Fluid Models for Content Distribution Systems

Florence Clévenot-Perronnin

Université de Nice - Sophia Antipolis – UFR Sciences
École Doctorale STIC

THÈSE

Présentée pour obtenir le titre de :
Docteur en Sciences de l'Université de Nice - Sophia Antipolis

Spécialité : Informatique

par

Florence Clévenot-Perronnin

Équipe d'accueil : MAESTRO – INRIA Sophia Antipolis

Fluid Models for Content Distribution Systems

Soutenue publiquement à l'INRIA le 3 octobre 2005 devant le jury composé de :

Directeur :	Dr. Philippe	Nain	INRIA
Rapporteurs :	Pr. R.	Srikant	University of Illinois, Urbana-Champaign
	Pr. Don	Towsley	University of Massachusetts, Amherst
Examinateurs :	Pr. Ernst	Biersack	Institut Eurécom
	Pr. Keith	Ross	Polytechnic University, New York
	Dr. Jean-Marc	Vincent	Université Joseph Fourier, Grenoble

Fluid Models for Content Distribution Systems

Florence Clévenot-Perronnin

Titre de la thèse en français :

Modèles Fluides pour l'Analyse des Systèmes de Distribution de Contenu

Acknowledgments

I would like to express my deepest gratitude to my advisor Philippe Nain for his dedication and guidance. I have been extremely fortunate to work with such a talented researcher, and his total confidence in my work has always been encouraging especially at the most critical moments.

During the course of this thesis I have been fortunate to work with external researchers. I am particularly indebted to Keith Ross for our fruitful collaboration on several chapters of this thesis, and for always pushing me to focus on the latest "killer problems". I also thank Marwan Krunz for asking the right question that helped me change the destiny of a so far unfortunate paper.

I wish to thank the members of my PhD defense committee for accepting this responsibility: R. Srikant and Don Towsley who reviewed my thesis, Ernst Biersack who presided the jury, and Keith Ross and Jean-Marc Vincent.

I would like to express my gratitude to all the present and past members of the MISTRAL/MAESTRO team at INRIA with whom I shared numerous discussions. In particular, I wish to thank Maria Ladoue and Urtzi Ayesta for their true friendship. Among the others I would also like to thank Robin Groenevelt, Nidhi Hegde, Sujay Sanghavi, Nicolas Bonneau, Victor Ramos and Rabea Boulifa for their help, advice, support, cheerfulness and kindness. Special thanks also to Ephie Deriche for her efficiency and helpful assistance.

I am grateful to my parents for letting me choose my own path and for helping me gain self-confidence. From them I have learned the value of work. To them and to my brother Pierre, thanks for making my life so cheerful.

Last but definitely not least, all my love and thanks to my husband Florent, without whom I would never even have started a PhD. Not only has he helped me in many practical aspects by reviewing my work and sharing his ideas, but he has

also given me the taste for research from the beginning and has patiently, endlessly encouraged me throughout these years. He fills my life with pride and happiness, and has always been my most important motivation. I did this thesis for him and thanks to him.

Table of Contents

List of Figures

List of Tables

Résumé

Les systèmes de distribution de contenu comme les caches web et les réseaux d'échanges de fichiers doivent pouvoir servir une population de clients à la fois très grande (centaines de milliers) et fortement dynamique (temps de connexion très courts). Ces caractéristiques rendent leur analyse très coûteuse par les approches traditionnelles comme les modèles markoviens ou la simulation. Dans cette thèse nous proposons des modèles fluides simples permettant de s'affranchir de l'une des dimensions du problème.

Dans la première partie, nous développons un modèle stochastique fluide pour les systèmes de caches distribués. Les documents stockés sont modélisés par un fluide augmentant avec les requêtes insatisfaites. Nous appliquons ce modèle aux "clusters" de caches et à Squirrel, un système de cache pair-à-pair. Dans les deux cas notre modèle permet de calculer efficacement et avec précision la probabilité de hit, et de mettre en évidence les paramètres clés de ces systèmes. Nous proposons également une approximation multiclasses pour modéliser la popularité des documents.

Dans la seconde partie de cette thèse nous étudions BitTorrent, un système d'échange de fichiers pair-à-pair. Nous proposons un modèle fluide multiclasses qui remplace les usagers par un fluide. Nous considérons deux classes d'usagers pour modéliser les différences de débits d'accès ou de qualité de service. Nous obtenons une formule close pour le temps de téléchargement dans chaque classe. Nous montrons également comment allouer la bande passante a chaque classe pour offrir un service différencié.

Abstract

Content distribution systems (CDS) such as web caches and file sharing systems are large-scale distributed systems that may serve hundreds of thousands of users. These highly dynamic systems exhibit a very large state space which makes them difficult to analyze with classical tools such as Markovian models or simulation. In this thesis we propose macroscopic fluid models to reduce the complexity of these systems. We show that these simple models provide accurate and insightful results on the performance of CDS.

In the first part we propose a generic fluid model for distributed caching systems. The idea is to replace cached documents with fluids that increase with unsatisfied requests. Caches may go up and down according to a birth-death process. We apply this model to study two caching systems: cache clusters and a P2P cooperative cache system called Squirrel. We derive an efficient and accurate expression for their hit probabilities and show how the model identifies the key tradeoffs of these systems. We also propose a multiclass approximation for taking into account document popularity.

In the second part of the thesis we consider file sharing systems such as BitTorrent. We propose a two-class fluid model which replaces downloaders with fluids. This simple deterministic model may reflect the problem of service differentiation or bandwidth diversity for instance. We provide a closed-form expression of the average download time for each class under the worst-case assumption that users leave the system immediately after completing their download. We also show how to allocate peers bandwidth between classes to achieve service differentiation.

Chapter 1

Introduction

Let us consider a collection of documents such as HTML pages, images, multimedia content offered by a set of web servers to a plurality of interested clients through a network. A Content Distribution System (CDS) is a system designed to facilitate the distribution of documents to the clients from the web servers, according to a target performance metric. The origin Web servers are sometimes also considered to belong to the class of CDS systems [SGD+02]. However, using our afore mentioned definition we will restrict a CDS to being a logical intermediary between Web clients and servers as shown in Figure 1.1.

Note that the representation in Figure 1.1 is purely logical. In its physical instantiation, a CDS may be implemented directly at the clients, as in a peer-to-peer network such as Kazaa [Kaz] or Gnutella [Gnu], or at the server level as in a content distribution network like Akamai [Aka, DMP+02]. It may also consist of a dedicated set of intermediary servers as in the caching paradigm. Therefore, the concept of a content distribution system overcomes the traditional client-server architecture which used to prevail in many Internet applications (FTP, Telnet, Web browsing...).

Having defined a content distribution system, we now classify them. Currently, there exist mainly three types of architecture designed to alleviate the load on origi-

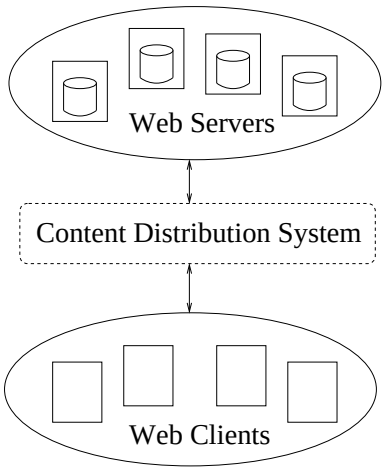

Figure 1.1: Logical representation of Content Distribution Systems

nating Web servers and/or facilitate the diffusion of content by bringing the desired documents closer to the set of users, where the notion of closeness may include geographical, topological or delay factors [KWZ01].

The first type of CDS is the class of Web caching sytems. These systems are widely used and easy to implement at proxy servers of virtually any existing private or institutional network. They rely on the simple observation that a recently accessed document is likely to be accessed again in the near future, especially given the skewness of the popularity distribution of objects [BCF$^+$99]. Typically, cache servers are physically placed between end users and web servers. They keep a copy of each accessed file to answer directly the future requests for these files, and save the users the delay of contacting the originating server.

