Towards federated social infrastructures for plug-based decentralized social networks

Resmi Ariyattu

ANNÉE 2017

UNIVERSITÉ DE RENNES 1

UNIVERSITE BRETAGNE LOIRE

THÈSE / UNIVERSITÉ DE RENNES 1
sous le sceau de l'Université Bretagne Loire

En Cotutelle Internationale avec

pour le grade de

DOCTEUR DE L'UNIVERSITÉ DE RENNES 1

Mention : Informatique

École doctorale Matisse

présentée par

Resmi ARIYATTU

préparée à l'unité de recherche INRIA
Institut National de Recherche en Informatique et Automatique
Université de Rennes 1

Towards federated social infrastructures for plug-based decentralized social networks

Thèse soutenue à Rennes
l05 July 2017

devant le jury composé de :

David BROMBERG
Professeur des Universités, U. Rennes 1 / *Président*
Fabrice HUET
Maître de Conférence, Université de Nice / *Rapporteur*
Sara BOUCHENAK
Professeur des Universités, INSA de Lyon / *Rapporteuse*
Barry PORTER
Lecturer, University of Lancaster, UK / *Examinateur*
Anne Cécile ORGERIE
Chargée de Recherche, CNRS, IRISA / *Examinatrice*
François TAIANI
Professeur des Universités, U. Rennes 1 / *Directeur de thèse*

If you save now you might benefit later.

Acknowledgement

With immense pleasure and deep respect, I place on record my heartfelt gratitude and unforgettable indebtedness to my thesis director François Taiani for the inspiring guidance and supervision as well as the unfailing help and support throughout the course of PhD work.

I express my deep sense of gratitude to Anne-Marie Kermarrec for providing me the opportunity to work in the ASAP team.

I am extremely thankful to the esteemed members of jury for having agreed to examine my work.

I also express my thanks to all the members of ASAP team as well as all employees of INRIA/IRISA for providing me a favourable atmosphere for completing my thesis work.

I acknowledge all people who contributed in different ways to this thesis.

On a personal note, no words sustain my heartfelt thanks to my husband Arjun Suresh, my parents and my friends for their love, moral support, personal sacrifices and constant prayers which helped me to complete this venture successfully. This thesis is dedicated to them.

Above all I humbly bow my head before the Almighty who blessed me with energy and enthusiasm to complete this endeavour successfully.

Contents

Résumé

Contexte

Notre société produit, diffuse et consomme une quantité croissante de données numériques. Cette croissante s'accompagne d'une augmentation considérable de l'adoption de services en ligne (courrier électronique, sites web, jeux en ligne, vidéos à la demande, et partage pair-à-pair). Par ailleurs, les appareils tels que les consoles de jeux, les téléphones cellulaires, les tablettes, les ordinateurs portables et les ordinateurs de bureau peuvent aujourd'hui participer au stockage ainsi qu'au traitement des données produites en ligne. L'arrivée d'ordinateurs à faible coût comme les plug computers (tels que le Raspberry Pi) permet d'envisager une augmentation supplémentaires du nombre d'utilisateurs capables de fournir des capacités de calcul et de stockage. Cependant, la plupart du temps, bon nombre des ressources à la disposition des utilisateurs restent sous-utilisées. Cette sous–utilisation offre une opportunités considérable: il est possible d'imaginer des systèmes construits à partir des ressources informatiques rendus disponibles par leurs utilisateurs, ouvrant la voie à la généralisation de systèmes informatiques distribués décentralisés complexes, sur le modèle des systèmes pair-à-pair.

Les systèmes distribués décentralisés, et notamment les systèmes pair-à-pair ont suscité une attention considérable dans le passé comme solution alternative au contrôle plus centralisé des données et des services. Les approches distribuées décentralisées sont capables d'offrir de fortes capacités de passage à l'échelle, une haute disponibilité, fiabilité et évolutivité. Elles facilitent le partage des données, tout en facilitant le contrôle des données partagées localement. Une forme d'architecture distribuée décentralisée, auto–organisée et dynamique sont les systèmes pair-à-pair (Peer-to-Peer ou P2P en anglais). Un système pair-à-pair répartit de façon équitable les tâches et la charge entre les participants (les pairs) d'un système. Les pairs jouent par défaut des rôles symétriques, et sont à la fois client et serveur du système. Chaque pair (ou noeud) fournit une partie de ses ressources, telles que sa puissance de calcul, sa capacité de stockage sur disque, ou sa bande passante réseau, et les rend accessibles aux autres

participants du système, sans aucune coordination centrale. Les systèmes P2P sont capables d'offrir des services et d'exécuter des tâches qui vont au-delà de celles qui peuvent être accomplies par des pairs individuels isolés. Bien que les systèmes P2P aient été largement étudiés depuis les années 90, ils ont progressivement perdu de leur attrait au cours de la dernière décennie. Les technologies émergentes comme l'informatique diffuse (fog computing), l'informatique de proximité (edge computing), et la disponibilité d'ordinateurs de connexion à faible coût ont suscité un regain d'intérêt pour les systèmes P2P, et plus généralement les architectures décentralisées.

Dans le contexte, cette thèse s'est plus particulièrement penchée sur la maîtrise de la topologie de tels systèmes à grande échelle, une capacité clé pour permettre leur démocratisation et leur programmabilité.

Topologie des systèmes décentralisés à grande échelle

Les systèmes décentralisés à grande échelle et les protocoles sur lesquels ils sont construits fonctionnent typiquement en échangeant des messages point-à-point de façon décentralisés. À l'heure actuelle, plusieurs mécanismes hautement efficaces sont fournis pour transférer des unités de données sous toutes ses formes (vidéos, son, texte, images) entre des machines géographiquement distantes. Le mécanisme de communication point-à-point communément adopté par ces systèmes distribués est communément appelé passage de message. Les chemins de routage traversés par ces messages peuvent être analysés selon deux topologies: physique et applicative. La topologie physique fait référence à l'ensemble des liens et des routeurs Internet qui relient physiquement les machines qui constituent le système distribué. Cette topologie physique permet de transporter les octets des messages entre les participants du système, en utilisant des protocoles comme TCP / IP. La topologie logique capture, elle, l'organisation logique des noeuds au sein de l'application qu'ils réalisent. Ces deux types de topologie se chevauchent, et le contrôle de leur superposition constitue une question de recherche essentielle pour la conception des systèmes décentralisés.

Ces deux couches de topologie jouent en effet un rôle clé dans la performance d'un système décentralisé à grande échelle. Ces performances sont limitées d'une part par les latences et la bande passante disponible des liens de la topologie physique. D'autre part, la topologie virtuelle détermine en grande partie les flux de messages entre noeuds. La topologie virtuelle doit donc être idéalement choisie pour faire face à l'équilibrage de charge sur les noeuds impliqués dans les échanges de messages et empêcher les goulots d'étranglement éventuels résultant de noeuds surchargés. Par ailleurs, des précautions appropriées doivent être prises pour faire face à la dynamique inhérente d'un réseau large-échelle, en particulier lors de défaillances de machines ou de liens réseau.

Motivations et objectifs

Les paragraphes précédents ont passé en revue quelques tendances actuelles des systèmes distribués à grande échelle. Les systèmes en ligne, tels que certaines applications P2P, ou les services de vidéo à la demande ont fortement contribué à l'augmentation constante du trafic Internet mondial. Dans cette thèse nous nous intéressons particulièrement aux réseaux logiques (overlay networks en anglais), une forme particulièrement populaire de systèmes large échelle décentralisés. Nous avons déjà discuté du rôle essentiel des topologies physique et logique de tels systèmes, et de leur impact sur les performances de ces systèmes. En particulier, l'alignement (ou non) de ces deux topologies peut avoir une influence drastique sur les performances et la latence du système résultant. La recherche présentée dans cette thèse est principalement motivée par la nécessité de contrôler les différentes topologies des systèmes décentralisés large échelle, notamment en optimisant la proximité physique d'entités logiques proches, et en facilitant le placement et la recherche de contenu. Notre objectif est ainsi de faciliter l'émergence de réseaux logiques auto-organisés efficaces. Nous nous sommes concentrés en particulier sur deux aspects principaux: l'alignement des réseaux logiques avec leur infrastructure physique, et l'édition collaborative distribuée.

Alignement des réseaux logique et physique Les réseaux logiques se retrouvent dans un nombre important de systèmes distribués, tels que les réseaux pair-à-pair ou certaines applications client-serveur. Leur principal avantage est de fournir des services additionnels qui ne sont à l'origine pas disponibles sur le réseau physique sous-jacent sur lequel ils s'exécutent. Il existe cependant de nombreux défis liés à l'utilisation efficace des réseaux logiques, et plus particulièrement liés aux réseaux logiques structurés. Il est très difficile de maintenir des propriétés souhaitables comme la disponibilité et la qualité de service (Quality of Service, QoS) dans la topologie d'un réseau logique structuré, malgré les défaillances et les surcharges. Ce problème peut être résolu par des protocoles de construction de topologie décentralisée évolutifs, robustes et généralement applicables à une large gamme de systèmes et de topologies de réseaux logiques. Malheureusement, de nombreux protocoles de construction de réseaux logiques ne tiennent pas compte du réseau physique sous-jacent sur lequel un réseau logique est déployé. Ceci est particulièrement problématique pour les systèmes dématérialisés (ou "en nuage", Cloud Computing en anglais), dans lesquels la latence peut varier considérablement. Ignorer cette hétérogénéité de performance peut avoir de profondes implications en termes de performance et de latence du système global. De par le passé, plusieurs travaux ont cherché à prendre en compte la topologie de l'infrastructure sous-jacente pour réaliser des réseaux logiques prenant en compte le réseau physique (network-aware overlays en anglais). Cependant, la plupart des solutions proposées sont spécifiques à certains ser-

vices ou protocoles, et sont difficilement adaptables à d'autres réseaux logiques. Dans ce contetxe, le premier objectif de cette thèse a été de fournir une solution efficace, robuste, et générique pour résoudre ce problème.

Édition collaborative distribuée Collaborer est essentiel pour permettre à des gens ou des organisations de réaliser un objectif commun. À l'ère d'Internet, les éditeurs collaboratifs distribués sont de plus en plus utilisés. Ces éditeurs permettent à plusieurs utilisateurs distants de contribuer simultanément au même document. La plupart des éditeurs collaboratifs distribués existant sont cependant centralisés et hébergés dans des environnements étroitement intégrés (comme des clouds publics, ou des centres de traitement). Ils passent mal à l'échelle, et tolèrent mal les pannes. Ces limitations peuvent être surmontées en utilisant des architectures décentralisées pair-à-pair. Cependant, la plupart des éditeurs collaboratifs décentralisés font l'hypothèse que tous leurs utilisateurs modifient le même ensemble de documents et ne passent pas très bien à l'échelle. Dans ces systèmes, propager les changements de chaque document à l'ensemble du système surcharge rapidement la bande passante, ce que nous souhaitons éviter. Une option consiste à utiliser un DHT, pour permettre aux éditeurs d'un même document de se retrouver rapidement. Cette option est cependant sous-optimale car elle ajoute un niveau d'indirection supplémentaire dans la procédure d'appariement entre documents et utilisateur et crée des points chauds potentiels pour les noeuds qui traitent des documents très populaires. Nous proposons dans une thèse une approche qui permet aux utilisateurs qui éditent le même document de se localiser les uns les autres afin d'échanger leurs mises à jour entre eux uniquement. Notre procédure de recherche est efficace, réactive aux changements et robuste aux pannes.

Contributions

Cette thèse propose deux contributions, l'une (Fluidify) portant sur le déploiement de réseaux logiques, et l'autre (Filament) sur l'appariement d'utilisateurs dans les éditeurs collaboratifs.

Fluidify Le chapitre 3 présente Fluidify, un nouveau mécanisme décentralisé pour le déploiement d'un réseau logique au-dessus d'une infrastructure physique qui maximise la proximité physique des noeuds logiques. Fluidify utilise une stratégie double qui exploite à la fois les liaisons logiques d'un réseau logique et la topologie physique de l'infrastructure sur laquelle le réseau logique s'exécute pour aligner progressivement l'une avec l'autre. Notre approche est entièrement décentralisée et n'assume aucune connaissance globale ou autre forme centrale de coordination. Le protocole résultant

est générique, efficace et passe fortement à l'échelle.

Ce travail a été publié et présenté dans les publications suivantes:

- Full paper - Ariyattu C. Resmi, François Taïani: Fluidify: Decentralized Overlay Deployment in a Multi-cloud World. In: 15th IFIP International Conference on Distributed Applications and Interoperable Systems (DAIS 2015), held as part of the 10th International Federated Conference on Distributed Computing Techniques, DisCoTec 2015, Grenoble, France, June 2015, Springer, pp. 1–15, 15 pages.
- Poster - In: EIT Digital Symposium on Future Cloud Computing, INRIA Rennes, France, 19-20 October 2015.

Filament La second contribution de cette thèse, présentée au chapitre 4, est Filament, une nouvelle approche de construction de cohorte d'utilisateurs dans les éditeurs collaboratifs décentralisés qui permet aux utilisateurs éditant les mêmes documents de se retrouver facilement entre eux. Filament élimine la nécessité de toute DHT intermédiaire, et permet ainsi aux noeuds éditant le même document de se retrouver d'une manière rapide, efficace et robuste en générant un champ de routage évolutif autour d'eux-mêmes. L'architecture de Filament repose sur un ensemble de réseaux logiques auto-organisés coopérant entre eux, qui exploitent une nouvelle métrique de similarité entre jeux de documents. En plus de ses mérites intrinsèques, la conception de Filament démontre par ailleurs comment la composition horizontale de plusieurs réseaux logiques auto-organisés peut permettre de réaliser des services de plus haut niveau de façon très efficace.

Ce travail a été publié et présenté dans les publications suivantes:

- Short paper - Ariyattu C. Resmi, François Taïani: Filament: a cohort construction service for decentralized collaborative editing platforms In: Conférence d'informatique en Parallélisme, Architecture et Système(Compas) in July 2016, Lorient, France.
- Full paper - Ariyattu C. Resmi, François Taïani: Filament: a cohort construction service for decentralized collaborative editing platforms. Accepted and will be presented in the 17th IFIP International Conference on Distributed Applications and Interoperable Systems (DAIS 2017), held as part of the 12th International Federated Conference on Distributed Computing Techniques, DisCoTec 2017, June 2017, Neuchâtel, Switzerland.

Introduction

1.1 Context

Our modern society produces, disseminates and consumes increasingly large amounts of data. This growth is, among other thing fueled by the wide-spread adoption and growth of web-based services including on-line social networks, online gaming, video streaming and peer-to-peer file-sharing. This deluge of data has been accompanied by a multiplication of personal devices such as game consoles, cell phones, tablets, laptops and desktops, which can participate in the storage as well as processing of on-line data. This trend has been compounded by the arrival of low cost plug computers (such as the Raspberry Pi), opening the prospect of many users being able to contribute computing and storage resources. Most of the time, however, much of these user-bound resources remain idle. This under-utilisation raises the possibility of building up systems able to thrive on the available, but yet untapped computational resources made available by users over the Internet. Such a revolution would democratize large-scale fully decentralized systems, shifting the predominant computing paradigm away from centralized cloud services, and thus fulfilling the original promises of peer-to-peer systems.

Decentralized distributed systems have attracted considerable attention in the past as an alternative to the more centralized control of data and services. Decentralized distributed approaches have been shown to deliver high levels of performance, availability, reliability and scalability. They are attractive because they offer an alternative to cloud-hosted and centralized solutions, in which users must typically surrender control of their data to a central entity. Decentralized solutions make it possible for users to retain control of their data, a growing concern in today's digital society. A form of distributed architecture which is decentralized, self-organized and dynamic are *Peer-to-Peer* (P2P) systems. P2P systems partition tasks or workloads between the nodes of a distributed system (known as *peers*). Peers act both as client and server and are equally privileged and equipotent. Peers make a portion of their resources, such as

processing power, disk storage or network bandwidth, directly available to other peers, without any central coordination. P2P systems are able to provide capabilities that go well beyond those that can be accomplished by individual peers, yet that are beneficial to all peers. Although P2P systems have been extensively studied during the 90's and the first decade of this century, researchers have gradually turned away from them over the last ten years. The recent rise of decentralized technologies such as fog computing [datc] and edge computing [datb], and the broad availability of low-cost plug computers have paved the way for P2P systems to regain their lost significance, and to trigger a renewed interest in decentralized architectures.

In this context, this PhD thesis focuses more precisely on the control of the topology of large scale decentralized systems, a key concern to insure their wide-spread adoption and their programmability.

Topology of large scale decentralized systems

The large scale decentralized systems and the protocols upon which they are built, typically rely on point-to-point message passing channels to communicate. Currently, several highly efficient mechanisms can be relied on to transfer data in all its forms (video, audio, text) across geographically remote machines. The routing paths traversed by the corresponding messages can be understood according to two topologies: the physical topology, and the logical topology. The physical topology represents the set of links and Internet routers that physically connect the machines that constitute a distributed system. The links in the physical topology transport the message bytes from one end to the other using protocols such as TCP/IP. Conversely, the logical (or application-level) topology captures the logical organization of nodes within the application they implement. The two topology layers exist one on top of another, and the control of their coupling and interaction is a key research question when designing large-scale decentralized systems [ALCR+10].

These two topology layers play a key role in the performance of large-scale decentralized systems. This performance is constrained on one hand by the bandwidth available on the links of the physical topology. On the other hand, the virtual topology determines to a large extent the flows of messages between nodes. The virtual topology must therefore be picked to ensure load balancing across nodes and prevent bottlenecks arising from nodes being excessively loaded. Moreover, appropriate precautions must be taken to withstand the inherent dynamism of large-scale distributed systems, especially in order to tolerate machines and network link failures.

1.2 Motivations and Objectives

The previous sections have discussed some of the current trends in large-scale distributed systems research. Online systems, such as some P2P applications, Internet based TV or Internet-based video-on-demand services have become some of the main contributors to world-wide Internet traffic [data, datd]. The work presented in this thesis focuses more specifically on *overlay networks*, a popular type of decentralized large-scale system. We have already discussed how the physical and logical topologies of these systems could have a key impact on their efficiency. In particular, the alignment (or misalignment) between these two kinds of topology can have a drastic effect on the performance of the resulting system. The research presented in this manuscript is mainly motivated by the need to control these two topologies, in particular by placing entities that are close logically on physically-close machines, and by facilitating content search. Our overall objective in doing so is to help create powerful self-organizing overlay networks. We have focused more precisely on two main aspects in our work: the coupling between the physical and logical topologies of decentralized systems, and distributed collaborative editing.

Network aware overlays

Overlay networks play an important role in many distributed systems, such as peer-to-peer networks and some client-server applications. Their main advantage is that they provide several diverse services which are otherwise not available from the underlying network. The efficient use of overlays networks raises, however, many challenges, in particular when considering structured overlays. It is very difficult to maintain some desirable properties, such as availability and Quality-of-Service (QoS) within the topology of structured overlay networks faced with failures and churn. This difficulty can be addressed through decentralized topology construction protocols [JMB09, VS05, BFG+10] that are scalable, robust, and in general applicable to a wide range of systems and overlay topologies. Unfortunately, many popular overlay construction protocols do not usually take into account the underlying network infrastructure on which an overlay is deployed [DEF13, JMB09, VS05], and those that do tend to be limited to a narrow family of applications or overlays [ZZZ+06, XTZ03]. This is particularly problematic for systems running in multiple clouds, in which latency may vary greatly, and ignoring this heterogeneity can have stark implications in terms of performance and latency. In the past, several works have sought to take into account the topology of the underlying infrastructure to realise network-aware overlays [ZZZ+06, XTZ03, WR03, RHKS02]. However, most of the proposed solutions are service-specific and they do not translate easily to other overlays. In this context, the first objective of this thesis is to provide a sound solution to solve this problem, i.e. to create an overlay construction protocol

that considers the underlying network infrastructure on which an overlay is deployed, and that does not tend to be limited to a narrow family of applications or overlays.

Distributed collaborative editing

Collaboration is essential for people or organization to realize or achieve a common objective successfully. In the present Internet era, distributed collaborative editors are now wide-spread and regularly used. These online editors allow several remote users to contribute concurrently to the same document. Most of the currently deployed distributed collaborative editors [DSM+15, WUM10, OMMD10, LM10, GSAA04] are centralized and hosted in tightly integrated environments. They show poor scalability as well as poor fault tolerance. One way of overcoming such limitations is to host collaborative editing platforms in decentralized peer-to-peer architectures [RFH+01, SMK+01, RD01, ZKJ01, FH10]. However, most of these approaches assume that all users in a system edit the same set of documents, and are not very scalable. If we propagate the changes about every document to the entire system, then the system will be overloaded with messages, which is something we wish to avoid. Another option is to use a DHT. This option is, however, sub-optimal as it adds an extra level of indirection in the document peering procedure, and creates potential hot-spots for the DHT nodes handling highly popular documents. Instead, we argue that users editing the same document should be able to locate each other as directly as possible in order to exchange updates between themselves. To fulfill this vision, this thesis proposes a novel decentralized service that connects together the users interested in the same document in order to exchange updates regarding the documents they edit. This new procedure is efficient, reactive to changes, and robust to failures.

1.3 Contributions

This thesis contributes to overlay deployment and collaborative editing:

Fluidify - Chapter 3 presents Fluidify, a novel decentralized mechanism for deploying an overlay network on top of a physical infrastructure while maximizing network locality. Fluidify uses a dual strategy that exploits both the logical links of an overlay and the physical topology of its underlying infrastructure to progressively align one with the other. Our approach is fully decentralized and does not assume any global knowledge or central form of co-ordination. The resulting protocol is generic, efficient, and scalable.

