requires a system or simulation in order to get the probability, as each Δ_i value between messages is different for each message tuple of messages, sender-receiver. Both issues can be solved by generalizing the models, with the rate of messages.

3.5 Generalization

In this section we generalize the previously defined FIFO and Causal order models. This generalization solves: (1) the ever growing number of operations to be processed; (2) the ties between the probabilities and the simulation. This is achieved by, first limiting the number of messages that have to be checked to the q and t most significant messages. Second, the rate of messages is used to define a new Δ_i. This generalization separates the equation from a specific simulation or experiment, and yields an approximation that is good enough. After defining these two changes, we need to update the equations.

Finally, after generalizing the equations, we get the probability of any message being ready for delivery, and not a specific message.

3.5.1 Window of events

The main reason for limiting the number of events is that it is costly to compute the probability that all past events have been received as the number of events grows unbounded. Furthermore, the older an event is the lower impact it has in the overall probability. This is because the probability, that a message is received, grows to one as $t \to \infty$. We limit the number of events according to two factors: the visibility time of an event with a certainty degree, $t(p)$; and the global rate of events of the system, r. Equation 3.18 gives $t(p)$, the waiting time at which a sent message has been received by all $n-1$ other nodes with a probability of p within an exponential latency model. With this time value, we can use the rate of events to calculate the window of events $t(p).r$, which is the maximum number of previous events that the probabilistic equations should take into account.

The proof of Equation 3.18 can be found in Proof A.3.

$$t(p) = \frac{-ln(1 - \sqrt[n-1]{p})}{\lambda} \tag{3.18}$$

For instance, in a system with 10 nodes and a λ of 0.3, $t(p)$ such that a message has been received by all nodes with a probability $p = 0.99$ is equal to 22.66. Assuming a rate of events $r = 30$, then we have to check the last 680 events. However, with a system with 5 nodes, and a rate of $r = 5$, we would have to check the last 100 events.

3.5.2 Rate op for Delta

We consider that the workload follows one of the following three distributions, exponential, uniform or normal. This random variable is the time elapsed between the issuing of two messages. The rate of operations is the mean value for any of the three distribution configurations.

When considering that the events are distributed with a period equal to the rate of messages we can define Δ_i as the time between i messages. Thus given that Δ_0 equals zero and Δ_k an arbitrarily large number. This contrasts with the previously defined Δ_i in which Δ_k was zero and Δ_0 the arbitrarily large number. This change makes sense as we changed the reference point between both cases. In the former, we ranged from zero to k, and in the latter we range from $-w$ to 0, where w is the window of events to check and zero, because they are the events that we need to check if they are ready for delivery.

With the two steps, window of messages and Δ_i completed, we can now update the previously described models, and generalize them.

3.5.3 FIFO

As previously described with the non generalized equations, we have three major cases we want to deal with. First, if a message is ready to be delivered in a specific node out of the whole group. Second, if the message is ready to be delivered in all the nodes. And finally, if the message is ready to be delivered in a subset of nodes of the system. Also, as was previously defined, we had the equations from the point of view of the sender and the receiver. However, due to the similarities in the changes in one of the equations, the changes for the other equations are done in the same fashion.

Generalizing: Sending to a specific node

The generalization process of the equation removes the need to specify which specific message we are referring to, as the Δ_i values have been generalized.

Furthermore, we just need to check the reception of the most significant messages, as determined by the window of events, $\lceil t(p) * r \rceil$. This produces the changes from Equation 3.7 to Equation 3.19.

$$P(F \leq t) = \prod_{i=0}^{\lceil t(p)*r \rceil} P(X_i \leq t + \Delta_i) \tag{3.19}$$

Δ_i is defined as $\frac{i \cdot n}{r}$, where n is the number of replicas, and r is the global rate of messages, as such the local rate of messages is r/n. Furthermore, it is assumed that the workload for each replica is the same, otherwise Δ_i would have to be changed accordingly.

3.5.4 Causal

The same changes used to obtain the FIFO generalization can be applied to the causal case. First we limit the number of messages by using a window of events that suits us. Second we generalize the time between incoming and outgoing events by using the new definition of Δ_i.

Recall that received messages were those that arrived to the replica but could not be applied. A received message can only be applied, delivered, if and only if all its dependencies have been already delivered.

We can consider that incoming and outgoing received events are uniformly distributed with a period equal to the global rate of messages. However, in contrast to the FIFO ordering where only the events from a replica had dependencies, here events from other replicas can also be the dependencies. For this reason, events that happened in other replicas are more likely to be concurrent if they happened at a close time, however they are more likely to be dependent the longer the time difference they have.

$P(D \leq t)$ gives the probability that a received event can be delivered before t, which means, that all its precedent messages have been already delivered.

We obtain $P(D)$ by using information available in the local node, and not contacting any other node. Algorithm 1 gives such probability by counting the number of remote messages whose delay falls between reception and delivery is smaller than or equal to t, and dividing such number by the total number of remote events.

To obtain Equation 3.20, we combine $P(D)$ with $\overline{P(R)}$, so that we get the probability "dependent and not received", and then we negate it $(\overline{\overline{P(R)}P(D)})$. We obtain $1-(1-P(R))P(D)$ which is the probability that a remote message is either not deliver to the node[5] or received on the other node.

$$P(C \leq t) = \prod^{i<t(p)\cdot r} \begin{cases} P(X_i \leq t + \Delta_i) & \text{if } i = 0(mod\ n) \\ (1-(1-P(R_i \leq t+\Delta_i)) \cdot P(D \leq \Delta_i)) & \text{else} \end{cases} \tag{3.20}$$

[5] As such, is not part of the dependencies of the considered message.

Algorithm 1 $P(D)$

```
function P(D)($delivery$, $reception$, $t$)
    $len = number\_rows(delivery)$
    $n = number\_columns(delivery)$
    $count = 0$
    for $i = 0$ to $i \leq len$ do
        for $j = 0$ to $j \leq n$ do
            if $(delivery[i, j] - reception[i, j]) \leq t$ then
                $count = count + 1$
            end if
        end for
    end for
    return $((count - len)/((n - 1) * len)$
end function
```

The $i = 0(mod\ n)$ condition – i equals 0 modulo n – allows the separation between local and remote events. Remember that we assume that the workload between nodes follows either an exponential, a normal or a uniform distribution.

Note that the changes to generalize the causal equation were made for getting the probability that a message is ready to be delivered to a specific replica. However, in order to have the probability that the message is ready to be delivered in all the replicas, $P(X_i)$ and $P(R_i)$ should be changed accordingly with the visibility equations.

This concludes this section, the generalization of the previous models, with a rate of operations that follows: uniform, exponential or normal distributions; and a key contribution of this thesis. So far we can get the probability that an event is ready to be delivered in one or many replicas, while enforcing either FIFO or causal ordering. Furthermore, we have presented a generalized version of the same equations that not only reduces the number of checked messages but also separates the equations from the specifics of a message to the general delivery status of the system.

We move now to another contribution of this thesis, a reactive error recovery mechanism that uses the aforementioned models. It reduces the use of acknowledgment messages, and sends negative acknowledgments to other nodes when it needs operations retransmitted.

3.6 Reactive Error Recovery

In this section we present the reactive error recovery mechanism, that uses the metadata of partial order enforcement to know when a message is missing and uses the previously defined equations to determine a waiting time threshold. When this time is over, an event is assumed lost and the mechanism will contact the original node for recovery. However, it is possible that we face false positives – i.e. messages that were assumed missing but were not – and in this section we will also introduce how to quantify them for a given waiting time.

3.6.1 Mechanism description

Once a message is received the system has to check if the message is deliverable since it may happen that there are previous messages that have not yet been received. Waiting before contacting the sender may be a good idea because the missing messages may arrive in between and contacting the original node may result in duplicated messages and network overuse. However, if the system waits too long and the message is lost, then the recovery mechanism is adding an unnecessary waiting time to the recovery process. So the trade-off that we are balancing is between the network overuse versus waiting time.

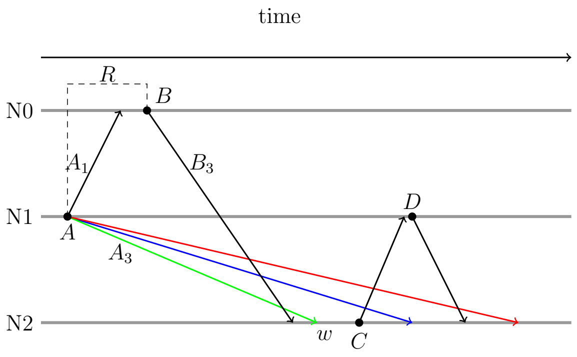


Figure 3.6: The three outcomes of a delayed message.

Figure 3.6 shows a message B which depends on message A since A was delivered to node N0 before B was issued. Therefore, when the message B arrives at N2, it is seen that the message A is not there yet. At this point, a retransmission timer (w) is started, and there are three possible outcomes. First, the message is received before the waiting time's out (green arrow). This outcome would not produce additional communication since the missing element has been recovered. If the waiting time runs out, N2 contacts N1 to get a retransmission of message A. However, if the original message A is received by N2 before (blue arrow) the resent message, the mechanism has produced an unnecessary request. Finally, if the original message A is received after (red line) the resent message, the mechanism successfully recovers from a missing element and obtains a delivery latency gain.

3.6.2 Probability of False Positives

A false positive message is defined as a message that was not lost but that the error recovery mechanism recovered, therefore duplicating a message and using unnecessary resources. An example can be seen in Figure 3.6, as the blue line outcome falls between message C and the return of message D.

Equation 3.21 gives the probability that given three events another event falls between the second and third. In the equation, there are two sets of events, A, and B, C, D. The equation gives the probability that the event A happens between the events C and D, which are issued in the following order B, C, D. And finally in order for the message A to be an FP it needs to intersect with the probability of the system being ready to deliver the message.

$$\begin{gathered} P(C_k \leq r) \cap P(B + r + w \geq A) \\ \cap P(B + r + w + C + D \leq A) \end{gathered} \tag{3.21}$$

The first part of the equation, $P(B + r + w \geq A) \cap P(B + r + w + C + D \leq A)$ can be obtained independently from the type of ordering that the system is enforcing, albeit it is only useful for causal ordering, as it only requires information for the latency model. Equation 3.22 gives the probability of message A arriving in between messages C and D with the exponential distribution as a latency model.

$$\begin{aligned} & P(A \geq B + r + w) \cap P(A \leq B + r + w + C + D) \\ & = \int_0^\infty \int_0^\infty \int_0^\infty \int_{b+w+r}^{b+w+r+c+d} \lambda^4 e^{-\lambda x} e^{-\lambda b} e^{-\lambda c} e^{-\lambda d} \, \mathrm{d}x \, \mathrm{d}b \, \mathrm{d}c \, \mathrm{d}d \\ & = \frac{3}{8} e^{-\lambda(r+w)} \end{aligned} \tag{3.22}$$

By using the defined Equation 3.21 and setting $P(C_k \leq r)$ as the latency models CDF, $1 - e^{-\lambda r}$ if the latency model is an exponential distribution, we can define the number of false positives that our system will have for a specific configuration.

Given a latency model with an exponential distribution and $\lambda = 1$, the FP surface function for the waiting time, w, and the time with the next event from another node once a message has arrived (time between A and B in Figure 3.6), r, is Equation 3.23.

$$f(w, r) = 3/8 * e^{-w-r} * (1 - e^{-r}) \tag{3.23}$$

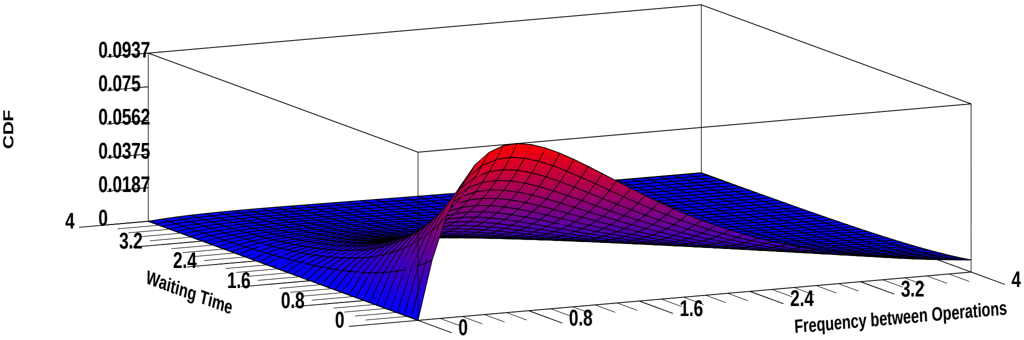


Figure 3.7: Surface plot of the false positive equation with exponential distribution latency model.

In order to locate the highest number of false positives for a given value of r, we obtain the partial derivative of Equation 3.23 over r, and find its solution. Later we use it to find the function of false positives dependent only on the waiting time.

$$\frac{\partial f}{\partial r} 3/8 * e^{-w-r} * (1 - e^{-r}) = -3/8 * (e^{r} - 2) * e^{-2r-w} \tag{3.24}$$

By finding the root of the Equation 3.24 we get its only real solution in $r = ln(2)$ and by substituting it in $f(w, r) = 3/8 * e^{-w-r} * (1 - e^{-r})$ we obtain $3/32e^{-w}$ which is the function that determines the number of false positives for a given waiting time in a system that used ordering metadata for error recovery, but didn't enforce it for message delivery. Such system does not need to delay the delivery of messages as a result of missing messages, but it keeps track of the messages that each node in the system has sent.

This result, $3/32e^{-w}$, gives us the probability of false positives that our mechanism will obtain for a given configuration. This allows us to tune the recovery of values more or less aggressively while being able to quantify the amount of resends, and false positives that the network will experience. Furthermore, for this example of $3/32e^{-w}$ we used a conservative upper bound with only the latency model, $1 - e^{-\lambda \cdot t}$. However, more accurate results for the number of false positives could be computed by using the equation of probability for each partial order.

In this section we have seen reactive error recovery mechanism. We have introduced the classification of operations depending on when they are received, as well as a mathematical upper bound on the number of false positives that we can expect for any give waiting time with an exponential latency model.

In this chapter we presented the work we have done to develop a mechanism that provides reliable communication without using acknowledgments. Furthermore, we have presented how we can quantify the number of false positives that the communications experience by tuning the waiting time before a message is asked for retransmission. To achieve this we proposed a set of equations that describe the probability for a message to be delivered, either from one replica to n replicas and either from the sender or receiver point's of view. Thanks to this, we have been able to describe the probabilities for a message that is enforcing a FIFO or causal partial order that it is deliverable in another replica.

In the following chapter, we present a simulator that we have built that allows us to run experiments and both validate the presented equations, an error recovery mechanism, and study the impact of the partial orders in distributed systems.

Chapter 4

Simulations

Contents

In this chapter we present the simulator we built as a sandbox to evaluate the equations previously described, as well as the reactive error recovery mechanism. All the code in the simulator was made using R language, which suited particularly well to generate the data and for further analysis of the results. The remainder of this chapter is structured as follows Section 4.1, which describes the inner workings of the simulator, Section 4.2, which describes the simulations that were run and that evaluates the behavior of the previously shown equations, and Section 4.3, which describes the reactive error recovery simulation results.

4.1 Structure

In this section we will detail how the simulator works. We divide its flow in four different steps, (1) input parameters, (2) data generation, (3) partial order enforcement, and (4) result analysis.

4.1.1 Input parameters

The input parameters are those configuration values that the simulation expects to create a new scenario to simulate. These are the parameters that we pass to Algorithm 4.1. We describe each of the parameters, but we might group some of them together.

- Total Number of operations: Positive integer number that defines the amount of operations that will be issued in the simulation. This is computed by $window\_op + 2 * padding\_op$.
  - `window_op`: The default value is 1000. This is the number of operations with which will will be computing the probabilities afterwards.
  - `padding_op`: The default a 10% of `window_op`, is assigned to `padding_op`. These are extra operations, that are place before and after the `window_op` to avoid a cold start and end for the simulations results.
- Rate of operations (rate_op): Positive float number that defines how many operations are expected to be send per unit of time. The unit of time is the same as in the latency model. Default value 10.
- Latency Model: Is the combination of distributions used to generate values to represent the communication latency.
  - Proportion of Exponential Distribution vs Pareto Distribution: Float between zero and one, where zero means all exponential distribution and one all Pareto distribution. Default value is 0. This means that the latency distribution is all exponential by default. The variable's name is `p_rpareto`.
  - Lambda (Exponential Dist.): Positive float number that is the inverse of the mean. The default is 1. The variable's name is `len_lambda`.
  - Scale (Pareto Dist.): Positive float number that indicates how spread is the distribution. There is no default value, as by default there is no Pareto distribution. The variable's name is `pareto_scale`.
  - Shape (Pareto Dist.): Positive float number, that has a big impact on the range of observed values, as with lower shape, the average maximum observed is smaller than for

larger shapes. There is no default value, as by default there is no Pareto distribution. The variable's name is `pareto_shape`.