A second class of CDS is the class of file sharing systems. The idea is that a popular file downloaded by a given client c_i may also be of interest for another client c_j of the same local network. If c_j can get the file directly from c_i, its latency is greatly reduced while also reducing the load on the originating server. This is the essence of the peer-to-peer (P2P) concept where clients (peers) also act as local servers for their neighbors. In this case the CDS is physically part of the client network. These peer-to-peer file sharing systems have recently become the main source of internet traffic (see, for instance, [AG04, KBB$^+$04]), mainly by making easily available highly popular multimedia content such as music files and video clips. In peer-to-peer systems, every peer keeps a number of documents that are made available to other peers. An object

may be localized through a variety of techniques, such as request flooding as in Gnutella [Gnu], the use of hash tables as in Chord [SMK+01] for instance, or even through a request to a centralized server as in the first version of Napster (see for instance [SGG03] for a description of Napster's architecture).

The third and last category of CDS is the class of Content Distribution Networks (CDNs). These networks are designed to speed up content delivery and reduce the load on Web servers by replicating their content and making it available to clients. The principle of a content distribution network is slightly different from the caching paradigm in the two following aspects. First, CDNs are privately owned networks that provide their service to Web servers, whereas a cache system is typically locally administrated by the client LAN or the Web server network. The typical CDN service includes strategic locations worldwide, server availability and handling of dynamical content, while caching systems only offer a local service and a limited range of cacheable document types. Second, content may be pushed by the Web server into the CDN replicas, whereas in the caching paradigm the copy is generally made upon a client request. A CDN may be a worldwide network of shared servers, which physically reflects the logical architecture of Figure 1.1, or it may be a server farm located at the server place, in which case it physically belong to the "server" entity from a network point of view.

Analyzing the performance of these CDSs is critical, for many reasons. First, regarding emerging technologies such as new P2P architectures for instance, it is crucial to evaluate the performance and scalability of the system early in the development process to avoid deploying inadequate systems and to anticipate possible causes of latency or overload. Performance analysis of these systems also allows one to identify the important tradeoffs and to dimension these systems properly. Finally, performance analysis is helpful, even for already deployed systems, for designing new features and services, or concurrent systems that may bring significant improvement. It may also be used for pricing problems.

However, CDSs exhibit an intrinsic complexity which makes their performance analysis a difficult problem. Indeed, these systems deal with highly dynamic, heterogeneous and increasingly numerous users, servers and documents. To give an order of magnitude of the typical dimension of a CDS, let us consider a few qualitative figures. For instance, institutional caching systems must be able to serve tens of thousands of users [WVS+99, DMF97] with total requests rates ranging from 12 to 178 requests

per second in large systems [WVS+99, DMF97], in a Web that contains billions of documents (about 8 billion pages referenced by Google in June 2005). Regarding CDNs, these systems are used by a significant fraction of the most popular web sites [KWZ01] and therefore need to face high request rates for rapidly changing sets of documents. P2P systems typically involve thousands to millions of users (statistics available at [Edo, Sly, IUKB+04]) that frequently interrupt and resume their download [IUKB+04]. The total traffic generated by these systems account for more than half of the total internet traffic [AG04]. In addition, hosts may fail and be repaired, which can modify both the cache, servers and user population, at nonnegligible rates: according to [LMG95], many hosts stay up for about a week begore going down, and then go back up after a short time. Though these figures were observed in 1995, churn rates have not decreased and are even increasing due to users joining and leaving the system several times a day in P2P systems for instance [BSV03].

For these reasons, classical analysis tools such as discrete Markovian models or discrete-event simulation, suffer from a too large state space and often require costly numerical methods or model simulations [ZA03, GFJ+03].

Inspired by the seminal work by Anick, Mitra and Sondhi in 1982 [AMS82] and the subsequent success of fluid modeling of packet networks (see for instance [EM92, EM93, KM01a, LZTK97, BBLO00, RRR02, LFG+01] and references therein), the central axis of this thesis is to propose a fluid approach for modeling content distribution systems, where the fluid approximation allows to reduce the discrete state space dimension of these systems.

The outline of this dissertation is as follows.

- In the first part we propose to replace content with a fluid for modeling distributed caching systems. This part is decomposed into four chapters:
  - in Chapter 2 we review existing work and introduce a generic fluid framework for modeling caching systems.
  - in Chapter 3 we apply the model to a cache cluster system. We show how the model exhibits some key properties of this system and quantitatively compare two possible request direction schemes as an illustration of the meaningfulness of the model. We then validate the model through a comparison with discrete-event simulation.

  - in Chapter 4 we apply the model to a novel cooperative web caching system called Squirrel [IRD02]. We use an Engset model [Kel79] to model the user behaviour. We underline the analytical differences with Chapter 3 and compute the expected hit probability of this new system. Again, we outline the important tradeoffs of this system and show how it can be expected to scale with the number of users.
  - in Chapter 5 we show how to overcome some limitations of the previous two chapters. We first address a scalability problem by using an $M/M/\infty$ user model instead of an Engset model. This new model provides the same numerical results as the Engset model but now allows us to cope with realistic network sizes (even millions of users). We then address the probability distribution of requested documents by a clustering approximation.

- In the second and last part, we propose a second fluid model designed for peer-to-peer file sharing systems. The idea is to take into account document replication among the CDS by considering the sharing of a single file, and modeling the downloaders by fluid. This part is composed of a unique chapter:

  - in Chapter 6 we propose a multiclass model of users based on [QS04]. Our approach allows us to evaluate and propose a service differentiation feature in BitTorrent-like networks. We also show how it is possible to optimize the protocol in presence of heterogeneous users.

Finally, Chapter 7 concludes this thesis.

Part I

A Document-Based Fluid Model

Chapter 2

A Document-Based Fluid Model

2.1 Introduction

In this chapter we propose a generic framework for modeling distributed caching systems. We first present an overview of caching systems and highlight the key features of these systems. Then we review existing work on the performance analysis of distributed caching systems. We finally introduce our generic fluid framework for modeling these systems: a document-based stochastic fluid model.

2.2 Caching Systems

Web caching systems are designed to save bandwidth and reduce Web latency by keeping copies of popular documents in servers (caches) that are "closer" to the end-users than the Web servers, where the notion of closeness ideally means a low latency.

The basic mechanism of caching is as follows. Let us take the common example of a proxy server located at the edge of a local area network (LAN). This proxy server

monitors all accesses of local clients to the Internet: it forwards requests to remote servers and sends the replies to the appropriate client. Since many clients are likely to request common documents (especially the most popular ones), the proxy server may keep (or cache) a copy of each requested document when it is first sent by the remote server. Thus, the proxy server will be able to answer directly all future requests for these documents and will save external bandwidth as well as external latency for the client. This event is called a "cache hit" and its frequency is one of the main performance indicators of caching systems. When a document is requested and is not in the proxy cache, the event is called a "cache miss". In this case, the proxy contacts the originating server, downloads the document and copies it into its cache before forwarding it to the requesting client.

Note that a cached object cannot be served forever to the requesting clients without running the risk of the original document having been updated since the first time it was requested. Therefore, cache systems need to know how long a document may be cached. In the absence of such an information, they typically use a heuristic to compute the time-to-live (TTL) of each cached document. In the typical freshness calculation heuristic, the lifetime is min(CONF_MAX, CONF_PERCENT×(Date-LastModified)) where CONF_PERCENT is a fraction typically limited to 10% and CONF_MAX is a default TTL value typically equal to a day, since HTTP/1.1 specifies that a cache must attach a warning to any response whose age is more than 24 hours [FGM+99]. When a cached document reaches it TTL, it is not necessarily immediately removed from the cache. Upon the next request for this document, the cache system attempts to validate its copy as follows. The cache issues a *conditional* GET request to the origin server, which answers with either a *Not-Modified* message or the document itself depending on whether the document has been changed since the cache downloaded it. This event is called a freshness miss, and typically incurs a latency close to that of a complete miss even if the document has not been modified [CK01a].

There are many issues involved in the caching paradigm. Designing a cache system needs to address many issues, including the following ones [Moh01]:

- Which documents should be cached? For instance, which types of objects, among Web pages, embedded objects, large files, dynamic pages (SQL query results for instance), and so on.
- Where should these objects be cached?