This work was published and presented in the following publications:

- Full paper - *Ariyattu C. Resmi, François Taïani*: Fluidify: Decentralized Overlay

Deployment in a Multi-cloud World. In: 15th IFIP International Conference on Distributed Applications and Interoperable Systems (DAIS 2015), held as part of the 10th International Federated Conference on Distributed Computing Techniques, DisCoTec 2015, Grenoble, France, June 2015, Springer, pp. 1–15, 15 pages.

- Poster - In: EIT Digital Symposium on Future Cloud Computing, INRIA Rennes, France, 19-20 October 2015.

Filament - The second contribution of this thesis, introduced in Chapter 4, presents Filament, a novel cohort-construction approach that allows users editing the same documents to rapidly find each other. Filament eliminates the need for any intermediate DHT, as well as allows the nodes editing the same document to find each other in a rapid, efficient, and robust manner by generating an adaptive *routing field* around themselves. Filament's architecture hinges around a set of collaborating self-organizing overlays exploiting a novel document-based similarity metric. Besides its intrinsic merits, Filament's design further demonstrates how the horizontal composition of several self-organizing overlays can lead to richer and more efficient services.

This work was published and presented in the following publications:

- Short paper - *Ariyattu C. Resmi, François Taïani*: Filament: a cohort construction service for decentralized collaborative editing platforms In: Conférence d'informatique en Parallélisme, Architecture et Système(Compas) in July 2016, Lorient, France.

- Full paper - *Ariyattu C. Resmi, François Taïani*: Filament: a cohort construction service for decentralized collaborative editing platforms. In: 17th IFIP International Conference on Distributed Applications and Interoperable Systems (DAIS 2017), held as part of the 12th International Federated Conference on Distributed Computing Techniques, DisCoTec 2017, June 2017, Neuchâtel, Switzerland, Springer, pp. 146–160, 15 pages.

1.4 Outline

The remainder of this thesis is organized as follows. Chapter 2 provides the background information needed for clearly understanding the relevance of this work. A brief overview of the current scenario of distributed systems and the motivation behind this work are presented in this chapter. Chapter 3 presents Fluidify - a novel decentralized mechanism for deploying an overlay network on top of a physical infrastructure

while maximizing network locality. This chapter introduces the problem and some of the theoretical concepts that we employ in designing the presented algorithm before going into a detailed description of the Fluidify algorithm. It also summarizes the experimental evaluation and highlights some of the important works in this area. Chapter 4 introduces Filament, a novel cohort-construction approach that allow users editing the same documents to rapidly find each other. This chapter discusses the problem and intuition behind the work before giving an elaborate explanation of Filament algorithm. It also provides a summary of experimental evaluation and an overview of the important works done in this area. Chapter 5 concludes this work by summarizing the main contributions, discussing their implications and sketching out possible directions for future work.

Chapter 2

Self Organization in Large Distributed Systems

This chapter mainly provides the background essential for understanding the details and importance of the remainder of this manuscript. The initial part of this chapter presents an overview of some emblematic large-scale distributed systems and decentralized distributed systems. In the later sections, we provide a brief introduction to P2P systems, cloud computing and overlay networks. We also discuss the problems in this area and the ones we concentrated on.

2.1 Distributed Systems

A distributed system [And00, CDKB11] is a collection of loosely coupled machines interconnected by a communication network. The actions of these components are typically coordinated via message passing. All these interactions are aimed at achieving a common goal which is usually to run an application or a service, so that the users perceive them as a single coherent system.

During the past decades, there has been significant increase in data production and consumption. This has resulted in an increase in the number of large distributed systems which are geo-distributed and geo-replicated. Some well known examples include the systems provided by Google, Facebook and Amazon. Google has more than 13 data centres and 900,000 servers processing data [ds3]. Facebook for instance has around 1 billion users, process around 750 TB of data everyday and has a data worth over 100 PB capacities in its facility. Similarly, Amazon which has more than 7 data centres has

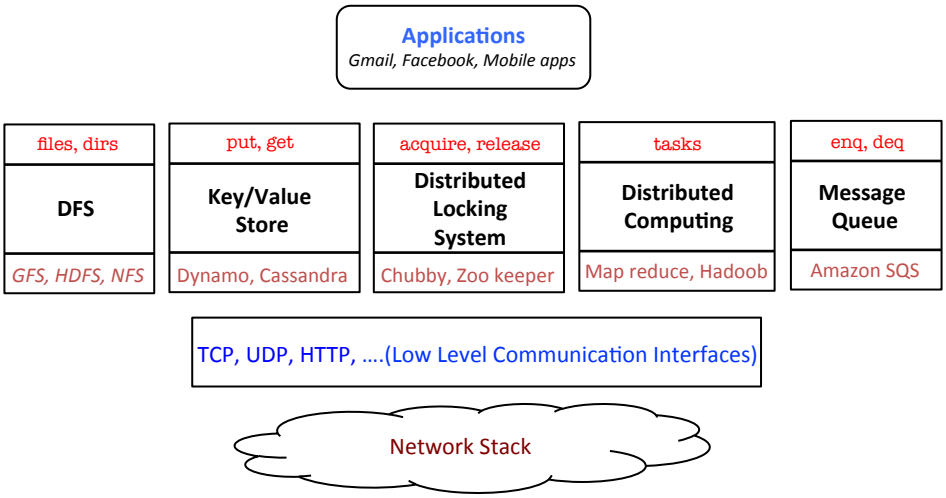


Figure 2.1: Distributed System Architecture

dedicated more than 50,000 servers to its cloud customers.

There are many reasons behind the popularity of distributed systems.

- **Location transparency** - It means that the user is not concerned about the actual location of data/services. E.g. the browser does not know which Google servers are serving the gmail to them right then. Depending on the network structure, users can easily obtain files that reside in another computer connected to a particular network. This means that the actual location of the resource doesn't matter to either the software developer or the end users. It creates the illusion of a single entity, like the entire system is located in a single computer. This simplifies the software development process greatly. It also makes the system flexible, as the system resources can be moved around at any time without disrupting any software systems running on them. In addition to location transparency we have access, migration, relocation, replication, concurrency and failure transparency.

- **Availability** - It helps to build a reliable system out of unreliable parts. Failures can happen in many ways : hardware failures, power outrage, disk failures, memory corruption, network switch failure, software failures, bugs, misconfiguration, etc. To achieve high availability, we need to replicate data/computation on many computers.

- **Scalable Capacity** - A typical data centre has around 200K machines and each service runs on more than 20K machines [ds3]. Load distribution helps in achieving enhanced system performance. Distribution makes it possible for the system

to have incremental growth and modular expandability as the computation power can be added in small increments rather than replacing the system components fully.

- **Resource Sharing and Openness** - Users can easily access remote resources or share their resources with others. A resource sharing service can be easily added and can be made available for use by a variety of client programs.

Distributed systems are usually built around complex middleware stacks [Ber96]. These stacks make the application building process easier by hiding the many complexities involved in a system and raising the level of abstraction at the implementation level. For modular functionality, an application is split into many simpler parts which may already exist or are easier to implement like authentication, indexing, locking, etc. Just by clicking on our Gmail inbox we are accessing a lot of components like load balance, authentication service, Gmail front end, storage service and ads service. Figure 2.1 illustrates this situation. The applications we use in our day-to-day life make use of distributed file systems, distributed locking systems and distributed computing each of which is contributing to the level of abstraction.

Peer-to-peer systems, SOA based systems and massively multiplayer online games are some of the currently well-known distributed systems. There are many challenges involved in designing a distributed system like finding the right interface, finding the optimal partitioning of the system, deciding how to co-ordinate machines, finding ways to authenticate clients and servers, deciding on how to protect the system in case of network and machine failures and make the system available at all times, exploring ways to maximize concurrency and reduce resource bottlenecks. Thus, distributed systems have always been an interesting area of research. The next section briefly describes some of the distributed systems like peer-to-peer systems and cloud.

2.2 Large Scale Systems: P2P systems and Cloud

Distributed systems that we know can be roughly categorized into two main categories - centralized and decentralized. Client-server systems and multi-tiered applications are examples of centralized distributed systems. They are hierarchical in nature with some nodes having higher importance compared to others. In contrast, P2P communication systems are decentralized distributed systems as all the nodes are of equal importance. These systems try to avoid single point of failure. In the following sections we discuss two of the large scale systems: P2P systems (Section 2.2.1) and cloud computing (Section 2.2.2).

2.2.1 Peer to Peer systems

Client-server architecture rose to pre-eminence during the early 1980's, but with the increase in the number of Internet users and the ever increasing demand for bandwidth it became difficult to cater to the application requirements with client-server architectures as it can lead to bottlenecks of resources. This gave rise to the development of P2P applications. In simpler terms, it is a system consisting of equal autonomous entities [RD10, SW05, BYL08]. The main aim is the shared use of distributed resources by partitioning tasks or workloads between peers.

Although the concept of P2P technology was explored since the early days, it was not widely known to the general public. The advantage of a P2P communication model became widely known in the late 1990s with the appearance of P2P music-sharing application Napster[1]. Napster allows its users to download music directly from other users in the network. The operating cost of Napster was low as the bandwidth-intensive music download was done in a distributed manner. With the widespread use of Napster, P2P systems caught the interest of industrial and research communities. Following this, many other applications were developed using the idea of cooperative resource sharing. With the addition of P2P applications like Gnutella [Rip01] and BitTorrent[2] [PGES05], P2P systems are now enjoying significant research attention and are being widely used in open software communities and industries.

P2P applications are being used in a multitude of areas as shown below :

- **Public File sharing** - File sharing helps to share information with a large number of users. Napster was one such application used for P2P file sharing. Napster was mainly used to share music among the peers and it also had an embedded search application. Napster had centralized control but gnutella which came later does not have any centralized component and is completely distributed. Tribler[3] is another example of P2P file sharing application.

- **Private file sharing** - Provides a simple and secure means of file transfer between peers using passwords and cryptographic keys. Pando [Gib] was an application used for private file sharing.

- **Data distribution** - BitTorrent is a well known P2P application used for data distribution. It helps to download large files quickly and efficiently by making use of the spare bandwidth of its peers.

[1] www.napster.com
[2] www.bittorrent.org
[3] www.tribler.org

- **Streaming media** - We can greatly reduce network costs by making use of the additional bandwidth of the participating peers. But, the main problem with streaming media is the strict timing requirements as the data needs to be delivered within a deadline. Freecast[4] is one such application.

- **Telephony** - Telephony applications help to make audio and video calls by using the unused resources of its participating peers. P2P telephony applications like Skype[5] help to provide audio visual connectivity for its users regardless of their geographical location and the type of internet connection.

- **Web search engines and Web crawler** - P2P web search engines and web crawlers are mainly used for searching peers in a P2P network for a desired content. FAROO[6] is an example.

- **Volunteer computing** - Here users donate their spare CPU cycles to scientific computations, mainly in fields such as astrophysics, biology, or climatology. BOINC[7] is a software platform for volunteer computing. SETI@home[8] was one BOINC application. When the users are inactive, their resources will be used for scientific data processing.

- **P2P Web content distribution networks (CDNs)** - CDNs helps to reduce the load associated with the server hosting the data. Participating nodes help to form web caches and also do content replication to reduce access delays and transport costs. BBC iPlayer[9] is an example of such a service.

Peer-to-peer systems have five very important features [RD10, SW05] that contribute to their widespread usage: decentralization, scalability, self-organization, fault tolerance, abundant and diverse resources.

Decentralization - The peers are considered equal autonomous entities that can act as both client and server at the same time. As the storage, processing and sharing of information is performed in a distributed manner, the system has higher availability, increased extensibility and improved resilience. Though decentralization is advantageous as we mentioned above, it also cause some difficulties. It is difficult to get and maintain a global view of the system state at any given time. Moreover, problems

[4] www.freecast.org
[5] www.skype.com
[6] www.faroo.com
[7] https://boinc.berkeley.edu
[8] http://setiathome.berkeley.edu
[9] www.bbc.co.uk/iplayer

arise when heterogeneous systems that have different characteristics, interfaces or use different protocols inter-operate.

Scalability - For the system to scale efficiently, bottlenecks must be identified and avoided at an early stage. Scalability is usually restricted by the amount of centralized operations. But, as a P2P system does not have any central servers they are inherently scalable. However, we still need to limit the number of control messages and use efficient search and distribution techniques to avoid flooding as it can increase the system load, leading to reduced scalability.

Self-organization - When a new node joins the system, little or no manual configuration should be needed for system maintenance. Even without any central management, the system is able to adapt to dynamic conditions like node heterogeneity and dynamic membership.

Fault-tolerance - The structure and organization of peer-to-peer systems are such as to make them inherently fault-tolerant. There is no central point of failure. System will be operational unless a large proportion of the nodes fails simultaneously. The loss of a peer or even a number of peers might decrease overall system performance but it can be easily remedied and the system will still be functional.

Abundant and diverse resources - P2P systems have an abundance of resources that are diverse in terms of their geographic location and hardware and software architecture. This diversity makes the system resilient to correlated failure, attack, and even censorship. The initial deployment cost of a P2P service is very low as it has little or no dedicated infrastructure. Resources can be easily added to the system as and when needed. We can also reduce the infrastructure up-gradation cost.

P2P technology faces many challenges. Due to the ad-hoc nature of P2P systems, nodes can join and leave the network at any point of time causing the availability of peers to vary considerably (known as *churn*). These fluctuating resources can affect the system performance. The lack of network bandwidth can also reduce the overall system performance. Therefore, the development of efficient replication and caching techniques which can provide high availability and performance are needed. Thus, various aspects of P2P systems like structure of overlay network, routing strategies, resource location and allocation, query processing, replication, and caching are still studied extensively.

2.2.2 Cloud Computing

Cloud computing [WAB+09, VRMCL08, BYV+09, MG+11] is a *pay-for-use* service provided over the internet. Cloud users can access a shared pool of computing resources (virtual machines, storage, applications, etc) on-demand. Clouds are used for a multi-

tude of purposes like storage, backup and recovery of data, hosting websites and blogs, streaming audio and video contents, launching new apps and services. Thus, we are making use of cloud computing on a daily basis without even knowing it, like while sending email, editing documents, watching movies, playing games and storing pictures.

Cloud computing has several attractive benefits that contribute to its widespread use in the industry [MG+11, VRMCL08]. The main benefits of cloud computing are pay-for-use model, elasticity and self-service provisioning.

Pay-for-use model - Cloud users can avoid the upfront infrastructure costs with the help of cloud computing. Users need to pay only for the resources and workloads they use. It also helps to avoid the cost of purchasing the infrastructure that may not be active all the time. Since none of the infrastructure is purchased, users do not have to worry about maintenance and other recurring costs.

Elasticity - Cloud computing services are able to scale elastically. The required amount of compute resources (computing power, storage, bandwidth) can be acquired as and when needed from a preferred geographic location. Cloud computing infrastructures can also cater for sudden workload spikes effectively. The worldwide network of secure data-centers that run the cloud services are usually equipped with the latest generation of computing resources. The developers can deliver applications in a faster manner as they do not need to worry about the cost and complexity of buying and managing the infrastructure required for the application development and provisioning.

Self-service provisioning - Cloud users can acquire compute resources, such as server time and network storage, for all types of workload on demand. There is no need for human interaction with any of the cloud service providers. The self-service feature of cloud provides a lot more business flexibility and it also eliminates the need for IT administrators to provision and manage computing resources. Cloud computing makes data backup, disaster recovery, and business continuity easier and less expensive, because data can be replicated at multiple sites on the cloud network.

Cloud computing services can be divided into three broad categories: *infrastructure as a service (IaaS)*, *platform as a service (PaaS)* and *software as a service (SaaS)*. They are referred to as the cloud computing stack and are built on top of one another.

- Infrastructure-as-a-service (IaaS) provides users with IT infrastructure(servers, networking, storage and data center space) on a pay-per-use basis, e.g. Xen[10], Oracle VM Virtual Box[11].

[10] https://www.xenproject.org
[11] https://www.virtualbox.org

- Platform as a service (PaaS) provides an on-demand cloud-based environment for developing, testing, delivering, and managing web-based applications, e.g. Microsoft Azure[12], Google App Engine[13].

- Software as a service (SaaS) - Users can use the cloud based applications that are in the cloud owned and operated by the cloud providers, via the internet, e.g. Office 365[14], Cisco WebEx[15].

Cloud computing resources can be deployed in three different ways:

Public cloud - In a public cloud, all hardware, software, and other supporting infrastructure are owned and operated by a third-party cloud service provider and services are rendered over this network for general public. Some well known public cloud service providers are Amazon Web Services (AWS), Microsoft and Google.

Private cloud - In a private cloud, all infrastructure is operated solely for a single business or organization. The infrastructure can be hosted in either an on-site data-center or a third-party cloud provider. Private cloud provides more control over the resources and there is no problems involving multi-tenancy.

Hybrid cloud - Hybrid clouds integrate the public and private clouds. The private cloud co-exist with public cloud by sharing data and applications between them, thereby providing greater flexibility as well as more deployment options.

Cloud computing adopts the features of many of the existing technologies and paradigms like grid computing and utility computing. But, we can say the starting of it was from virtualization. Virtualization isolates hardware from software, making it easier to reallocate the resources across servers based on demand. With the release of Amazon Elastic Compute Cloud (EC2) in 2006 and Microsoft Azure in 2008, clouds gained popularity. It also helps to broaden the horizons across organizational boundaries. Even though cloud computing provides so many features, there are so many challenges that need to be overcome for its smooth functioning. One of the main challenge is providing data protection and confidentiality. Since third parties are involved in the usage of cloud, strict measures are taken to provide privacy for each user. There are also some management issues that deal with the placement of data like how to provide load balancing, where the data should be placed to provide replication and availability, how to migrate the data to a different cloud provider and how to resolve software compatibility issues.

[12]https://azure.microsoft.com/
[13]https://cloud.google.com/appengine/
[14]https://www.office.com
[15]https://www.webex.com

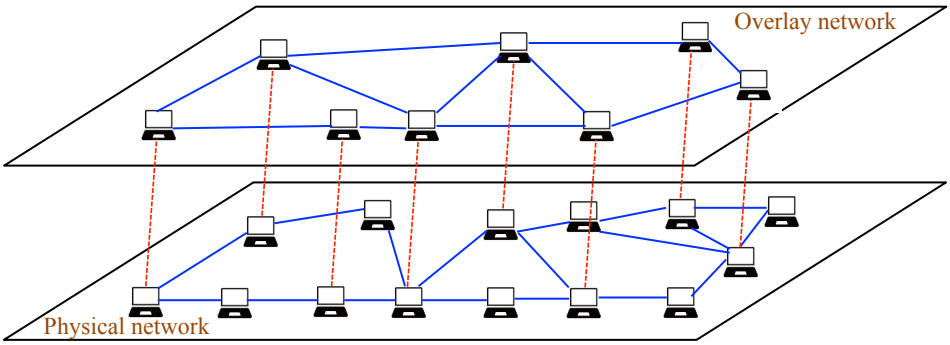


Figure 2.2: Overlay Network

2.3 Self Organization

In the previous section we discussed decentralized distributed systems and their importance. Most decentralized distributed systems rely on overlay networks for providing their services [RD01, SW05, LM10]. The structure of the underlying overlay network can greatly affect the performance of these systems. The characteristics of the environment and the application requirements can change over time. An overlay network has to take these into consideration and adapt/organize itself dynamically at run time. In this section we discuss the importance of overlay maintenance, content placement and distribution in the context of network aware self-organizing overlay network.

2.3.1 Overlay Maintenance

An overlay network [LXQ+08, FGK+09, GSAA04, KT13, RFH+01, SMK+01, LM10] is a virtual network of nodes and logical links, built on top of an existing physical network. Figure 2.2 shows an example of such an overlay network. Nodes in the overlay are usually a subset of nodes in the physical network and are able to communicate with each other via the logical overlay links. It is clear from the figure that nodes can be logical neighbours even if they are not neighbours in the underlying physical network. Overlays are used to implement a network service that is not available in the underlying physical network like mobility, addressing, customized routing, security and multicast. Thus, overlay networks play a particularly interesting role in modern distributed systems and care must be taken to make it secure, scalable and load balanced. We can broadly classify overlay networks as unstructured and structured based on how the nodes are linked to each other and how the resources are indexed and located.

Unstructured overlays

In an unstructured overlay [Rip01,FH10,JVG+07,PGES05], peers are connected by loose rules (e.g. stochastic rules), thereby forming a random graph. Gnutella [Rip01], Gossip [JVG+07] and BitTorrent [PGES05] are some of the well-known unstructured P2P protocols. The lack of structure makes it highly robust and easier to build an unstructured overlay. A new node typically joins the network via random walks. A node degree which helps to maintain the connectivity in the graph is chosen. The node degree is carefully chosen so that it does not cause any significant overlay maintenance overhead. The nodes connect randomly among themselves based on this degree. But there are certain limitations for the unstructured networks, one of which is inefficient and unreliable searching. In order to search for a desired content, the search query must be flooded through the whole network. This can lead to excessive use of network bandwidth and there is no guarantee that the search query will be resolved.

Structured overlays

Structured overlay networks [CDHR02,RFH+01,SMK+01,RD01,ZKJ01] reduce the search cost by constraining overlay structure and the placement of data. DHT (Distributed Hash Table)-based protocols make use of this structured approach. Each node is given a uniform random node identifier (NodeId) from a large id space, for example the set of 160-bit integers. Data items are also assigned unique identifiers known as keys, chosen from the same ID space. The node identifier determines the node's position in the overlay structure. Each key is mapped to a live peer in the system, usually to the one with a NodeId corresponding to the unique key. Even though we need to spend some additional resources to maintain the structure of the overlay, structured overlays makes storage and retrieval of data very efficient. A store operation (put(key,value)) and retrieval operation (value=get(key)) can be invoked for the storage and retrieval of the data item corresponding to the key. This also involves key-based routing. If a node n_0 initiates a search for a key k in the network, the key-based routing protocol KBR(n_0, k) helps to find a path from node n_0 to the node responsible for k. Each peer maintains a small routing table consisting of its neighbouring peers NodeIds and IP addresses. A lookup query is usually forwarded through the network in a progressive manner, each time getting more and more closer to the NodeId corresponding to the key. DHT-based systems may differ in their routing strategies, data organization, key space selection. However, it usually provides a deterministic guarantee that a data object can be located in O(logN) overlay hops on average, where N is the number of nodes in the system. If we require the benefits provided by key-based routing like efficient and fault tolerant routing, data location and load balancing, it might not be bad to pay the cost associated with maintaining the overlay structure.