- Number of replicas: Positive integer value that defines the number of replicas in the simulation. The default value is 5. The variable's name is `num_repl`.
- Weight per replica: Vector of positive floats that describes the weight of a replica at issuing operations. A replica with a weight of 2 when the other replicas have 1, will issue twice as many operations as another node. The variable's name is `weight_repl_op`.
- Ordering Policies: Different ordering policies that will result in different output matrices of times per operation.
  - Causal Order: Causal order is enforced for the delivery of operations. The function that we use to process the scenario is `matrix_causal_delivery_time_bt`.
  - FiFO Order: FIFO order is enforced for the delivery of operations. The function that we use to process the scenario is `matrix_fifo_delivery_time`.
  - No Order: No ordering is enforced. Reception time is also delivery time. The function that we use to process the scenario is `matrix_delivery_time`.
- Alternative latency models: `get_scenario` also allows to overwrite the latency model generator with the function in `f_tot`. While this is still very experimental and not very robust, when done correctly it allows to surpass some of the limitations imposed by the more rigid configuration parameters. The default configuration is NULL, which disables the function overwrite.
- Waiting Times: There are two waiting times that can be configured. Both of the waiting times are used in the reactive error recovery mechanism.
  - `c1_dt`: This is the retransmission timeout value. Since a message is received, and we notice that there are dependencies missing, until we send the retransmission message.
  - `c2_dt`: Since we send the retransmission message until we receive the retransmitted answer. By default we use the TCP RTO configuration for this timer, which depends on the latency model.
- Error Recovery: Reactive mechanism to recover operations which communication delay is too large, by using ordering information. `matrix_causal_delivery_time_bt_with_error_recovery`. For each simulation we only allow to use the error recovery mechanism in either FIFO or Causal. There is boolean named `b_fifo`, by default FALSE. If it is TRUE the error recovery will enforce a FIFO partial order, and if FALSE a Causal order.

```
get_scenario <- function (
  f_tot = NULL,
  num_repl = 5,
  weight_repl_op = rep.int(1, num_repl),
  rate_op = 10,
  window_op = 1000,
  padding_op = round(0.1*window_op),
  len_lambda = 0.1,
  p_rpareto = 0,
  pareto_shape = NA,
  pareto_scale = NA,
  c1_dt = NA,
  c2_dt = NA
)
```

Algorithm 4.1: Header of Get Scenario Function

4.1.2 Data Generation

In this section we will discuss more specifically how is the data generated and in which structures are they stored. In order to do this, we will use the names described in the previous section.
The data that is generated is:

- `t_op`: which contains the time at which each operation was sent. In order to generate this data we used the total number of operations ($window\_op + 2 \cdot padding\_op$) divided by `rate_op` to get the time the simulation lasts and we generate the event times for each operation following a uniform, exponential or normal distributions. However other distributions can be used to generate the timestamp of the events by using the parameter f_tot, name a function that accepts only the number of operations, and that should return the latency for these operations. However this is highly experimental and not very robust, as anything is accepted as a function, and can crash the simulator.

- `repl_op`: which contains the replica id from which each operation is sent. To generate this data we sampled from the pool of all replica ids and each of the ids had the same probability to be chosen. However, the probability can be changed with `weight_repl_op`, and new weights can be assigned to each replica.

- `dis_op`: which contains the communications delay to each replica for each operation. These values are generated using the latency model, and the only restriction is that the values have to be greater than zero.

Limitations

With the current version of the simulator we can generate configurations that are mostly homogeneous. While we can change the ratio of events generated by a replica, we do not allow multiple latency models. At the same time the same latency model needs to be ran for the whole simulation and we do not consider crash failures. Recall that our model allows arbitrarily large latencies, but no losses. We do not allow the change of number of replicas during the simulation or to change the rate of operations. However those features are needed in more general purpose network simulators like PeerSim [57], but for the task that we had at hand there was no need to build them. We have not focused on reproducibility, and while it is possible to set a specific seed when creating a new scenario, we do not implement it. However, it would be interesting to add this feature.

4.1.3 Partial Order enforcement

In the simulator there are two partial orders that can be enforced: Causal or FIFO; and the no order, which would be the time of generation of an operation plus its communication delay.

For each ordering policy a matrix is created with $Number\ of\ Operations \times Number\ of\ Replicas$ dimensions. Each matrix cell contains the delivery time of an operation to a replica according its dependencies.

4.1.4 Result analysis

In order to process a scenario with the simulator that we implemented, we use Algorithm 4.2. This requires the configuration that Algorithm 4.1 provides. After processing all the functions on the data generated, the simulator returns an R list which contains four elements. Each element is either a matrix of size $Number\ of\ Operations \times Number\ of\ Replicas$ or NULL, which means that partial order has not been processed with this configuration. By default all functions in Algorithm 4.2 would have a NULL value, however in this configuration we process the configuration with the following orderings: causal order, causal order with error recovery and no order enforcement. If the FIFO ordering function is left at NULL, it will not be processed.

```
  process_scenario <- function (
          config , % config from get_scenario()
          f_dc=matrix_causal_delivery_time_bt , % function that processes causal order enforcement
          f_dnc=matrix_delivery_time , % function that processes no order enforcement
          f_fifo=NULL, % function that processes fifo order enforcement
          f_dc_er=matrix_causal_delivery_time_bt_with_error_recovery , % function that processes error recovery
          b_fifo=FALSE, % in either fifo or causal depending on this
          show_time=FALSE % flag to print the time it takes to run the function
  )
```

Algorithm 4.2: Header of Process Scenario Function

One of the ways that we process the simulations, in order to compare them with the predictions, is to compute the probability that a messaged has been received with the different partial orders. We do this in two different steps. First, for each operation we calculate its delivery time (Algorithm 4.3). This is done by calculating the time difference between the sender of the message and the delivery time in the receivers and then take the max, in order to get the visibility. Once we have all the delivery time, we can compute the probability, by calculating the number of message that have been delivered before a time t.

```
get_operation_delivery_time <- function(m_data) {

  return(apply(m_data , 1, function(row) max(row - min(row))))
}
```

Algorithm 4.3: Header and function to compute the time of delivery for all operations in the simulation.

```
get_operation_latency_time <- function(m_data) {
  return (t(apply(m_data , 1, function(x) x - min(x))))
}
```

Algorithm 4.4: Header and function to compute the time of reception for all operations in the simulation.

```
get_operation_m_latency_time <- function(m_data, m=nrow(m_data) ) {
  return(apply(m_data, 1, function(x) sort(x)[m] - min(x)))
}
```

Algorithm 4.5: Header and function to compute the time of delivery for the m first nodes for each operation in the simulation.

There are other functions, like Algorithm 4.4 or Algorithm 4.5, that process time of the events differently. Algorithm 4.4 processes the reception time for each operation instead of the delivery time, whilst Algorithm 4.5 computes a quorum like latencies were can take the m largest values for each operation.
Now that we have seen how the simulator works, we will explain the simulations that we have done, and the results.

4.2 Equation Experiments

In this section we present the experiments that were conducted with the simulator. There are two goals in this experiments. The first is to validate the simulator with the results from the equations from the previous chapter. Therefore we will be checking if the prediction of readiness for delivery matches the simulation in different scenarios, and we will compute the difference between the two. The second goal is to observe the behavior of the reactive error recovery mechanism. This needs to be done in the simulator and not directly using the models, as the error recovery mechanism influences the latency model, as it recovers the found errors.

4.2.1 Statistical Error between Equations and Simulations

In the previous chapter we presented a set of equations to compute the probability that a message that enforces a specific partial order policy, can be delivered in other nodes t time after being sent. In this section we will present the experiments done with the simulator that we have defined in this chapter and we will compute the statistical error between the predicted value, from the equation, and the observed value, from the simulation. We run the simulations several times in order to obtain a more stable value, and we average the results from each simulation together. The number of iterations was decided by running shorter simulations with different iteration values and observing the standard deviation between the different runs. Figure 4.1 shows the variability of the simulation for the different number of iterations. This is why we run between 25 and 50 iterations for each simulation, because while running more iterations will yield a lower variability the increased cost in simulation time its not worth the improvement.

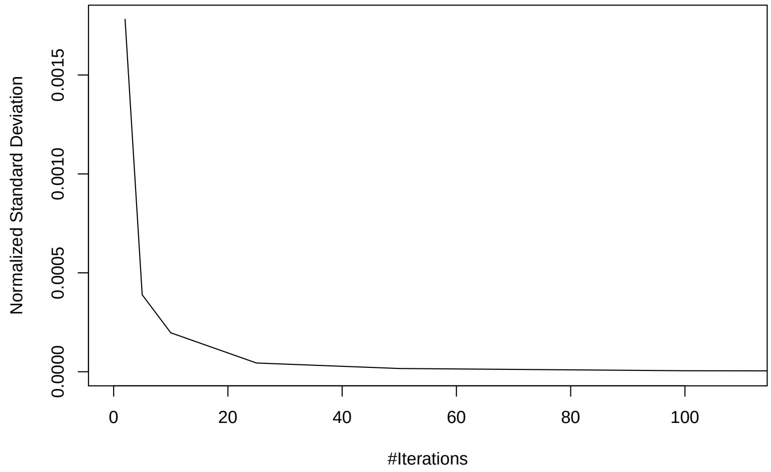


Figure 4.1: Normalized Variation of the simulations as the number of iterations in the simulator increases.

The same kind of trade-off appears when we decide the number of operations that will need to be run for each iteration. We tried with various configurations from 100 to 100000 operations, and we cease to see a significant improvement around 10000 operations. However the time it takes for the simulation does increase.

P2P FIFO

Figure 4.3 contains both the predicted and the observed values side by side, while the Figure 4.2 is the difference between them whose is 0.72%. The experiment configurations were the following: 5 nodes, exponential distribution with lambda 1 for latency model, and a rate of operations of 10.

P2P Causal

For calculating the error in Equation 3.16 we use the same configuration as in the previous experiment, but with a causal partial order, Figure 4.5 shows such results. Furthermore, the observed RMSE for this configuration is 0.73%, with a maximum divergence error between the predicted and the observed of 3.5%. Figure 4.4 shows the difference between the values observed in the simulation and the values predicted by the model.

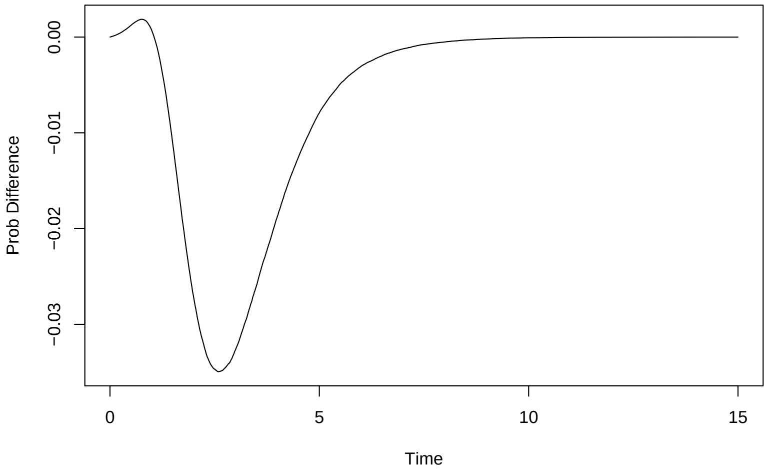


Figure 4.2: Difference between the model and the simulation for the FIFO partial order.

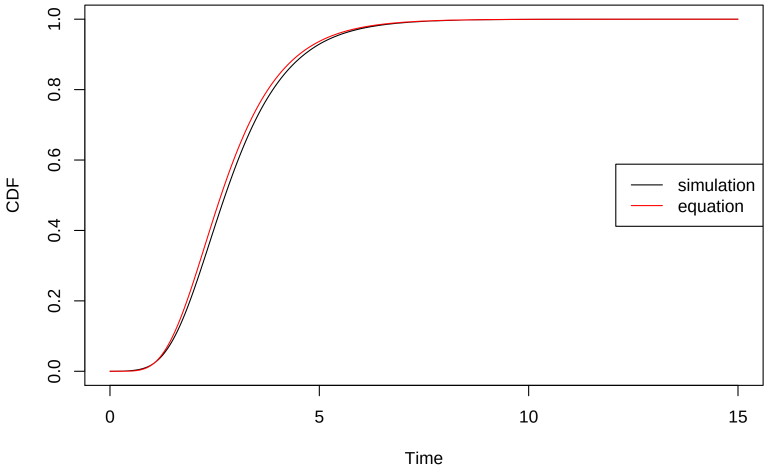


Figure 4.3: Comparison between the CDF of the model and the simulation of the FIFO partial order.

Quorum FIFO

The aim of Equation 3.14 was to give the probability that, a message that was sent to a group of n nodes, has been received by at least m of those. This equation becomes very useful when modeling

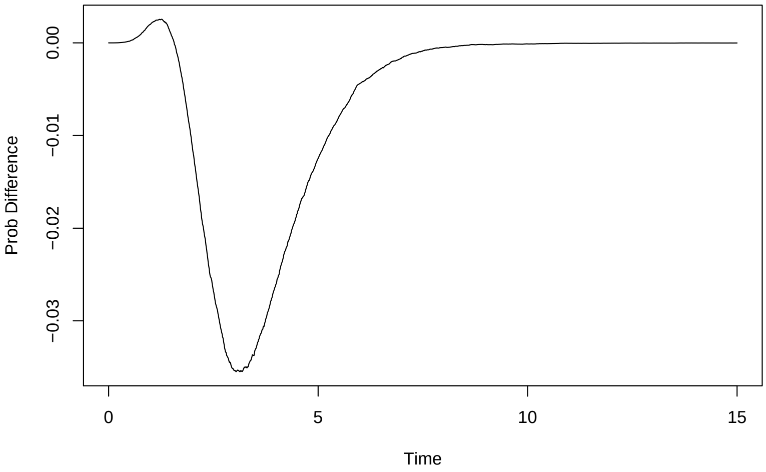


Figure 4.4: Difference between the model and the simulation for the Causal partial order.

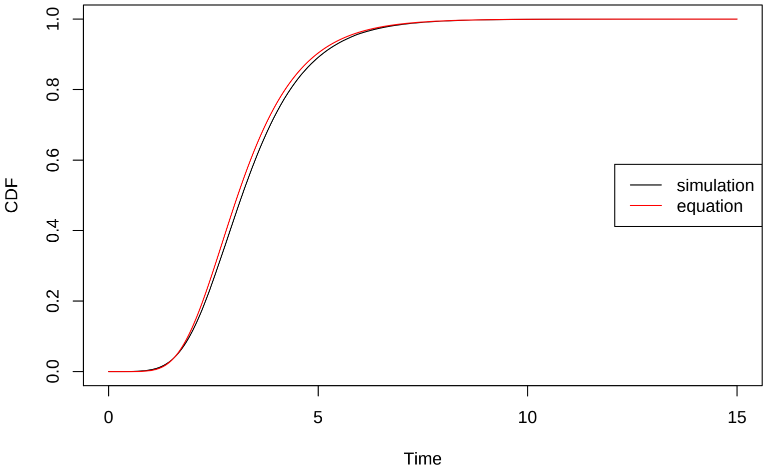


Figure 4.5: Comparison between the CDF of the model and the simulation of the causal partial order.

scenarios that involve quorum systems.

So, in this scenario the aim was to compute the error between the FIFO partial order quorum equation and the same simulation. Recall that quorum allowed to fine grain the description of how many nodes had received a message, from one to all. Figure 4.7 shows the superposition of the prediction and the simulation whilst on Figure 4.6 we have the difference between both. It has to be noted, again, that the range of error is between 1% and −3%, and even if it looks very aggressive in the plot, it is due to the axis size more than the significance of the error. If the Y-axis of the error plot had the same range as the plot, the error would be barely noticeable.

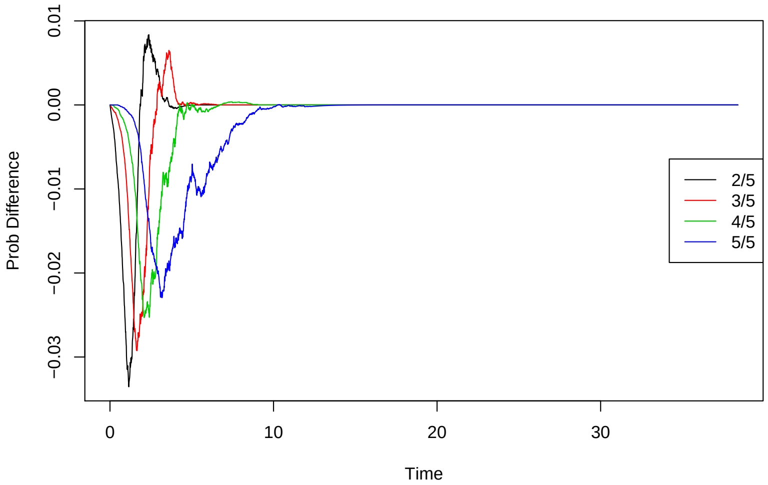


Figure 4.6: Difference between the model and the simulation for the quorum FIFO partial order.

Generalized and Non-Generalized vs Simulation

We show how different are the predictions, both generalized and not generalized, from the simulation, for both partial orders, FIFO and causal.