  - locally at the client host (local cache),
  - at a site proxy (e.g., attached to a LAN)
  - on an organizational proxy server : global server for universities, companies, government agencies, ...
  - at a national or larger level (Internet service providers (ISPs) for instance)
  - locally at web servers
- How should the cache servers be dimensioned and what replacement policy should be used (FIFO, LFU, LRU...)?
- How long should a document be kept in cache? Hit rate vs. freshness tradeoff
- How to anticipate requests (prefetching, refreshment) to avoid miss latencies?
- How to prevent the proxy server to be a single point of failure (availability, bandwidth, CPU...)?
- In case of multiple proxy servers, how to coordinate servers?

As a result, there exists a number of caching technologies and systems. In this section we will present an overview of distributed caching systems, i.e., caching systems using several servers. An exhaustive review is out of the scope of this dissertation due to the large and rapidly evolving body of existing work in the area. The interested reader can refer to other surveys [Wan99, RS02]. We will thus focus on the most significant decentralized architectures and on some interesting novel approaches. We will particularly emphasize the description of two caching systems (hash routing schemes and Squirrel) that will be the target applications of the three next chapters.

2.2.1 Cache clusters

A single cache proxy may be simultaneously a bottleneck and a single point of failure for a network. To address this issue, a simple idea is to use a cluster of servers, which increases availability as well as hardware resources. In this architecture, all cache servers are at an equal level and are called “siblings” or “neighbors”. They may go up and down at random times, due to disk failures, software bugs, updates, or misconfigurations [BSV03, LMG95]. Several schemes have been proposed for this distributed architecture, in particular to decide to which cache server an incoming request needs to be routed.

In the remainder of this section we will consider the cache cluster to be built at an LAN or organizational level. In particular, we will not present Web server side caching and mirroring systems such as Backslash [SMB02] or Seres [VR02] for instance.

2.2.1.1 ICP

A first protocol for coordinating Web caches is the Internet Cache Protocol (ICP) which is described in [WC97c, WC97b]. This protocol allows communication between Web caches through ICP queries and replies. The ICP protocol uses UDP as a transport layer protocol. In the context of a cluster of equal web caches, the ICP caching system globally works as follows. A request for a document is sent to one of the caches. In case of a hit the document is simply sent by this cache to the requesting user. In case of a miss, the cache first queries all other caches in the cluster with ICP query messages. If one of the sibling caches has the document, the first cache retrieves the document from that sibling (e.g., the first to respond with an ICP hit message). Then it stores a copy of the document and sends it to the client. If no cache in the cluster has the document, the first cache retrieves it from the remote Web server, keeps a copy in its cache and sends it to the client.

Potential problems can arise from this protocol. First, the most popular documents will be replicated among many caches, which results in a waste of storage space. Second, in the case of a miss, the latency seen by the client is increased as the first cache has to wait for all ICP replies before concluding to a miss and fetching the document from the originating server. Third, ICP messages consume processing resources of all siblings. On the other hand, with ICP the stored content of the cache cluster is only lightly affected in the event of a cache failure, thanks to the replication feature of the protocol.

2.2.1.2 Hash routing schemes

Another approach for using web cache clusters is to use a hash function at clients which maps URLs to a hash space which is then divided among the caches.

The detailed behavior of hash routing schemes is as follows. When a client in the organization makes a request for an object, the request is sent to one of the up caches.

If the up cache receiving the request has the object, it immediately sends a copy to the client. Otherwise, the cache retrieves a copy of the object from the origin server, stores a copy, and sends a copy to the client. Because caches are going up and down at relatively slow time scales compared to requests, we assume that each client always knows which caches are up, that is, each client tracks the set of active caches. (This is typically done by configuring each browser to retrieve a *proxy automatic configuration (PAC) file* each time the browser is launched. The PAC file indicates which caches are currently up, and also implements the direction policy as discussed later in this section.)

It remains to specify how a client requesting a particular object determines to which cache it should direct its request. This is specified by the *direction policy*. Ideally, to avoid object duplication across caches, we want requests from different clients for the same object to be directed to the same cache in the cluster. This ensures that at most one copy of any object resides in the cache cluster. Also, we would like the request load to be evenly balanced among the caches in the cluster. These two goals are often achieved by using a common mapping at all the clients. When a client wants an object, it maps the object name (typically a URL) to a specific cache in the cluster, and directs its request to the resulting cache. This mapping can be created with a hash function as follows. Let $h(\cdot)$ be a hash function that maps object names to the set $[0, 1)$. Let i be the number of up caches. Partition $[0, 1)$ into i intervals of equal length, $\Psi_1 = [0, 1/i)$, $\Psi_2 = [1/i, 2/i)$, ..., $\Psi_i = [1-1/i, 1)$. Associate one up cache with each of these intervals. Then when a client makes a request for object o, it calculates $h(o)$ and determines the interval Ψ_j for which $h(o) \in \Psi_j$. It then directs its request for object o to the jth cache. We refer to this direction policy as *partition hashing*. If the hash function has good dispersion properties, partition hashing should balance the load among the caches in a more-or-less equitable manner.

Partition hashing has a serious flaw, however. When a new cache is added or goes down, approximately 50% of all the cached objects are cached in the wrong caches [Ros97]. This implies that after an up/down event, approximately 50% of the requests will be directed to the wrong up cache, causing "misses" even when the object is present in the cluster. Furthermore, partition hashing will create significant duplication of objects after an up/down event. Because the caches employ cache replacement policies, such as *least recently used (LRU)*, this duplication will eventually be purged from the system.

To solve this problem, independent teams of researchers have proposed refined hashing techniques, including CARP and consistent hashing, which route requests to their correct caches with high probability even after a failure/installation event [VR97, KSB+99]. Such robust hashing techniques have been used in Microsoft and Netscape caching products, and also appear to have been implemented in the Akamai content distribution network. We now briefly describe CARP; consistent hashing is similar. CARP uses a hash function $h(o, j)$ that is both a function of the object name o and the cache name j. When the client wants to obtain object o, it calculates the hash function $h(o, j)$ for each j, and finds the cache j^* that maximizes $h(o, j)$. We henceforth refer to this technique as *winning hashing*. The principal feature of winning hashing is that relatively few objects in the cluster become misplaced after an up/down event [Ros97]. Specifically, when the number of active caches increases from j to $j + 1$, only the fraction $1/(j + 1)$ of the currently correctly-placed objects become incorrectly placed; furthermore, when the number of up nodes decreases from $j + 1$ to j, none of the currently correctly-placed objects become misplaced.

Globally, hash routing has been shown to be more efficient than ICP for single-level cache clusters [Ros97], regarding both client-perceived latency and processing overhead for caches.

2.2.1.3 Other systems

Apart from the ICP communication protocol and the hash routing scheme, there exist many other creative proposals for cache clusters architectures.

The Cachemesh [WC97a] architecture resembles hash routing schemes in the sense that cache servers try not to replicate content. The key difference is that request routing to the corresponding cache is now done using routing tables instead of hash functions: each cache server maintains a routing table with a list of Web sites and the corresponding cache to which it should forward requests. As a result, since only cache servers are equipped with routing tables, a client may first send its request to a cache which is not responsible for the document. The choice of a designated cache for a given Web site is also made through the use of the routing table, including a default route for unknown sites. It is also possible for a cache server to indicate a list of Web sites it wants to be responsible for. As a result, Cachemesh is flexible but requires the potentially heavy cost of maintaining routing tables for Web sites, and does not

provide load balancing features among the caches in the cluster. The Relais Project [Gro98] proposes a very similar architecture in which each node maintains a shared directory of the documents stored by all other caches. This directory is updated each time a cache server notifies an addition or removal of document in its own cache, which generates update messages between cache servers in addition to the request messages (similar to ICP messages). Unlike ICP however, this protocol generates little overhead in comparison since only one server is queried instead of the whole cluster. This protocol mainly suffers from very high memory consumption for the maintenance of the directory at each node.