Some of the well known self-organizing P2P overlay networks such as CAN, Chord, Pastry and Tapestry are briefly discussed below. An understanding about the working

of topology construction algorithms like Chord will be helpful while discussing the two contributions of this thesis - Fluidify (Section 3) and Filament (Section 4).

- *CAN* - CAN [RFH+01] is a distributed decentralized P2P infrastructure. It provides hash table functionality for large scale systems. The architecture of CAN is a virtual d-dimensional Cartesian coordinate space on a multi-torus. This completely logical d-dimensional space is partitioned dynamically among all the nodes in the system so that each node has its own distinct individual zone within the space. The architecture of CAN makes it self-organizing, scalable and fault tolerant. Each CAN node has a routing table associated with it which holds the IP address and virtual coordinate zone of each of its neighbors. When routing a message, each node checks the destination coordinates associated with the message and forwards the message to the neighbouring peer that is closest to the destination coordinates. The routing table entries are constricted in the sense that it contains only specific neighbouring nodes in each dimension. Unlike other DHT based protocols, the network size does not affect the routing table size but it can increase the hop count associated with routing. When a new node joins the system, it picks a random coordinate point and sends a join request. This join request is routed through the network and received by the node owning the destined region in the coordinate space. The node then splits its zone in half, keeping a half for itself and giving a half to the newly joined peer. The routing tables associated with the neighbouring peers are updated after this. When a node departs, its zone is either merged to a neighbouring zone or taken over. The routing complexity of a CAN network is $O(d.N^{1/d})$. We can increasing the routing performance of CAN in many ways like increasing the dimensionality of the hypercube, by having multiple peers managing the same zone and by performing network aware routing (Section 2.3.2). CAN is mainly used in large storage management systems.

- *Chord* - Like CAN, Chord [SMK+01] also provides DHT functionality thereby making it useful for combinatorial searches, distributed indexes and time shared storage. Chord uses a unidirectional and circular NodeId space. Peers and data keys are assigned an m-bit identifier. These identifiers are obtained by hashing the IP address and the data key correspondingly. The value of m must be large enough to avoid collisions. Data keys and peers are mapped to a circular identifier ring which can be thought of as a circle of numbers from 0 to $2m - 1$. Key k is assigned to the first peer whose identifier is equal to or follows k in the identifier space. This peer is called the successor node of key k, denoted by *successor(k)*. In the circular identifier ring, also called the Chord ring, it will be the first peer clockwise from k. Consistent hashing is used to assign keys to the peers. This

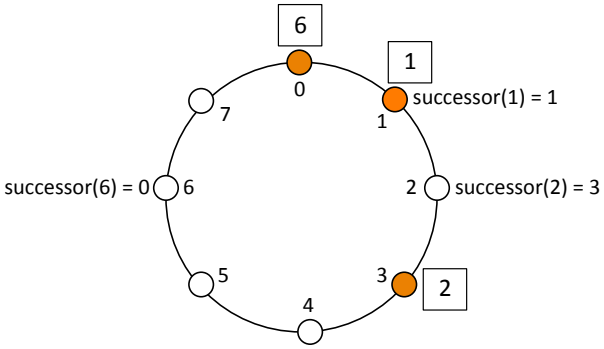


Figure 2.3: An Chord ring consisting of the three nodes 0, 1 and 3

supports the dynamic nature of the network and also helps in load balancing since each peer will be receiving nearly the same number of keys. Figure 2.3 shows a chord ring with $m = 3$ and having three nodes 0,1 and 3. Key 1 would be located at node 1 as the successor of identifier 1 is node 1. Similarly, key 2 would be located at node 3 and key 6 at node 0. In each routing step, the Chord protocol forwards the message to a node that is numerically closer to the key. It forwards messages only in clock-wise direction in the circular id space. Each peer knows how to contact its successor. To provide fault tolerance, a node can store its first r successors instead of just one. For a key lookup, the lookup message is forwarded in a circular manner from the initiating node until it reaches the node that stores the information corresponding to the key. The query visits every node in the path of initiating node and the node matching the key. To easily forward the messages, each node maintains routing state information of about $O(logN)$ other peers, referred to as the finger table. The i-th entry in the finger table of node n refers to peer with the smallest NodeId that succeeds n by at least 2^{i-1} in the circular id space. The first entry in the finger table of node n points to its successor while the subsequent entries points to nodes at repeatedly doubling distance from n. Chord guarantees a query response in logarithmic number of hops. It also ensures logarithmic update for node join/leave. When a new node n joins the system, some of the keys initially assigned to successor of n needs to be reassigned to n. Similarly, when a node n leaves the network, all the keys in it are reassigned to its successor. A background process maintains the overlay structure.

- *Pastry* - Pastry [RD01] is mainly used for application level routing and object location in a potentially large overlay network. The architecture of Pastry is a Plaxton-style global mesh network. The NodeIds are 128-bit unsigned integers and it represents a position in a circular id-space. The NodeIds are assigned

randomly and they are uniformly distributed. The NodeIds and keys are usually hexadecimal numbers. Hence the routing table associated with each node has approximately $log_{16}N$ rows and 16 columns. Each row has 15 entries. The 15 entries in row n of the routing table refers to a peer whose NodeId shares the first n digits with the current peer's NodeId, but whose $(n+1)$-th digit is m if it is in column m. The entry with the same digit as the current NodeId is left empty. At each routing step, a node forwards the routing message to a node, whose nodeId shares with the key a prefix that is at least one digit longer than the prefix that the key shares with the current node's Id. If no such node is there, the message is forwarded to a node whose Id is numerically closer to the key than the current node's Id. In addition to the routing table each node also maintains a neighbourhood list and a list of leaf nodes. The neighbourhood list M contains the NodeIds and IP addresses of the $|M|$ closest peers of the current node and helps to maintain locality. A suitable proximity metric can be chosen to increase the performance. The leaf set L contains a set of $|L|/2$ numerically closest peers in both direction of the circular id space in relation to the current node's Id. Therefore, deterministic delivery guarantees can be ensured with good reliability and fault tolerance, unless all the $|L|$ peers in the leaf set fail simultaneously. Since the routing overlay is built on top of the DHT we can switch the routing metric to one with highest bandwidth, lowest latency or shortest hop count. Pastry provides logarithmic routing complexity.

- *Tapestry* - Tapestry [ZKJ01] provides DHT functionality and can also be used for application level routing and multicasting. Tapestry also uses a variant of Plaxton-style mesh network. Though it looks similar to Pastry in first glance, the way of handling network locality and data replication is entirely different. Tapestry constructs locally optimal routing tables in order to achieve load distribution and routing locality. Tapestry also facilitates multicasting and data distribution based on application requirement. Each node is assigned a unique NodeId uniformly distributed in a 160-bit identifier space. Usually a data object is connected to a single root node but Tapestry provides scalability, availability and fault tolerance by assigning multiple root nodes to each data object. Routing is based on suffix matching in NodeId. Routing map has multiple levels where each level contains pointers to the closest peers that match the NodeId suffix corresponding to that level. This ensures logarithmic routing capacity. Each peer also maintains a neighbour list in addition to the routing table. In case of data replication, as the pointers to all nodes containing the same data object is maintained we can select one based on a suitable selection criteria based on latency or locality.

2.3.2 Network Aware Overlays

Overlay networks have been repeatedly proposed as a promising technology for network-sensitive applications such as content dissemination [GSAA04, KT13], distributed web caching [RS04, LGB03, SMB02] and stream processing [LXQ+08, FGK+09]. A crucial requirement when constructing overlay networks for such purposes is self-organization [MKG03, AB05]. It means that the system should be adaptable to dynamic changes such as node heterogeneity and dynamic memberships. Self-organization also helps to make the system fault tolerant, easily manageable and efficient. Gossip-based paradigms [FGK+09, JMB09, LM99] are typically used to implement self-organizing distributed protocols and services. Gossip-based protocols rely on periodic exchange of information between pairs of peers and the ability of a node to make local decisions. These protocols help to implement a variety of overlay network services such as membership management, dissemination, overlay structure management and key-based routing.

P2P systems are typically built on top of overlay networks. Unless specific measures are taken, the topology of *application-level / logical* overlays is independent of the underlying physical network in most situations. In an unstructured system, a new node randomly chooses some existing nodes in the system as its logical neighbors; while in a structured one, a new node will get an identifier and forms connections with other nodes according to some specific rules based on their identifier values. Thus, the proximity of two nodes in the logical overlay does not inherently reflect proximity in the underlying physical network. A message routing path with a small number of logical hops may have a long delay as it may have large number of physical hops. In order to build an effective large-scale overlay network, we need to overcome the topology mismatch between the logical overlay and physical network [MKG03, WR03, CDHR03]. The topology mismatch can also lead to a large amount of unnecessary traffic as the same message may traverse the same physical link multiple times. The efficiency and scalability of the designed overlay can be increased by having some network awareness [RHKS02, ZZZ+06, XTZ03]. A network-aware overlay (NAO) exploits locality for effective routing [CDHR03]. Applications make use of network-aware overlays to optimize network metrics such as latency, bandwidth and packet loss by reducing the number of routing hops. Maintaining network-awareness in spite of failures and churn is the main challenge when building such applications.

The main aim of DHT is to provide uniform data and load distribution. Therefore they are usually network-oblivious. Systems like CAN, Chord, Pastry and Tapestry provide scalable and fault tolerant DHTs, thereby forming self-organizing overlay networks. These systems differ in the approach they take to exploit locality in the underlying network. Both Tapestry [ZKJ01] and Pastry [RD01] measure a proximity metric

among pairs of nodes and fill their routing tables with nearby nodes to exploit network locality. Thus, the average total distance traveled by a message in Tapestry or Pastry is only a constant factor larger than the actual distance between source and destination in the underlying network. The main disadvantage of proximity based routing is that they require an expensive overlay maintenance protocol and may also compromise the load balance in the overlay network. The original design of Chord does not consider network proximity. But an extension to Chord [SMK+01] replaces its distant finger table entries with closer ones thereby providing locality. In CAN [RFH+01], each node measures its network delay to a set of landmark nodes. These measures are then used to determine the relative position of nodes in the network and construct a topology aware overlay.

In this subsection, we are only providing a brief overview of network-aware overlays. In chapter 3, we discuss our approach to creating network-aware overlays.

2.3.3 Content Placement, Search and Distribution

The rate of production, dissemination and consumption of data is steadily increasing in this digital era. Significant research is being done to conceive efficient and robust data dissemination systems providing high storage capacity, data availability and good performance [BHO+99, CDK+03, CDKR02, EGH+03]. P2P content distribution systems rely on data replication on more than one node to improve the content availability and performance [IN04]. If every node maintains a copy, then the read operation will be fast but insert and delete will be expensive. Moreover, it also has high storage overhead. So the content distribution algorithms should be efficient with respect to both time and space. There are many simple solutions for the scalable distribution of quasi-static contents while only very few scalable and consistent approaches are there in the case of highly dynamic content such as wikis.

P2P techniques for content distribution can be categorized as tree-based or epidemic protocols. In tree-based techniques [CDKR02, CDK+03], information is spread using distribution trees that are formed either with the aid of a structured overlay or embedded in an unstructured overlay. These techniques are highly efficient due to the rigid structure of the tree, but need to be rebuilt constantly under failure and churn. This constant rebuilding can affect the robust dissemination and continuity of service. Only the interior nodes in the tree contribute to data dissemination, which can lead to poor load balancing, as most of the leaf node resources remain unused. Brisa [MSF+13] is an efficient, robust and scalable data dissemination system. Some other well-known dissemination services that combine a tree structure for dissemination with an epidemic-based service for optimization are Scribe

[CDKR02] , SplitStream [CDK+03], Chunkyspread [VYF06], Bullet [KRAV03], Rappel [PRGK09], MON [LKGN05], GoCast [TCW05]. On the other hand, epidemic techniques usually form unstructured overlays where information is spread via multicast [BHO+99, HJB+09]. These techniques make use of redundancy to achieve guaranteed delivery. The main problem is the increased bandwidth and processor usage due to the transmission/processing of duplicates.

Resource discovery is another challenging issue in peer-to-peer networks. Resource discovery schemes can be broadly classified into three main categories: forwarding-based, cache-based and overlay optimization. In forwarding-based mechanisms, a peer forwards the message to a subset of neighbors rather than sending it to all its neighbors. An example is k-random walk mechanism [LCC+02] where the querying node forwards the query to k of its neighbors. After the initial step, the query is forwarded to a randomly selected neighbor. Blind search techniques like random walks are non-deterministic as the success rate greatly depends on the random choice of neighbors the query is forwarded to [OO06, YGM02]. By making use of some additional information obtained from the previous queries we can reduce the traffic overhead but the query coverage is limited. Combinations of flooding [LCC+02, DP06] and random walks [LCC+02, GMS06] were also tried to increase the search efficiency [GMS05, DLOP07]. Using blind flooding alone is not scalable because of its high communication cost. Works like [DNV06, LQGL06] are examples of cache based resource discovery technique. Some solutions that take advantage of the network topology are GUESS [DF02] and Gnutella2 [Sto02]. Plaxton routing schemes [RD01, ZKJ01] also come under this category.

Gnutella [Rip01] and Freenet [CSWH01] are two well known distributed search techniques. Though both of them have similar architecture (random DHT), the resource discovery mechanisms employed by both are different - Gnutella uses broadcast while Freenet uses chain routing. The main drawbacks of these techniques are slow information discovery and high communication cost. Gnutella faces serious scaling and reliability problems with very large network size [DLOP07]. Freenet has better scalability but search efficiency is low because of long query traversal paths. [FH10] improve the search performance in unstructured peer-to-peer networks. It employs a caching technique along with a dynamic Time-To-Live (TTL) to redirect queries towards the right direction and to reliably discover rare resources.

In this subsection, we are only providing a brief overview of content placement, search and distribution techniques. In chapter 4, we discuss our approach to peer clustering in collaborative editing.

2.4 Summary

In this chapter we presented some of the essential background needed for the clear understanding of the rest of the thesis. The significant increase in the data production and consumption during the past decades has resulted in an increase in the number of large distributed systems. They are geo-distributed and geo-replicated. P2P communication systems are an example of decentralized distributed systems with all nodes having equal importance. The study of large scale systems like P2P systems and cloud holds great importance. Most of the decentralized distributed systems rely on overlay networks for providing their services. The structure of the underlying overlay network can greatly influence the performance of these systems. Thus, a network aware self-organizing overlay can be very helpful for decentralized distributed systems. So there is a need for efficient overlay maintenance techniques and care must be given to content placement, search and distribution.

Chapter 3

Fluidify: Decentralized Overlay Deployment in a Multi-Cloud World

In the previous chapter we have mentioned the importance of network aware overlays and the general lack of awareness of the underlying physical network topology by decentralized overlay based systems. In this chapter we are presenting our approach to address this problem. As overlays get deployed in large, heterogeneous systems with stringent performance constraints, their logical topology must exploit the locality present in the underlying physical network. In this chapter, we propose a novel decentralized mechanism—Fluidify—for deploying an overlay network on top of a physical infrastructure while maximizing network locality. Fluidify uses a dual strategy that exploits both the logical links of an overlay and the physical topology of its underlying network to progressively align one with the other. The resulting protocol is generic, in the sense that it does not make any assumptions regarding the logical topology to be deployed, or the underlying physical topology on which the deployment occurs. Fluidify is efficient, scalable and can substantially improve network overheads and latency in overlay based-systems. Simulation results show that in a network of 25,600 nodes, Fluidify is able to produce an overlay with links that are on average 94% shorter than that produced by a standard decentralized approach based on slicing and 97% shorter compared to that produced by a decentralized approach based on PROP-G, while demonstrating a sub-linear time complexity.

Section 3.1 introduces the problem and some important works in this area. Section 3.2 presents our intuition. Section 3.3 presents the Fluidify algorithm. Section 3.4 sum-

marizes the experimental evaluations of Fluidify in terms of proximity and convergence time.

3.1 Background and Problem Statement

As discussed in chapter 2, overlays are increasingly used as a fundamental building block of modern distributed systems. Overlay networks organize peers in logical topologies on top of an existing network to extend its capabilities, with application to storage [RFH+01, SMK+01], routing [GSAA04, KT13], recommendation [VS05, BFG+10], and streaming [LXQ+08, FGK+09]. Although overlays were originally proposed in the context of peer-to-peer (P2P) systems, their application today encompasses wireless sensor networks [GHP+08] and cloud computing [DHJ+07, LM10].

3.1.1 The problem: building network-aware overlays

One of the challenges when using overlays, in particular structured ones, is to maintain desirable properties within the topology, in spite of failures and churn. This challenge can be addressed through decentralized topology construction protocols [JMB09, VS05, MJB05, LPR07], which are scalable and highly flexible. Unfortunately, such topology construction solutions are not usually designed to take into account the infrastructure on which an overlay is deployed. This brings clear advantages in terms of fault-tolerance, but is problematic from a performance perspective, as overlay links may in fact connect hosts that are far away in the physical topology. This is particularly likely to happen in heterogeneous systems, such as multi-cloud deployment, in which latency values might vary greatly depending on the location of individual nodes.

Cloud infrastructures, with its inherent elasticity and scalability, help to process large amounts of scientific data. Cloud resources for computation and storage are spread among globally distributed datacenters. This distribution provides redundancy and ensures reliability in case of site failures. In order to optimally use the full computation power of clouds, we have to exploit locality across multiple sites as there are frequent large-scale data movements between them. Inefficient data management across geographically distributed datacenters can cause high and variable latencies among sites which is costly. To achieve reasonable QoS and optimize the cost-performance in a federated cloud setting it is critical to effectively use the underlying storage and network resources.

For instance, Fig. 3.1(a) depicts a randomly connected overlay deployed over two cloud providers (rounded rectangles). All overlay links cross the two providers, which

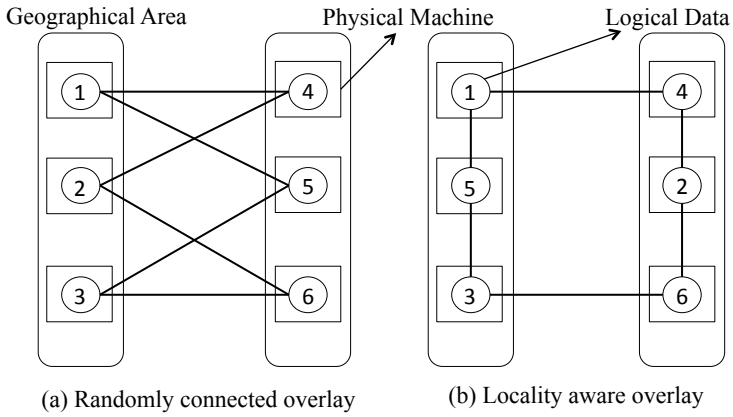


Figure 3.1: Illustration of a randomly connected overlay and a network-aware overlay

is highly inefficient. By contrast, in Fig. 3.1(b), the same logical overlay only uses two distant links, and thus minimizes latency and network costs.

This problem has been explored in the past [ZZZ+06, XTZ03, WR03, RHKS02, QCY+07], but most of the proposed solutions are either tied to a particular service or topology, or limited to unstructured overlays, and therefore cannot translate to the type of systems we have just mentioned, which is exactly where the work we present in this chapter comes in.

3.1.2 Existing approaches to building network-aware overlays

Fully decentralized systems have been extensively studied by many researchers. This also led to the study of network aware overlays and ways to create them. Some of the important works in this field are discussed below. We first give a brief overview of the types of network aware overlays. Then we discuss how locality awareness is provided for P2P systems - approaches specific to structured overlays, generic approaches and those specific to unstructured overlays. The last part is about making use of topology aware protocols for routing and data dissemination.

Network aware overlays can be either reactive or proactive. Reactive overlays initiate network measurements to determine node distances when routing a message. This can cause a large overhead when overlay usage is high. An example of a reactive network aware overlay is Meridian [WSS05]. Each Meridian node organizes a fixed set of neighbours into exponentially increasing rings according to network distance. Routing is done based on the current latency measurements but each query has a large mea-

surement overhead. Meanwhile, proactive overlays periodically perform measurements in the background to maintain up-to-date routing state. Even though it can reduce the measurement overhead associated with overlay usage, it can suffer heavily from stale information. Tulip [ABB+05] is an example of proactive network aware two hop routing overlay. Each node selects a random color and maintains a color list with all nodes of its color and a vicinity list containing one node of every other color. A node routes a message by sending it to the node in the vicinity list of the target's color. That node routes the message directly to the target using its color list. Tulip makes use of gossip and direct node-to-node measurements to ensure locality of its vicinity list. This results in a large maintenance overhead.

Studies were performed to provide locality awareness for P2P systems. We know that P2P systems can be structured or unstructured. Structured P2P overlays, such as CAN [RFH+01], Chord [SMK+01], Pastry [RD01], and Tapestry [ZKJ01], are designed to enhance search performance by giving some importance to node placement. But, as pointed out in [RS02], structured designs are likely to be less resilient, because it is hard to maintain the structure required for routing to function efficiently when hosts are joining and leaving at a high rate. A detailed description of all the four protocols is given in Sec. 2.3.1. The original design of Chord does not consider network proximity at all. CAN, Pastry, and Tapestry proposed some mechanisms to provide locality to some extent. In CAN, a landmark based similarity is used to find the relative position of a node in the Internet. Tapestry and Pastry exploit locality by measuring the proximity between pairs of nodes, and adding the closest ones to the corresponding routing tables. However, these mechanisms are typically specific to the overlay in which they are employed and come at the expense of a significantly more expensive overlay maintenance protocol. It also compromises the load balance in the overlay network.