We start by viewing the FIFO partial order. Figure 4.2 shows the difference between the generalized model and the simulation, as well as, the non-generalized model and the simulation. To obtain the non-generalized we ran a simulation with the same parameters, to obtain the time at which the operations were generated and were received by the different nodes. We then use these times in the models to obtain the probability that the message is ready to be delivered.

Figure 4.9 shows the CDF for the probability that the message is ready to be delivered, but as the difference between the models and the simulation is very small, we can appreciate better the difference in Figure 4.8. We observe a maximum difference of 0.04 for the generalized model, and 0.03 for the non-generalized. With a maximum distance between the generalized and the non-generalized

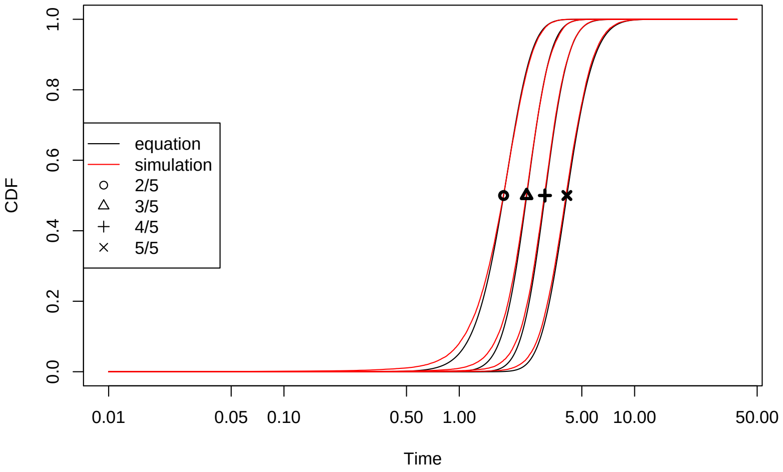


Figure 4.7: Comparison between the CDF of the model and the simulation of the quorum FIFO partial order.

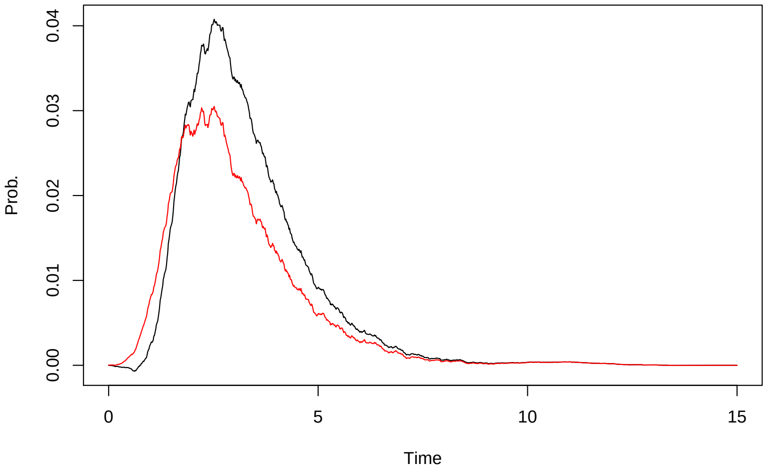


Figure 4.8: Difference between the non-generalized model and simulation, and the generalized model and the simulation for the FIFO partial order.

of 0.01

However when we observe the generalized and non-generalized model for causal partial order en-

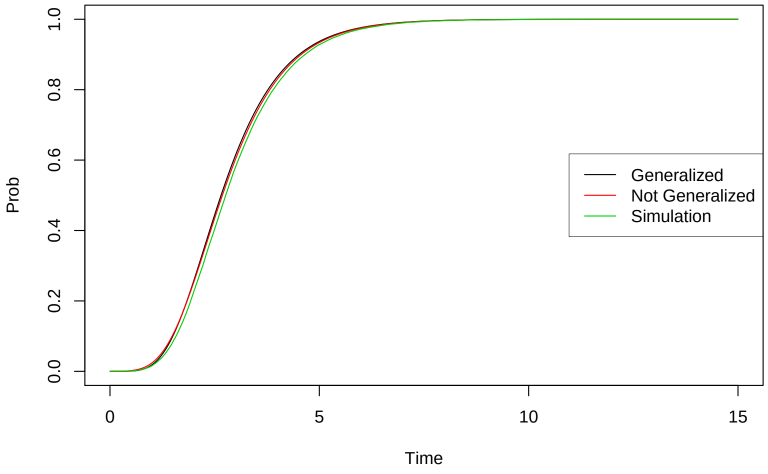


Figure 4.9: Comparison between the CDF of the generalized and non-generalized models and the simulation of the FIFO partial order.

forcement, we see, from Figure 4.10, that while the maximum difference between generalized model and the simulation is close to 0.06, for the non-generalized model it is closer to a 0.01.

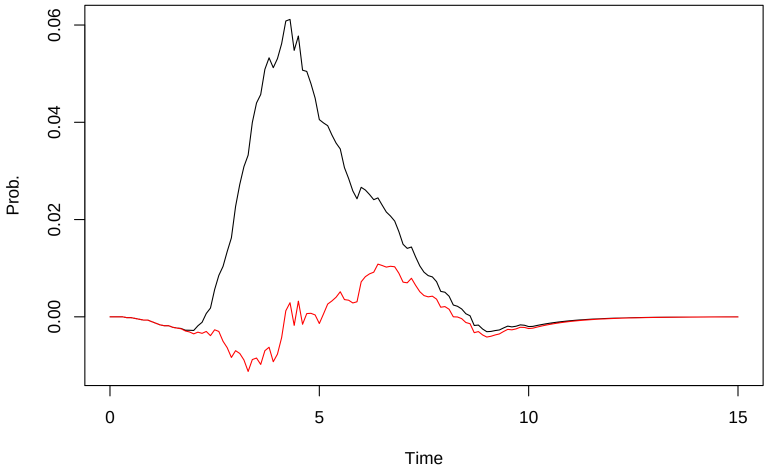


Figure 4.10: Difference between the non-generalized model and simulation, and the generalized model and the simulation for the causal partial order.

Figure 4.11 represents the CDF comparison of all three, generalized, non-generalized and simulation for the causal partial order model.

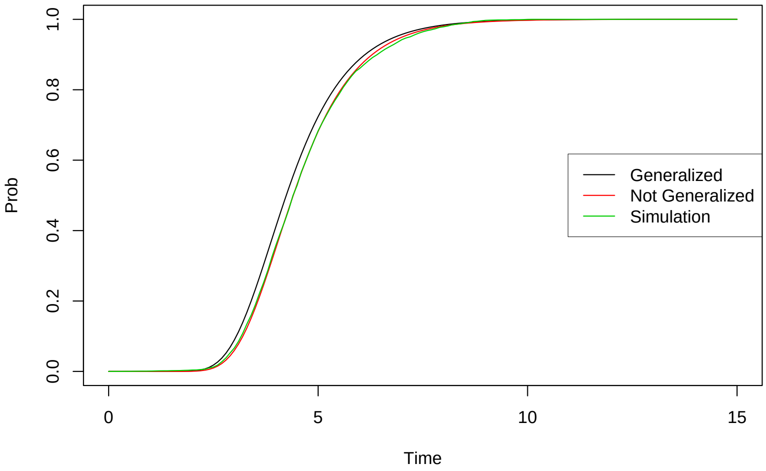


Figure 4.11: Comparison between the CDF of the generalized and non-generalized models and the simulation of the causal partial order.

We have seen the comparison between a generalized model and a non-generalized one. While the non-generalized model usually outperforms the generalized one, this is due to the fact that the times used are the same ones that come from the simulation. However the difference observed between both of them is small enough. That gives us confidence in the generalization process. With the generalized model we have no need for intermediate simulations in order to get the probabilities, and it gives the probability that the system is ready, instead of just a specific operation. At the same time with the generalized model we do not have to keep track of all past operations, just the most significant as defined by Equation A.3. At the same time, the comparison between the FIFO and the causal order models, we observe a smaller error with the FIFO due to its lack of dependencies in other nodes, that were more challenging to model in the causal model. In the generalized model we use a simulation to obtain the $P(D)$ probabilities, however once the vector of probabilities is obtained, it can be used for as many operations as needed. In a production system, this vector could be obtain using local information from previously delivered operations.

4.2.2 Impact of partial order enforcement

Figure 4.12 compares the probability that a message is ready for delivery under three different partial order enforcements: causal, FIFO and no order; with respect to the time. All three lines represent the CDF, therefore the probabilities that go from zero to one monotonically.

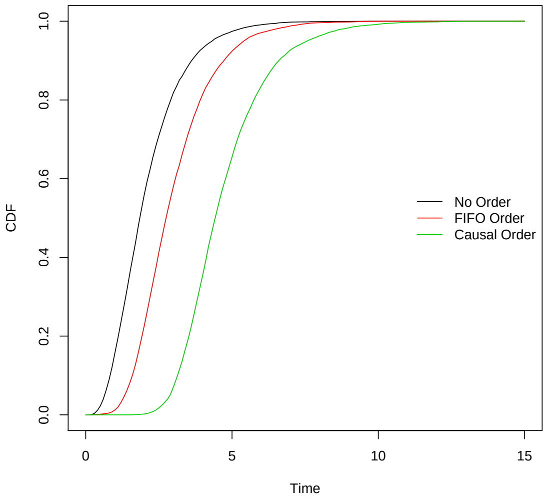
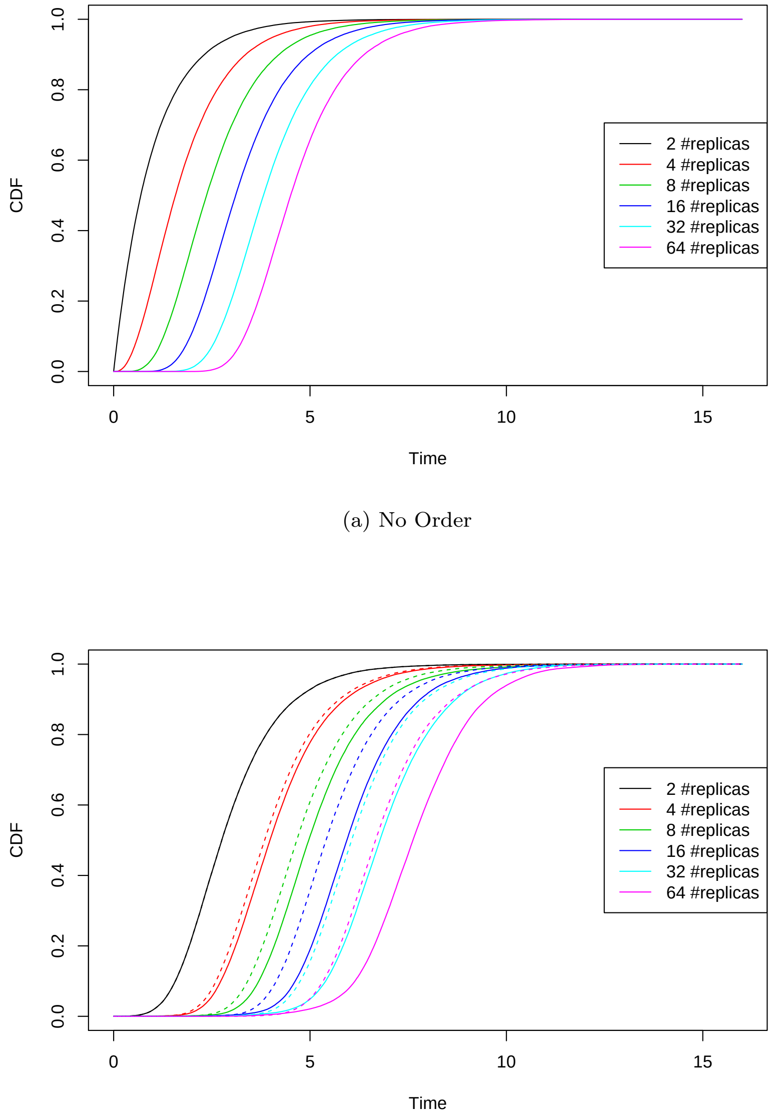


Figure 4.12: Comparison on the probability for a message to be ready for delivery under three different partial order enforcements. [The configuration for the test is a rate operations 10, exponential distribution with λ 1, and 5 nodes.]

Figure 4.12 also shows how the more strict a partial order is, the more delay it adds to the communication in order to provide message delivery.

Different Number Replicas

Another set of simulations is devised to check the impact that partial orders have under scenarios with different number of replicas. Figure 4.13 contains four different plots. Three of them compare four different number of replicas configurations: three, five, seven and nine replicas, for these orderings. Whilst the fourth one is a comparison between all three partial orders for a given number of replicas. Figure 4.13a shows the impact that different replicas have in a system where no order is enforced. We see how the larger the scenario is the more delay the operations experience. This is due the fact that we are showing the visibility time, and so the more replicas the larger the delay.
Figure 4.13b shows the comparison of both FIFO and Causal partial order side by side. When



(a) No Order

(b) FIFO and Causal Order

Figure 4.13: Different Number of Replicas. [The configuration for this test is: rate of operations is 10, and the latency model is an exponential distribution with λ 1.]

comparing the lowest number of replicas, 2, we see both of them behaving the same way. This is due to the causal order in a two-node system is equivalent to the FIFO order. However as we increase the number of replica we can notice how the causal order has a significant higher impact on the delivery latency than FIFO. Up to the point where FIFO with 32 nodes has a similar latency curve than causal with 16 nodes.

Different Rate of Operations

The aim of this set of simulations was to see the impact that partial orders have in a system with different rates of operations. The different rates of operations are 1, 10, 25, 50, 100. As seen with the previous comparison, with the different number of replicas, three of the four plots check all the different rate of operations for each partial order enforcement.

Figure 4.14a shows the impact that the growth in rate of operations has in a system where no partial order enforced. In contrast with Figure 4.13a, here as all tests have the same number of replicas it does not affect when we change the rate of operations. This causes the different results to overlap.

Figure 4.14b compares both FIFO and Causal partial orders. In contrast with Figure 4.13b the smaller the rate of operations the further away they find themselves, but as the rate increases, the cost of enforcing FIFO or causal order become indistinguishable.

Different Latency Model Parameters

Another more naive set of experiments was to change the latency model parameters of an exponential distribution to different λ values. This value, λ^{-1} represents the mean value that an observation of an exponential distribution will take, therefore the bigger the value the smaller the expected latency in a system.

We check the impact of causality with four different λ configurations, 0.1, 0.2, 0.5 and 1, which are 10ms, 5ms, 2ms and 1ms respectively.

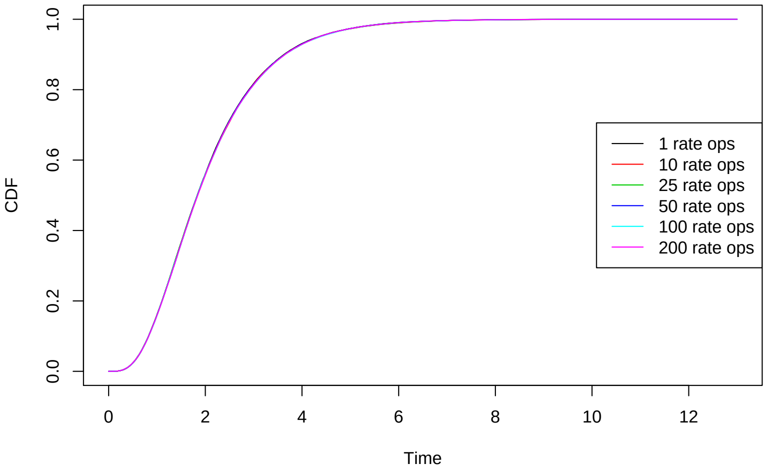
As expected, the faster a lambda is, the better it performs. We can see in all three partial orders configurations. We can see that given a specific lambda then the more restrictive a partial order is the more latency it observes. However we can normalize the observed latency across the different lambdas in order to observe the additional latency that comes from the partial order enforcement, apart from the latency from the latency model.

We will normalize the different observed quantile latencies by the excepted latency value of each configuration. In exponentials distributions this yields the exact same value.