The architecture of the CRISP cache system [GRC97] lies midway between hash routing and cache routing tables as in Cachemesh. A client sends its request for an object to one of the caches, which is determined by the browser by using a Proxy Automatic Configuration (PAC) file for instance. This cache belongs to the cache cluster and forwards the request to a central authority called a "mapping server". This mapping server maintains a directory which indicates for any URL which cache server of the cluster holds a copy of the document. In case of a hit at the peer cache, the cache server that was contacted in first place directly retrieves the document from the peer cache and forwards it to the requesting client. In case of a miss, when a document has never been requested for instance, the chosen cache server which will store a copy is determined using a partition of the URL space, for instance with a hash function. To ensure consistency of the directory table, all caches in the cluster notify the mapping server each time they add or remove an object in their local cache. The single point of failure arising at the mapping server is not so damaging as in the case of a unique centralized proxy because only the cache feature becomes unavailable, while Internet access is still provided by the proxies of the cluster. However, this architecture also exhibits the cost of maintaining a directory table, introduces additional processing delays at each step (first proxy, mapping server, then home node) and especially, requires a strong geographical locality to exhibit acceptable latencies in the proxy/mapping server communications.

Another architecture maintains locally at each cache a summary representation of other caches in the cluster which is updated periodically with a modified ICP. This architecture is the core of the Summary Cache [FCAB98] and of the Cache Digest [RW98] proposals that were developed independently in 1998. We briefly describe the Summary Cache protocol; Cache Digest is based on the same principle and only differs in small details such as the update mechanism. Cache servers keep a summary of

all other cache servers' content through the compact representation of Bloom Filters. This representation is an efficient compression of the complete directory and provides very low false hit probabilities. The main advantage of this representation is that it saves both local memory as well as bandwidth consumption during periodical directory updates between nodes of the cluster. These updates typically happen when a predefined fraction of the total locally cached objects have been modified/added/removed. Therefore the Summary Cache saves both the ICP overhead and the replication cost of Cachemesh and Relais. The remaining cost is the consistency tradeoff between update messages overhead and false hits/misses, as well as the compression tradeoff in the Bloom filters between memory consumption and false hit probability. Note that in this system, the partition of the URL space is not done a priori but in an ad-hoc fashion: the cache server responsible for a given object will be the first server to receive a request for that object.

2.2.2 Hierarchical architectures

Designed to alleviate the load on access links and to take advantage of the large bandwidths available in the core portions of the Internet, hierarchical cache structures have been proposed. The most widespread hierarchical scheme is the Harvest architecture [CDN+96], or its derivative Squid [Wes98]. In this hierarchical structure, caches are placed at different levels of the Internet, for example: local level (browser cache), institutional level (proxy server), regional and national level [RSB01]. When a request is not satisfied by the local browser cache, it is forwarded to the institutional cache, which in turn either answers with the document of forwards the request to the regional cache. The latter finally forwards the request to the national cache in case of a miss, and the national cache will in the end contact the origin Web server if it does not hold a copy of the object. When the document is sent from the origin server, it travels down the hierarchy and a copy is made at every level for future requests. A cache at a given level is said to be the child (respectively the parent) of the cache of the upper (respectively lower) level. Several caches of the same level are said to be siblings, as in the case of cache clusters. The interest of the hierarchical architecture also lies in the fact that a high level cache may pool documents that can serve a number of children that may share common interests. An additional feature of this architecture is that at each level (except the local browser cache), a cache that does not have a copy of a requested object will contact all other siblings, typically through an ICP query message, in parallel

to the request to the parent cache. The protocol inside a given level cache group is exactly the ICP protocol for cache clusters described in Section 2.2.1.1. In case of a hit at a parent or sibling cache, the first queried cache retrieves the object from the first cache that responded with a hit, i.e. chooses the closest cache based on ICP latency. Some of the main drawbacks of this architecture are [RSB01]:

- every level of hierarchy introduces additional latency
- upper level caches may become a bottleneck
- documents are replicated at various levels, resulting in a waste of storage space.

Several modifications to this hierarchical system have been proposed. In particular, to save memory consumption, two hierarchical directory schemes have been proposed [PH97, TDVK99]. In [PH97] the authors propose that caches do not store copies of objects but only location hints as to where the object can be found. When a client requests an object, the first queried cache looks in a directory table whether it is aware of another client that might hold a copy of the object. If this is the case, it returns the address of that client to the requesting client which will in turn directly download the document from the peer client. Otherwise, the request is forwarded to upper levels until either a hit is found and a client address is returned, or the request results in a miss and the requesting client directly contacts the origin Web server. This scheme is therefore half way between hierarchical caching and peer-to-peer caching that will be described in the next section. In [TDVK99], the directory principle is the same except that it is translated one level higher. Indeed, all caches of the hierarchy hold directory tables, except the institutional caches which act as the CRISP cache system, in which the mapping server would be replaced by parent caches.

2.2.3 Peer-to-peer architectures

Peer-to-peer architectures take advantage of the individual resources of clients, which, though small if considered separately, may outperform any powerful centralized architecture, when pooled together in a large scale distributed system. In addition, these resources are often already present in any network and simply represent unutilized resources of clients, for instance overprovision in memory or CPU at idle times. The result

of this observation is that a peer-to-peer system may implement large scale functionalities, including content distribution, at a very low cost, with no dedicated hardware to purchase or maintain. This paradigm has been already applied to a number of applications, in particular distributed computing and file sharing. Regarding web caching, several systems have been developed to take advantage of peer-to-peer architectures. Indeed, a few megabytes (e.g., 10MB) of storage space available at each client of an organization, when organized in a completely decentralized cache, can perform as well as a dedicated cache system with sufficiently large storage capacity [IRD02] in terms of external bandwidth usage, but without the cost of creating and administrating a dedicated cache cluster.

We first present a hybrid scheme which is a mix between a centralized proxy server, the CRISP architecture, and a peer-to-peer design. Then we will turn to completely decentralized systems which are purely peer-to-peer in their design. There exist several proposals for a peer-to-peer caching system: Squirrel [IRD02], BuddyWeb [WNO+02] and a P2P caching application based on the Kelips overlay [LGB03]. We will describe Squirrel in detail in Section 2.2.3.2. Differences in the two other designs will be given at the end of the section.

2.2.3.1 Browsers-aware proxy server

A first (partially) peer-to-peer architecture is the Browsers-Aware Proxy Server [XZX02]. Though equipped with a central proxy and therefore not purely peer-to-peer, this caching system relies on its own clients to improve the performance by sharing their own private browser caches. The principle is the following one. The proxy server works as any centralized cache server, but also maintains a directory table of its clients individual browser caches. When a request cannot be satisfied from the proxy's cache, the proxy looks for a corresponding entry in the directory table. In case of a hit, the proxy replies with the address of the client that holds a copy of the object, and the requesting client directly retrieves the object from its peer client. This system is therefore very similar in principle to [PH97] except for the hierarchical structure at upper levels. In case of a miss, the proxy contacts the remote Web server and sends the file back to the requesting client upon reception. Clients may update the directory table of the proxy server either periodically or upon changes in their browser cache. Note that the authors of [XZX02] also propose a scheme in which, in the event of a hit in the directory table, the proxy itself downloads the file from the client and forwards it to the requesting

client. This alternative scheme does not present a peer-to-peer aspect anymore since the clients do not communicate directly together and all management of objects is done by the proxy server.

2.2.3.2 Overview of Squirrel

Squirrel [IRD02] is a decentralized, peer-to-peer Web cache that uses Pastry [RD01] as a location and routing protocol. When a client requests an object it first sends a request to the Squirrel proxy running on the client's machine. If the object is uncacheable then the proxy forwards the request directly to the origin Web server. Otherwise it checks the local cache, like every Web browser would do, in order to exploit locality and reuse. If a fresh copy of the object is not found in this cache, then Squirrel tries to locate one on another node. To do so, it uses the distributed hash-table and the routing functionalities provided by Pastry. First, the URL of the object is hashed to give a 128-bit object identity (a number called *object-Id*) from a circular list. Then the routing procedure of Pastry forwards the request to the node with the identity (called *node-Id*; this number is assigned randomly by Pastry to a participating node) which is the closest one to *object-Id*. This node then becomes the *home node* for this object. Squirrel then proposes two schemes from this point on: *home-store* and *directory* schemes.