Some of the approaches are generic in nature. They can be used for both structured and unstructured overlays. One such approach to creating network aware overlays is to use network coordinates (NC) [PLMS06]. They provide a decentralized construction mechanism for adaptive network aware overlays. Each node calculates its position in a virtual coordinate space using a small number of inter-node network latency measurements. Nodes then continuously adjust their NCs to adapt to the changes in the underlying network. NCs have many attractive properties like low run-time overhead, low embedding error and they also provide geometric primitives for solving problems like nearest cache selection, content distribution and resource placement. But the main problem is that they have too low accuracy and too high measurement overhead [PLMS06].

Another approach used to bridge the gap between physical and overlay node proximity is landmark clustering. Peers measure the latency between them and a set of stable

internet servers called *landmarks* and use these measured latencies to determine proximity. It can be used in combination with both structured and unstructured overlays. Ratnasamy et al. [RHKS02] use landmark clustering in an approach to build a topology-aware CAN [RFH+01] overlay network. Although the efficiency can be improved, this solution needs extra deployment of landmarks and produces some hotspots in the underlying network when the overlay is heterogeneous and large. Some [ZZZ+06] [XTZ03] have proposed methods to fine-tune the landmark clustering for overlay creation. The main disadvantage with landmark systems is that there needs to be a reliable infrastructure to offer these landmarks with a high availability.

Proximity neighbour selection [CDHR03] which is similar to landmark clustering was proposed to organise and maintain the overlay network with improved routing speed and load balancing. Waldvogel and Rinaldi [WR03] proposed an overlay network (Mithos) that focuses on reducing routing table sizes for structured overlays. Here the addresses are directly mapped into a subspace of IPv6 address space thereby making it suitable for native deployment and efficient forwarding. To maximize the accuracy and efficiency without the use of multi-dimensional coordinate space and full mesh probing, it uses every node in the network as a topology landmark. It is a bit expensive and only very small overlay networks are used for simulations and the impact of network digression is not considered. As the routing tables do not contain long distance links, the hop count is also high.

Many approaches were proposed solely for unstructured overlays. Application layer multicast algorithms construct a special overlay network that exploits network proximity. The protocol they use are often based on a tree or mesh structure. Mesh structures such as Narada [CRSZ02] are suitable for small overlays but are often not scalable. The mesh (an unstructured overlay) that is constructed in a decentalized manner is later optimized according to several network metrics (load, distance, node capacity). Narada requires each node to maintain a global membership information and to update it as nodes leaves and join the system. In a tree structure such as NICE [BBK02], a new node who wants to join the overlay will first contact the highest level host. This will create a hot spot. If by any chance higher-level hosts fail, the system becomes unstable and might need a rather long time to recover. This also limits the scalability of the approach.

In [KW00, KW01] authors use the concept of topology-awareness. However, their method directly uses the border gateway protocol (BGP) to route information. For an overlay network which is built on top of an application layer, it might not be practical to use that kind of information. Zhang et al. [ZZZ+06] construct a network aware overlay by using the group concept. They use dynamic landmarks to avoid hot spots and to achieve load balance. Another method known as Locality aware Topology

Matching (LTM) which aims to alleviate the mismatch problem was proposed by Liu et al. [ZLX+05]. In LTM, each peer issues a detector (a control message) so that the peers receiving the detector can record relative delay information. Based on the delay information, a node can detect and cut most of the inefficient and redundant logical links and add closer nodes as its direct neighbors.

Works like [MSF+13] and [DEF13] combines the robustness of epidemics with the efficiency of structured approaches in order to improve the data dissemination capabilities of the system. Gossip protocols which are scalable and inherent to network dynamics can do efficient data dissemination. Frey et al. [FGK+09] uses gossip protocols to create a system where nodes dynamically adapt their contribution to the gossip dissemination according to the network characteristics like bandwidth and delay. Directional Gossip [LM99] creates an overlay targeting wide-area networks. It computes a weight for each neighbour which reflects the connectivity and the network topology. But, this can lead to a static hierarchy sensitive to failures. Kermarrec et al. [GKW13] use gossip protocols for renaming and sorting. Here nodes are given id values and numerical input values. Nodes exchange these input values so that in the end the input of rank k is located at the node with id k. This slicing method [PMRS14] [JK06] was used in resource allocation. Specific attributes of network (memory, bandwidth, computation power) are taken into account to partition the network into slices. Network aware overlays can be used in cloud infrastructure [TCW+14] to provide efficient data dissemination.

Most of the works on topology aware overlays are aimed at improving a particular service such as routing, resource allocation or data dissemination. What we are proposing is a generalized approach for overlay creation giving importance to data placement in the system. Our approach is scalable, robust and has very low maintenance cost. The simulated annealing, slicing approach and PROP-G that we used as baselines is motivated mainly by the works [PMRS14], [GKW13], [JK06], [QCY+07]. But these works concentrated mainly on improving a single network service while we concentrate on a generalized solution that can significantly improve all the network services.

3.2 Our intuition: a dual approach

Our proposal, Fluidify, uses a dual strategy that exploits both an overlay's logical links and its physical topology to incrementally optimize its deployment.

We model a deployed overlay as follows: each node possesses a physical index, representing the physical machine on which it runs, and a logical index, representing its logical position in the overlay. Each node also has a physical and logical neighbourhood:

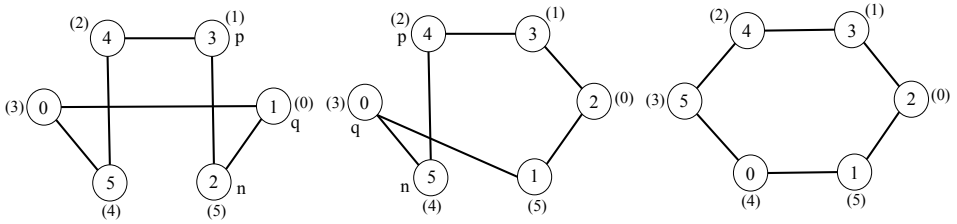


Figure 3.2: Example of basic Fluidify approach on a system with n=6 and d=2

the physical neighbors of a node are its d closest neighbors in the physical infrastructure, according to some distance function $\mathsf{d}_{\text{net}}()$ that captures the cost of communication between nodes. The logical neighbors of a node are the node's neighbors in the overlay being deployed. For simplicity sake, we model the physical topology as an explicit undirected graph between nodes, with a fixed degree. We take d to be the fixed degree of the graph, and the distance function to be the number of hops in this topology.

This model is illustrated in Fig. 3.2. Logical indices are shown inside the nodes, and physical indices outside. The physical infrastructure is modelled as a ring (not shown) in which the physical index i is connected to the indices $(i-1) \mod 6$ and $(i+1) \mod 6$. The logical overlay being deployed is also a ring (solid lines). Fig. 3.2(a) shows an initial configuration in which the overlay has been deployed without taking into account the underlying physical infrastructure. The two logical indices 0 and 1 are neighbors in the overlay, but are diametrically placed in the underlying infrastructure. By contrast Fig. 3.2(c) shows an optimal deployment in which the logical and physical links overlap.

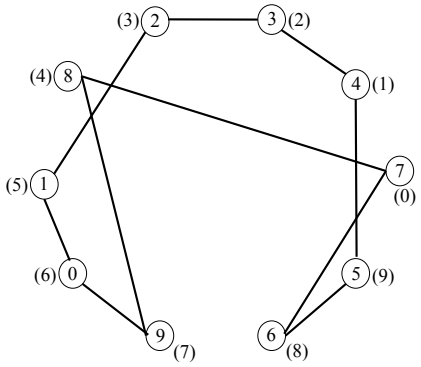


Figure 3.3: Example of local minimum of a system with n=10 and d=2

Our intuition, in Fluidify, consists of exploiting both the logical and physical neighbors of individual nodes, in a manner inspired from epidemic protocols, to move from the configuration of Fig. 3.2(a) to that of Fig. 3.2(c). Our basic algorithm is organized in asynchronous rounds, and implements a greedy approach as follows: in each round, each node n randomly selects one of its *logical* neighbors (noted p), and considers the *physical* neighbor of p (noted q) that is closest to itself. n evaluates the overall benefit of exchanging its logical index with that of q. If positive, the exchange occurs (Fig. 3.2(b), and then Fig. 3.2(c)).

Being a greedy algorithm, this basic strategy carries the risk of ending in a local minimum (Fig.3.3). To mitigate such situations, we use simulated annealing (taking inspiration from recent works on epidemic slicing [PMRS14]), resulting in a decentralized protocol for the deployment of overlay networks that is generic, efficient, and scalable.

3.3 The Fluidify algorithm

We now present in more detail how we realize the intuition we have just sketched, by precising our system model (Sec. 3.3.1) before moving on to the algorithm implementing Fluidify (Sec. 3.3.2).

3.3.1 System model

We consider a set of nodes $N = \{n_1, n_2, .., n_N\}$ in a message passing system. Each node n possesses a physical index(n.net) and a logical index (n.data). n.net represents the machine on which a node is deployed. n.data represents the role n plays in the overlay, e.g. a starting key in a Chord ring [MJB05, SMK+01].

Table 3.1 summarizes the notations we use. We model the physical infrastructure as an undirected graph $\mathcal{G}_{\text{net}} = (N, E_{\text{net}})$ and capture the proximity of nodes in this physical infrastructure through the distance function $\mathsf{d}_{\text{net}}()$. Similarly, we model the overlay being deployed as an undirected graph $\mathcal{G}_{\text{data}} = (N, E_{\text{data}})$ over the nodes N. In a first approximation, we use the hop distance between two nodes in $\mathcal{G}_{\text{net}}$ for $\mathsf{d}_{\text{net}}()$, but any other distance would work. The hop distance is clearly a simplified view of a real network, in which latency and bandwidth depend on many other variables, and might not behave symmetrically between a pair of nodes. However, this simplification fits our aim, which is to demonstrate the principles of our approach, rather than propose a detailed modelling of current networks which is a highly complex task. Let us note however, that our approach would continue to apply to any distance metric.

Our algorithms use the k-NN neighborhood of a node n in a graph $\mathcal{G}_x$, i.e. the

Table 3.1: Notations and Entities

n.net	physical index of node n
n.data	logical index of node n
d_{net}	distance function to calculate the distance between two nodes in physical space
$\mathcal{G}_{\text{net}}$	the physical graph (N, E_{net})
$\mathcal{G}_{\text{data}}$	the logical graph (N, E_{data})
$\Gamma^k_{\text{net}}(n)$	k closest nodes to n in $\mathcal{G}_{\text{net}}$, in hop distance
$\Gamma^k_{\text{data}}(n)$	k closest nodes to n in $\mathcal{G}_{\text{data}}$, in hop distance

Table 3.2: Parameters of Fluidify

k_{net}	size of the physical neighborhood explored by Fluidify
k_{data}	size of the logical neighborhood explored by Fluidify
K_0	initial threshold value for simulated annealing
$r_{\max}$	fade-off period for simulated annealing (# rounds)

k nodes closest to n in hop distance in $\mathcal{G}_x$, which we note as $\Gamma^k_x(n)$. In case of a tie, nodes are chosen randomly. We assume that these k-NN neighborhoods are maintained with the help of a topology construction protocol [JMB09, VS05, BFG+10]. In the rest of the chapter, we discuss and evaluate our approach independently of the topology construction used, to clearly isolate its workings and benefits. Under the above model, finding a good deployment of $\mathcal{G}_{\text{data}}$ onto $\mathcal{G}_{\text{net}}$ can be seen as a graph mapping problem, in which one seeks to optimize the cost function $\sum_{(n,m)\in E_{\text{data}}} \mathsf{d}_{\text{net}}(n, m)$.

3.3.2 Fluidify

The basic version of Fluidify (termed Fluidify (basic)) directly implements the ideas discussed in Sec. 3.2 (Algorithm. 1): each node n first chooses a random *logical* neighbor (noted p, line 2), and then searches for the *physical* neighbor of p (noted q) that offers the best reduction in cost (arg min operator at line 3)[1]. The code shown slightly generalises the principles presented in Sec. 3.1, in that the nodes p and q are chosen beyond the 1-hop neighborhood of n and p (lines 2 and 3), and consider nodes that are k_{data} and k_{net} hops away, respectively.

The potential cost reduction is computed by the procedure $\Delta(n, u)$ (lines 5-8), which returns the cost variation if n and u were to exchange their roles in the overlay. The decision whether to swap is made in $\mathsf{conditional\_swap}(n, q, \delta_{\text{lim}})$ (with $\delta_{\text{lim}} = 0$ in Fluidify Basic).

[1] $\arg\min_{x\in S}\big(f(x)\big)$ returns one of the x in S that minimizes $f(x)$.

Algorithm 1 Fluidify (basic)

1: **In round**(r) **do**
2: $p \leftarrow$ random node from $\Gamma^{k_{\text{data}}}_{\text{data}}(n)$
3: $q \leftarrow \arg\min_{u \in \Gamma^{k_{\text{net}}}_{\text{net}}(p)} \Delta(n, u)$
4: conditional_swap$(n, q, 0)$
5: **Procedure** $\Delta(n, u)$
6: $\delta_n \leftarrow \sum_{(n,r) \in E_{\text{data}}} \mathsf{d}_{\text{net}}(u, r) - \sum_{(n,r) \in E_{\text{data}}} \mathsf{d}_{\text{net}}(n, r)$
7: $\delta_u \leftarrow \sum_{(u,r) \in E_{\text{data}}} \mathsf{d}_{\text{net}}(n, r) - \sum_{(u,r) \in E_{\text{data}}} \mathsf{d}_{\text{net}}(u, r)$
8: **return** $\delta_n + \delta_u$
9: **Procedure** conditional_swap$(n, q, \delta_{\text{lim}})$
10: **if** $\Delta(n, q) < \delta_{\text{lim}}$ **then**
11: swap n.data and q.data
12: swap $\Gamma^{k_{\text{data}}}_{\text{data}}(n)$ and $\Gamma^{k_{\text{data}}}_{\text{data}}(q)$
13: **end if**

To mitigate the risk of local minima, we extend it with simulated annealing [PMRS14], which allows two nodes to be swapped even if there is an increase in the cost function. We call the resulting protocol Fluidify (SA), shown in Algorithm 2. In this version, we swap nodes if the change in the cost function is less than a limit, $\Delta_{\text{limit}}(r)$, that gradually decreases to zero as the rounds progress (line 4). $\Delta_{\text{limit}}(r)$ is controlled by two parameters, K_0 which is the initial threshold value, and $r_{\max}$ which is the number of rounds in which it is decreased to 0. In the remainder of this thesis, we use Fluidify to mean Fluidify (SA).

Algorithm 2 Fluidify (SA)

1: **In round**(r) **do**
2: $p \leftarrow$ random node from $\Gamma^{k_{\text{data}}}_{\text{data}}(n)$
3: $q \leftarrow \arg\min_{u \in \Gamma^{k_{\text{net}}}_{\text{net}}(p)} \Delta(n, u)$
4: conditional_swap$(n, q, \Delta_{\text{limit}}(r))$
5: **Procedure** $\Delta_{\text{limit}}(r)$
6: **return** $\max\big(0, K_0 \times (1 - r/r_{\max})\big)$

3.4 Evaluation

3.4.1 Experimental Setting and Metrics

Unless otherwise indicated, we use rings for both the infrastructure graph $\mathcal{G}_{\text{net}}$ and overlay graph $\mathcal{G}_{\text{data}}$. We assume that the system has converged when the system remains stable for 10 rounds.

The default simulation scenario is one in which the system consists of 3200 nodes, and use 16-NN logical and physical neighborhoods ($k_{\text{net}} = k_{\text{data}} = 16$) when selecting p and q. The initial threshold value for simulated annealing (K_0) is taken as $|N|$. $r_{\max}$ is taken as $|N|^{0.6}$ where 0.6 was chosen based on the analysis of the number of rounds Fluidify (basic) takes to converge.

We assess the protocols using two metrics:

- **Proximity** captures the quality of the overlay constructed by the topology construction algorithm. Lower values denote a better quality.
- **Convergence time** measures the number of rounds taken by the system to converge.

Proximity is defined as the average network distance of logical links normalized by the diameter of the network graph $\mathcal{G}_{\text{net}}$:

$$\textit{proximity} = \frac{\mathop{\mathbb{E}}_{(n,m)\in E_{\text{data}}} \mathsf{d}_{\text{net}}(n,m)}{\text{diameter}(\mathcal{G}_{\text{net}})} \tag{3.1}$$

where $\mathbb{E}$ represents the expectation operator, i.e. the mean of a value over a given domain, and *diameter*() returns the longest shortest path between pairs of vertices in a graph, i.e. its diameter. In a ring, it is equal to $N/2$.

3.4.2 Baselines

The performance of our approach is compared against three other approaches. One is Randomized (SA) (Algorithm. 3), where each node considers a set of random nodes from N for a possible swap. The other is inspired from epidemic slicing [PMRS14,JK06], and only considers the physical neighbors of a node n for a possible swap (Slicing (SA), in Algorithm. 4). The third approach is similar to PROP-G [QCY+07], and only considers logical neighbours of a node n for a possible swap (PROP-G (SA), in Algorithm. 5).

In all these approaches simulated annealing is used as indicated by (SA). The only difference between the above four approaches is the way in which the swap candidates are selected.

Algorithm 3 Randomized (SA)

1: **In round**(r) **do**
2: $S \leftarrow k_{\text{net}}$ random nodes from N
3: $q \leftarrow \arg\min_{u \in S} \Delta(n, u)$
4: conditional_swap$\big(n, q, \Delta_{\text{limit}}(r)\big)$

Algorithm 4 Slicing (SA)

1: **In round**(r) **do**
2: $q \leftarrow \underset{u \in \Gamma_{\text{net}}^{k_{\text{net}}}(n)}{\arg\min} \Delta(n, u)$
3: conditional_swap$\big(n, q, \Delta_{\text{limit}}(r)\big)$

Algorithm 5 PROP-G

1: **In round**(r) **do**
2: $S \leftarrow \Gamma_{\text{data}}^{k_{\text{data}}}(n)$
3: $q \leftarrow \arg\min_{u \in S} \Delta(n, u)$
4: conditional_swap$(n, q, 0)$

To provide further comparison points, we also experimented with some combinations of the above approaches. Algorithm. 6 (termed *Data-Net & Net*) is a combination of Fluidify (basic) with Slicing (SA). Algorithm. 7 (termed *Data-Net & R*) is a combination of Fluidify (basic) with Randomized (SA). We also tried a final variant, *combination-R*, in which once the system has converged using Fluidify (basic) (no more changes are detected for a pre-determined number of rounds), nodes look for random swap candidates like we did in Algorithm. 3.

3.4.3 Results

All the results (Figs. 3.4-3.11 and Tables 3.3-3.5) are computed with Peersim [MJ09] and are averaged over 30 experiments. When shown, intervals of confidence are computed at a 95% confidence level using a student t-distribution. The source code is made available in http://armi.in/resmi/fluidify.zip.

Algorithm 6 Data-Net & Net

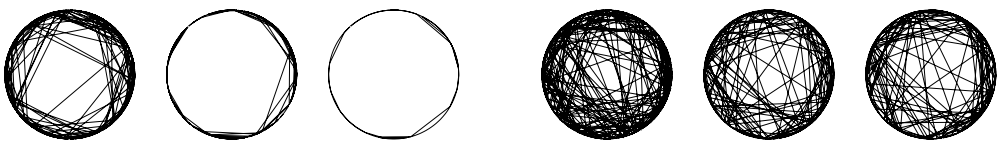
1: **In round**(r) **do**
2: $p \leftarrow$ random node from $\Gamma_{\text{data}}^{k_{\text{data}}}(n)$
3: $S \leftarrow \Gamma_{\text{net}}^{\frac{k_{\text{net}}}{2}}(p) \cup \Gamma_{\text{net}}^{\frac{k_{\text{net}}}{2}}(n)$
4: $q \leftarrow \arg\min_{u \in S} \Delta(n, u)$
5: conditional_swap$(n, q, 0)$

Algorithm 7 Data-Net & R

1: **In round**(r) **do**
2: $p \leftarrow$ random node from $\Gamma_{\text{data}}^{k_{\text{data}}}(n)$
3: $S \leftarrow \Gamma_{\text{net}}^{\frac{k_{\text{net}}}{2}}(p) \cup \left\{ \frac{k_{\text{net}}}{2} \text{ rand. nodes } \in N \setminus \Gamma_{\text{net}}^{\frac{k_{\text{net}}}{2}}(p) \right\}$
4: $q \leftarrow \arg\min_{u \in S} \Delta(n, u)$
5: conditional_swap$(n, q, 0)$

3.4.3.1 Evaluation of Fluidify (SA)

The results obtained by Fluidify (SA) and the three baselines on a ring/ring topology are given in Table 3.3 and charted in Figs. 3.5 and 3.6. In addition, Fig. 3.4 illustrates some of the rounds that Fluidify (SA) and Slicing (SA) perform. Fig. 3.5 shows that Fluidify clearly outperforms the other three approaches in terms of proximity over a wide range of network sizes. For Fluidify, varying the network size from 100 to 25,600 results in the reduction of proximity from 4% to 0.43%. But the reduction is from 7.7% to 1.79% and 10.46% to 6.99% respectively, for Randomized (SA) and Slicing (SA). But for the PROP-G (SA) approach, increasing the network size is not showing much of an effect. In the case of Fluidify, the converged proximity value for network sizes above 6400 is around 0.5% and does not show much of a variation thereafter.