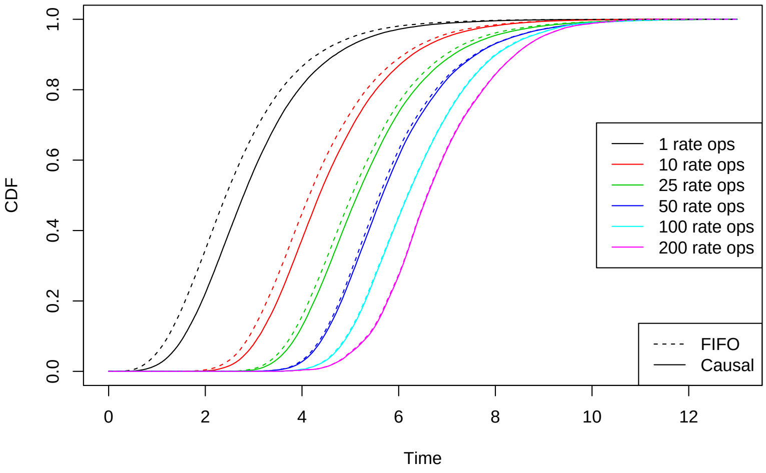
$$\frac{Q_1(p)}{\lambda_1} = \frac{Q_2(p)}{\lambda_2} \tag{4.1}$$

This shows a proportionality between lambdas in the exponential distributions[1]. And by normalizing

[1] A proof of this can be found in Proof A.6



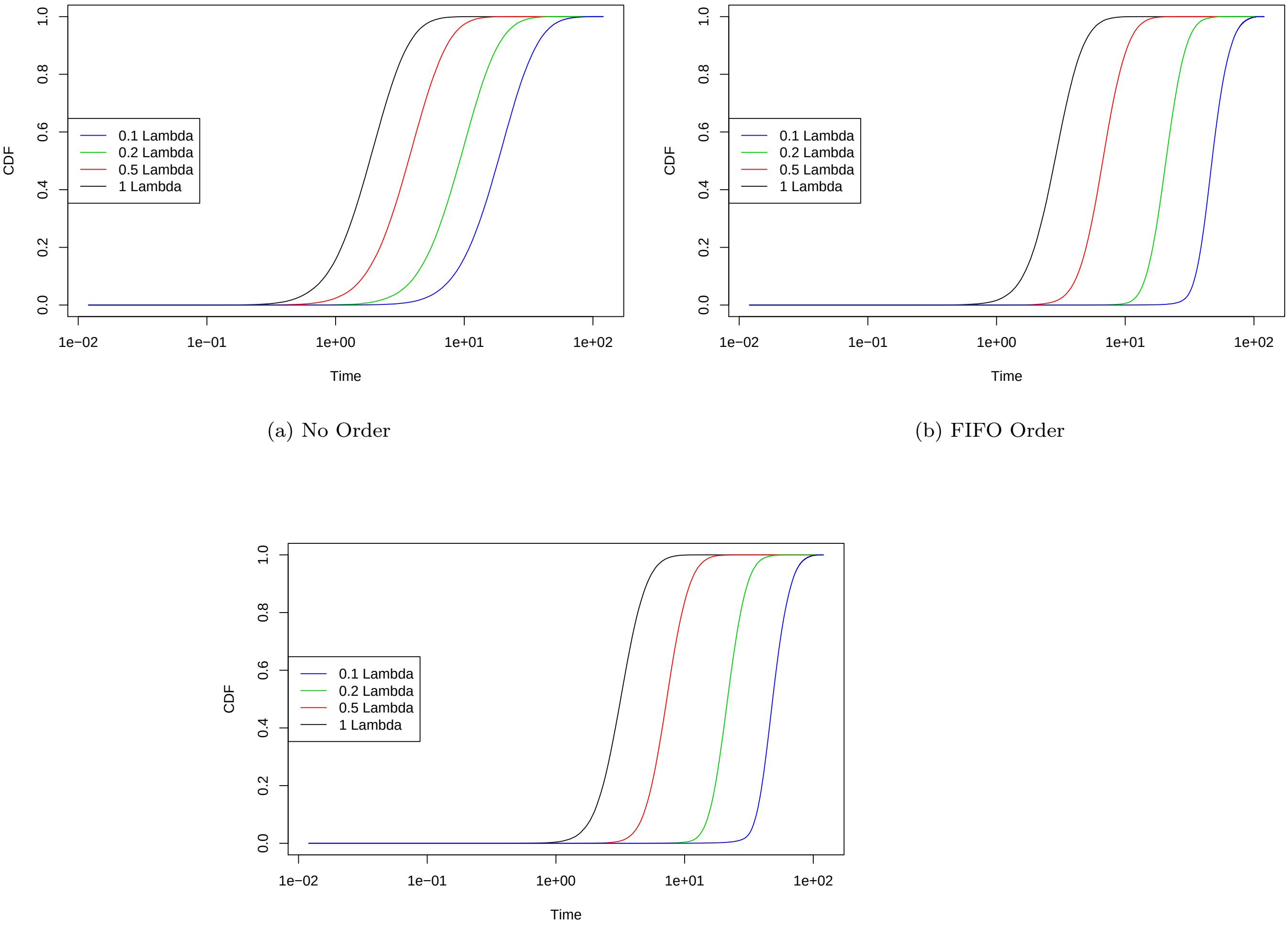
(a) No Order



(b) FIFO Order

Figure 4.14: CDF for different Rate of Operations. [The configuration for this test is: 5 nodes per simulation, and the latency model is an exponential distribution with λ 1.]

the observed latencies by the expected value, we will be able to see the additional delay caused by the partial order enforcement.



(a) No Order

(b) FIFO Order

(c) Causal Order

Figure 4.15: Different LM Parameters. [The configuration for this test is: 5 nodes per simulation, and a rate of operations of 10.]

Figure 4.16 shows a normalized comparison between different latency models that have enforced causal partial order. Given that all latencies had the same experiment configuration, except for the latency model, according to Equation 4.1 when normalized they would have to give the same results. While this is true when no partial order is enforced, we have seen a maximum standard deviation of 0.01 between the means, it does not hold when partial order are enforced. With partial order enforcement, we notice that the higher a latency model is, the more it will cause dependencies between messages that in turn will create additional latency. That, is due to the increased probability that one of the dependencies will have a higher latency given that there are more dependencies, even if the latency between events is independent. This causes the shift between the normalized latency model seen in Figure 4.16.

From that same figure the baseline line is obtained by computing the mean of the latencies for each latency model, but without enforcing any partial order.

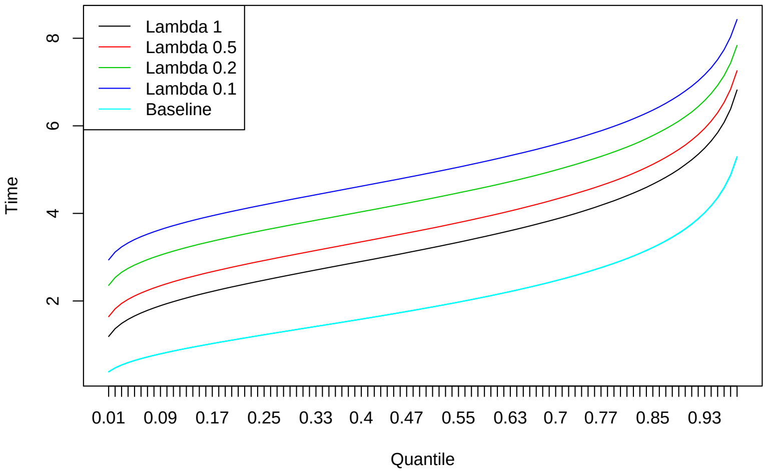


Figure 4.16: Normalized quantile comparison between different lambdas under causal partial order enforcement.

These results show that, the lower the communication latency the lower the additional delay caused by causal order enforcement, which in turn makes the overall system faster. However a slower latency, has a larger additional delay due to the enforcement. This can be explain as follows. A larger latency, for the same throughput of operations, has a more operations than can be overlapped from the latencies variability. However with a lower communication latency there are less operations received out of order and thus having lower additional latency for partial order enforcement.

One of the goals of modeling partial orders was to gain some insight on the behavior of stronger constraints under eventual consistency. The models that we provided yield the probability that a message is ready to be delivered under different situations. In this section we have shown how this probability changes under the different variation of its parameters. We have also seen the impact that generalizing the model has on its results and how it compares to the non-generalized version, and while given the similarity results between both, we could have computed the impact from the model without the need of simulation. Nonetheless the conclusions extracted and presented would be the same.

To study the impact of partial order enforcement we have made isolated variations on three different parameters: (1) the rate of operations, (2) the number of replicas, and (3) the latency model. We have seen how an increase on the number of operations or number of replicas will reflect in an increased additional latency due to the partial order enforcement. Similarity, we have seen how a decrease in the communication mean latency will increase the additional latency due to order enforcement in a non-linear way.

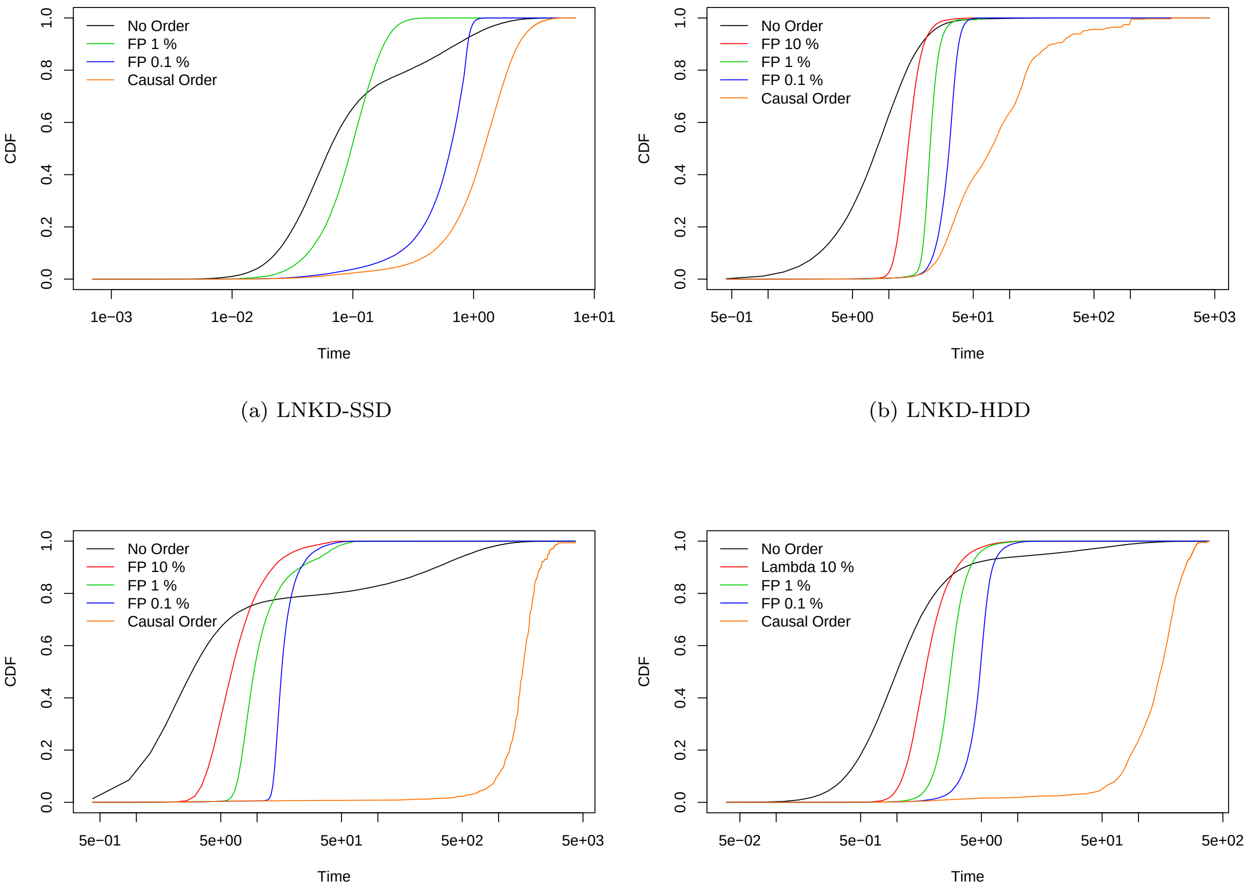
4.3 RER Experiments

In this section, we present the simulations done for reactive error recovery. The aim of this simulations is to present how the RER performs and how without it a system that does not recover from errors is rendered unusable when enforcing some type of partial order. Recall that from our model's assumptions all failures are transient and are represented in the latency model as having higher latency values.

In order to experiment our reliable error recovery mechanism, we evaluated it using production latency statistics [13] from two Internet-scale companies, LinkedIn and Yammer.

We ran all four latency models presented in Table 4.1. Figure 4.17 shows the graphical results of such simulations, one for each different latency model. The four simulations share the same configuration (5 nodes and 50 as rate of operations), except for the latency model values. Each plot contains five cumulative latency lines in logarithmic scale : two without RER, and three with RER. Among the two without RER, we present the communication latency without ordering (black curve) and delivery latency with causal ordering (cyan curve). For the error recovery we defined three different waiting times that correspond to 10%, 1%, and 0.1% false positives – see Section 3.6.2 – represented in respectively red, green and blue curves.

The top left graphic uses the configuration parameters of LKND-SSD, which has the lowest mean time of all four latency models : 0.0764 ms for transmission and 0.2369 ms for visibility. However, what makes it faster than the other is that it has a small std dev in its latency distribution. This configuration is the only one that has no 10% FP line, as a waiting time of zero corresponds to a lower FP value that 10%. When enforcing the partial order, the mean delivery time is 1.34 ms if no error recovery mechanism is active. However, by only using the 0.1% FP configuration we reduce the mean delivery time to a 45% and we reduce it to a 17.51% by using the more aggressive waiting



(a) LNKD-SSD

(b) LNKD-HDD

(c) YMMR-W

(d) YMMR-R

Figure 4.17: RER results on production-like latency models.

time corresponding to 1% FP.

The top right graphic uses the configuration parameters of LKND-HDD, and is significantly slower, two orders of magnitude, than its counterpart the LNKD-SSD when enforcing a partial order without error recovery. However in this scenario, the non-enforced ordering performs better until just after the 90th and before the 99th both 10% and 1% error recovery configurations perform better and by 99.9th all three configurations perform significantly better, as the slowest of all three achieves the 99.9th a 47.89% faster.

The bottom left graphic uses the configuration parameters of YMMR-W. Without error recovery, it

Name	P.Pareto	x_m	α	λ	wt 10% FP	wt 1% FP	wt 0.1% FP
LNKD-SSD	0.9122	0.235	10	1.66	NA	0.039	0.817
LNKD-HDD	0.38	1.05	1.51	0.183	4.75	17.61	32.56
YMMR-W	0.939	3	3.35	0.0028	1.125	5.046	12.23
YMMR-R	0.982	1.5	3.8	0.0217	0.39	1.97	4.46

Table 4.1: Extracted from PBS[13] and the waiting times, wt, for each configuration

takes 1990 ms for the 99.9*th* of its messages to be delivered without any order enforcement. However using a waiting time that yields a 10% FP it takes 3.8214 ms to delivered from the 0.1*th* to the 99.9*th*, and by increasing the waiting time so that we have a 0.1 % FP increases the latency 1.29 times.

The bottom right graphic uses the configuration parameters of YMMR-R, which has a faster mean that its counterpart the YMMR-W. In this case, it can be seen how the non-ordered plot achieves a higher delivery rate just until 2.9336 ms, when the solution with error recovery and a waiting time at 10% FP outperforms it. Later 1.3 ms at 4.2074 ms it is outperformed by the solution with a 1% FP and at 6.6392 ms by the 0.1% FP.

The results show that (1) enforcing a partial order without a reliable delivery mechanism that recovers from errors renders the whole system unusable as the mean delay grows several orders of magnitude higher than the systems that have an error recovery mechanism. (2) A system that does not recover from errors, even when no ordering has to be enforced, will have a higher performance for as long as no errors are present, but as soon as there are errors its performance will decay. (3) Solutions with three arbitrary false positive configuration values, perform within the same order of magnitude. The ones with a higher number of false positives perform better at the expense of higher network usage.

To recapitulate, in this chapter we have presented the simulator with which we have validated the previously described models. We have also validated the reactive error recovery mechanism using latency models obtained from real world production systems. Finally we have presented the impact that applying causal order enforcement has on the systems latency and how it scales, when the number of replicas, rate of operations or mean communication latency grows.

In the next chapter we present the evaluation of the reactive error recovery mechanism using an implementation deployed in Amazon Web Services. Then we will compare the observed false positives from the evaluation with the results from the simulation.

Chapter 5

Experimental Validation

Contents

In this chapter, we present the validation experiments done for the Reactive Error Recovery mechanism. We used virtual machines from Amazon Web Services, at different EC2 regions. We sat up two main scenarios: communication within a datacenter, and across datacenters. For each of these two scenarios, we ran different configurations for the error recovery mechanism. Over the course of this chapter we explain the motivations for the experiment, the configuration and decisions, as well as the imposed limitations. Finally, we explain and analyze the results that we have obtained.

5.1 Motivations

The aim of this set of experiments is to create a proof of concept for the proposed Reactive Error Recovery mechanism. With it we compare the error recovery properties of the mechanism under different conditions against the well-known TCP.

Observe the behavior of the error recovery mechanism under more realistic conditions. While we have seen the behavior of RER in the simulator, an evaluation using physical machines gives us access to information such as the exact bandwidth used, and a comparison with TCP.

The result provided by this evaluation ought to be more compelling than those from the simulation, as there are many variables that the simulation does not take into account that the real-life evaluation does implicitly. For example, in the simulation there is no cost associated to processing an operation. Therefore a node can process an unlimited number of messages without any overhead. In real life, it takes some time to process a message it needs to be serialized, de-serialized, validated, and then go through the application's logic.

To evaluate the RER we need to create an application to deploy on top of AWS[1]. The application consists on different clients exchanging messages with each other.

For this application we built two different communications layers. The first one is our error recovery mechanism on top of UDP, which does not provide any reliability properties except a checksum of the message. The second communication layer option is TCP. Both of them need to enforce a causal order delivery for the exchanged messages. Now that we have seen the motivation behind this evaluation experiment, we will discuss the configuration setup in AWS for the different scenarios.