In the home-store scheme, objects are stored both at client caches and at their home nodes. The client cache may either have no copy of the requested object or a stale copy. In the former case the client issues a GET request to its home-node, and it issues a *conditional* GET (cGET) request in the latter case. If the home-node has a fresh copy of an object then it forwards it to the client or it sends a not-modified message to the client depending on which action is appropriate. If the home-node has no copy of the object or has a stale copy in its cache, then it issues a GET or a cGET request, respectively, to the origin server. The origin server then either forwards a cacheable copy of the object or sends a not-modified message to the home-node. Then, the home-node takes the appropriate action with respect to the client (i.e. sends a not-modified message or a copy of the object).

In the directory scheme the home-node for an object maintains a small directory of pointers to nodes that have recently accessed the object. A request for this object is sent randomly to one of these nodes. We will not go deeper into the description of this scheme since from now on we will only focus on the home-store scheme. We do so

mainly because the latter scheme has been shown to be overall more attractive than the directory scheme [IRD02].

In a Squirrel network (a corporate network, a university network, etc.), like in any peer-to-peer system, clients arrive and depart the system at random times. There are two kinds of failures (or departures): abrupt and announced failures. Each failure has a different impact on the performance of Squirrel. An abrupt failure will result in a loss of objects. To see this, assume that node i is the home-node for object O. If node i fails, then a new home-node for object O has to be found by Pastry, as explained above, the next time object O is requested. Assume that the copy of object O was fresh when node i failed and consider the first GET request issued for O after the failure of node i. The GET request is therefore forwarded to the new home-node for object O (say node j). This request will result in a miss if j has no copy of O or if its copy is stale. In this case, the failure of node i will yield a degradation in the performance since node j will have to contact the origin server to get a new copy of object O or a not-modified message, as appropriate. If a node is able to announce its departure and to transfer its content to its immediate neighbors in the node-Id space before leaving Squirrel (announced failure), then no content is lost when the node leaves.

When a node joins Squirrel then it automatically becomes the home node for some objects but does not store those objects yet (see details in [IRD02]). In case a request for one of those objects is issued, then its two neighbors in the node-Id space transfer a copy of the object, if any. Therefore, we can consider that there is no performance degradation in Squirrel due to a node arrival, since the transfer time between two nodes is supposed to be at least one order of magnitude smaller than the transfer time between any given node and the origin server.

2.2.3.3 Other peer-to-peer proposals

We now briefly compare BuddyWeb and the Kelips-based architecture proposed by Linga et al. [LGB03]. In BuddyWeb, routing is based on similarity of interest between peers. The P2P network dynamically reconfigures itself based on periodical information sent by peers to name-lookup servers which contain a representation of their interest (keywords, <title> HTML field of browsed pages...). This system also provides a keyword search functionality.

The Linga proposal [LGB03] is closer to the Squirrel system in the architecture. However, this system is based on the Kelips overlay, in which nodes lie in affinity groups (which are initially arbitrarily computed with a consistent hashing function). Each node keeps a structured view of the network as follows: a node has a complete view of its affinity group (with various information on peers such as topology concerns, trust, round-trip times...) and the name of a contact node in each other group. When a document is added into the local cache of a peer, it is assigned an affinity group (by hashing the document name). The name and location of the document is then advertised to the corresponding affinity group contact node. Each affinity group thus maintains a directory table for each cached object belonging to its own group. When the object is requested again, the request is sent by the client to the contact node of the object's affinity group - or itself if the node's affinity group is the same as the client's. The contact node then looks up the directory table for a valid entry for that object. If such an entry is found, the contact node sends the location of the object to the requesting node, which will in turn directly retrieve the object from that location.

We conclude this section by mentioning Pseudoserving [KG97, KG99]. When first presented in 1997, this proposal was an early peer-to-peer solution for content distribution, in which clients obtain the desired file in exchange from serving it, in turn, to other clients. However, though the proposal is presented as based on caching principle, its management at the Web server side, as well as its file-centric model, make it a file-sharing system rather than a caching system in our CDS classification.

2.3 Related Work on Performance Analysis of Distributed Caching Systems

We now briefly review related work on performance analysis of caching systems. Many early studies purely concern centralized proxy caching but lay the basis for later performance studies of distributed caching schemes. Therefore, we will first review these works.

Most performance studies of Web caching systems are trace-based simulations [Dav99]. For Web proxy caching, Kroeger et al. try to quantify bounds on latency savings due to caching and prefetching techniques [KLM97]. While the results are quite impressive – only 26% latency reduction achievable with caching, although external

latency accounts for 77% of the total latency – we must keep in mind that trace-based conclusions are valid for a traffic pattern which may change as fast as Internet usage, and which may be site dependent. In [FCD+99], the trace-based simulation focuses on low-level details such as cookies, aborted connections and their effect on latency and bandwidth. In [DFKM97], caching is not directly modeled or simulated, but several key factors are estimated in a proxy log analysis. The idea is to extract the main traffic patterns that can strongly impact cache performance: for example, the frequency of reaccess or the rate of change of documents. A very important work on Web caching performance is [BCF+99] in which Breslau et al. exhibit the Zipf-like distribution of Web object popularity, and derive a discrete analytical model of proxy caching that computes the hit ratio for a finite cache or a finite request stream.

Cooperative caching has also been given some attention. Several trace-driven simulations of hierarchical caches [DMF97, CK01a] observe a number of performance factors, including cache size, request rates, and consistency mechanisms. Analytical models have also been derived. In [RSB01], the authors develop a discrete model of hierarchical (without cooperation inside a given level), distributed (ICP cluster at institutional level) and hybrid (hierarchical scheme with ICP cooperation at each level) schemes. They compare these schemes according to three metrics: latency, bandwidth usage and required capacity. Their model is simple and tractable but does not account for object expiration nor cache churn rates (i.e., join and leave rate). In [Ros97], Ross proposes an analytical model designed to compare cache processing overhead and latency for two cache cluster schemes: ICP and hash routing. This study shows that hash routing outperforms ICP in the absence of a hierarchy, because of the complete replication of objects among ICP caches and because of their numerous signaling messages. The model does not take into account document expiration nor the caches churn rates. Finally, in [WVS+99], Wolman et al. propose a double performance analysis of cooperative caching systems: they first investigate potential benefits of cooperative caching through a trace-driven simulation. While the authors conclude that cooperative caching is particularly efficient for small populations where a single proxy could suffice, they acknowledge that these conclusions are specific to the Web characteristics of their trace (1 week in 1990 and 1999). Then they propose an analytical model based on Breslau's model [BCF+99] with some enhancements: their model supports cooperative caching (namely, Squid, CARP and Summary Cache), takes into account document rate of change, and focuses on the steady-state instead of finite streams. They use the model to compare the latency reduction and required storage capacity for all three cooperative architectures over various client population sizes.

We observe that few studies have developed an analytical model for performance evaluation of distributed cache systems, and in particular, none of them addresses the crucial issue of churn rates. This issue which was introduced by the distributed design of these cooperative schemes, is particularly challenging. Indeed, cache join/leave events occur at a much slower time scale (e.g., once a day) than requests (typically hundreds per second). As a consequence, the state space of discrete models becomes very large which renders classical tools such as Markovian analysis and simulation untractable. In the next section, we propose a novel analytical fluid model for distributed caching, which is designed to reflect the impact of cache nodes joining and leaving the system.

2.4 A General Stochastic Fluid Model

Our generic framework for modeling dynamic distributed cache systems is essentially based on the observation that requests occur at a much faster time scale (typically hundreds per second) than node join/leave events (e.g. once a day or even less frequently). Therefore we can approximate the request process by a fluid flow when considering the system at the slowest time scale. We expect the long-run average performance of the fluid model to be similar to that of the real, discrete-time system, where requests occur with any distribution.