(a) Fluidify (SA) (b) Slicing (SA)

Figure 3.4: Illustrating the convergence of Fluidify (SA) & Slicing (SA) on a ring/ring topology. The converged state is on the right. ($N = K_0 = 400$, $k_{\text{net}} = k_{\text{data}} = 16$)

Table 3.3: Performance of Fluidify against various baselines(with simulated annealing)

Nodes	**Proximity(%)**				**Convergence (rounds)**			
	Fluidify	Slicing	Random	PROP-G	Fluidify	Slicing	Random	PROP-G
100	*4.06*	10.46	7.70	13.88	18.10	17.16	23.80	*17.03*
200	*2.70*	10.12	6.27	12.99	28.50	26.33	43.43	*25.13*
400	*1.71*	9.76	5.35	12.65	42.50	39.20	85.36	*38.06*
800	*1.26*	9.34	4.83	12.14	64.13	58.93	136.76	*57.16*
1,600	*0.86*	8.80	4.41	11.57	96.80	90.56	198.03	*85.13*
3,200	*0.69*	8.47	3.82	11.31	144.40	138.20	274.80	*128.14*
6,400	*0.51*	8.13	3.07	11.27	216.10	203.40	382.10	*198.24*
12,800	*0.46*	7.66	2.28	11.01	324.00	292.10	533.67	*263.32*
25,600	*0.43*	6.99	1.79	10.02	485.00	418.60	762.13	*392.81*

Fig 3.6 charts the convergence time against network size in loglog scale for Fluidify and its competitors. Interestingly all approaches show *a polynomial convergence time* with a sub linear slope, of which Randomized (SA) taking clearly longer than the others. This shows the scalability of Fluidify even for very large networks. If we turn to Tab. 3.3, it is evident that as the network size increases, the time taken for the system to converge also increases. Both Fluidify and Slicing (SA) converges around the same time with Slicing (SA) converging a bit faster than Fluidify. Randomized (SA) takes much longer (almost twice as many rounds). PROP-G (SA) converges faster compared than all other approaches.

The better convergence of PROP-G (SA) and Slicing (SA) can be explained by the fact that both approaches run out of interesting swap candidates more rapidly than Fluidify. It is important to note here that all approaches are calibrated to consider the same number of candidates per round. This suggests that PROP-G (SA) and Slicing (SA) runs out of potential swap candidates because they consider candidates of lesser quality, rather than considering more candidates faster.

Fig. 3.7 shows how the proximity varies with round for our default system settings. Initial avg. link distance was around $N/4$ where N is the network size and this is expected as the input graphs are randomly generated. So the initial proximity was approximately equal to 50%. Fluidify was able to bring down the proximity from 50% to 0.7%. A steep decrease in proximity was observed in initial rounds and later it decreases at a lower pace and finally settles to a proximity value of 0.7% as shown in Fig 3.7. The randomized version was also able to perform well in the initial stages but later on the gain in proximity decreases and converges to a proximity value of 3.8% after a very long time. Slicing (SA) is unable to get much gain in proximity from the start itself and converges to a proximity value of 8.4%. PROP-G (SA) performs really well in the initial rounds but after a while the performance becomes worse compared

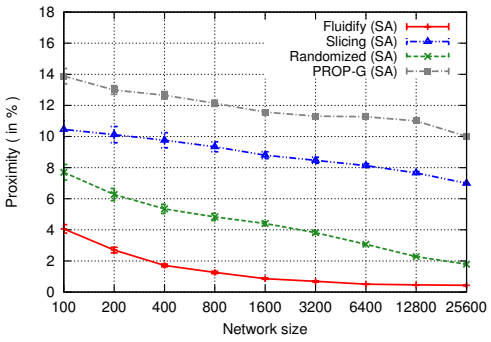


Figure 3.5: Proximity. Lower is better. Fluidify (SA) clearly outperforms the baselines in terms of deployment quality.

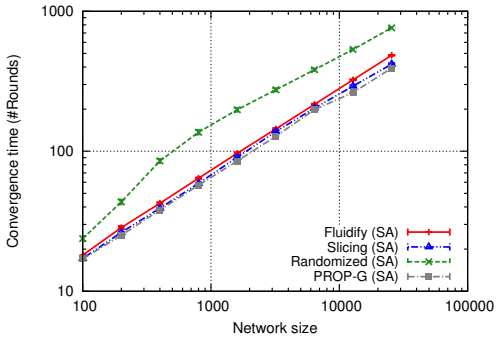


Figure 3.6: Convergence time. All three approaches have a sublinear convergence ($\approx 1.237 \times |N|^{0.589}$ for Fluidify).

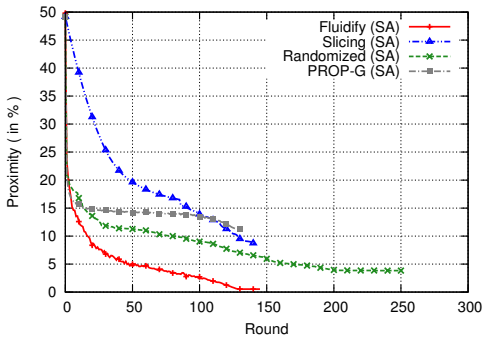


Figure 3.7: Proximity over time ($N = K_0 = 3200$, $k_{\text{net}} = k_{\text{data}} = 16$). Fluidify (SA)'s optimization is more aggressive compared to other baselines.

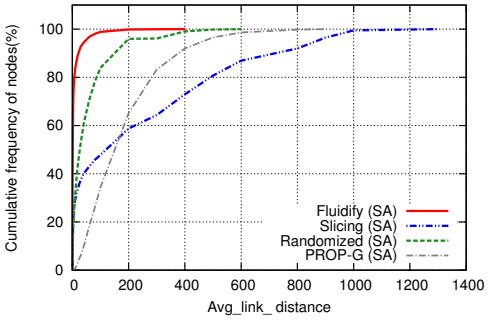
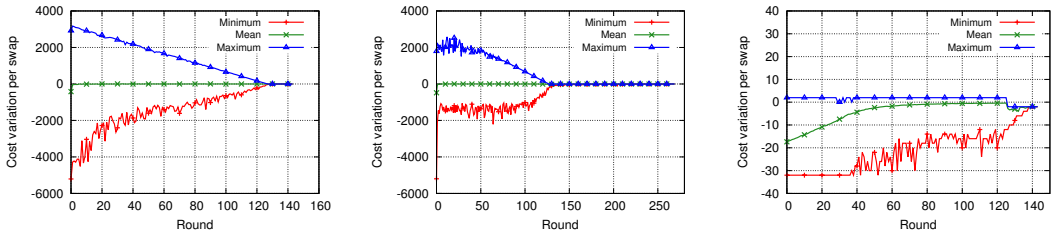


Figure 3.8: Average link distances in converged state ($N = K_0 = 3200$, $k_{\text{net}} = k_{\text{data}} = 16$). Fluidify (SA)'s links are both shorter and more homogeneous.

to Randomized (SA) and Fluidify (SA). The cumulative distribution of nodes based on the average link distance in a converged system for all the three approaches is depicted in Fig. 3.8. It is interesting to see that nearly 83% of the nodes are having an average link distance less than 10 and 37% of the nodes are having an average link distance of 1 in the case of Fluidify. But for Slicing (SA) even after convergence, a lot of nodes are having an average link distance greater than 200. Slicing (SA) clearly fails in improving the system beyond a limit.

The maximum, minimum and the mean gain obtained per swap in a default sys-



(a) Fluidify (SA) (b) Randomized (SA) (c) Slicing (SA)

Figure 3.9: Variation of the cost function per swap over time. Lower is better. ($N = K_0 = 3200$, $k_{\text{net}} = k_{\text{data}} = 16$, note the different scales) Fluidify (SA) shows the highest amplitude of variations, and fully exploits simulated annealing, which is less the case for Randomized (SA), and not at all for slicing.

tem setting using Fluidify is shown in Fig. 3.9(a). As the simulation progresses the maximum, minimum and the mean value of the cost function per swap in each round starts getting closer and closer and finally becomes equal on convergence. The highest gains per swap (negative cost) are obtained in the initial rounds of the simulation. The maximum value obtained by the cost function is expected to gradually decrease from a value less than or equal to 3200, which is the initial threshold value for simulated annealing, to 0. From Fig. 3.9(b) it is clear that the variation of the cost function for Randomized (SA) also shows a similar behaviour. Here the system progresses with a very small gain for a long period of time. As shown in Fig.3.9(c), the most interesting behaviour is that of Slicing (SA). It does not benefit much with the use of simulated annealing. The maximum gain that can be obtained per swap is 32 and the maximum negative gain is 2. This is because only the physically closer nodes of a given node are considered for a swap and the swap is done with the best possible candidate.

The *message cost* per round per node is equal to the amount of data that a node exchanges with another node. In our approach the nodes exchange their logical index and the logical neighbourhood. We assume that each index value amounts to 1 unit of data. So the message cost will be $1+k_{\text{data}}$ which will be 17 in default case. The *communication overhead* in the network per cycle will be equal to the average number of swaps occurring per round times the amount of data exchanged per swap. A single message costs 17 units. So a swap will cost 34 units. In our default setting, an average of 2819 swaps happen per round and this amounts to around 95846 units of data per round.

All the four approaches that we presented here are generic and can be used for any topology. Table. 3.4 shows how the four approaches fares for various topologies in a

Table 3.4: Performance on various topologies

Physical topology	Logical topology	Approach	Proximity(%)	Convergence(#Rounds)
torus	torus	Fluidify(SA)	**2.4**(±0.05)	162(±2.34)
torus	torus	Slicing(SA)	4.5(±0.05)	**130**(±2.16)
torus	torus	Randomized(SA)	3.82(±0.08)	423(±2.41)
torus	torus	PROP-G(SA)	4.6(±0.05)	132(±2.34)
torus	ring	Fluidify(SA)	**2.6**(±0.03)	171(±3.6)
torus	ring	Slicing(SA)	5.2(±0.02)	**128**(±3.26)
torus	ring	Randomized(SA)	4.05(±0.04)	464(±3.28)
torus	ring	PROP-G(SA)	5.6(±0.03)	130(±3.6)
ring	torus	Fluidify(SA)	**1.8**(±0.06)	156(±2.36)
ring	torus	Slicing(SA)	9.5(±0.08)	143(±4.1)
ring	torus	Randomized(SA)	2.7(±0.05)	442(±3.82)
ring	torus	PROP-G(SA)	10.1(±0.06)	**128**(±2.36)

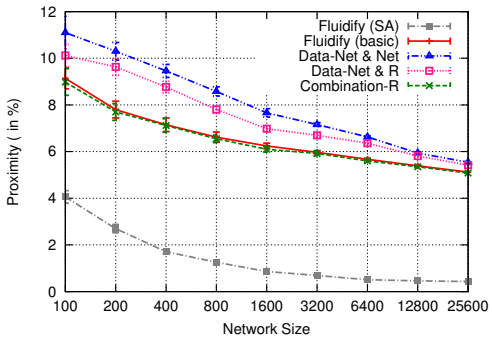


Figure 3.10: Comparison of different variants of Fluidify - Proximity

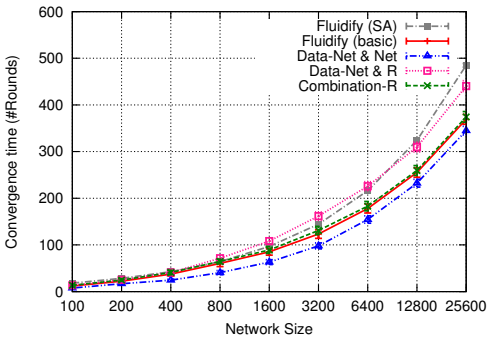


Figure 3.11: Comparison of different variants of Fluidify - Convergence

default setting. Fluidify clearly outperforms the other approaches in terms of proximity.

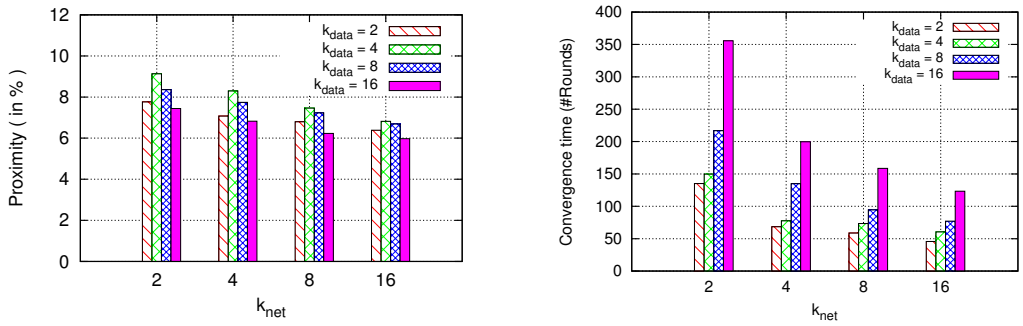
3.4.3.2 Effects of variants

Figure. 3.10 shows that compared to its variants like Fluidify (basic), combination-R, Data-Net & Net (Algorithm. 6) and Data-Net & R (Algorithm. 7), Fluidify (SA) is far ahead in quality of convergence. Here also, we consider a ring/ring topology with default setting. The convergence time taken by Fluidify is slightly higher compared to its variants as shown in Fig. 3.11.

Table 3.5 shows, how varying the initial threshold value for Fluidify affects its performance. From the table, it is clear that as the initial threshold value increases the

Table 3.5: Impact of K_0 on Fluidify (SA)

K_0	**Proximity (%)**	**Convergence (rounds)**
320	2.4	156
640	1.6	145
1600	1.1	146
3200	0.7	144



(a) Proximity(lower the better)

(b) Convergence time

Figure 3.12: Comparison of performance with varying k_{net} and k_{data} values

proximity that we obtain also improves. With a higher threshold value, more swaps will occur and therefore there is a higher chance of getting closer to the global minimum. The threshold value that gives the best performance is used for all our simulations.

Fig. 3.12 shows the performance of Fluidify (basic) with varying k_{net} and k_{data} values. Proximity improves slightly as we improve the search space from 4 to 16 which is clear from the Fig. 3.12(a). We conjecture that the rise in proximity from 2 to 4 is due to the creation of new local minima. As the value of k_{net} grows, we consider more potential swaps in each round, improving our choice, so the convergence speed increases which is evident from Fig. 3.12(b). When k_{data} grows beyond 2, we no longer optimize 1-hop links in $\mathcal{G}_{\text{data}}$ which slows down the convergence. The best performance is obtained for the combination where k_{net} and k_{data} are taken as 16.

3.5 Summary

In this chapter, we presented Fluidify, a novel decentralized mechanism for network aware overlay deployment. Fluidify works by exploiting both the logical links of an overlay and the physical topology of its underlying network to progressively align one

with the other and thereby maximizing the network locality. The proposed approach can be used in combination with any topology construction algorithm. The resulting protocol is generic, efficient, scalable and can substantially improve network overheads and latency in overlay based-systems. Simulation results show that in a ring/ring network of 25,600 nodes, Fluidify is able to produce an overlay with links that are on average 94% shorter than those produced by a standard decentralized approach based on slicing and 97% shorter compared to that produced by a decentralized approach based on PROP-G.

Chapter 4

Filament: A Cohort Construction Service for Decentralized Collaborative Editing Platforms

In chapter 2 we discussed the need for efficient content placement, search and distribution techniques. In this chapter we concentrate on decentralized collaborative editing platforms and present an approach to increase their efficiency. Distributed collaborative editors allow several remote users to contribute concurrently to the same document. Only a limited number of concurrent users can be supported by most of the currently deployed editors. A number of peer-to-peer solutions have therefore been proposed to remove this limitation and allow a large number of users to work collaboratively. These approaches however, tend to assume that all users edit the same set of documents, which is unlikely to be the case if such systems should become widely used and ubiquitous. To avoid flooding the system with updates that most users are not interested in, such decentralized collaborative editors therefore need to implement a form of user peering procedure in order to connect together users editing the same documents.

To realize this peering procedure, we propose in this chapter a novel cohort construction approach—Filament—that allows users editing the same documents to rapidly find each other. Our proposal utilizes the semantic relations between the sets of documents edited by individual peers to construct a set of self-organizing overlays to route search requests. The resulting protocol is efficient, scalable and provides beneficial load-balancing properties over the involved peers. We evaluate our approach and compare it against a standard Chord-based DHT approach. Our approach performs as well as a DHT-based approach, but provides better load balancing. Simulation results show

that in a network of 2^{12} nodes, Filament is able to reduce the document latency by around 20% compared to a Chord-based DHT approach.

Section 4.1 introduces the problem and some important works in this area. Section 4.2 presents our intuition. Section 4.3 presents the Filament algorithm. Section 4.4 summarizes the experimental evaluations of Filament in terms of document latency and load per node. Section 4.5 concludes the chapter.

4.1 Background and Problem Statement

A new generation of low-cost computers known as *plug computers* have recently appeared, offering users the possibility to create cheap nano-clusters of domestic servers, host data and services and federate these resources with other users. These nano-clusters of autonomous users brings closer the vision of *self-hosted on-line social services*, as promoted by initiatives such as ownCloud [own] or diaspora [dia]. But, the initiatives so far are primarily focused on the sharing and diffusion of *immutable* data (pictures, posts, chat messages) and offer much less in terms of real-time collaborative tools such as collaborative editors. In order to fill this gap, several researchers have proposed promising approaches [DSM+15, WUM10, OMMD10] to realize decentralized peer-to-peer collaborative editors.

In the following we discuss decentralized collaborative editors and introduce the problem of decentralized cohort construction (Section 4.1.1). We then review existing works related to this problem, such as P2P pub-sub and search systems (Section 4.1.2).

4.1.1 The problem: collaborative editing and cohort construction

Distributed collaborative editors allow several remote users to contribute concurrently to the same document. Most of the currently deployed distributed collaborative editors are centralized, hosted in tightly integrated environments and show poor scalability [Gdo, Eth] as well as poor fault tolerance. For instance, typical collaborative editors such as Google Doc [Gdo] or Etherpad [Eth] are limited in the number of users they can support concurrently.

To overcome this limitation, researchers have been looking into P2P collaborative editing platforms [DSM+15, OUMI06, WUM10, IN04, OMMD10, UPVS07] for some time. Deploying a collaborative editing framework on a P2P network provides several advantages like high availability, performance enhancement and censorship resilience [OMMD10]. Some of the existing approaches that deploy a collaborative system on a distributed network include Wooki [WUM07], DistriWiki [Mor07], Piki [MLS08],

Scalaris [SSR08], DTWiki [DB08], Distributed Version Control Systems like Git [git]. The collaboration can be synchronous or asynchronous [IN04]. In synchronous collaboration, the same documents are being edited by the group members and all the modifications can be seen in real-time by other members. In asynchronous collaboration, the copies of the documents are modified in isolation and synchronized afterwards to reestablish a common view. This led to the study of consistency and conflict resolution during object replication [OUMI06, SJZ+98].

Most of the approaches in decentralized peer-to-peer collaborative editing assume, however, that all users in a system are editing the same set of documents which may not be the case in most systems. If these systems become widely used, this is unlikely to be the case, and most users will only be editing a small subset of all the documents that exist in the system. Ideally, in order to limit communication costs and bandwidth consumption, only the users editing a particular document should be involved in the synchronization of changes to this document, precluding the use of indiscriminate flooding techniques. Instead, a user contributing for the first time to an existing document should be able to rapidly locate the group (or *cohort*) of other users editing this same document, a cohort construction problem linked to decentralized search which we discuss in the next section.

4.1.2 Existing approaches to cohort construction and decentralized search

Finding other users editing the same set of documents to construct an editing cohort is a particular case of peer-to-peer search, which has been extensively researched in the past both in unstructured [LCC+02, GMS06, OO06, DLOP07] and structured systems, in particular in DHT [RFH+01, SMK+01, RD01, ZKJ01]. Unstructured approaches have probabilistic guarantees: a resource might be present in the system, but it may not get found unless a flooding or exhaustive multicast strategy [FH10, CSWH01] is used, which might be very costly in massive systems.

A straightforward choice to realize such a cohort-construction mechanism consists of using a DHT (*Distributed Hash Table*) [ZKJ01, RFH+01, SMK+01] to act as an intermediate rendezvous point between nodes editing the same document. This choice is however sub-optimal. DHTs typically have deterministic guaranties in the sense that they are correct and complete, but they assume that the number of items to be stored is much higher than the number of storage nodes available. This is in stark contrast to distributed collaborative platforms, in which the number of documents being edited is smaller than the number of users. Furthermore, these systems use consistent hashing techniques in which a node's role in the system is independent of this node's particular

interests (in our case here documents), thus adding an additional layer of redirection. In case of a highly requested resource, DHTs use load-balancing techniques [RLS+03, KR04] that typically use virtual nodes or modified hash function [DHJ+07] to spread the load more evenly. These functions are however reactive, and well suited for content that is mostly read, but less suitable when interest in a document might vary rapidly. This is exactly where the work we present in this chapter comes in.

Our problem is also very similar to peer clustering seen in publish/subscribe systems. Publish/subscribe systems are mainly used for content distribution and selective content delivery. In pub/sub systems, subscribers express their interest by registering subscriptions and are then notified of any events (issued by publishers) which match their subscription. Pub/sub systems may be either topic-based or content-based. In topic based pub/sub systems, a topic name is used to represent a category of event and subscriptions occur against a particular topic. By contrast, in content based pub/sub systems, arbitrary predicates on attributes are used express subscriptions [CMTV07b, ASS+99]. P2P architectures are natural candidates for large scale pub/sub systems due to their scalability and self organizing properties. Structured P2P overlays [RFH+01, SMK+01, ZKJ01, RD01] are often used to implement content based pub/sub systems. This is made possible by mapping the attribute space of the latter to the identifier space of the former. Associating an attribute with one specific peer provides efficient routing but the peers hosting popular attributes unfortunately tend to become overloaded. Meghdoot [GSAA04], SCRIBE [CDKR02] and Bayeux [ZZJ+01] are examples of pub/sub systems built on top of DHT overlays.