5.2 Experimental Setup

In this section we explain the configuration in the Amazon Web Services, the nodes that we have used, the clients and server. To allow repeatability of our experiment, we describe the different technical parts that need to be managed in the cloud provider to allow for the communication to work. We also describe the configuration files that we used and the key parameters involved in each of them, as well as, the data recovered from each experiment launch.

There are many different configurations that could be tested. First of all, we have the parameters that specify the scenario in which we test the error recovery mechanism. These ones are, the number of nodes in the system, the number of different regions and the amount of replicas per each region, as well as the placement of such nodes. Furthermore, whilst we have decided to use Amazon Web Services EC2 as a cloud platform where to deploy the evaluation, there are specific configurations

[1] Even if the applications logic is empty, as in our case

Model	vCPU	Mem(GiB)	Storage
t2.micro	1	1	EBS-Only

Table 5.1: Specification of the nodes used in the evaluation.

that come from using this service.

5.2.1 AWS

We chose to use Amazon Web Services because it is one of the biggest cloud infrastructure providers in the world. There are available regions all over the world, from Sidney and Singapore to Ohio and Sao Paolo. However we later noticed that the network stability and reliability provided is counterproductive for analyzing our reactive error recovery mechanism. Due to that, we ended up adding different levels of extra latency in order to observe message reordering. In this section, we present the different nodes that we use by this cloud provider, and that are key components for our evaluation. These nodes are either a Client, a node that generates and receives messages; or a Server, a node that either relays messages to clients or acts as membership server and informs about clients addresses'.

Node

A node is the deployment unit that we have for the experiment. It can either be a client or the server, and in both cases they are deployed in AWS EC2. We used t2.micro[2] as the size for the virtual machine. Even though they are the smallest unit available in EC2, they are big enough for the experiment, and using more powerful machines would only increase the cost of the experiment. Each EC2 instance is linked to a security group[3] configured as described in Table 5.2. This rules allow both TCP and UDP connections from any IP to port 5005, ssh connection for management in port 22, as well as ICMP to ping if the machines are available.

The operating system used for the nodes, is the same one for both client and server. Is is an Ubuntu Server 14.04 64 bit, in which we added the necessary packages needed to run the application's code. Table 5.3 contains the AMI[4] for all the images used in the evaluation. We generated our own image based of an Ubuntu server, for each region that ran the evaluation. Even though creating this images is time consuming, it is worth it as it speeds up the deployment of the experiments for the different

[2] Specifications in Table 5.1

[3] an amazon's term to define the firewall set of rules

[4] AMI: Amazon Machine Image

Protocol	Dst Address	Port
TCP	0.0.0.0	5005
UDP	0.0.0.0	5005
ssh	0.0.0.0	22
ICMP	0.0.0.0	NA

Table 5.2: Security Group configuration for the nodes used in the evaluation.

Region	Original AMI	Modified
eu-central-1	ami-9c09f0f3	ami-31d91c5e
us-east-2	ami-b7075dd2	ami-c16832a4
sa-east-1	ami-900b96fc	ami-0c1e8160
ap-southeast-1	ami-1c2e887f	ami-9c4deeff
ap-southeast-2	ami-bf3d00dc	ami-18a7987b
eu-west-1	ami-f95ef58a	ami-708ec403

Table 5.3: AMIs of the OS used for nodes in the evaluation.

configurations [5].

Client

In this evaluation, the clients communicate between them by sending messages across the network. If the communication is with UDP sockets then the client sends the message to the server the broadcast the message to all connected clients, in case of a new message, or relays the message in case of message recovery. In the case of TCP the clients use the server as a membership service and later connect to each other client in a P2P fashion. This implies that after being connected, a client does not contact the server anymore.

There is specific configuration for each client, that for this evaluation are homogeneous across all clients. We can separate the configuration values in two categories: code parameters, and experiment parameters. The former contains the IP address of the server, the service port, how many entries has the vector clock, as well as the log names and the path to the script that runs as the client. The latter contains the configuration for the RER waiting time, as well as the number of operations that each client needs to send and with which workload. The workload is defined with a distribution

[5] However, considering software evolution we would have to study the trade-off between pre-generating the images and installing the needed dependencies at deployment time, or creating a docker-like solution for our images.

configuration. The workload defines the time between two operations are send, and the mean of the distribution is the period, or the inverse of the rate of operations. We also allow to configure a drop probability of messages. A dropped message is not send but it is created and logged as if it had. When a message is dropped in the client side, all nodes are forced to recover it.

Server

For the evaluation, the server is used in two different ways depending on the socket type. In a TCP communication the server will behave as a membership service, and updates the client on the addresses of the other clients so that they can connect to them P2P. However in the UDP communication it takes a more relevant task, as it relays all the messages between clients.
Furthermore in UDP communication we added some transmission control features to force the system to generate additional latency, which results in message reordering, and message drops. The message drop functionality is homologous to the one described in the client section. However in the server side a message can be dropped and affect only one client, and not the whole group.
The server is the first node to be available, and the last one to leave. We enforce this behaviour to ensure that the client nodes have a server to connect to when they are available.

Admin

In order to automate the testing of different configurations, we created several scripts that allow batching experiments and generate a descriptive unique name for the result logs retrieved from every experiment. In order to manage these scripts, we used additional configuration files, that define the type of machines that need to be launch from AWS EC2, as well as the security groups the virtual machine belongs to, and the AMI for the machine in each different available regions. These configuration files also include security practices like the management of the ssh keys for the different regions and monitor machines.
All communication to the machines is done using ssh connections, and managing scripts that are uploaded to the machines.
After all the messages have been delivered, or the system has timeout from an unrecoverable error, the monitor connects to all nodes, clients and server, and retrieves the logs that were generated.

Those are the three types of nodes related to the experiment, the two first, client and server, will be the nodes that generate the data and that run the application. The third node, the admin, is the machine that puppeteers the experiments by launching new instances when needed, rebooting and cleaning after each experiment, and finally collecting the data for later post-processing.

# DC	# Nodes per DC	Location
1	6(5+1)	eu-west-1 (6)
6(5+1)	1	eu-central-1 (1), us-east-2 (1), ap-southeast-1 (1) , sa-east-1 (1), ap-southeast-2 (1), eu-west-1 (1)

Table 5.4: Number of nodes and datacenters and its amazon code location.

The code and scripts need to launch this experiment are provided in `https://github.com/jmartori/rerval` under GNU General Public License v3. In the same repository we added the scripts to process the logs and to generate the plots and figures.

5.2.2 Parameters

Now we present the different configuration parameters that can be used for each experiment. For any given experiment there are three key sets of parameters that can be tuned. These are location or node placement, message workloads, and waiting time for the reactive error recovery mechanism.

Location

We work with two main location configurations, single and multiple datacenter scenarios. Table 5.4, contains two different configuration scenarios tested in this experiment. First communication within a single datacenter, and the second within multiple datacenter across the globe. Even though we could change the placement for the single datacenter, or which datacenter plays the role of server in the multi-datacenter scenario, we decided it was better to pin this configuration. This way we could minimize the noise coming from the placement, as it is constant across the experiments.

Workload

The workloads are defined by three key items. First the added latency model in the shape of exponential, Pareto or uniform distributions. Second, the delay between which operations are issued per node that can follow an exponential, uniform or normal distribution. Third is the probability that a message will be dropped. A message can be dropped both in the client side and in the server side. If it is the former, then the message is dropped for all the nodes, however in the latter, as the probability is independent for each message, it does not have to affect all nodes. We use the same rate of operations and number of replicas for all the experiments, unless told otherwise. The rate of operations is 10 operations per second per client node, and we have five clients plus a server for a total of six nodes per experiment.

RER (wt)

For both single and multiple data center experiments we use the following different waiting time for the RER mechanism, 0.01, 0.05, 0.1, 0.2, 0.5, 1, 2 and 5 seconds. This contains a sample of different waiting times for the amount latency, including the additional latency for reordering, that the system is exposed to.

5.2.3 Data Recovered

From each run a compressed file is obtained. The name of the file follows the same following structure "sdc-lm_exp_5-loss_0.00-wt_5-ro_10-date_1481257787.tar.gz". From it we can extract the basic configuration used in the experiment, **sdc** means that the experiment was within a single datacenter, with **lm**, latency model of a exponential distribution with a lambda of $5ms$, with a 0.00 percentage of voluntary message loss, and, **wt**, waiting time of $5s$, in which each replica has a **ro** – rate of operations – of 10 operations per second. Finally, the **date** of the file creation. The contents of the file are:

- **operations log**: For each node a log is created that contains when each operation was received, and from which replica it was. Another log is also created for the server.
- **monitor log**: For each node a different monitor log is created. It contains the output of a tcpdump for all the messages inbound and outbound through the port 5005 from each node.
- **configurations (Global and Local)**: Each experiment uses two different configuration files. One for the generation of the cluster and one for each node. Both configuration files are included in the results file, so that all the details needed to run an equivalent experiment are stored.
- **placement and IP**: A file containing the AWS region and the ip of each machine used in the experiment.
- **summary log**: At the end of each run, a summary file is generated by the node, that contains some rough statistics about that run. This includes the number of actively drop messages, send and delivered, as well as messages delayed or recovered.

Having seen the key parameters that shape the experiment, we are now ready to explore the results yielded by each experiment.

5.3 Results

This section contains the results for the evaluation ran in amazon web services EC2. The section is structured as follows:

- Results from the experiments where all the nodes are in the same data center.
- Results from the experiments where all the nodes are geo-distributed.
- Comparison results from the single datacenter scenarios with the results obtained from the simulator described in the previous chapter.

5.3.1 Data Processing

Before presenting the experiment's results, we will delve in how those results are obtained. As we have said before, from each run we obtain a set of logs that contain when was each operation sent, received and delivered to each replica, and from which replica was that operation from. We process those logs pairwise between a sending node and a receiving node. We obtain a matrix with the number of operations issued per replica, and four columns, query time, send time, received time, and delivery time, plus three more columns for the error recovery that are for time that a message was asked for retransmission, the time the message was received from retransmission and the time of delivery from retransmission. Another matrix is generated with the amount of messages of each type, which allows us to obtain how aggressive a waiting time is. The aggressivity of a waiting time value depends on the latency model, as it will be seen in Figure 5.3. An aggressive waiting time will ask for unnecessary retransmissions, wasting resources, but we will observe lower delivery times, as the errors are corrected sooner.

The second step for obtaining the results is classifying each operation according to four categories. This classification is explained in Figure 3.6. This four categories are: A0, A1, A2, and A3. First, $A0$, if they are received in the expected order, thus avoiding any delay. We add an ϵ value in order to classify these operations. Therefore if a message is received and can be delivered within that ϵ, we consider it an $A0$. We are aware that due to this extra value, we could wrongly classify a message, but for an arbitrarily small ϵ, this probability decreases and it highly improves and simplifies the classification process. Second, $A1$, if they are received out of order but before causing any recovery message – within the waiting time. Third, $A2$, if they are received after a retransmission was issued but before the recovery message has been received. We consider this messages as false positives. Fourth, $A3$, if the message is either received after recovery is successful or if the message is never

received.
Another step is processing the monitoring logs that contain the amount of bandwidth that has been used from each node.
Finally the last step is the comparison between the number of false positives obtained from the simulator and from the experiment. Section 5.3.4 describes more about the problems faced and the results obtained.
All the data processing has been done using scripts coded in R[6].

5.3.2 Single Datacenter

The results presented in the evaluation within a Single Datacenter are the following:

- Classification of operations for different waiting time configurations and latency models.
- Aggressivity of the different waiting time values under specific latency models configurations.
- Bandwidth usage by the different scenarios using the Reactive Error Recovery mechanism and TCP connections.
- Message delay caused by unresolved dependencies for different waiting times and latency models.

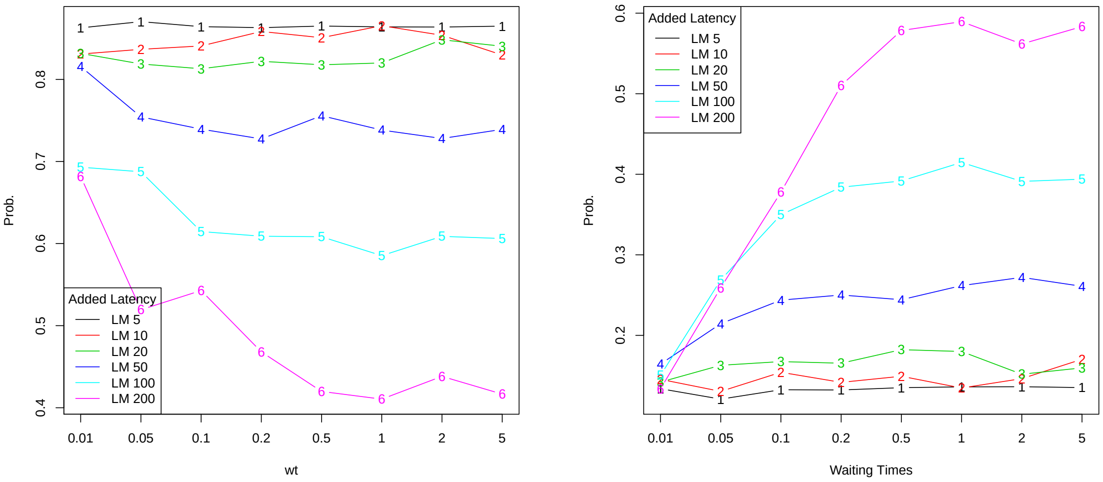
Classification of operations into A0, A1, A2, A3

In the following figures, we compare the observed ratio, for every category, with different waiting time configurations of RER. Furthermore we have added extra delay to the processing of the message to generate additional reordering. To achieve that we have added delay following an exponential distribution.

Figure 5.1a shows the results for the messages that fell in the A0 category, for different latency models and waiting times. We observe how as more latency is added, more messages arrive out of order and have to be delayed. At the same time, the more waiting time, the more time the message is delayed before it is recovered. But when the waiting time is too large, around 0.1s for LM100, it does not perform worse. However this "does not perform worse" depends to the latency model, as for larger additional latency, we would need larger waiting times to observe that it does not perform worse.

[6] While processing the raw logs is a long task, several hours, however once the preprocessed matrices are obtained the overall processing is much more faster, to a matter of seconds.

We do not observe much change across different waiting times when the latency model is very small, as to for a message to be received out of order it has to arrive sooner than the previous message. I.e. in a FIFO partial order enforcement with a rate of operations of 10, we can assume an operation every 100ms. That means that for an operation to be received out of order it has to experience at least 100ms of delay more than the next. With exponential distributions with low means, this is not very likely, and even when it happens once, it does not affect the system much. For higher means, as it happens often enough it does affect the system.



(a) Ratio of operations received in order for different latency reordering in a single datacenter for different waiting times ($A0$).

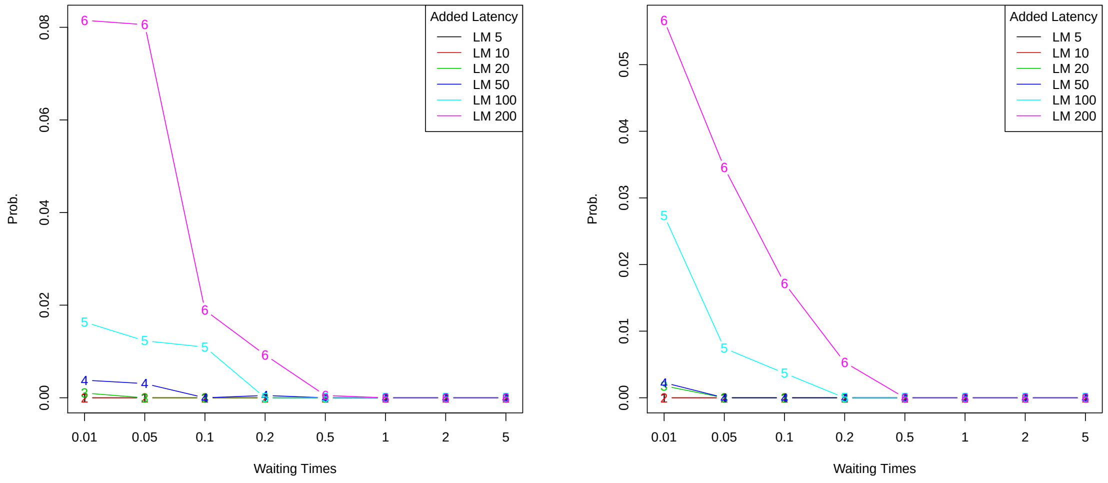
(b) Ratio of operations received in waiting time for different latency reordering in a single datacenter for different waiting times ($A1$).

Figure 5.1: Operations classified as A0 or A1 in a single datacenter under different latency models and waiting times.

It should be noted that the following figures have different y-axis limits, that while it provides a maximized view of their range, it can give a false sense of significance if the scale values are not compared carefully.

Figure 5.1b shows the results for the messages that fall in the A1 category. The plot shows how the longer the waiting time, the more messages fall in that category. However this behavior is seen better in those latency models with high reordering, as for small latency models, most of the messages where already received in order.