2.4.1 Review of fluid modeling

Beginning with the seminal work of Anick, Mitra and Sondhi in 1982 [AMS82], stochastic fluid models have been successfully applied to a variety of packet-switching systems over the past 20 years (e.g., see [EM92, EM93, KM01a, LZTK97, BBLO00, RRR02]). In these papers, detailed models of system behavior, which involve random arrivals of packets of discrete size to network nodes, are replaced by macroscopic models that substitute fluid flows for packet streams. The rates of the fluid flows are typically modulated by a stochastic process (such as a Markov process), thereby resulting in a "stochastic fluid model". Although the resulting stochastic fluid models ignore the detailed, packet-level interactions, they are often mathematically tractable and accurate, and provide significant insight into the qualitative properties of the original system.

In the past years, fluid approximations have also been used for efficient simulation

of networks [KM01b, LFG+01, KSCK96]. Discrete-event fluid simulations of packet networks replace the packet flows by fluid streams, smoothing the cell-level behavior at buffers for instance to a piecewise-linear occupancy [KM01b]. [KSCK96] proposes a Markov-modulated fluid simulation of ATM networks. In [LFG+01], Liu et al. compare the performance of fluid simulation and packet-level simulation. They show that the event-rate gain of using fluid simulation is not systematic and depends on a mechanism called ripple effect, which in turn depends on the network scenario and on the source sending rates.

Work on TCP modeling also frequently uses fluid models regarding the window size evolution. In particular, our fluid approach was partly inspired by [AAB00], which uses linear fluid models for the window size, modulated by a stationary ergodic loss process.

2.4.2 General framework

We now introduce an original fluid model for distributed caching. Our model assumes a single level of caching, and therefore is best suited for cache clusters and peer-to-peer cooperative caching schemes. Two important assumptions are required for this model. First, we assume good load balancing between cache nodes. Second, we assume that a cached document is present at only one node of the system, i.e., the distributed caching system does not replicate documents across the participating nodes. In particular, ICP based cache clusters do not satisfy this assumption because they duplicate already cached documents. However, these assumptions are verified by a number of caching systems, such as hash routing schemes, CRISP, or Squirrel.

2.4.2.1 Modeling the node dynamics

We first address the macroscopic event of the model, i.e., the node join/leave events. These events may be due, for instance, to host failures (e.g., software crash), software updates, or user disconnection in the case of peer-to-peer schemes.

We assume that nodes go up and down independently of each other. We denote by $N(t)$ the number of active nodes at time t. The model assumes that participating nodes follow a general birth-death process $N(t)$. We respectively denote by λ_i and μ_i

the birth and death rates of $N(t)$ when i nodes are active, i.e., when $N(t) = i$. The sequence of jump times of this process is denoted by T_n, $n \geq 1$.

Let us denote by N^∞ the stationary number of participating nodes. The stationary distribution of $N(t)$ is $\mathbb{P}[N^\infty = i]$. We also introduce $N_n \stackrel{\text{def}}{=} N(T_n^+)$, the stationary number of participating nodes just *after* the occurrence of the n-th event/jump (i.e. join or leave of a node). The stationary distribution of N_n will be denoted by $\pi_i \stackrel{\text{def}}{=} \lim_{n\to\infty} \mathbb{P}[N_n = i]$, $i \geq 0$. Note that a priori $\pi_i \neq \mathbb{P}[N^\infty = i]$.

2.4.2.2 Modeling the document dynamics

We replace the discrete set of cached objects with fluid. Specifically, let x_j denote the number of objects currently stored in node j. Between up/down events we suppose that x_j grows at a continuous rate. This growth corresponds to a request directed to node j and not being immediately satisfied. Node j then retrieves the object from the origin server and stores a copy, causing x_j to increase. This growth is slowed down by object expirations, which can also be modeled as a fluid flowing out of the system.

We further simplify the fluid model by supposing that the caching protocol balances the load across all of the up nodes in the cluster. As a result, the amount of fluid at each node is an equal share of the total fluid in the system, thereby allowing us to summarize this distributed state by a single variable $X(t)$: the *global* amount of fluid in the system. With this simplified description, the state becomes $(N(t), X(t))$, Also, c denotes the total amount of fluid in the universe (i.e., the total number of documents in the universe). Therefore we have, for all t, $0 \leq X(t) \leq c$. Similarly to N_n, we define X_n as the total amount of cached fluid just *after* the occurrence of the n-th event/jump of the node process $\{N(t)\}$.

This quantity of fluid will increase when objects are downloaded in the network from the origin server and added to their home node, i.e. whenever there is a cache miss. It may happen that two concurrent requests for the same document will generate two misses but only one cached copy. This event is assumed rare enough to be neglected. We validate this claim in the experimental section of the three following chapters. Cache misses occur at a rate proportional to the global request rate $\sigma(t)$ seen by the caching system at time t, and to the miss probability.

On the other hand, the amount of fluid decreases as cached objects become stale. We assume that cached objects have the same constant time-to-live in cache, given by $1/\theta$. This assumption is made both for the sake of simplicity and because most caches use a time-to-live calculation heuristic for objects without any specified expiration date (about 70% of requested objects [CK01a]), which is generally subject to a default maximum value. The usual default value is 24 hours (see [CK01a] for more details).

We now make an additional assumption regarding cache storage capacity. We assume that each node can store an unlimited number of objects. Indeed, disk storage capacity is abundant for most caching systems, and capacity misses are very rare as compared to freshness misses. In peer-to-peer systems, though individual nodes would probably not dedicate too much memory to the collaborative cache, even reasonable cache sizes are sufficient to avoid losses due to a full cache. One reason for this is that cached objects become stale fast enough to avoid continuous increase of the content. For centralized caches, the largest size needed to avoid most capacity misses is dictated by the clients request rates [DMF97] and is fairly small.

2.4.2.3 Characterizing the evolution of fluid

Let us now describe the evolution of the fluid in the model introduced in Sections 2.4.2.1 and 2.4.2.2.

We have already observed that between two consecutive jumps, the amount of fluid grows (continuously) with miss events, and decreases as cached copies expire. At jump times on the other hand, a node is added or removed from the cooperative system. When a node leaves the system, its content may be lost for the global system. This results in a brutal loss of fluid. Similarly, when a node joins the system, it may become suddenly responsible for caching a fraction of objects (such as in DHT schemes: CARP, Squirrel...), while these objects may be already cached in another node which was formerly responsible for them. If the new node joins with an empty cache, this may also result in a decrease of fluid: even if these objects are still physically present in one of the nodes, the fact that this node will no more be requested for these objects results in an apparent decrease of the total available fluid. Let us now translate this behavior into a mathematical model.

For the sake of generality we introduce two mappings, $\Delta_u(i)$ and $\Delta_d(i)$, that give

the fluid *reduction* generated by a node up and down event, respectively, given that i nodes were connected before this event. In other words, if the amount of fluid is x and that i nodes are connected before a leave (resp. join) then the amount of fluid just after this event will be $x\Delta_d(i)$ (resp. $x\Delta_u(i)$). Note that it is theoretically possible at this stage that $\Delta_u(i)$ and $\Delta_d(i)$ exceed unity, which would mean jump events might actually *add* fluid into the system. This will be discussed for each specific application to which we apply our model.

Between two consecutive jumps, fluid increases continuously, provided that at least one node is active. When $N_n = 0$ (all nodes are inactive in (T_n, T_{n+1})), then the amount of fluid remains constant and equal to zero in this time-interval, namely $X(t) = 0$ for $T_n < t < T_{n+1}$ when $N_n = 0$. In particular, the hit probability is equal to zero during such a time-interval. Let us denote $\mathbb{P}[\text{hit}|i, x]$ as the steady-state hit probability when there are i connected nodes containing the fluid x. (We shall indicate how $\mathbb{P}[\text{hit}|i, x]$ is modeled shortly.) The content increases whenever there is a miss event. Therefore, a natural model for the rate at which the fluid increases in the system between up/down events is $\sigma(t)[1-\mathbb{P}[\text{hit}|i, x]]$. However, we have seen that the content decreases at the constant rate θ due to object expirations. Then the variation rate of the amount of fluid is

$$\frac{dx}{dt} = \sigma(t)\left(1 - \mathbb{P}[\text{hit}|i, x]\right) - \theta x \tag{2.1}$$

The resulting fluid process $\{X(t)\}$ is therefore a piecewise-continuous process.