A trivial alternative to the use of a structured overlay consists of flooding each new event in the system and then filtering out events that do not match local subscriptions at each single node. Flooding reaches all potential subscribers, but at a very high communication cost. Instead, it is more efficient to confine the dissemination of each event to the set of matching subscribers (Interest Clustering). If clusters are formed by subscribers having common interests then once an event reaches a member of the cluster, its dissemination can be limited to that cluster by following a simple flooding scheme or more sophisticated routing techniques. This helps to reduce the involvement of non interested subscribers.

Voulgaris et al. [VS05] for instance have proposed an epidemic protocol that clusters peers with similar content. The work [VRKS06] similarly proposes Sub-2-Sub, a solution to implement a content based pub/sub system on an unstructured overlay network. Subscribers sharing the same interests are clustered to form a ring-shaped overlay network which is updated continuously by analyzing the interests of users. The publishers are also slowly navigated to clusters of matching subscriptions. This work mainly focuses on interest clustering and the content dissemination. Some other P2P-based

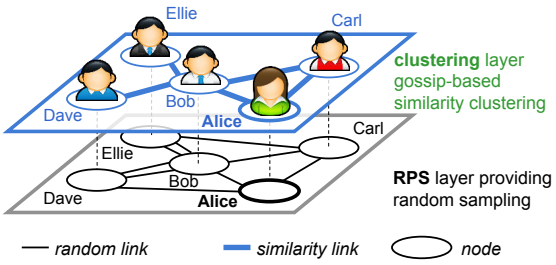


Figure 4.1: Overlay Architecture

implementations of content-based pub/sub are DPS [AGD+06] and GosSkip [GHK04]. Starting from this initial trend, several other papers [CT15] have appeared in this line of research.

The TERA system (Topic-based Event Routing for p2p Architectures) [BBQ+07] was designed with a general overlay (similar to Filament's helper overlay, which we discuss below) that is used to keep track of given topic ids used to maintain topic-overlays and perform topic based routing. Spidercast [CMTV07a], a decentralized topic-based pub/sub communication system also uses a similar clustering technique. In this case, a single overlay is built but the connectivity between users interested in the same topic is guaranteed (the overlay is topic-connected). Each SpiderCast node has some knowledge on the interests of other nodes in the system. Based on this, each node tries to locally determine to which nodes to connect in order to achieve topic connectivity. The routing and communication overheads can be reduced as the nodes that are not interested in a certain topic need not take part in routing messages of that topic. Generally, a separate overlay is maintained for each topic [BEG04, CDKR02]. Although such systems scale well with the number of nodes, they might not scale with the number of subscriptions per-node. Many of these search and routing techniques can be adapted for collaborative editing systems but this adaptation is likely to be sub-optimal because of the structural difference between collaborative editing and pub/sub systems.

4.2 Our intuition: self-organizing overlays

We propose a novel decentralized service called Filament, that connects together users interested in the same document without relying on the additional indirection implied by DHTs, while delivering deterministic guarantees, contrary to unstructured networks. Our solution exploits self-organizing overlays with a novel document based similarity

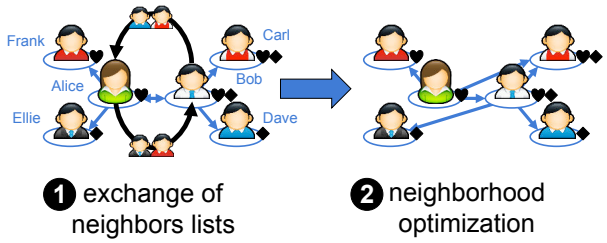


Figure 4.2: P2P neighborhood optimization

measure and is proactively load balancing, in that nodes working on the same documents naturally add their resources to help route their requests to the corresponding document editing community (which we call a *document cohort*) and more generally illustrate how an advanced behaviour can be obtained by combining several sub self-organizing overlays to create a routing structure that matches both the expected load and document interests of individual nodes. In this section, we discuss our intuition behind Filament (Sec. 4.2).

Our proposal, called Filament, composes together several self-organizing overlay networks to deliver its service. Overlay networks connect computers (aka *nodes* or *peers*) on top of a standard point-to-point network (e.g. TCP/IP) in order to add additional properties and services to this underlying network [RFH+01,SMK+01,RD01, DSM+15,WUM10]. A self-organizing overlay [VS05,JMB09] seeks to organize its nodes so that each node is eventually connected to its k closest other nodes, according to some similarity function.

A self-organizing overlay typically uses a two-layer structure to organize peers (Figure 4.1). Each layer provides a peer-to-peer overlay, in which users (or peers) maintain a fixed list of neighbors (or *views*). For instance, in Figure 4.1, Alice is connected to Bob, Carl, and Dave in the bottom RPS (Random Peer Sampling) layer, and to Carl and Bob in the upper layer (clustering). Periodically, each peer selects one or more users from its view and exchanges information about its neighbors with the final goal of converging to an optimal topology of similar users in the top layer (*clustering*).

The bottom RPS layers allows each peer to periodically obtain a random sample of the rest of the network and thus guaranties the convergence of the second layer (clustering), while making the overall system highly resilient against churn and partitions. This is achieved by having peers exchange and shuffling their neighbors list in *periodic gossip rounds* to maximise the randomness of the RPS graph over time [JVG+07]. For reason of efficiency, each peer does not however communicate with all its neighbors in each round, but instead randomly selects one of its neighbors in its RPS view to inter-

act with. Alice might for instance request Carl's list of RPS neighbors (which includes Ellie), and randomly decide to replace Dave by Ellie (received from Carl) in her RPS view.

The clustering layer sits on top of the RPS layers, and implements a local greedy optimisation procedure that leverages both neighbors returned by the RPS, and current neighbors from the clustering views [JMB09, VS05]. A peer (say Alice in Figure 4.1) will periodically update its list of similar neighbors with new neighbors found to be more similar to her in the RPS layer. This guarantees convergence under stable conditions, but can be particularly slow in large systems. This mechanism therefore is complemented by a swap mechanism in the clustering layer (Figure 4.2), whereby two neighboring peers (here Alice and Bob) exchange their neighbors lists (so peers that are already close to them, Step 1), and seek to construct a better neighborhood based on the other peer's information (Step 2 in Figure 4.2).

In Figure 4.2(1) for instance the interests of each user is shown as a symbol associated with them. As we can see Frank, Alice, Bob and Carl share the same interests. So instead of a communication link to Ellie as shown in the random network, it is beneficial for Alice to have a communication link to Carl who shares the same interest as shown in Figure 4.2(2). Bob applies a similar procedure, and decides to drops Alice for Ellie.

4.3 The Filament algorithm

In a large CE system, users editing the same document need to find each other in order to propagate modifications between themselves. Our approach *Filament* relies on a novel similarity measure, and exploits self-organizing overlays to allow the rapid, efficient, and robust discovery of document communities in large scale decentralized collaborative editing platforms. Each node in the system further maintains a specific view for each document it is currently editing, in order to rapidly propagate the edits: the aim of Filament is to fill this view as rapidly as possible. In addition to this we also need mechanisms that help the system react to changes, and reconnect nodes as required i.e. in cases where a new node joins the system or in cases where a new document is added to a node in the system.

4.3.1 System model

We consider a network consisting of a large number of nodes representing users $\mathcal{N} = \{n_1, n_2, .., n_N\}$. The network is dynamic: nodes may join or leave at anytime. Nodes are

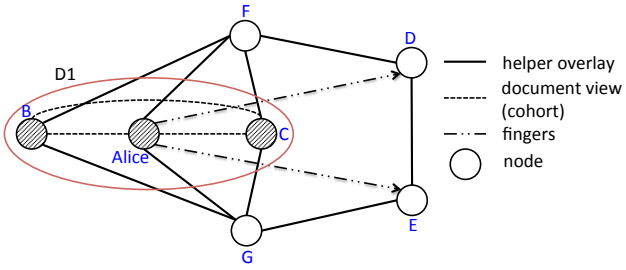


Figure 4.3: Overlay view

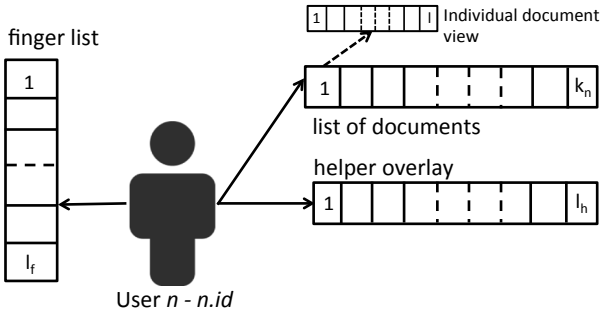


Figure 4.4: Illustration of the system model

assigned unique identifiers and communicate using messages over an existing network, such as the Internet, allowing every node to potentially communicate with any other node as long as it knows the other node's identifier. Nodes are organized in a set of interdependent overlay networks (termed *suboverlays* in the following). For each suboverlay, each individual node knows the identifiers of a set of other nodes, which forms its *neighbourhood* (or *view*) in this suboverlay. This neighbourhood can change over time to fulfil the overlay network's objectives. Each node/user n is editing a set of zero or more documents (noted $n.\mathcal{D}$) at any given time according to its interests. For the sake of uniformity, both the node ids and document ids are taken from the same id space.

4.3.2 Filament

As mentioned previously, our approach makes use of a hierarchy of self-organizing overlays in order to allow the rapid, efficient and robust discovery of document communities. All the nodes in the system are part of several suboverlays as shown in Fig. 4.3. A *helper overlay* ($\mathcal{H}$) is associated with each node. This helper overlay provides short

Table 4.1: Notations and Entities

$n.id$	node identifier of node n
k_n	number of documents being edited by node n
$n.\mathcal{D}$	list of documents edited by node n depicted as $\{d_1^n, d_2^n, ..., d_{k_n}^n\}$
$n.\mathcal{H}$	helper overlay associated with node n
$n.\mathcal{F}$	fingers of node n
$n.view(d)$	set of collaborators for document d contained in node n
$F_n[i]$	node which is the i^{th} finger of node n
l	maximum size of the collaborators list associated with each document
lh	size of helper overlay
lf	size of finger list

distance routing links within the system and it relies on a document-based similarity function, i.e. a similarity function that uses the set of documents edited by individual nodes in order to compute whether two nodes are close or far. The helper overlay view is initially filled using random peers taken from Random Peer Sampling layer (RPS). As the system executes, $n.\mathcal{H}$ is progressively filled with nodes that are similar to but not identical to node n in terms of the documents they edit.

Each node in the system further maintains a specific view for each document it is currently editing, in order to rapidly propagate new edits on these documents (These views can then be used to maintain a converged document state at each interested node using existing algorithms [DSM+15, WUM10, OMMD10]). In Fig. 4.3, nodes Alice, B and C will form a document cohort as all of them are editing document D1. Likewise, the system should insure that a node takes part in all the document cohorts pertaining to the documents it is currently editing.

In addition to the above helper and document overlays, each node maintains a set of *fingers* ($\mathcal{F}$), which act as long distance links within the system, in order to create a small world topology and provide fast routing. Similar to a traditional ring-based DHT, these links also help to rapidly locate collaborating nodes, and to avoid disjoint partitions. A simplified view of the system model is shown in Fig. 4.4. It shows the overlays (helper overlay, document overlays,fingers) that are associated with any given node in the system. Table 4.1 summarizes the notations that we use to describe Filament.

The basic working of our approach is shown using Alg. 8 and Alg. 9. Algorithm 8 shows how the system is initialized while Alg. 9 shows how the system proceeds after initialization and what it does in each round.

The proposed approach hinges on a novel similarity measure based on document ids. This similarity measure is described in procedure $\Delta(n, u)$ in Alg. 9. Each node n has a list of documents $n.\mathcal{D}$ associated with it. This list contains the set of documents

for which node n is a collaborator.

Given two nodes and the list of documents being edited by those nodes, the similarity measure in our approach is the smallest distance between the non-identical documents contained by it. For example, suppose node A is editing documents 5, 3 and 8, while node B is editing documents 3, 11 and 9, then the similarity between them is taken as 1 which is the difference between 8 and 9. The identical documents being edited by them are not taken into consideration. The key to the faster convergence of our system is the novel similarity measure which helps in finding nodes which are similar but not identical in their interests.

Algorithm 8 Initialization

1: **System initialization**
2: $n.\mathcal{H} \leftarrow$ R.P.S of size lh
3: **for all** $d \in n.\mathcal{D} : n.view(d) \leftarrow n.\mathcal{H}$
4: $Update\_Overlay(F[0], \Delta, n.\mathcal{H}, 1, 1, 1)$
5: **for** i from 1 to $\log \Delta(F[0], n)$
6: $Update\_Overlay(F[i], \Delta, n.\mathcal{H}, 1, 1, \Delta(F[0], n)/2^i)$

The initialization stage is pretty straightforward. The helper overlay associated with each node is filled randomly using Random Peer Sampling. The number of nodes in the helper overlay is truncated to lh. The documents are uniformly distributed throughout the system. In the initial stage, as we don't know the collaborators, the helper overlay is used to fill all the document views associated with a node. The node which is the farthest in the helper overlay forms the first entry of the finger list. Based on how far this node is, the other finger list entries are also filled.

Algorithm 9 shows how our system progresses after initialization. All the sub overlays contained in the system follow the same generic procedure. In each cycle, all the sub overlays get updated so as to reach an optimal stage. Procedure $Update\_Overlay$ $(O, dist, c, s, so, base)$ (lines 27-28) is used for updating the overlay networks. Six arguments are passed to this function. Here, O represents the overlay being updated. $dist$ is the function used for calculating the similarity measure. s is the size of the resulting overlay. so is the sort order. $base$ is used to remove identical nodes and to give priority to the nodes which are similar but non-identical (like a threshold; 0 represents identical nodes). The argument c is the candidate list of nodes that is used to update the overlay. For generality, we are truncating the candidate list to the desired size(s) of the resulting overlay. A good set of candidate nodes can significantly affect the convergence speed of the system. Thus $Update\_Overlay()$ returns a set of nodes sorted in ascending or descending order on the basis of similarity measure.

In each round, node n randomly selects a node p from its helper overlay and gets the neighbourhood information of p (lines 2). $p.\mathcal{H}$ along with one randomly selected node in the system is used as the candidate list for the updation of helper overlay associated with node n (line 3). A random entry is added with the hope that the system converges faster. Measures are taken to remove n from the candidate list of nodes associated with the updation of overlays associated with node n. The randomly filled helper overlay of node n gets modified as the simulation progresses to include nodes similar to n but non-identical. Likewise, the finger list is also updated with a set of carefully selected candidate list (lines 5-7). Fingers help in providing links to non-similar nodes; in other words they provide long distance routing links to nodes further away. They also help in preventing disjoint clusters. The finger lists are used mainly in cases where a node needs to find collaborators for a newly added document. A node can look in its finger list in order to find someone editing the newly added document or to find some one who might be editing a document similar to the newly added document. Individual document views are also updated in each round (lines 8-13). If the current document view already has a node with that document, then that node's document view is used to update the document overlay or else a randomly selected node is made use of.

After a certain number of rounds, the system "kind of" stabilizes i.e., all the document views get filled. Procedure $\delta(d, n, u)$ helps when a new document gets added to a node or when a new node is added to the system. When a new document d gets added to a node n, what we aim to do is to find its collaborators in a fast manner. Procedure $\delta(d, n, u)$ checks whether the document d which is newly added to node n is present in node u. If it is present then n uses the document view of u to find collaborators for d. We can use $Update\_Overlay(n.view(d), \delta(d, -, -), n.\mathcal{F} \cup n.\mathcal{H}, l, 1, 0)$ for this purpose. If none of the nodes in the candidate list contains document d, then node n makes use of the similarity measure Δ to find collaborators.

4.4 Evaluation

4.4.1 Experimental Setting and Metrics

Unless otherwise indicated, the default network size is taken as 2^{12}. We assume that the system has converged when all the document sub-overlays are filled i.e. all the nodes have successfully found collaborators for the documents they are currently editing. For generality, the value of l (document view) and lh (size of the help overlay) is taken as 10 in all the experiments. For all the network sizes, we assume that a total of 10 documents are there in the system. It is also assumed that each document is being edited by 10% of the network size number of nodes. The results obtained during the

Algorithm 9 Filament

1: **In round(r) do**
2: $p \leftarrow$ random node from $n.\mathcal{H}$
3: $ch \leftarrow p.\mathcal{H} \cup \{\text{one random R.P.S}\} \setminus \{n\}$
4: $Update\_Overlay(n.\mathcal{H}, \Delta, ch, lh, -1, 1)$
5: **for** i from 1 to lf
6: $cf \leftarrow F[i].\mathcal{F} \cup F[i].\mathcal{H} \cup \{\text{one random R.P.S}\} \setminus \{n\}$
7: $Update\_Overlay(F[i], \Delta, cf, 1, 1, 1)$
8: **for all** d from $n.\mathcal{D}$
9: **if** $\exists p \in n.view(d)$ so that $d \in p.\mathcal{D}$
10: select p ; $c \leftarrow p.view(d)$
11: **else** select a random node p from $n.view(d)$
12: $c \leftarrow p.\mathcal{H} \cup p.\mathcal{F} \cup \{\text{one random R.P.S}\} \setminus \{n\}$
13: $Update\_Overlay(n.view(d), \Delta, c, l, -1, 0)$
14: **Procedure** $\Delta(n, u)$
15: $S_1 \leftarrow n.\mathcal{D} \setminus u.\mathcal{D}$
16: **if** $S_1 = \emptyset$ **then** $S_1 \leftarrow n.id$
17: $S_2 \leftarrow u.\mathcal{D} \setminus n.\mathcal{D}$
18: **if** $S_2 = \emptyset$ **then** $S_2 \leftarrow u.id$
19: $S_3 \leftarrow S_1 \times S_2$
20: $m \leftarrow min(|x - y|) \forall (x, y) \in S_3$
21: **return** m
22: **Procedure** $\delta(d, n, u)$
23: **if** $d \in n.\mathcal{D} \cap q.\mathcal{D}$
24: **return 0**
25: else
26: **return** $\Delta(n, u)$
27: **Procedure** $Update\_Overlay(O, dist, c, s, so, base)$
28: **return** $argmax\_s_{p \in c}(dist(n, p) - base) * so$

evaluation are shown in this section.

We assess the performance of our approach along two metrics:

- **Document latency** - captures the number of rounds it takes for the system to find l collaborators for a newly added document.
- **Load per node** - measures the load associated with each node based on the communication cost associated with them. This is directly related to the number

of times a node is accessed during simulation.

4.4.2 Baselines

The performance of our approach is compared against a chord-based DHT approach. The main reason for this is that a DHT is commonly used in similar applications and they perform really well providing deterministic guaranties. The document id is hashed and based on the hash value obtained, a node gets selected. The collaborators list for that document gets stored in the selected node. So in order to find the collaborators for a document all we have to do is hash the document id and send a message to the corresponding node for the collaborators list. The main delay here is to find a node given its node id. Chord based topology helps in this by providing faster routing. Node ids are ordered in an ID space modulo 2^t. We say that id a follows id b in the ring, if $(a - b + 2^t) \bmod 2^t < 2^{t-1}$; otherwise a precedes b. Given an id a, its successor is defined as the nearest node whose id is equal to a or follows a in the ring. The notion of predecessor is defined in a symmetric way. Each node maintains two sets of neighbors, called leaves and fingers. Leaves of node n are its lh nearest successors. For each node n, its j^{th} finger is defined as successor$(n + 2^j)$, with $j \in [0, t-1]$. Routing in Chord works by forwarding messages in the ring following the successor direction; when receiving a message targeted at node k, node n forwards it to its furthest leaf or finger that precedes successor(k). Fingers help in reducing the number of nodes traversed to reach the destination node.

4.4.3 Results

All the results (Figures 4.5 - 4.10 and Tables 4.2 - 4.3) are computed with Peersim [MJ09] and are averaged over 10 experiments. The comparison to the baseline is done with the help of a base case setting. When shown, intervals of confidence are computed at a 95% confidence level using a student t-distribution. The source code is made available in http://armi.in/resmi/ce1.zip.

4.4.3.1 Evaluation of Filament

Figure 4.5 shows the convergence time of Filament with varying network sizes. As the network size increases the time taken for the system to converge also increases. We assume that the system is converged when all the document overlays are completely filled. From the graph it is clear that Filament works well for very large network sizes. Figure 4.6 shows the cumulative frequency distribution of converged nodes for Filament

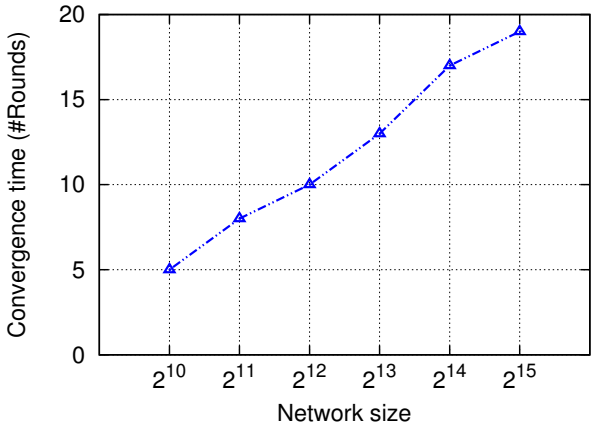


Figure 4.5: Convergence time of Filament for varying network sizes

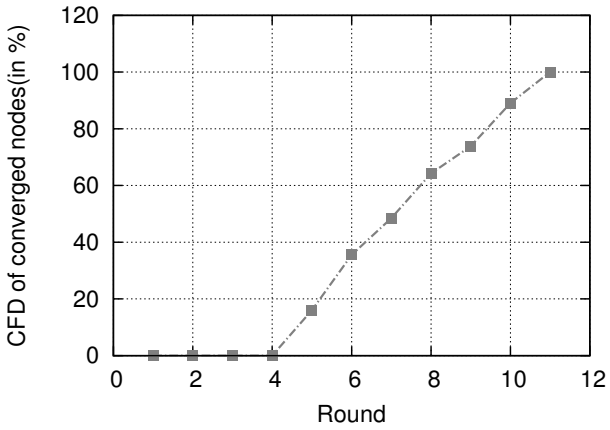


Figure 4.6: Cumulative frequency distribution of converged nodes for Filament in the base case

in the base case. A small number of converged nodes causes a chain effect causing a larger number of nodes to converge in the following rounds. Thus once the nodes start converging, the system progresses towards convergence in a faster manner. Figure 4.7 shows the number of nodes in the document view of n when a new document is added to n and it tries to find l collaborators.