Figure 5.2a shows the results for the messages that fall in the A2 category, are considered as false positives. Recall that a false positive is a message that, by the time a recovery message is received, the original has already been received. This causes unnecessary usage of the network and the nodes. As it can be observed, the smaller waiting time have a higher percentage of messages in this category. A high number of messages in this category indicates that the waiting time can be improved. If we increase the waiting time the number of false positives will decrease as we will be receiving the original message that fell between the recovery. However if we reduce the waiting time, we will force a recovery before and the original will fall in the next category. This aggressive setting makes the system more responsive.



(a) Ratio of operations received as false positives for different latency reordering in a single datacenter for different waiting times ($A2$).

(b) Ratio of operations received after recovery for different latency reordering in a single datacenter for different waiting times ($A3$).

Figure 5.2: Operations classified as A2 or A3 in a single datacenter under different latency models and waiting times.

Figure 5.2b shows the results for the messages that fall in the A3 category. A3 type messages are those that the original message is either never received, or it is received after the recovery has been delivered. This type of message are a recovery success. We can see however, how the longer the waiting time the less probabilities that a message is received. This is due the latency model following an exponential distribution, as the chances that a message is added with long additional latency are small.

If we compare the results from a single datacenter with the ones from the multiple datacenter, they

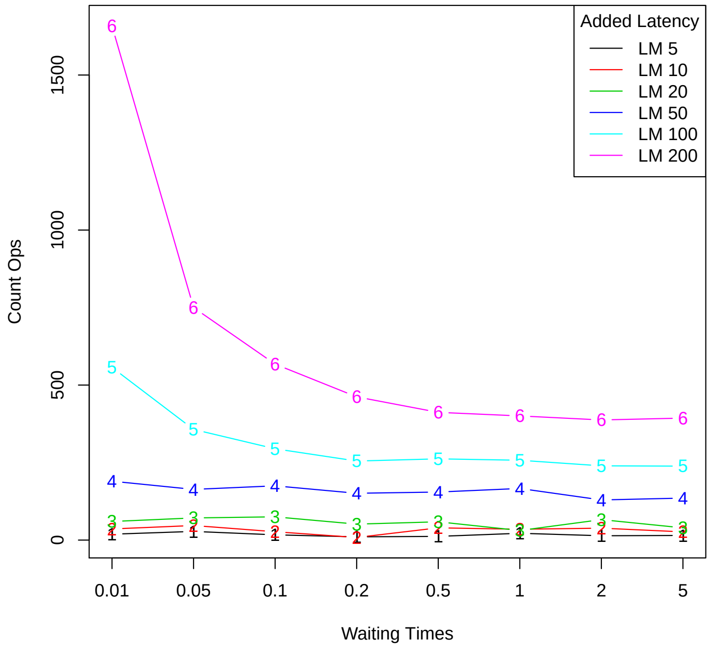


Figure 5.3: Number of resent operations for different latency reordering in a single datacenter for different waiting times.

give the impression that the multiple datacenter outperforms the other one. This is striking as multi-datacenter scenarios usually suffer more latency. However in our scenarios, as the latency between datacenters in very constant, it pipelines the messages and we do not see as many messages out of order as we could expect.

Aggressivity

We define aggressivity, as the amount of additional resources that a given waiting time uses. Figure 5.3 shows how more aggressive waiting times use more messages. This behavior is not surprising at all, as short waiting times will cause slightly unordered operations to be asked for retransmission when they were not lost. We can also observe how given the amount of reordering that this experiment has been subjected to, most of the reordering is recovered by waiting one second. This causes that waiting more time have similar results to a one second wait, but at the cost of added latency to the messages delivery.

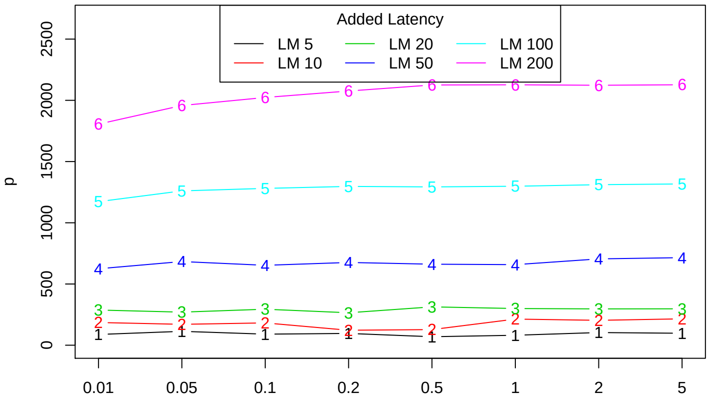


Figure 5.4: Number of delayed operations for different latency reordering in a single datacenter for different waiting times.

Delayed operations

Figure 5.4 shows the amount of messages that have been delayed as they were received out of order. As it can be observed, the higher the reorder delay added, the more messages that are stored for later delivery when the dependencies have been received. At the same time the larger the waiting time the more messages that are also delayed as the missing dependencies have not yet been recovered, the messages in the A1, A2, and A3 categories.

Bandwidth consumed

For every message sent there is an amount of bandwidth consumed. Figure 5.5 shows the typical amount of kilobytes used for each communication between all five clients, in which a thousand messages are delivered. As it should be expected the more aggressive that a waiting time configuration is, the more bandwidth is used. At the same time the more reordering that is observed the more recovery messages are issued and therefore more bandwidth is necessary. In order to compare the bandwidth used, we ran another version of the experiment with TCP as a transport layer, but without adding any additional latency. As it can be seen the RER implementation uses less bandwidth as long as its reordering latency is lower that $10ms$, and is very close the $20ms$ performance in most of the different waiting time configurations. We also added the expected amount of bytes used by a UDP communication where no messages were lost or out of order, and therefore there was no need

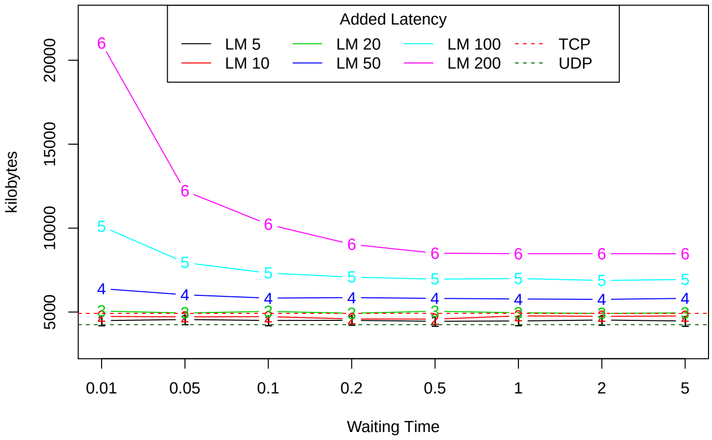


Figure 5.5: Bandwidth usage comparison between different waiting times under different latency reordering in a single datacenter vs TCP.

to use the error recovery mechanism. To achieve this we took the mean value of bytes per message, in a configuration with low additional latency, and multiplied those bytes by the expected number of messages, that are issued in an experiment by each node. This gives a baseline that is very close to the lowest latency model.

Trade-off between bandwidth and delayed operations

There exists a trade-off between the aggressivity of the waiting time, which as seen in Figure 5.5, influences the amount of bandwidth used, and the amount of delayed operations, as seen in Figure 5.4. For the same latency model configuration, a smaller waiting time will send more messages, and more network resources used, but less messages will be delayed.

This trade-off becomes more clear the more reordering the network is experiencing. This is because the more constant the network latency is, the more pipelined the messages are received, reducing the amount of out of order messages. In a two node scenario, or which a constant latency between all the nodes, constant delay does not change the order between message but just delay them. In systems with multiple nodes, and different constant latencies, it could happen that the messages appear to be received out of order. Imagine a scenario from Figure 5.6, any broadcast message from A, will arrive at C after B has notified it. This may cause the node to believe that the message from A is missing.

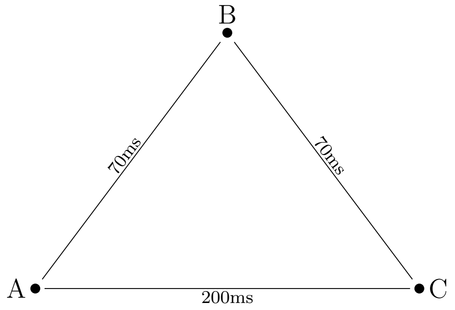


Figure 5.6: Three node scenario with asymmetric constant latency.

5.3.3 Multiple Datacenters

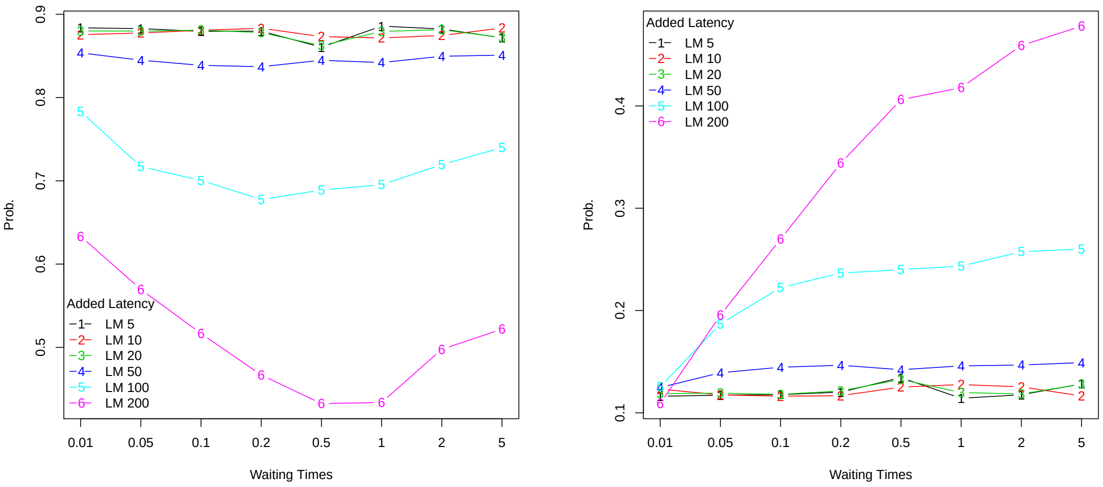
In this section we present the results for the inter-datacenters communication scenarios. We present the following results:

- Classification of operations for different waiting time configurations and latency models.
- Aggressivity of the different waiting time values under specific latency models configurations.

Classification of operations into A0, A1, A2, A3

In this section we describe the classification results from the evaluation in the geo-distributed configuration. The definition of each category can be found in Section 5.3.2.

Figure 5.7a shows the results for the messages that fall in the $A0$ category. We observe how with the lowest additional latencies, 5, 10, and 20, we have around 10% messages that need to be delayed. This is caused by the distributed nature of the experiment, and the asymmetry in latencies between the different nodes. We observe that when the waiting time is small, the errors are recovered sooner, which avoids the delay for other incoming operations. But when the waiting time increases, specially with larger additional latencies, there are less messages arriving in the correct order, as they are recovered in waiting time. But after 1 second of waiting time we notice that ratio of messages that arrive in order increases. This, while being unintuitive, is caused by the asymmetry of latency models between replicas. The average round trip latency for a message between the furthest two nodes, is around 950 ms for LM200, or 750 ms for LM100. This is why when the waiting is higher than $1s$ the ratio of messages received in order increases.
Figure 5.7b shows the results for the messages that fall in the $A1$ category. Most messages that did not fall in the category $A0$, arrive within the defined waiting time. As the waiting time increases,



(a) Ratio of operations received in order for different latency reordering across multiple datacenter for different waiting times ($A0$).

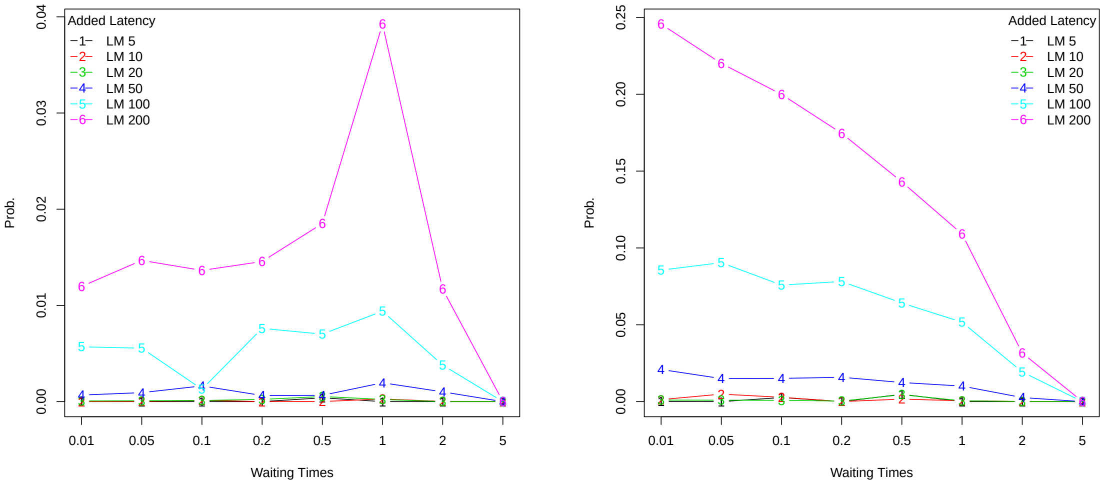
(b) Ratio of operations received in waiting time for different latency reordering across multiple datacenter for different waiting times ($A1$).

Figure 5.7: Operations classified as A0 or A1 in multiple datacenter scenario under different latency models and waiting times.

the number of operations recovered within waiting time increases with it. However as it can be seen in both blue lines, LM50, and LM100, for this given configurations, at a given waiting time 0.1 for LM50 and 0.2 for LM100 the additional waiting time duration does not have a significant recovery impact. For lower latency models this also true after the lowest waiting time value.

Figure 5.8a shows the results for the messages that fall in the $A2$ category. In contrast with Figure 5.2a we notice that the number of false positives remains fairly constant across the increasing waiting times, up to 5 seconds, when all messages are either received in $A0$ or $A1$. We believe that the disparity in the behavior between single datacenter and multiple datacenters is the asymmetric latency between the nodes. As we have seen for the classification of $A0$ messages after $1s$ of waiting time, having a waiting time close to the expected round-trip value, causes additional false positives messages.

Figure 5.8b shows the results for the messages that fall in the $A3$ category. Messages that fall in this category are those that we consider recovered. Seeing the line decrease shows that the error recovery mechanism recovered a message before it was received therefore gaining time. All this messages classified as $A3$ allow us to reshape the network latency model into the observed latency model by



(a) Ratio of operations received as false positives for different latency reordering across multiple datacenter for different waiting times ($A2$).

(b) Ratio of operations received after recovery for different latency reordering across multiple datacenter for different waiting times ($A3$).

Figure 5.8: Operations classified as A2 or A3 in multiple datacenter scenario under different latency models and waiting times.

the system, where the errors and delays are purged from it.

Aggressivity

Figure 5.9 shows how more aggressive waiting times use more messages. Aggressive waiting times, are those that cause unnecessary retransmits of messages. In the case of LM 200 in waiting time 0.01, it makes the system use more recovery messages than the sent by the original messages. At the same time we can observe that those configurations with low reordering have lower errors, and are therefore in lower need of recovering.

5.3.4 Fitting

One of the side-goals of this evaluation was to corroborate that the results obtained from the model, and simulation, were similar to the results from the experiment[7].

[7] It goes without saying that if they were to be significantly different there would be a problem somewhere.

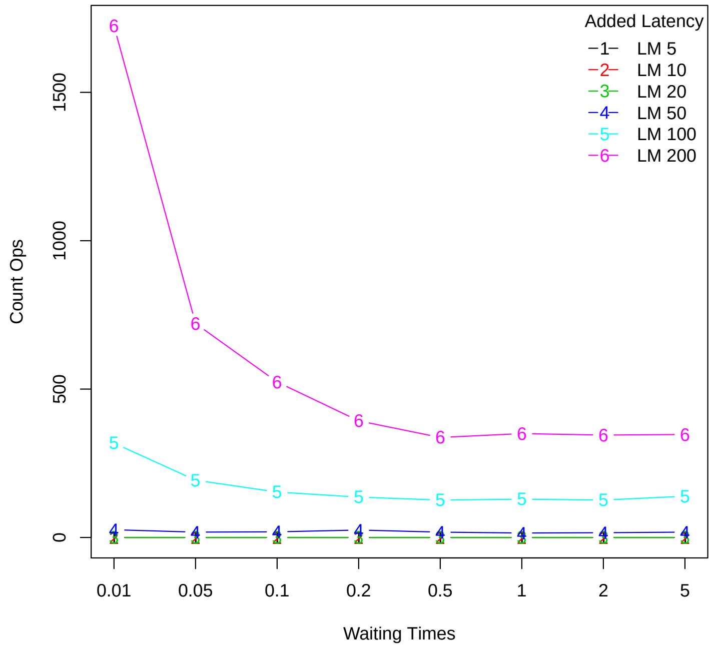


Figure 5.9: Number of resent operations for different latency reordering across multiple datacenter for different waiting times.