We now define an appropriate model for the hit probability function $\mathbb{P}[\text{hit}|i, x]$. Recall that c is the total number of objects that can ever be requested (i.e., the total amount of existing fluid in the universe). Since x is the quantity of cached fluid, a very simple model for the hit probability is

$$\mathbb{P}[\text{hit}|i, x] = \frac{x}{c} \tag{2.2}$$

However, this linear function does not take into account the fact that some objects may be requested more often than others and thus are more likely to be present in the network. Since the popularity of Web objects follows a Zipf-like distribution [BCF$^+$99], we can also model $\mathbb{P}[\text{hit}|i, x]$ as a concave function of the type

$$\mathbb{P}[\text{hit}|i, x] = \left(\frac{x}{c}\right)^{\beta} \tag{2.3}$$

which reflects the fact that:

– When the amount of fluid is low, popular documents are quickly retrieved, resulting in a fast increase of the fluid.

– When most popular objects are present in the system, the fluid can then only increase with requests for rare objects.

For easy reference, the main definitions and notation have been collected in Table 2.1.

Table 2.1: Notation

$N(t)$	Number of active nodes at time t
T_n	Jump times of the $\{N(t)\}$ process
$N_n = N(T_n^+)$	Number of active nodes just after the n-th jump
$X(t)$	Total amount of cached fluid at time t
$X_n = X(T_n^+)$	Total amount of fluid just after the n-th jump.
λ_i	Birth rate of the node process when $N(t) = i$
μ_i	Death rate of the node process when $N(t) = i$
π	Stationary distribution of $\{N_n\}_n$
$\sigma(t)$	Total request rate at time t
θ	Expiration rate of cached objects
c	Total number of objects in the universe (i.e. total amount of fluid)
$\Delta_d(i)$	Fluid reduction after a node departure when there were $i \geq 1$ connected nodes.
$\Delta_u(i)$	Fluid reduction after a node join when there were $i \geq 0$ connected nodes.
$\mathbb{P}[\text{hit}\|i, x]$	hit probability when $N(t) = i$ and $X_t = x$

2.5 Conclusion

In this chapter, we have reviewed the major approaches for cooperative caching system. We classified them into 3 categories: cache clusters, multi-level hierarchies, and peer-to-peer schemes. We have also reviewed existing work on performance analysis of such systems, and found that most performance studies rely on trace-driven simulation -

which generally restrict the applicability of conclusions to a given traffic profile that may evolve rapidly. More importantly, performance studies of distributed caching systems do not estimate the impact of nodes joining and leaving the system (churn rates). We propose a general stochastic fluid framework for modeling single-level cooperative caching systems that takes into account churn rates. Our fluid model will be used in the next two chapters to analyze the performance of two different caching systems.

Chapter 3

Application to Cache Clusters

3.1 Introduction

In this chapter, we specialize the model introduced in Chapter 2 to analyze the performance of cache clusters. We consider a hash routing scheme such as CARP (cf. Section 2.2.1.2) and show how to compute the expected hit probability in the presence of cache dynamics.

Section 3.2 shows how to specialize our generic framework to model cache clusters. Section 3.3 provides the principal contributions of the chapter. We describe the evolution of fluid and show that the hit probability can be easily obtained from a tridiagonal linear system of dimension N where N is the number of caches in the cluster. We provide explicit, closed-form expressions for $N = 2$ in Section 3.4, which provide insight into performance issues of cache clusters. Our analysis shows that two key systems parameters largely determine the performance of the system. We also use the results of the stochastic fluid model to compare two natural direction policies, namely, "partitioning" and "winning". In Section 3.5 we compare the theoretical results from our fluid model with a discrete-event simulation of a CARP based cache cluster. We find that the fluid model is largely accurate and has the same qualitative behavior as

the detailed model.

3.2 Specializing the Model to Cache Clusters

First of all, note that hash routing satisfies the main assumptions of our generic model. First, the hash function generally provides the load balancing of requests among active nodes. Second, each cached object is available in at most one cache of the cluster. Indeed, even if at some time a node joins the cluster and becomes responsible for a number of objects that are already stored elsewhere, these objects are no longer seen by clients at their former locations. Therefore, once they have been requested again after the node join event, and retrieved from the new node, they are considered to be only present at this new node. Therefore, our fluid model only takes into account *effectively available* documents, which are unique, instead of those actually stored in all of the node caches, which may include useless duplicates.

We now refine our fluid model to fit the behaviour of cache clusters. Let N denote the maximum size of the cluster, i.e., the total number of caches, including inactive ones. We assume that nodes go up and down independently of each other, and that the time until a given up (respectively down) node goes down (respectively up) is exponentially distributed with rate μ (respectively λ). The resulting process $N(t) \in \{0, 1, \ldots, N\}$ is a particular birth-death process, known in the literature as the *Engset* (or Ehrenfest) model. Setting

$$\rho \stackrel{\text{def}}{=} \frac{\lambda}{\mu} \tag{3.1}$$

we have [Kel79, p. 17]

$$\mathbb{P}[N^\infty = i] = \binom{N}{i} \frac{\rho^i}{(1+\rho)^N}. \tag{3.2}$$

In particular, the expected number of caches which are up in steady-state is

$$\mathbb{E}\left[N^\infty\right] = \frac{N\rho}{1+\rho}. \tag{3.3}$$

Recall that $\pi_i = \lim_{n\uparrow\infty} \mathbb{P}[N_n = i]$ is the steady-state probability that there are i active

caches just after a jump. We show in appendix A.1 that

$$\pi_0 = \frac{1}{2(1+\rho)^{N-1}} \tag{3.4}$$

$$\pi_i = \frac{i+\rho(N-i)}{2i(1+\rho)^{N-1}} \binom{N-1}{i-1} \rho^{i-1}, \quad 1 \leq i \leq N. \tag{3.5}$$

We now characterize the fluid dynamics of the system. We assume that the cache cluster handles a global, constant request flow with rate σ. We also assume a linear hit probability model (2.2). Our goal is to determine the steady-state hit probability of the system. Let us denote by p_H this probability.

It remains to determine $\Delta_d(i)$ and $\Delta_u(i)$, the performance degradation factors that affect the amount of fluid when a node leaves or joins the system. As discussed in Section 2.2.1.2, for partition hashing it is natural to define $\Delta_d(i) = 1/2$ for $i = 1, \ldots, N$ and $\Delta_u(i) = 1/2$ for $i = 0, \ldots, N-1$. For winning hashing, it is natural to define $\Delta_d(i) = (i-1)/i$ when $i > 0$ and $\Delta_u(i) = i/(i+1)$ for $i < N$. In the next section we will determine the hit probability for general $\Delta_d(i)$ and $\Delta_u(i)$, and use this to compare partition hashing with winning hashing.

We summarize the newly introduced parameters as well as affected values in Table 3.1.

Table 3.1: Parameters for Cache Clusters

N	Maximum number of active nodes
λ	Birth rate of each node
μ	Death rate of each node
ρ	λ/μ
σ	Total request rate seen by the cluster
$\Delta_d(i)$	$(i-1)/i$ for winning hashing 1/2 for partition hashing
$\Delta_u(i)$	$i/(i+1)$ for winning hashing 1/2 for partition hashing
$\mathbb{P}[\text{hit}\|i,x]$	x/c
p_H	stationary hit probability of the cache cluster

3.3 Hit Probability Analysis

In this section we compute the hit probability associated with the fluid model. Using the specific model detailed in Section 3.2, the fluid arrival process described by (2.1) is now defined by:

$$\frac{d}{dt}X(t) = \sigma\left(1 - \frac{X(t)}{c}\right) - \theta X(t) = \sigma - \left(\frac{\sigma}{c} + \theta\right)X(t) \tag{3.6}$$

for $T_n < t < T_{n+1}$ and $N_n \in \{1, 2, \ldots, N\}$.