Figure 4.8 and Table 4.2 show how our approach fares compared to a chord based DHT approach. Our approach has lower document latency compared to a DHT. The document latency varies from 4.8 to 8.1 as the network size grows from 2^{10} to 2^{15} for

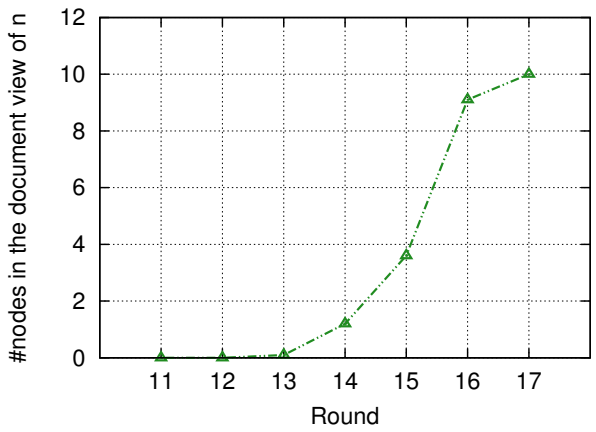


Figure 4.7: No: of nodes in the document view of n for Filament in the base case

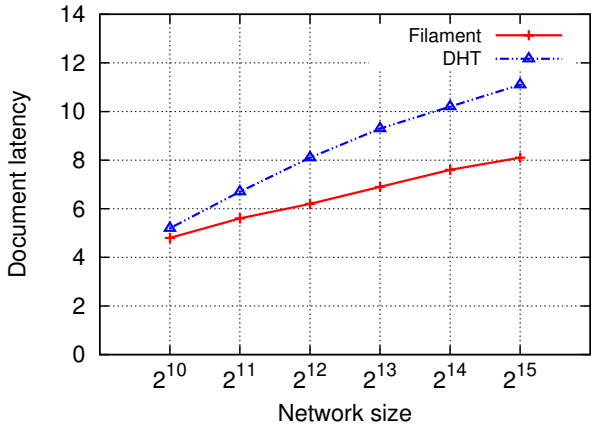


Figure 4.8: Filament vs DHT based on document latency

Filament while it varies from 5.2 to 11.1 for DHT. DHT provides an additional level of indirection. The document id is used for hashing and the collaborator list associated with a document might be stored in a node which is not editing that document at all. Moreover DHT is not exactly an optimal solution in this scenario as the number of documents being edited is significantly smaller compared to the number of nodes in the system. The latency in the case of DHT is mainly associated with routing to the node with the collaborators list. Compared to DHT, Filament shows a better performance with the help of document sub-overlays and finger list.

The Table 4.3 shows the maximum, minimum and mean load associated with a node for both Filament and DHT when a new document is added to a converged system.

Table 4.2: Filament vs DHT based on document latency

Network Size	Filament	DHT
2^{10}	4.8(±1.3)	5.2(±1.4)
2^{11}	5.6(±1.2)	6.7(±1.3)
2^{12}	6.2(±1.1)	8.1(±1.2)
2^{13}	6.9(±1.1)	9.3(±1.1)
2^{14}	7.6(±1.1)	10.2(±1.1)
2^{15}	8.1(±0.9)	11.1(±0.9)

Table 4.3: Load associated with nodes for Filament and DHT

Approach	Minimum (bytes)	Mean (bytes)	Maximum (bytes)
Filament	8	64	176
DHT	8	96	880

When a new document is added to a node, the node tries to find l collaborators for that document. In order to do that, it has to exchange messages with other nodes. Here, we assume that a single message has a size of 8 bytes which is the size of node id. The results show the case when a document d is added to a node that doesn't contain it and 10 experiments are conducted with the same document id. The cumulative result is shown in the table. In the case of DHT the same node is getting accessed multiple times for the collaborators list of d while in the case of Filament the load is divided as all the nodes editing the document will have collaborators list in them. The average load associated with a node is slightly lesser for Filament. The maximum load of DHT is very high which can lead to bottle necks in the network.

4.4.3.2 Effects of variants

Figure 4.9 shows the effect of varying the number of documents in the system. As we can see, increasing the number of documents in the system helps it to converge in a faster manner. This is to be expected as the number of sub-overlays associated with each node increases with the increased number of documents. Making use of these additional sub-overlays, a node can optimize its neighbourhood and finger list. But there is also a disadvantage associated with this; the amount of overlays to be managed in each round increases leading to an increased load for the nodes.

Figure 4.10 shows the effect of varying the number of nodes editing a document or in other words the size of collaborators in the system. From the graph it is clear that as the number of nodes editing a given document increases it helps the system to

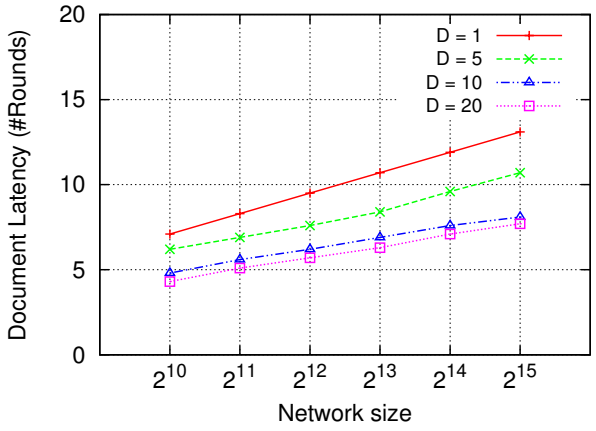


Figure 4.9: Effect of varying the number of documents for Filament

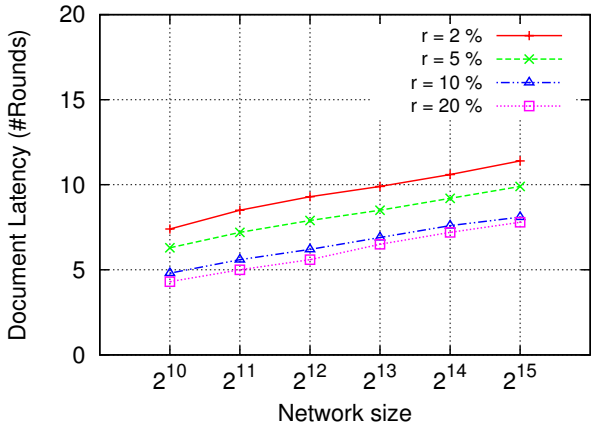


Figure 4.10: Effect of varying the number of nodes editing a document

converge faster. This is mainly because we can easily get the information about the collaborators if more and more nodes are editing the same document.

The Table 4.4 shows the effect of finger list and document views on the overall system performance. The document latency in the presence and absence of each of the feature is shown. As we can see, the effect of finger list is much less compared to the effect of document views. The document views clearly have a high impact on latency as we can get all the needed collaborators with the help of a document view. In the absence of it, all the needed collaborators are to be found with the help of helper

Table 4.4: Document latency in the presence and absence of document views and finger list

Feature	**#Rounds (with)**	**#Rounds (without)**
Document views	6.2	11.8
Finger list	6.2	7.1

overlay. But finger list mainly helps for routing, so the effect of it is less compared to document views.

4.5 Summary

In this chapter, we presented Filament, a *decentralized cohort-construction protocol* adapted to the needs of large-scale collaborative editors. Filament eliminates the need for any intermediate DHT, and allows nodes editing the same document to find each other in a rapid, efficient and robust manner by generating an adaptive *routing field* around themselves. Filament's architecture hinges around a set of collaborating self-organizing overlays exploiting a novel document-based similarity measure. Beyond its intrinsic merits, Filament's design further demonstrates how the horizontal composition of several self-organizing overlays can lead to richer and more efficient services. Simulation results show that in a network of 2^{12} nodes, Filament is able to reduce the document latency by around 20% compared to a Chord-based DHT approach.

Chapter 5

Conclusion

5.1 Summary of Contributions

We presented two contributions in this thesis. Both contributions aim at solving problems in the area of decentralized distributed systems. One problem concentrates on network aware overlays while the other concentrates on collaborative editing systems.

In Chapter 3, we presented *Fluidify*, an approach to incorporate network awareness in overlay networks. Overlay networks play a central role in the deployment of complex distributed systems on a global scale. They provide some diverse services which are not available from the network mainly in the areas of storage, routing, recommendation and streaming. The design of an overlay and the role each node plays in the network can significantly impact the system performance and latency. Unfortunately, many popular overlay construction protocols do not usually take into account the underlying network infrastructure on which an overlay is deployed and those that do, tend to be limited to a narrow family of applications or overlays. Our first objective was to provide a sound solution to this problem.

Our proposal, *Fluidify*, is a novel decentralized mechanism for overlay deployment which enhances network awareness. Fluidify works by exploiting both the logical links of an overlay and the physical topology of its underlying network to progressively align one with the other and thereby maximizing the network locality. Fluidify can be used in combination with any topology construction algorithm. The resulting protocol is generic, efficient and scalable. It can also substantially reduce network overheads and latency in overlay based systems. From our simulation results, it can be shown that in a network of 25,600 nodes, Fluidify is able to produce an overlay with links that are on an average 94% shorter than that produced by a standard decentralized approach based

on slicing and 97% shorter compared to that produced by a decentralized approach based on PROP-G, while demonstrating a sub-linear time complexity.

In chapter 4, we presented *Filament*, an approach to increase the efficiency of distributed collaborative editing platforms. Collaborative editors allow several remote users to contribute concurrently to the same document. Most of the currently deployed collaborative editors are centralized, hosted in tightly integrated environments and show poor scalability and poor fault tolerance. A number of peer-to-peer solutions have therefore been proposed to overcome this limitation and allow a large number of users to work collaboratively. However, most of these works, generally assume that all nodes in the system edit the same document or the same set of documents which is unlikely to be the case in very large systems. As these systems typically propagate updates using a uniform broadcast primitive there is a chance of overloading the system. Another option is to use a DHT. This option sub-optimal as it adds an extra level of indirection in the document peering procedure and creates potential hot spots for nodes handling highly popular documents. We argue that users editing the same document should be able to first locate each other in a manner which is efficient, reactive to changes and robust to failures. Our second objective was to suggest a solution to this problem.

To solve this problem, we proposed *Filament*, a decentralized cohort-construction protocol adapted to the needs of large-scale collaborative editors. Filament's architecture hinges around a set of collaborating self-organizing overlays exploiting a novel document-based similarity measure. Filament allows the nodes editing the same document to find each other in a rapid, efficient and robust manner by generating an adaptive routing field around themselves. This mechanism is also proactively load balancing, in that, nodes working on the same documents naturally add their resources to help route their requests to the corresponding document editing community (which we call a document cohort). Filament's design demonstrates how the horizontal composition of several self-organizing overlays can lead to richer and more efficient services. Simulation results show that in a network of 2^{12} nodes, Filament is able to reduce the document latency by around 20% compared to a Chord-based DHT approach.

5.2 Discussion and Future Work

5.2.1 Limitations

In our study we mainly concentrated on experimental results for both Fluidify and Filament. Our simulations were mainly performed on synthetic data sets. We have made the synthetic data sets as generic as possible so as to resemble a real data set,

but the experimental results might not always reflect how the system would behave under a real workload. An experimental study of how the system behaves when given real data sets will be useful in finding the flaws in our system. A thorough analytical study is also needed for better validation of our system. Moreover it might be nice if we could deploy both Fluidify and Filament on a decentralized distributed system like a P2P network formed by connecting a group of plug computers.

5.2.2 Further Extensions

We now discuss some future works to improve both our contributions. In Fluidify we concentrated on creating network aware overlays by aligning both the logical and physical overlays. We can extend this by trying out several variations to the Fluidify algorithm like mapping more than one logical node to a physical node and trying out some other techniques to overcome the problem of local minima. Filament deals with finding collaborators in a collaborative editing environment. We can extend it further to deal with all the dynamicity in the network – like the nodes and the documents being active and passive. The performance of the system can also be calculated for some other similarity measures. In addition to all these extensions, using some real data traces to test the system and deploying the above two systems on a plug based P2P system or some other decentralized distributed system would bring additional and useful insights into the strengths and weaknesses of both systems.

We also envision future works that can extend our contributions instead of just overcoming the limitations they have. To effectively deploy overlay networks on a cloud infrastructure and to ensure network awareness we need to take in to account the features of a cloud. Moreover in a collaborative editing environment an effective consistency technique is needed to ensure that no concurrent edits are lost. Like this both our contributions can be extended to deal with problems in their corresponding areas.

Appendix

Publications

The contributions in this thesis led to the following publications :

- Full paper - *Ariyattu C. Resmi, François Taïani*: Fluidify: Decentralized Overlay Deployment in a Multi-cloud World. In: 15th IFIP International Conference on Distributed Applications and Interoperable Systems (DAIS 2015), held as part of the 10th International Federated Conference on Distributed Computing Techniques, DisCoTec 2015, Grenoble, France, June 2015, Springer, pp. 1–15, 15 pages.
- Full paper - *Ariyattu C. Resmi, François Taïani*: Filament: a cohort construction service for decentralized collaborative editing platforms. Accepted and will be presented in the 17th IFIP International Conference on Distributed Applications and Interoperable Systems (DAIS 2017), held as part of the 12th International Federated Conference on Distributed Computing Techniques, DisCoTec 2017, June 2017, Neuchâtel, Switzerland.
- Short paper - *Ariyattu C. Resmi, François Taïani*: Filament: a cohort construction service for decentralized collaborative editing platforms In: Conférence d'informatique en Parallélisme, Architecture et Système(Compas) in July 2016, Lorient, France.
- Poster - In: EIT Digital Symposium on Future Cloud Computing, INRIA Rennes, France, 19-20 October 2015.

Glossaire

DHT : Distributed Hash Table
CE : Collaborative Editing
IP : Internet Protocol
TCP : Transmission Control Protocol
NC : Network Coordinates
BGP : Border Gateway Protocol
P2P : Peer to Peer
SOA : Service Oriented Architecture
CDN : Content Distribution Network
IT : Information Technology
VM : Virtual Machine
RPS : Random Peer Sampling
QoS : Quality of Service.

Bibliography

[AB05] Sven Apel and Klemens Bohm. Self-organization in overlay networks. In *In Proceedings of 1st CAISE'05 Workshop on Adaptive and Self-Managing Enterprise Applications*, 2005.

[ABB+05] Ittai Abraham, Ankur Badola, Danny Bickson, Dahlia Malkhi, Sharad Maloo, and Saar Ron. Practical locality-awareness for large scale information sharing. In *4th Annual International Workshop on Peer-To-Peer Systems (IPTPS '05)*, 2005.

[AGD+06] Emmanuelle Anceaume, Maria Gradinariu, Ajoy Kumar Datta, Gwendal Simon, and Antonino Virgillito. A semantic overlay for self-peer-to-peer publish/subscribe. In *Distributed Computing Systems, 2006. ICDCS 2006. 26th IEEE International Conference on*, pages 22–22. IEEE, 2006.

[ALCR+10] Hussam Abu-Libdeh, Paolo Costa, Antony Rowstron, Greg O'Shea, and Austin Donnelly. Symbiotic routing in future data centers. *ACM SIGCOMM Computer Communication Review*, 40(4):51–62, 2010.

[And00] Gregory R. Andrews. *Foundations of multithreaded, parallel, and distributed programming.* Addison-Wesley, Reading (Mass.), Menlo Park (Calif.), New York, 2000.

[ASS+99] Marcos K Aguilera, Robert E Strom, Daniel C Sturman, Mark Astley, and Tushar D Chandra. Matching events in a content-based subscription system. In *Proceedings of the eighteenth annual ACM symposium on Principles of distributed computing*, pages 53–61. ACM, 1999.

[BBK02] Suman Banerjee, Bobby Bhattacharjee, and Christopher Kommareddy. Scalable application layer multicast. In *Proceedings of the 2002 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, ACM SIGCOMM '02, pages 205–217, 2002.

[BBQ+07] Roberto Baldoni, Roberto Beraldi, Vivien Quema, Leonardo Querzoni, and Sara Tucci-Piergiovanni. Tera: Topic-based event routing for peer-to-peer architectures. In *Proceedings of the 2007 Inaugural International Conference on Distributed Event-based Systems*, DEBS '07, pages 2–13, 2007.

[BEG04] Sébastien Baehni, Patrick Th Eugster, and Rachid Guerraoui. Data-aware multicast. In *Dependable Systems and Networks, 2004 International Conference on*, pages 233–242. IEEE, 2004.

[Ber96] Philip A. Bernstein. Middleware: A model for distributed system services. *Commun. ACM*, 39(2):86–98, February 1996.

[BFG+10] Marin Bertier, Davide Frey, Rachid Guerraoui, Anne-Marie Kermarrec, and Vincent Leroy. The gossple anonymous social network. In *Middleware'10*, 2010.

[BHO+99] Kenneth P Birman, Mark Hayden, Oznur Ozkasap, Zhen Xiao, Mihai Budiu, and Yaron Minsky. Bimodal multicast. *ACM Transactions on Computer Systems (TOCS)*, 17(2):41–88, 1999.

[BYL08] John Buford, Heather Yu, and Eng Keong Lua. *P2P Networking and Applications*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2008.

[BYV+09] Rajkumar Buyya, Chee Shin Yeo, Srikumar Venugopal, James Broberg, and Ivona Brandic. Cloud computing and emerging {IT} platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Computer Systems*, 25(6):599 – 616, 2009.

[CDHR02] Miguel Castro, Peter Druschel, Y Charlie Hu, and Antony Rowstron. Exploiting network proximity in distributed hash tables. In *International Workshop on Future Directions in Distributed Computing (FuDiCo)*, pages 52–55, 2002.

[CDHR03] Miguel Castro, Peter Druschel, Y. Charlie Hu, and Antony Rowstron. Topology-aware routing in structured peer-to-peer overlay networks. In *Future Directions in Distributed Computing*, pages 103–107. Springer-Verlag, 2003.

[CDK+03] Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, Animesh Nandi, Antony Rowstron, and Atul Singh. Splitstream: high-bandwidth multicast in cooperative environments. In *ACM SIGOPS Operating Systems Review*, volume 37, pages 298–313. ACM, 2003.

[CDKB11] G.F. Coulouris, J. Dollimore, T. Kindberg, and G. Blair. *Distributed Systems: Concepts and Design*. International computer science series. Addison-Wesley, 2011.

[CDKR02] Miguel Castro, Peter Druschel, A-M Kermarrec, and Antony IT Rowstron. Scribe: A large-scale and decentralized application-level multicast infrastructure. *IEEE Journal on Selected Areas in communications*, 20(8):1489–1499, 2002.

[CMTV07a] Gregory Chockler, Roie Melamed, Yoav Tock, and Roman Vitenberg. Spidercast: A scalable interest-aware overlay for topic-based pub/sub communication. In *Proceedings of the 2007 Inaugural International Conference on Distributed Event-based Systems*, DEBS '07, pages 14–25, 2007.

[CMTV07b] Gregory V. Chockler, Roie Melamed, Yoav Tock, and Roman Vitenberg. Constructing scalable overlays for pub-sub with many topics. In *Proceedings of the Twenty-Sixth Annual ACM Symposium on Principles of Distributed Computing, PODC 2007, Portland, Oregon, USA, August 12-15, 2007*, pages 109–118, 2007.

[CRSZ02] Yang-Hua Chu, Sanjay G. Rao, Srinivasan Seshan, and Hui Zhang. A case for end system multicast. *IEEE Journal on Selected Areas in Communications*, 20(8):1456–1471, 2002.

[CSWH01] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong. Freenet: A distributed anonymous information storage and retrieval system. In *International Workshop on Designing Privacy Enhancing Technologies: Design Issues in Anonymity and Unobservability*, pages 46–66, 2001.

[CT15] Chen Chen and Yoav Tock. Design of routing protocols and overlay topologies for topic-based publish/subscribe on small-world networks. In *Proceedings of the Industrial Track of the 16th International Middleware Conference*, Middleware Industry '15, 2015.

[data] Cisco - trends and analysis. URL: http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/vni-hyperconnectivity-wp.html.

[datb] Wikipedia - edge computing. URL: https://en.wikipedia.org/wiki/Edge_computing.

[datc] Wikipedia - fog computing. URL: https://en.wikipedia.org/wiki/Fog_computing.

[datd] Wikipedia - internet traffic. URL: https://en.wikipedia.org/wiki/Internet_traffic.

[DB08] Bowei Du and Eric A Brewer. Dtwiki: a disconnection and intermittency tolerant wiki. In *Proceedings of the 17th international conference on World Wide Web*, pages 945–952. ACM, 2008.

[DEF13] Benjamin Doerr, Robert Elsässer, and Pierre Fraigniaud. Epidemic algorithms and processes: From theory to applications. *Dagstuhl Reports*, 3(1):94–110, 2013.

[DF02] Susheel Daswani and A Fisk. Gnutella udp extension for scalable searches (guess) v0. 1, 2002.

[DHJ+07] 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. In *SOSP '07*, 2007.

[dia] diaspora. URL: https://en.wikipedia.org/wiki/Diaspora_(software).

[DLOP07] Reza Dorrigiv, Alejandro Lopez-Ortiz, and Paweł Prałat. Search algorithms for unstructured peer-to-peer networks. In *32nd IEEE Conference on Local Computer Networks*, pages 343–352. IEEE, 2007.