One of the results obtained from the evaluation was the number of false positives that the RER mechanism was obtaining for a given waiting time. We had seen in the simulations that a more aggressive waiting time, lower value, would have higher false positives values as it would be recovering more than necessary. In the other hand, higher waiting time would obtain smaller false positives values, as ordering faults would be recovered by waiting.

In communication latencies we could describe two types of latency. One would be the latency that the system experiences if no recovery is done. This latency can be named base latency model, and can work as a rough base line. This latency model includes the errors that have not recovered, and that if left unrecovered would render the system unusable, if we have to enforce some partial order. The second latency model is the one that we observe when we have an error recovery mechanism in place, and we can call it observed latency model. This latency model does not include errors in it, as the recovery mechanism has purged them away.

For the simulation, we will be using the added latencies as latency models. This will cause a smaller latency than the evaluation counterpart and therefore we should observe less false positives.

These latencies follow an exponential distribution with the following means: $5ms$, $10ms$, $20ms$, $50ms$, $100ms$, and $200ms$. As both rate of operations and lambda need to be in the same order of magnitude, we convert from ms to seconds. Finally, as need the λ of the distribution and not the rate, or β we inverse it, obtaining the following result of: 200, 100, 50, 20, 10, 5 respectively to the previous means.

Figure 5.10 shows the results from the simulation for each of the configurations used in the evaluation experiment within a datacenter. This figure compares with the results presented in Figure 5.2a.

Even if both of them have different results, the trend that is obtained in the simulator is also observed in the evaluation. This trend shows as follows:

- the more waiting time provided the less chances are for obtaining a false positive as the operation is likely to be recovered during the waiting time.
- the more communication delay that the messages observe, the more likely it is for the system to generate false positives.
- the evaluation should experience more false positives for a given latency model-waiting time tuple.

Given that the simulator has only the additional delay as latency model, and that the evaluation has the additional latency, plus the real communication, and the mechanism used to add the latency, it is fair to assume that the simulator has less latency than the evaluation for the same configuration. This causes the simulation to observe less false positives than its realistic counterpart. However at the same time the amount of latency that we add to lower latency values has less variability than in the larger latencies. This causes the system to pipeline messages which in time causes to observe less false positives for smaller latencies, as the messages arrive less out of order.

Using NTP

The fitting of some data to a distribution consists in giving the parameters for a distribution that will reduce $rmse$ from the observed data with the simulated data. As we are fitting the observed latency to an exponential distribution, we compute the mean latency of the message from their sending to their reception. In order to compare their times, we use NTP to synchronize the clocks

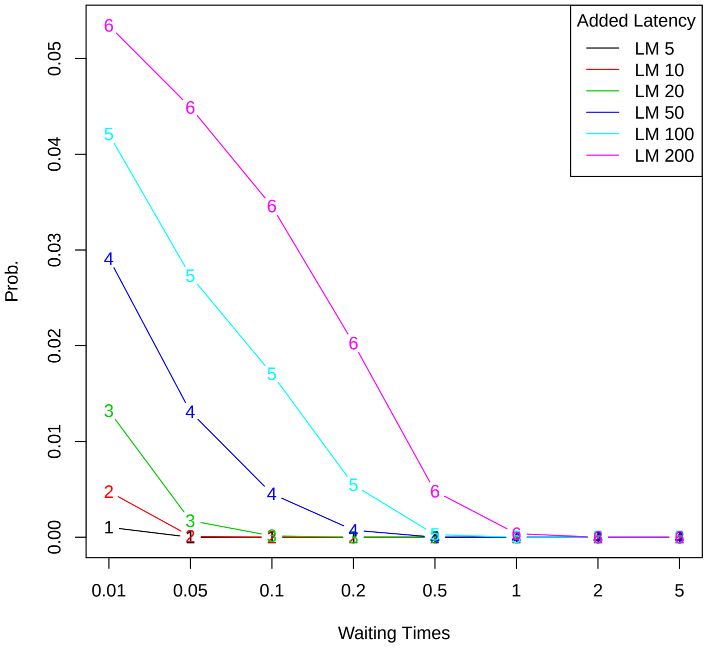


Figure 5.10: Ratio of operations received as false positives for different latency reordering for different waiting times obtained from a simulation.

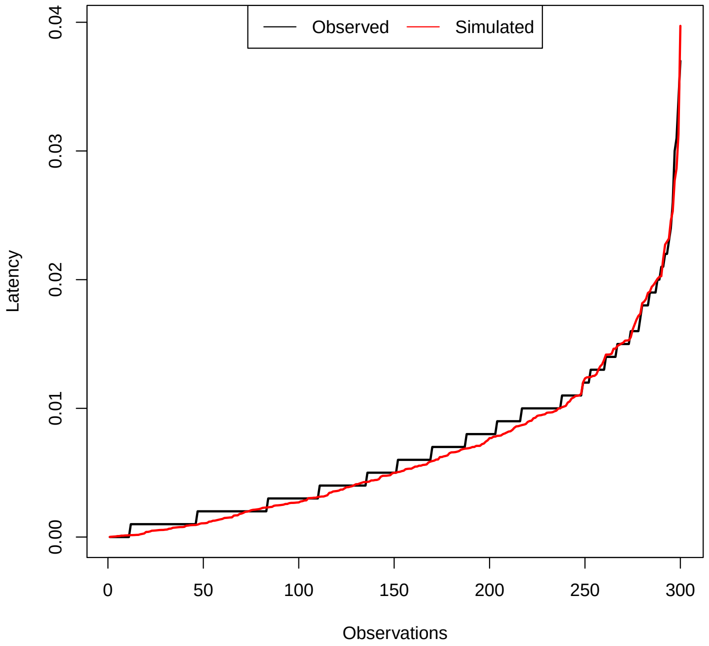


Figure 5.11: Observed and Simulated Latency with the same configuration from a single datacenter experiment.

of the machines within a datacenter and cherry pick the ones with the lowest offset between the client's clock and the server's to extract the latency values with the lowest available error. When using an NTP server it that forces all nodes to synchronize their clocks with the server. When this happens within a datacenter, as the latency between the nodes is very small, the variability between the nodes and the server is within the same order as the latency. In this case we obtained the communication latency with the difference between the receiver and the sender log entries. Furthermore, as the added latency follows an exponential distribution, we fit the latency data into an exponential distribution, that produces Figure 5.11 as a result.

Once the latency model is available for simulation we will assume that all nodes within a datacenter follow the same latency model.

5.4 Limitations

In this section we acknowledge a series of shortcomings observed in our evaluation, that even though we do not believe they diminish the results and conclusions, we believe we should address each one of them accordingly.

5.4.1 Global Clock

In contrast with the proposed simulator in the previous chapter, while doing the evaluation experiment we lacked a global clock to synchronize log operations in the results processing stage. Even though this has no impact in the experiment, as are the logical vector clocks the ones that manage causality enforcement, it would be interesting for some results, as well as to compare with the simulator, to obtain clocks that are closely synchronized.
We obtained the observed latency model for a configuration that had all the nodes within the same data center. Furthermore we configured a NTP, Network Time Protocol, server that would serve its local time to the other nodes in the cluster. Given that they are within the same region, the latency between nodes was on the milliseconds order.

In order to obtain the observed latency, we took from each experiment a tuple of client nodes that had the lowest time divergence with the server, around two milliseconds.

5.4.2 Performance

We have to remember that the aim of this experiment was not to be an implementation that could compete with real world deployments. We achieve enough operations per second to replicate the configuration observed in PBS [?] by Linkedin or Yammer, which was 50op/s. The maximum throughput observed in our experiments was of 3000 operations per second per node.

5.4.3 Not a real case-study

The workloads that we have generated are synthetic, and they follow the models that we have previously defined. This means that time between operations sent by a client is either a fixed, from a uniform, exponential or normal distribution. We have not, for lack of access to this kind of data and infrastructure, experimented our error recovery mechanism in a production-like scenario. Which while the cloud nodes, and links are the same that would be used, the application itself is lacking the access patterns to databases and read/write workloads from a realistic scenario. Nonetheless, we believe that this first approach was an initial step to check the validity of our claims, whilst opening a door to further experiments done by people with access to such data or infrastructure.

In this chapter we have presented the validation experiments done in AWS EC2 that corroborates the results previously obtained by the model and the simulator. We showed that the causal metadata can be used to achieve reliability without the use of positive acknowledgments. At the same time we have shown how even under additional latency the proposed mechanism uses less bandwidth than TCP without the additional latency. We have checked the behavior of the mechanism within a datacenter and across a geo-distributed. Furthermore we have compared the observed results with the ones predicted by the simulator.

In the next chapter we will extend the conclusions regarding the overall thesis, while at the same time presenting the future works that could follow this one.

Chapter 6

Conclusions

With this chapter we conclude this dissertation by presenting the conclusions, and proposing further works that could extend this one.
The contributions for this thesis were the following:

1. We proposed a model that yields the probability that a message is ready for delivery under different partial orders, causal and FIFO; with different latency models, exponential and Pareto distribution, and a mix of both.

2. We generalized the previous models, with rate of operations that follow a uniform, exponential or normal distributions.

3. We used the previously mentioned models, to propose a reactive error recovery mechanism, that reduces the use of acknowledgment messages, and sends negative acknowledgments to other nodes when he needs operations retransmission.

4. We built a simulator using R to validate the models and the reactive error recovery mechanism. We used production-like latency models for the error recovery mechanism validation.

5. We deployed an evaluation experiment in Amazon Web Services EC2 to demonstrate the behavior of the reactive error recovery mechanism. We compared the bandwidth usage with TCP.

For contribution (1), we proposed a set of models that yield the probability that a message is ready for delivery under partial order enforcement. We used different latency models, to model the networks latency, a key factor in message communication in distributed systems. We settled for using exponential distribution, Pareto distribution and a mix of both. The main limitation with this model is that we were giving the probability for just one operation, and not system wide, and we were using the history of all previous operations for partial order enforcement. Contribution (2) fixed these shortcomings by generalizing the models with a rate of operations that could follow uniform, exponential or normal distributions. With the generalized version, we give the probability that the system is ready for delivery. These models have increased our understanding of partial order enforcement under eventual consistency systems, and provide some probabilistic bounds where the message readiness for delivery can be achieved.

Contribution (3) spawns from the previous models, and we propose an error recovery mechanism that reduces the use of acknowledgment for received message. It uses the metadata information used for partial order enforcement to determine that a message is missing. To achieve this it uses an extended version of the models to determine the waiting time of the retransmission timer before contacting the sender replica, with a negative acknowledgment, for retransmission.

Contribution (4) focused on validating the models as well as the error recovery mechanism. We built a simulator with which we could generate operations, following different workloads and distribution, in order to experiment with our models. We obtained the results using production-like latency models from Internet companies and showed how not recovering errors under partial order enforcement yielded the system unusable.

Contribution (5) extended the simulation validation by deploying the error recovery mechanism Amazon Web Services, a cloud environment. We compared the bandwidth used between our error recovery mechanism and TCP, under different waiting time configurations, and showed how our mechanism could provide reliable communication while using less bandwidth, and under more straining configurations. We added additional latency to our connections while we left TCP untouched. This results showed that under approximately 20 ms of added mean latency our mechanism would consume less bandwidth, while with additional latencies between 50 and 200 ms, given to the additional reorderings, we would consume more bandwidth than TCP untouched.

6.1 Future Works

In this section we will introduce some interesting points that we believe that if pursued could yield interesting results, and extend the research done so far.

We will begin by work that would extend the job done until now.

1. Adding the Pareto and a mixed distribution of exponential and Pareto, to the generalized causal models, and to the Reactive Error Recovery mechanism to predict the amount of false positives, would allow to use this work in more realistic scenarios.

2. Adding additional workload models like Pareto, patterns more linked with causal behaviors, would result in stronger predictions.

3. Reevaluating the Reactive Error Recovery mechanism using a more realistic case study.

4. Move from the homogeneity to more heterogeneous scenarios. This has an impact on the models that need to be extended to express the different configurations and relationship between nodes.

5. Following partial quorums with causal ordering, for the modeling, and adding an evaluation that uses the partial quorums using well known data stores.

6. Generalize $P(D)$ used in the generalization of the causal model. By removing the need of a simulation to compute $P(D)$ we would allow the models to face changes in the latency model or the workloads in a more dynamic way.

Other of lines of further work are the following topics:

Garbage collection for CRDT: As CRDT need to be monotonic, in some cases the data types need to add tombstones[1] to produce the monotonicity effect. In those cases the amount of metadata used can grow unbounded and needs to be garbage collected by synchronizing the different replicas [5]. This results in a very consuming operation that requires all replicas to be active and does not tolerate failures.
However we believe that a continuous garbage collecting system could be achieved by using the models that have been provided in this thesis, and it could provide an heuristic method to delete already seen messages from all the replicas without need for acknowledgment.

Scheduling message delivery with best effort constraints: In many real time communication systems there is a window limit that if passed renders the dialog difficult to follow [69]. We believe that the models we have presented could be useful to predict if a message needs to be retransmitted or wait for the original delivery. This case study follows quite close to our work as all the messages need to be delivered in the same order that they were generated,

[1] A tombstone is an element flagged as removed and kept hidden to keep track of it.

however the window constraint may allow this dependencies to be broken in order to avoid disrupting the conversation.

Bibliography

[1] Basho Riak. `http://basho.com/products/`. Accessed: 2017-3-07.

[2] Cisco global cloud index: Forecast and methodology, 2015–2020 – white paper. Technical Report C11-738085, CISCO, 2015.

[3] Ittai Abraham and Dahlia Malkhi. Probabilistic quorums for dynamic systems. *Distributed Computing*, 18(2):113–124, nov 2005.

[4] Mustaque Ahamad, Gil Neiger, James E. Burns, Prince Kohli, and Phillip W. Hutto. Causal memory: definitions, implementation, and programming. *Distributed Computing*, 9(1):37–49, 1995.

[5] Mehdi Ahmed-Nacer, Claudia-Lavinia Ignat, Gérald Oster, Hyun-Gul Roh, and Pascal Urso. Evaluating crdts for real-time document editing. In ACM, editor, *ACM Symposium on Document Engineering*, page 10 pages, San Francisco, CA, USA, september 2011.

[6] Amitanand S. Aiyer, Lorenzo Alvisi, and Rida A. Bazzi. Byzantine and Multi-writer K-Quorums. In *Lecture Notes in Computer Science*, volume 4167, pages 443–458. 2006.

[7] Amitanand S. Aiyer, Lorenzo Alvisi, and Rida A. Bazzi. Byzantine and multi-writer k-quorums. In *Proceedings of the 20th International Conference on Distributed Computing*, DISC'06, pages 443–458, Berlin, Heidelberg, 2006. Springer-Verlag.

[8] Mark Allman and Vern Paxson. On estimating end-to-end network path properties. *SIGCOMM Comput. Commun. Rev.*, 29(4):263–274, August 1999.

[9] Paulo Sérgio Almeida, Ali Shoker, and Carlos Baquero. Delta state replicated data types. *CoRR*, abs/1603.01529, 2016.

[10] Asonge. A crdt library with d-crdt support. `https://github.com/asonge/loom`, Feb 2015.

[11] Peter Bailis and Ali Ghodsi. Eventual consistency today: Limitations, extensions, and beyond. *Queue*, 11(3):20:20–20:32, March 2013.

[12] Peter Bailis, Shivaram Venkataraman, Michael J. Franklin, Joseph M. Hellerstein, and Ion Stoica. Probabilistically bounded staleness for practical partial quorums. *Proc. VLDB Endow.*, 5(8):776–787, April 2012.

[13] Peter Bailis, Shivaram Venkataraman, Michael J. Franklin, Joseph M. Hellerstein, and Ion Stoica. Probabilistically bounded staleness for practical partial quorums. *Proc. VLDB Endow.*, 5(8):776–787, April 2012.

[14] Valter Balegas, Cheng Li, Mahsa Najafzadeh, Daniel Porto, Allen Clement, Sérgio Duarte, Carla Ferreira, Johannes Gehrke, João Leitão, Nuno Preguiça, Rodrigo Rodrigues, Marc Shapiro, and Viktor Vafeiadis. Geo-Replication: Fast If Possible, Consistent If Necessary. *IEEE Data Engineering Bulletin*, 39(1):12, March 2016.

[15] David Bermbach and Stefan Tai. Eventual consistency: How soon is eventual? an evaluation of amazon s3's consistency behavior. In *Proceedings of the 6th Workshop on Middleware for Service Oriented Computing*, MW4SOC '11, pages 1:1–1:6, New York, NY, USA, 2011. ACM.

[16] Kenneth P Birman. The process group approach to reliable distributed computing. *Communications of the ACM*, 36(12):37–53, 1993.

[17] Kenneth P. Birman, Mark Hayden, Oznur Ozkasap, Zhen Xiao, Mihai Budiu, and Yaron Minsky. Bimodal multicast. *ACM Trans. Comput. Syst.*, 17(2):41–88, May 1999.

[18] K.P. Birman. *Building Secure and Reliable Network Aplications*. Manning Publishing Company and Prentice Hall, January 1997.

[19] E.A. Brewer. Towards robust distributed systems. In *Proceedings of the Annual ACM Symposium on Principles of Distributed Computing*, volume 19, pages 7–10, 2000.

[20] Eric Brewer. Spanner, truetime and the cap theorem. Technical report, 2017.

[21] I. Briquemont, M. Bravo, Z. Li, and P. V. Roy. Conflict-free partially replicated data types. In *2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 282–289, Nov 2015.

[22] Lucas Brutschy, Dimitar Dimitrov, Peter Muller, and Martin Vechev. Effective serializability for eventual consistency. Technical report, 2016.

[23] John W. Byers, Michael Luby, Michael Mitzenmacher, and Ashutosh Rege. A digital fountain approach to reliable distribution of bulk data. *SIGCOMM Comput. Commun. Rev.*, 28(4):56–67, October 1998.

[24] Punit Chandra and Ajay D. Kshemkalyani. *Causal multicast in mobile networks*, pages 213–220. 2004.

[25] George Colouris, Jean Dollimore, Tim Kindberg, and Gordon Blair. *Distributed Systems: Concepts and Designs.* Addison-Wesley, 1988 - 2012. ISBN 978-0-13-214301-1.

[26] V. Jacobson R. Scheffenegger D. Borman, B. Braden. RFC 7323: Tcp extensions for high performance. RFC 7323, RFC Editor, September 2014.

[27] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: amazon's highly available key-value store. *SIGOPS Oper. Syst. Rev.*, 41(6):205–220, 2007.

[28] Peter van Hulten Didier Liauw. Demystification of crdt. Technical report, April 2016.

[29] Dropbox. Dropbox website. `https://www.dropbox.com/`, March 2017.

[30] C. J. Fidge. Timestamps in message-passing systems that preserve the partial ordering. *Proceedings of the 11th Australian Computer Science Conference*, 10(1):56–66, 1988.

[31] Seth Gilbert and Nancy Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *SIGACT News*, 33:51–59, June 2002.

[32] Theo Haerder and Andreas Reuter. Principles of transaction-oriented database recovery. *ACM Comput. Surv.*, 15(4):287–317, December 1983.

[33] Mark G. Hayden. The ensemble system. *PhD. dissertation Cornell University Dept. of Computer Science*, January 1998.

[34] Maurice Herlihy and Nir Shavit. On the nature of progress. In *Proceedings of the 15th International Conference on Principles of Distributed Systems*, OPODIS'11, pages 313–328, Berlin, Heidelberg, 2011. Springer-Verlag.

[35] Hugh W. Holbrook, Sandeep K. Singhal, and David R. Cheriton. Log-based receiver-reliable multicast for distributed interactive simulation. *SIGCOMM Comput. Commun. Rev.*, 25(4):328–341, October 1995.

[36] Patrick Hunt, Mahadev Konar, Flavio P. Junqueira, and Benjamin Reed. Zookeeper: Wait-free coordination for internet-scale systems. In *Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference*, USENIXATC'10, pages 11–11, Berkeley, CA, USA, 2010. USENIX Association.

[37] V. Jacobson. Congestion avoidance and control. *SIGCOMM Comput. Commun. Rev.*, 18(4):314–329, August 1988.

[38] Phillippe Kalitine and Matthieu Nicolas. Mute. `https://www.npmjs.com/package/mute-structs`.

[39] Anne-Marie Kermarrec, Laurent Massoulié, and Ayalvadi J. Ganesh. Probabilistic reliable dissemination in large-scale systems. *IEEE Trans. Parallel Distrib. Syst.*, 14(3):248–258, March 2003.

[40] A.M.Odlyzko K.G.Coffman. Growth of the internet. `http://www.dtc.umn.edu/~odlyzko/doc/oft.internet.growth.pdf`, July 2001. AT&T Labs – Research.

[41] Tim Kraska, Martin Hentschel, Gustavo Alonso, and Donald Kossmann. Consistency rationing in the cloud: pay only when it matters. 2009.

[42] Avinash Lakshman and Prashant Malik. Cassandra - a decentralized structured storage system. In *Proceedings of The 3rd ACM SIGOPS International Workshop on Large Scale Distributed Systems and Middleware (LADIS)*, Big Sky, MT, USA, October 2009. SIGOPS.

[43] Leslie Lamport. The part-time parliament.

[44] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565, 1978.

[45] Leslie Lamport. Paxos made simple, fast, and byzantine. In *OPODIS*, pages 7–9, 2002.

[46] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4/3:382–401, July 1982.

[47] George J. Lee and Lindsey Poole. Diagnosis of tcp overlay connection failures using bayesian networks. In *Proceedings of the 2006 SIGCOMM Workshop on Mining Network Data*, MineNet '06, pages 305–310, New York, NY, USA, 2006. ACM.

[48] Mihai Letia, Nuno Preguiça, and Marc Shapiro. CRDTs: Consistency without concurrency control. In *SOSP W. on Large Scale Distributed Systems and Middleware (LADIS)*, pages 29–34, Big Sky, MT, USA, October 2009. sigops, acm.

[49] Cheng Li, Daniel Porto, Allen Clement, Johannes Gehrke, Nuno Preguiça, and Rodrigo Rodrigues. Making geo-replicated systems fast as possible, consistent when necessary. In *Presented as part of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12)*, pages 265–278, Hollywood, CA, 2012. USENIX.

[50] Wyatt Lloyd, Michael J. Freedman, Michael Kaminsky, and David G. Andersen. Don't settle for eventual: scalable causal consistency for wide-area storage with cops. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, SOSP '11, pages 401–416, New York, NY, USA, 2011. ACM.

[51] E. Blanton M. Allman, V. Paxon. RFC 5681: Tcp congestion control. RFC 5681, RFC Editor, September 2009.

[52] S.Floyd A. Romanov M. Mathis, J. Mahdavi. RFC 2018: Tcp selective acknowledgment options. RFC 2018.

[53] Prince Mahajan, Lorenzo Alvisi, and Mike Dahlin. Consistency, availability, and convergence. *University of Texas at Austin TR-11-22 (May)*, pages 1–31, 2011.

[54] Dahlia Malkhi, Michael K Reiter, Avishai Wool, and Rebecca N Wright. Probabilistic Quorum Systems. *Information and Computation*, 170(2):184–206, nov 2001.

[55] Friedemann Mattern. Virtual time and global states of distributed systems. In Michel Cosnard et al., editor, *Proceedings of the International Workshop on Parallel and Distributed Algorithms*, pages 215–226, Château de Bonas, France, October 1989. Elsevier Science Publishers.

[56] C. Mohan, B. Lindsay, and R. Obermarck. Transaction management in the r* distributed database management system. *ACM Trans. Database Syst.*, 11(4):378–396, December 1986.

[57] Alberto Montresor and Márk Jelasity. PeerSim: A scalable P2P simulator. In *Proc. of the 9th Int. Conference on Peer-to-Peer (P2P'09)*, pages 99–100, Seattle, WA, September 2009.

[58] M. Paulitsch, J. Morris, B. Hall, K. Driscoll, E. Latronico, and P. Koopman. Coverage and the use of cyclic redundancy codes in ultra-dependable systems. In *2005 International Conference on Dependable Systems and Networks (DSN'05)*, pages 346–355, June 2005.

[59] V. Paxon and M. Allman. RFC 2988: Computing tcp's retransmission timer. RFC 2988, RFC Editor, Novembre 2000.

[60] V. Paxson, M. Allman, J. Chu, and M. Sargent. Computing TCP's Retransmission Timer. RFC 6298, RFC Editor, June 2011.

[61] Syncfree EU Project. Antidotedb. `http://syncfree.github.io/antidote/`. Accessed: 2017-3-07.

[62] Michal Ptaszek. Scaling league of legends chat to 70 million players. `http://highscalability.com/blog/2014/10/13/how-league-of-legends-scaled-chat-to-70-million-players-it-t.html`, September 2014.

[63] Marc Shapiro, Nuno Preguiça, Carlos Baquero, and Marek Zawirski. Conflict-free replicated data types. In Xavier Défago, Franck Petit, and V. Villain, editors, *Stabilization, Safety, and Security of Distributed Systems (SSS)*, volume 6976, pages 386–400, Grenoble, France, October 2011.

[64] Liuba Shrira, Hong Tian, and Doug Terry. Exo-leasing: Escrow synchronization for mobile clients of commodity storage servers. Technical report, August 2008.

[65] Mukesh Singhal and Ajay Kshemkalyani. An efficient implementation of vector clocks. *Inf. Process. Lett.*, 43(1):47–52, August 1992.

[66] Madhu Sudan. Decoding of reed solomon codes beyond the error-correction bound. *J. Complex.*, 13(1):180–193, March 1997.

[67] Roshan Sumbaly, Jay Kreps, Lei Gao, Alex Feinberg, Chinmay Soman, and Sam Shah. Serving large-scale batch computed data with project voldemort. In *Proceedings of the 10th USENIX Conference on File and Storage Technologies*, FAST'12, pages 18–18, Berkeley, CA, USA, 2012. USENIX Association.

[68] Qixiang Sun and Daniel C. Sturman. A gossip-based reliable multicast for large-scale high-throughput applications. In *Proceedings of the 2000 International Conference on Dependable Systems and Networks (Formerly FTCS-30 and DCCA-8)*, DSN '00, pages 347–, Washington, DC, USA, 2000. IEEE Computer Society.

[69] Tim Szigeti and Christina Hattingh. *End-to-End QoS Network Design: Quality of Service in LANs, WANs, and VPNs.* Cisco Press, 2004.

[70] D.B. Terry, A.J. Demers, K. Petersen, M.J. Spreitzer, M.M. Theimer, and B.B. Welch. Session guarantees for weakly consistent replicated data. In *Proceedings of 3rd International Conference on Parallel and Distributed Information Systems*, pages 140–149. IEEE Comput. Soc. Press, 1994.

[71] Douglas B. Terry, Marvin M. Theimer, Karin Petersen, Alan J. Demers, Mike J. Spreitzer, and Carl H. Hauser. Managing update conflicts in Bayou, a weakly connected replicated storage system. In *Proceedings of the fifteenth ACM symposium on Operating systems principles - SOSP'95*, pages 172–182. ACM Press, 1995.

[72] Robert H. Thomas. A majority consensus approach to concurrency control for multiple copy databases. *ACM Trans. Database Syst.*, 4(2):180–209, 1979.

[73] Werner Vogels. Eventually consistent. *Commun. ACM*, 52(1):40–44, January 2009.

[74] Stéphane Weiss, Pascal Urso, and Pascal Molli. Logoot: a P2P collaborative editing system. Rapport de recherche INRIA RR-6713, INRIA, December 2008.

[75] Haifeng Yu and Amin Vahdat. The costs and limits of availability for replicated services. *ACM SIGOPS Operating Systems Review*, 35(5):29, dec 2001.

[76] Haifeng Yu and Amin Vahdat. Design and evaluation of a conit-based continuous consistency model for replicated services. *ACM Transactions on Computer Systems*, 20(3):239–282, aug 2002.

List of Figures

List of Tables

List of Algorithms

Appendices

Appendix A

Proof

A.1 Reception in n nodes

Proof. $P(max(X_2, X_3, \ldots, X_n) - X_1 \leq t)$
When a node sends a message it sends it broadcasts it to all the nodes in the system, this gives us the probability that all nodes have received their message from the point of view of a node that has already received its message. n is the total number of messages sent, which is one less than the total number of nodes.
Assuming that it was the X_1 message that was received, $max(X_2, X_3, \ldots, X_n)$ is the last node to receive the message and by subtracting the node that already has it, we get the time since it was received by the node until all the nodes have it.
By assuming that all the events are i.i.d then,
$max(X_2, X_3, \ldots, X_n)$ has cdf F^{n-1}, where F is the cdf of X_i
so the desired probability is $\int F(t+x)^{n-1} f(x)dx$ where f is the density of X_1
given that X_i are event following an exponential distribution,

for $f(x) = \lambda e^{-\lambda x}$
and $F(t+x) = 1 - e^{-\lambda(t+x)}$

$$\int_0^{+\infty} F(t+x)^{n-1} f(x)dx = n - 1^{-1} e^{\lambda t}(1 - (1 - e^{-\lambda t})^{n-1})$$

□

A.2 Reception in 3 nodes

Proof. We have two random variables named X and Y which follow an exponential distribution with the same mean value.The pdf for each one is $\lambda e^{-\lambda x}$ and $\lambda e^{-\lambda y}$, and as they are independent, we can define combined pdf as: $\lambda^2 e^{-\lambda x} e^{-\lambda y}$.

As $P(R \leq t)$ is the cdf of the combined pdf, then:

$$P(R \leq t) = P(X - Y \leq t) = \iint \lambda^2 e^{-\lambda x} e^{-\lambda y} \, \mathrm{d}x \, \mathrm{d}y$$

For $Y \leq X + t$:

$$P(R \leq t) = \lambda^2 \int_0^\infty e^{-\lambda x} (\int_0^{x+t} e^{-\lambda y}) \, \mathrm{d}y \, \mathrm{d}x$$

By solving the first integral we get:

$$P(R \leq t) = \lambda \int_0^\infty (e^{-\lambda x} - e^{-\lambda t} e^{-2\lambda x}) \, \mathrm{d}x$$

That results in:

$$P(R \leq t) = \lambda(\frac{1}{\lambda} - \frac{e^{-\lambda t}}{2\lambda}) = 1 - 0.5e^{-\lambda t}$$

□

A.3 Window of Events Exponential Distribution

Proof. To obtain the window of events we multiply r by $t(p)$, where r is the rate of operations and $t(p)$ is the quantile function for the exponential distribution, in which $n-1$ is the number of nodes to which a message is send.
Therefore after time $t(p)$ there is a p probability that the message has been received in all $n-1$ nodes.

The quantile function for the exponential distribution is:

$$Q(p) = \frac{-ln(1-p)}{\lambda}$$

The probability that a message is received by all $n-1$ nodes is defined as $P(X_g \leq t)$ as is equal to the $n-1$ probabilities that a message is received by a node multiplied, $P(X_l \leq t)^{n-1}$.
If we want $p = P(X_g \leq t(p))$ then $p = \sqrt[n-1]{P(X_l \leq t(p))}$

Combining the probability that a message is received by all $n-1$ nodes and the quantile function, we get:

$$t(p) = \frac{-ln(1-\sqrt[n-1]{p})}{\lambda}$$

□

A.4 Window of Events Pareto Distribution

Proof. To obtain the window of events we multiply r by $t(p)$, where r is the rate of operations and $t(p)$ is the quantile function for the Pareto distribution, in which $n-1$ is the number of nodes to which a message is send.
Therefore after time $t(p)$ there is a p probability that the message has been received in all $n-1$ nodes.

The quantile function for the exponential distribution is:

$$Q(p) = \frac{1}{(1-p)^{-\alpha}}$$

The probability that a message is received by all $n-1$ nodes is defined as $P(X_g \leq t)$ as is equal to the $n-1$ probabilities that a message is received by a node multiplied, $P(X_l \leq t)^{n-1}$.
If we want $p = P(X_g \leq t(p))$ then $p = \sqrt[n-1]{P(X_l \leq t(p))}$

Combining the probability that a message is received by all $n-1$ nodes and the quantile function, we get:

$$t(p) = \frac{1}{(1 - \sqrt[n-1]{p})^{-\alpha}}$$

The reasoning behind this proof is equivalent to the one used for Proof A.3, but changing the exponential quantile function for the Pareto one. □

A.5 Reception Equivalence

Proof. We want to proof that Equation 3.6 is the generalization of Equation 3.5. In order to show this; we check that for $n = 3$, which is the scenario for Equation 3.5, both equations are equal.

$$1 - 0.5e^{-\lambda t} = e^{\lambda t}(1 - (1 - e^{-\lambda t})^{n-1})/n - 1$$

When $n = 3$:

$$1 - 0.5e^{-\lambda t} = e^{\lambda t}(1 - (1 - e^{-\lambda t})^{2})/2$$

$$1 - 0.5e^{-\lambda t} = e^{\lambda t}(1 - (1 - 2e^{-\lambda t} + e^{-2\lambda t})/2$$

$$1 - 0.5e^{-\lambda t} = e^{\lambda t}(2e^{-\lambda t} - e^{-2\lambda t})/2$$

$$1 - 0.5e^{-\lambda t} = 1 - 0.5e^{-\lambda t}$$

As both sides are equal, we can say that Equation 3.6 generalizes Equation 3.5 for $n = 3$. □

A.6 Quantile Equivalence

Proof. The claim is that:

$$\frac{Q_1(p)}{\lambda_1} = \frac{Q_2(p)}{\lambda_2}$$

For two exponential distributions with lambdas λ_1 and λ_2 respectively.

The quantile function for an exponential distribution can be defined as:

$$Q(p) = \frac{-ln(1-p)}{\lambda}$$

Therefore, by dividing the each quantile function by its lambda, we get:

$$\frac{\frac{Q_1(p)}{\lambda_1}}{\lambda_1} = \frac{\frac{Q_2(p)}{\lambda_2}}{\lambda_2}$$

Then we can remove the lambdas at each side, as they get cancelled by the lambda division. And we get:

$$-ln(1-p) = -ln(1-p)$$

□