[DNV06] Christos Doulkeridis, Kjetil Norvag, and Michalis Vazirgiannis. Schema caching for improved xml query processing in p2p systems. In *Peer-to-Peer Computing, 2006. P2P 2006. Sixth IEEE International Conference on*, pages 73–74. IEEE, 2006.

[DP06] Vassilios V Dimakopoulos and Evaggelia Pitoura. On the performance of flooding-based resource discovery. *IEEE Transactions on Parallel and Distributed Systems*, 17(11):1242–1252, 2006.

[ds3] Storage servers. URL: https://storageservers.wordpress.com/2013/07/17/facts-and-stats-of-worlds-largest-data-centers/.

[DSM+15] Alan Davoust, Hala Skaf-Molli, Pascal Molli, Babak Esfandiari, and Khaled Aslan. Distributed wikis: a survey. *Concurrency and Computation: Practice and Experience*, 27(11):2751–2777, 2015.

[EGH+03] P Th Eugster, Rachid Guerraoui, Sidath B Handurukande, Petr Kouznetsov, and A-M Kermarrec. Lightweight probabilistic broadcast. *ACM Transactions on Computer Systems (TOCS)*, 21(4):341–374, 2003.

[Eth] Etherpad. URL: https://en.wikipedia.org/wiki/Etherpad.

[FGK+09] Davide Frey, Rachid Guerraoui, Anne-Marie Kermarrec, Boris Koldehofe, Martin Mogensen, Maxime Monod, and Vivien Quéma. Heterogeneous gossip. In *Middleware*, pages 42–61, 2009.

[FH10] Imen Filali and Fabrice Huet. Dynamic ttl-based search in unstructured peer-to-peer networks. In *Cluster, Cloud and Grid Computing (CCGrid), 10th IEEE/ACM International Conference on*, pages 438–447. IEEE, 2010.

[Gdo] Google docs. URL: https://en.wikipedia.org/wiki/Google_Docs,_Sheets,_and_Slides.

[GHK04] Rachid Guerraoui, SB Handurukande, and A-M Kermarrec. Gosskip: a gossip-based structured overlay network for efficient content-based filtering. Technical report, 2004.

[GHP+08] Paul Grace, Danny Hughes, Barry Porter, Gordon S. Blair, Geoff Coulson, and Francois Taiani. Experiences with open overlays: A middleware approach to network heterogeneity. In *Eurosys'08*, 2008.

[Gib] Mark Gibbs. Pando makes light work of big files. URL: http://www.networkworld.com/article/2296312/software/pando-makes-light-work-of-big-files.html.

[git] Git. URL: https://git-scm.com.

[GKW13] George Giakkoupis, Anne-Marie Kermarrec, and Philipp Woelfel. Gossip protocols for renaming and sorting. In *DISC*, pages 194–208, October 14–18 2013.

[GMS05] Christos Gkantsidis, Milena Mihail, and Amin Saberi. Hybrid search schemes for unstructured peer-to-peer networks. In *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, volume 3, pages 1526–1537. IEEE, 2005.

[GMS06] Christos Gkantsidis, Milena Mihail, and Amin Saberi. Random walks in peer-to-peer networks: Algorithms and evaluation. *Perform. Eval.*, 63(3):241–263, March 2006.

[GSAA04] Abhishek Gupta, Ozgur D Sahin, Divyakant Agrawal, and Amr El Abbadi. Meghdoot: content-based publish/subscribe over p2p networks. In *Middleware'04*, 2004.

[HJB+09] Kenneth Hopkinson, Kate Jenkins, Kenneth Birman, James Thorp, Gregory Toussaint, and Manu Parashar. Adaptive gravitational gossip: A gossip-based communication protocol with user-selectable rates. *IEEE Transactions on Parallel and Distributed Systems*, 20(12):1830–1843, 2009.

[IN04] Claudia-Lavinia Ignat and Moira C. Norrie. CoDoc: Multi-mode Collaboration over Documents. In *Proceedings of the 16th International Conference on Advanced Information Systems Engineering (CAiSE'04)*, pages 580–594, June 2004.

[JK06] Mark Jelasity and Anne-Marie Kermarrec. Ordered slicing of very large-scale overlay networks. In *P2P 2006*, 2006.

[JMB09] Márk Jelasity, Alberto Montresor, and Ozalp Babaoglu. T-man: Gossip-based fast overlay topology construction. *Comput. Netw.*, 53(13):2321–2339, August 2009.

[JVG+07] Márk Jelasity, Spyros Voulgaris, Rachid Guerraoui, Anne-Marie Kermarrec, and Maarten van Steen. Gossip-based peer sampling. *ACM Trans. Comput. Syst.*, 25, August 2007.

[KR04] David R Karger and Matthias Ruhl. Simple efficient load balancing algorithms for peer-to-peer systems. In *Proceedings of the sixteenth annual ACM symposium on Parallelism in algorithms and architectures*, pages 36–43. ACM, 2004.

[KRAV03] Dejan Kostić, Adolfo Rodriguez, Jeannie Albrecht, and Amin Vahdat. Bullet: High bandwidth data dissemination using an overlay mesh. In *ACM SIGOPS Operating Systems Review*, volume 37, pages 282–297. ACM, 2003.

[KT13] Anne-Marie Kermarrec and Peter Triantafillou. Xl peer-to-peer pub/sub systems. *ACM Computing Surveys (CSUR)*, 46(2), 2013.

[KW00] Balachander Krishnamurthy and Jia Wang. On network-aware clustering of web clients. In *SIGCOMM '00*, pages 97–110. ACM, 2000.

[KW01] Balachander Krishnamurthy and Jia Wang. Topology modeling via cluster graphs. In *Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement*, IMW '01, pages 19–23, 2001.

[LCC+02] Qin Lv, Pei Cao, Edith Cohen, Kai Li, and Scott Shenker. Search and replication in unstructured peer-to-peer networks. In *Proceedings of the 16th international conference on Supercomputing*, pages 84–95. ACM, 2002.

[LGB03] Prakash Linga, Indranil Gupta, and Ken Birman. A churn-resistant peer-to-peer web caching system. In *Proceedings of the 2003 ACM Workshop on Survivable and Self-regenerative Systems: In Association with 10th ACM Conference on Computer and Communications Security*, SSRS, pages 1–10, 2003.

[LKGN05] Jin Liang, Steven Y Ko, Indranil Gupta, and Klara Nahrstedt. Mon: On-demand overlays for distributed system management. In *WORLDS*, volume 5, pages 13–18, 2005.

[LM99] Meng-Jang Lin and Keith Marzullo. Directional gossip: Gossip in a wide area network. In *Proceedings of the Third European Dependable Computing Conference on Dependable Computing*, EDCC-3, pages 364–379, 1999.

[LM10] Avinash Lakshman and Prashant Malik. Cassandra: a decentralized structured storage system. *SIGOPS Oper. Syst. Rev.*, 44(2), 2010.

[LPR07] Joao Leitao, Jose Pereira, and Luis Rodrigues. Epidemic broadcast trees. In *SRDS'07*, pages 301–310, October 2007.

[LQGL06] Xucheng Luo Xucheng Luo, Zhiguang Qin Zhiguang Qin, Ji Geng Ji Geng, and Jiaqing Luo Jiaqing Luo. Iac: Interest-aware caching for unstructured p2p. In *Semantics, Knowledge and Grid, 2006. SKG'06. Second International Conference on*, pages 58–58. IEEE, 2006.

[LXQ+08] Bo Li, Susu Xie, Yang Qu, Gabriel Y Keung, Chuang Lin, Jiangchuan Liu, and Xinyan Zhang. Inside the new coolstreaming: Principles, measurements and performance implications. In *IEEE INFOCOM 2008*, 2008.

[MG+11] Peter Mell, Tim Grance, et al. The nist definition of cloud computing. 2011.

[MJ09] Alberto Montresor and Márk Jelasity. Peersim: A scalable p2p simulator. In *P2P 2009*, 2009.

[MJB05] Alberto Montresor, Márk Jelasity, and Ozalp Babaoglu. Chord on demand. In *P2P'2005*, 2005.

[MKG03] Laurent Massoulié, A-M Kermarrec, and Ayalvadi J Ganesh. Network awareness and failure resilience in self-organizing overlay networks. In *Reliable Distributed Systems, 2003. Proceedings. 22nd International Symposium on*, pages 47–55. IEEE, 2003.

[MLS08] Patrick Mukherjee, Christof Leng, and Andy Schürr. Piki-a peer-to-peer based wiki engine. In *Peer-to-Peer Computing, 2008. P2P'08. Eighth International Conference on*, pages 185–186. IEEE, 2008.

[Mor07] Joseph C Morris. Distriwiki:: a distributed peer-to-peer wiki network. In *Proceedings of the 2007 international symposium on Wikis*, pages 69–74. ACM, 2007.

[MSF+13] Miguel Matos, Valerio Schiavoni, Pascal Felber, Rui Oliveira, and Etienne Rivière. Lightweight, efficient, robust epidemic dissemination. *J. Parallel Distrib. Comput.*, 73(7):987–999, 2013.

[OMMD10] Gérald Oster, Rubén Mondéjar, Pascal Molli, and Sergiu Dumitriu. Building a collaborative peer-to-peer wiki system on a structured overlay. *Computer Networks*, 54(12):1939–1952, 2010.

[OO06] Francis Otto and Song Ouyang. Improving search in unstructured p2p systems: Intelligent walks (i-walks). In *Proceedings of the 7^{th} International Conference on Intelligent Data Engineering and Automated Learning (IDEAL'06)*, volume 4224 of *Lecture Notes in Computer Science*, pages 1312–1319. Springer, September 2006.

[OUMI06] Gérald Oster, Pascal Urso, Pascal Molli, and Abdessamad Imine. Data consistency for p2p collaborative editing. In *Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work*, pages 259–268. ACM, 2006.

[own] owncloud. URL: https://owncloud.org/.

[PGES05] Johan Pouwelse, Paweł Garbacki, Dick Epema, and Henk Sips. The bittorrent p2p file-sharing system: Measurements and analysis. In *International Workshop on Peer-to-Peer Systems*, pages 205–216. Springer, 2005.

[PLMS06] Peter R. Pietzuch, Jonathan Ledlie, Michael Mitzenmacher, and Margo I. Seltzer. Network-aware overlays with network coordinates. In *26th International Conference on Distributed Computing Systems Workshops (ICDCS 2006 Workshops), 4-7 July 2006, Lisboa, Portugal*, page 12, 2006.

[PMRS14] Mathieu Pasquet, Francisco Maia, Etienne Rivière, and Valerio Schiavoni. Autonomous multi-dimensional slicing for large-scale distributed systems. In *DAIS*, pages 141–155, 2014.

[PRGK09] Jay A Patel, Étienne Rivière, Indranil Gupta, and Anne-Marie Kermarrec. Rappel: Exploiting interest and network locality to improve fairness in publish-subscribe systems. *Computer Networks*, 53(13):2304–2320, 2009.

[QCY+07] Tongqing Qiu, Guihai Chen, Mao Ye, Edward Chan, and Ben Y. Zhao. Towards location-aware topology in both unstructured and structured p2p systems. In *ICPP*, page 30. IEEE Computer Society, 2007.

[RD01] Antony I. T. Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware*, pages 329–350, 2001.

[RD10] Rodrigo Rodrigues and Peter Druschel. Peer-to-peer systems. *Communications of the ACM*, 53(10):72–82, 2010.

[RFH+01] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A scalable content-addressable network. *SIGCOMM Comput. Commun. Rev.*, 31(4):161–172, 2001.

[RHKS02] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker. Topologically-aware overlay construction and server selection. In *INFOCOM'02*, volume 3, pages 1190– 1199. IEEE, 2002.

[Rip01] Matei Ripeanu. Peer-to-peer architecture case study: Gnutella network. In *Peer-to-Peer Computing, 2001. Proceedings. First International Conference on*, pages 99–100. IEEE, 2001.

[RLS+03] Ananth Rao, Karthik Lakshminarayanan, Sonesh Surana, Richard Karp, and Ion Stoica. Load balancing in structured p2p systems. In *Peer-to-Peer Systems II*, pages 68–79. Springer, 2003.

[RS02] Sylvia Ratnasamy and Scott Shenker. Can heterogeneity make gnutella scalable? In *IPTPS'02*, 2002.

[RS04] Venugopalan Ramasubramanian and Emin Gün Sirer. Beehive: O (1) lookup performance for power-law query distributions in peer-to-peer overlays. In *Nsdi*, volume 4, pages 8–8, 2004.

[SJZ+98] Chengzheng Sun, Xiaohua Jia, Yanchun Zhang, Yun Yang, and David Chen. Achieving convergence, causality preservation, and intention preservation in real-time cooperative editing systems. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 5(1):63–108, 1998.

[SMB02] Tyron Stading, Petros Maniatis, and Mary Baker. Peer-to-peer caching schemes to address flash crowds. In *International Workshop on Peer-to-Peer Systems*, pages 203–213. Springer, 2002.

[SMK+01] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM '01*. ACM, 2001.

[SSR08] Thorsten Schütt, Florian Schintke, and Alexander Reinefeld. Scalaris: reliable transactional p2p key/value store. In *Proceedings of the 7th ACM SIGPLAN workshop on ERLANG*, pages 41–48. ACM, 2008.

[Sto02] M Stokes. Gnutella2 specifications part one. *Rapport technique*, 2002.

[SW05] Ralf Steinmetz and Klaus Wehrle. 2. what is this "peer-to-peer" about? In *Peer-to-peer systems and applications*, pages 9–16. Springer, 2005.

[TCW05] Chunqiang Tang, Rong N Chang, and Christopher Ward. Gocast: Gossip-enhanced overlay multicast for fast and dependable group communication. In *Dependable Systems and Networks, 2005. DSN 2005. Proceedings. International Conference on*, pages 140–149. IEEE, 2005.

[TCW+14] Radu Tudoran, Alexandru Costan, Rui Wang, Luc Bougé, and Gabriel Antoniu. Bridging Data in the Clouds: An Environment-Aware System for Geographically Distributed Data Transfers. In *IEEE/ACM CCGrid*, Chicago, May 2014.

[UPVS07] Guido Urdaneta, Guillaume Pierre, and Maarten Van Steen. A decentralized wiki engine for collaborative wikipedia hosting. In *WEBIST (1)*, pages 156–163, 2007.

[VRKS06] Spyros Voulgaris, Etienne Rivière, Anne-Marie Kermarrec, and Maarten Van Steen. Sub-2-sub: Self-organizing content-based publish subscribe for dynamic large scale collaborative networks. In *In IPTPS'06: the fifth International Workshop on Peer-to-Peer Systems*, 2006.

[VRMCL08] Luis M. Vaquero, Luis Rodero-Merino, Juan Caceres, and Maik Lindner. A break in the clouds: Towards a cloud definition. *SIGCOMM Comput. Commun. Rev.*, 39(1):50–55, December 2008.

[VS05] S. Voulgaris and M. v. Steen. Epidemic-style management of semantic overlays for content-based searching. In *Euro-Par'05*, 2005.

[VYF06] Vidhyashankar Venkataraman, Kaouru Yoshida, and Paul Francis. Chunkyspread: Heterogeneous unstructured tree-based peer-to-peer multicast. In *Network Protocols, 2006. ICNP'06. Proceedings of the 2006 14th IEEE International Conference on*, pages 2–11. IEEE, 2006.

[WAB+09] Christof Weinhardt, Arun Anandasivam, Benjamin Blau, Nikolay Borissov, Thomas Meinl, Wibke Michalk, and Jochen Stößer. Cloud computing – a classification, business models, and research directions. *Business and Information Systems Engineering*, 1(5):391–399, 2009.

[WR03] M. Waldvogel and R. Rinaldi. Efficient topology-aware overlay network. In *SIGCOMM/CCR'03*, 2003.

[WSS05] Bernard Wong, Alex Slivkins, and Emin Gun Sirer. Meridian: A lightweight network location service without virtual coordinates. In *ACM SIGCOMM*, August 2005.

[WUM07] Stéphane Weiss, Pascal Urso, and Pascal Molli. Wooki: a p2p wiki-based collaborative writing tool. In *International Conference on Web Information Systems Engineering*, pages 503–512. Springer, 2007.

[WUM10] Stéphane Weiss, Pascal Urso, and Pascal Molli. Logoot-undo: Distributed collaborative editing system on P2P networks. *IEEE Transactions on Parallel and Distributed Systems*, 21(8):1162–1174, 2010.

[XTZ03] Z. Xu, C. Tang, and Z. Zhang. Building topology-aware overlays using global soft-state. In *ICDSC'03*, May 2003.

[YGM02] Beverly Yang and Hector Garcia-Molina. Improving search in peer-to-peer networks. In *Distributed Computing Systems, 2002. Proceedings. 22nd International Conference on*, pages 5–14. IEEE, 2002.

[ZKJ01] B.Y. Zhao, J. Kubiatowicz, and A.D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. *Computer*, 74, 2001.

[ZLX+05] Xiaodong Zhang, Yunhao Liu, Li Xiao, Xiaomei Liu, and Lionel M. Ni. Location awareness in unstructured peer-to-peer systems. *IEEE Transactions on Parallel and Distributed Systems*, 16:163–174, 2005.

[ZZJ+01] Shelley Q Zhuang, Ben Y Zhao, Anthony D Joseph, Randy H Katz, and John D Kubiatowicz. Bayeux: An architecture for scalable and fault-tolerant wide-area data dissemination. In *Proceedings of the 11th international workshop on Network and operating systems support for digital audio and video*, pages 11–20. ACM, 2001.

[ZZZ+06] Xin Yan Zhang, Qian Zhang, Zhensheng Zhang, Gang Song, and Wenwu Zhu. A construction of locality-aware overlay network: moverlay and its performance. *IEEE J.Sel. A. Commun.*, 22(1):18–28, September 2006.

List of Figures

List of Tables

List of Algorithms

Résumé

Dans cette thèse, nous abordons deux problèmes soulevés par les systèmes distribués décentralisés - le placement de réseaux logiques de façon compatible avec le réseau physique sous-jacent et la construction de cohortes d'éditeurs pour dans les systèmes d'édition collaborative.

Bien que les réseaux logiques (overlay networks) été largement étudiés, la plupart des systèmes existant ne prennent pas ou prennent mal en compte la topologie du réseau physique sous-jacent, alors que la performance de ces systèmes dépend dans une grande mesure de la manière dont leur topologie logique exploite la localité présente dans le réseau physique sur lequel ils s'exécutent. Pour résoudre ce problème, nous proposons dans cette thèse Fluidify, un mécanisme décentralisé pour le déploiement d'un réseau logique sur une infrastructure physique qui cherche à maximiser la localité du déploiement. Fluidify utilise une stratégie double qui exploite à la fois les liaisons logiques d'un réseau applicatif et la topologie physique de son réseau sous-jacent pour aligner progressivement l'une avec l'autre. Le protocole résultant est générique, efficace, évolutif et peut améliorer considérablement les performances de l'ensemble.

La deuxième question que nous abordons traite des plates-formes d'édition collaborative. Ces plates-formes permettent à plusieurs utilisateurs distants de contribuer simultanément au même document. Seuls un nombre limité d'utilisateurs simultanés peuvent être pris en charge par les éditeurs actuellement déployés. Un certain nombre de solutions pair-à-pair ont donc été proposées pour supprimer cette limitation et permettre à un grand nombre d'utilisateurs de collaborer sur un même document sans aucune coordination centrale. Ces plates-formes supposent cependant que tous les utilisateurs d'un système éditent le même jeu de document, ce qui est peu vraisemblable. Pour ouvrir la voie à des systèmes plus flexibles, nous présentons, Filament, un protocole décentralisé de construction de cohorte adapté aux besoins des grands éditeurs collaboratifs. Filament élimine la nécessité de toute table de hachage distribuée (DHT) intermédiaire et permet aux utilisateurs travaillant sur le même document de se retrouver d'une manière rapide, efficace et robuste en générant un champ de routage adaptatif autour d'eux-mêmes. L'architecture de Filament repose sur un ensemble de réseaux logiques auto-organisées qui exploitent les similarités entre jeux de documents édités par les utilisateurs. Le protocole résultant est efficace, évolutif et fournit des propriétés bénéfiques d'équilibrage de charge sur les pairs impliqués.

Abstract

In this thesis, we address two issues in the area of decentralized distributed systems: network-aware overlays and collaborative editing.

Even though network overlays have been extensively studied, most solutions either ignores the underlying physical network topology, or uses mechanisms that are specific to a given platform or applications. This is problematic, as the performance of an overlay network strongly depends on the way its logical topology exploits the underlying physical network. To address this problem, we propose Fluidify, a decentralized mechanism for deploying an overlay network on top of a physical infrastructure while maximizing network locality. Fluidify uses a dual strategy that exploits both the logical links of an overlay and the physical topology of its underlying network to progressively align one with the other. The resulting protocol is generic, efficient, scalable and can substantially improve network overheads and latency in overlay based systems.

The second issue that we address focuses on collaborative editing platforms. Distributed collaborative editors allow several remote users to contribute concurrently to the same document. Only a limited number of concurrent users can be supported by the currently deployed editors. A number of peer-to-peer solutions have therefore been proposed to remove this limitation and allow a large number of users to work collaboratively. These decentralized solution assume however that all users are editing the same set of documents, which is unlikely to be the case. To open the path towards more flexible decentralized collaborative editors, we present Filament, a decentralized cohort-construction protocol adapted to the needs of large-scale collaborative editors. Filament eliminates the need for any intermediate DHT, and allows nodes editing the same document to find each other in a rapid, efficient and robust manner by generating an adaptive routing field around themselves. Filament's architecture hinges around a set of collaborating self-organizing overlays that utilizes the semantic relations between peers. The resulting protocol is efficient, scalable and provides beneficial load-balancing properties over the involved peers.