Let us now introduce two parameters that will play a role in understanding the system behavior.

$$\alpha \stackrel{\text{def}}{=} \frac{\theta c}{\sigma} \quad \text{and} \quad \gamma \stackrel{\text{def}}{=} \frac{\sigma}{\mu c}\,. \tag{3.7}$$

For the sake of convenience we also introduce

$$\eta \stackrel{\text{def}}{=} \frac{c}{1+\alpha}. \tag{3.8}$$

Integrating (3.6) gives

$$X(t) = \eta + (X_n - \eta)\, e^{-(t-T_n)\sigma/\eta} \tag{3.9}$$

for $T_n < t < T_{n+1}$ provided that $N_n \in \{1, 2, \ldots, N\}$. Clearly, if $N_n = 0$ then $X(t) = 0$ for $T_n < t < T_{n+1}$. At time T_n a jump occurs in the process $\{X(t)\}$ as described in Section 2.4.2.3. Note that from (3.9), $X(t)$ satisfies $0 \le X(t) < \eta$ for all $t > 0$ as long as $0 \le X_0 < \eta$.

If T_n corresponds to a node join or leave event then the amount of cached fluid is reduced respectively as follows

$$\text{join event: } X_n = \Delta_u(N_n)X(T_n-) \tag{3.10}$$

$$\text{leave event: } X_n = \Delta_d(N_n)X(T_n-) \tag{3.11}$$

Therefore, $\{X(t)\}_t$ is a piecewise (exponential) process, with randomness at jump times $\{T_n\}_n$. A sample path of the process $\{(N(t), X(t))\}_t$ is represented on Figure 3.1.

From now on we will assume without loss of generality that $N_0 = 0$ and $X_0 = 0$. Under the aforementioned assumptions $\{(N(t), X(t))\}$ is an irreducible Markov process

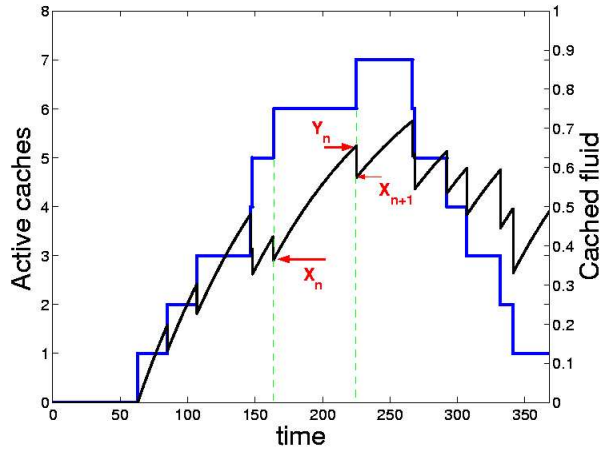

Figure 3.1: Sample path of $\{(N(t), X(t))\}$ for cache clusters.

on the set $\{0,0\} \cup \{\{1,2,\ldots,N\} \times [0,\eta)\}$. Let us denote by X^∞ the stationary regime of $X(t)$.

Our objective in this section is to compute the hit probability p_H, defined as

$$p_H = \frac{\mathbb{E}\left[X^\infty\right]}{c} \tag{3.12}$$

Proposition 3.3.1 below gives an expression for p_H. (Note that $\mathbf{v}^T$ denotes the transpose vector of the vector $\mathbf{v}$.)

Proposition 3.3.1 *Assuming that*

$$0 \leq \Delta_u(i)\Delta_d(i+1) \leq 1, \quad \textit{for } i = 0,1,\ldots,N-1, \tag{3.13}$$

the hit probability p_H is given by

$$p_H = \frac{1}{(1+\alpha)(1+\rho)^N} \sum_{i=1}^{N} \binom{N}{i} \rho^i \, v_i \tag{3.14}$$

where the vector $\mathbf{v} = (v_1,\ldots,v_N)^T$ is the unique solution of the linear equation

$$A\,\mathbf{v} = \mathbf{b} \tag{3.15}$$

with $\mathbf{b} = (b_1,\ldots,b_N)^T$ a vector whose components are given by $b_i = \gamma(1+\alpha)$ for i=1,2,...,N, and $A = [a_{i,j}]_{1\leq i,j\leq N}$ a $N \times N$ tridiagonal matrix whose non-zero elements

are

$$a_{i,i} = \gamma(1+\alpha) + i + \rho(N-i), \qquad 1 \le i \le N \tag{3.16a}$$

$$a_{i,i-1} = -i\Delta_u(i-1), \qquad 2 \le i \le N \tag{3.16b}$$

$$a_{i,i+1} = -\rho(N-i)\Delta_d(i+1), \qquad 1 \le i \le N-1. \tag{3.16c}$$

◇

Proof. The idea of the proof is to first compute the expected amount of cached fluid just before a jump in the process $\{N(t)\}$ conditioned on the value of $N(t)$ just before this jump, and then to invoke Palm calculus to deduce the expected amount of cached fluid at *any time*. Let Y_n be the amount of correctly cached fluid just before the $(n+1)$-th event, i.e.,

$$Y_n = X_{T_{n+1}^-} \tag{3.17}$$

The quantities X_n and Y_n are illustrated on Figure 3.1.

We first compute $\mathbb{E}\left[Y_n | N_n = i\right]$ for $1 \le i \le N$. With (3.9) we have

$$\begin{aligned} \mathbb{E}\left[Y_n \,|\, N_n = i\right] &= \mathbb{E}\left[\eta + (X_n - \eta)\, e^{-(T_{n+1}-T_n)\sigma/\eta} \,|\, N_n = i\right] && (3.18) \\ &= \eta\, \frac{\gamma(1+\alpha) + (\rho(N-i)+i)\,\eta^{-1}\,\mathbb{E}\left[X_n \,|\, N_n = i\right]}{\rho(N-i) + i + \gamma(1+\alpha)} && (3.19) \end{aligned}$$

To derive (3.19) we have used the fact that, given $N_n = i$, the random variables X_n and $T_{n+1} - T_n$ are independent, and $T_{n+1} - T_n$ is exponentially distributed with parameter $(N-i)\lambda + \mu i$.

Let us now evaluate $\mathbb{E}[X_n \,|\, N_n = i]$. We define

$$v_i \stackrel{\text{def}}{=} \frac{\lim_{n\to\infty} \mathbb{E}\left[Y_n \,|\, N_n = i\right]}{\eta} \tag{3.20}$$

Conditioning on N_{n-1} and using (3.5) and the definition of v_i, we have

$$\begin{aligned}
&\lim_{n\uparrow\infty} \mathbb{E}[X_n|N_n = i] = \\
&\qquad \lim_{n\uparrow\infty} \mathbb{E}\left[X_n|N_n = i, N_{n-1} = i-1\right] P\left(N_{n-1} = i-1|N_n = i\right) \\
&\qquad + \lim_{n\uparrow\infty} \mathbb{E}\left[X_n|N_n = i, N_{n-1} = i+1\right] P\left(N_{n-1} = i+1|N_n = i\right) \mathbb{1}_{[i<N]} \quad (3.21) \\
&= \Delta_u(i-1) \lim_{n\uparrow\infty} \mathbb{E}[Y_{n-1}|N_{n-1} = i-1] \frac{\pi_{i-1}}{\pi_i} \frac{\rho(N-i+1)}{\rho(N-i+1)+i-1} \\
&\qquad + \Delta_d(i+1) \lim_{n\uparrow\infty} \mathbb{E}[Y_{n-1}|N_{n-1} = i+1] \frac{\pi_{i+1}}{\pi_i} \frac{i+1}{\rho(N-i-1)+i+1} \mathbb{1}_{[i<N]} \quad (3.22) \\
&= \eta \frac{\Delta_u(i-1)v_{i-1}\, i + \Delta_d(i+1)v_{i+1}\, \rho(N-i)}{\rho(N-i)+i} \quad (3.23)
\end{aligned}$$

Finally, introducing (3.23) into (3.19) yields

$$\big(\rho(N-i)+i+\gamma(1+\alpha)\big)\, v_i = \gamma(1+\alpha) + i\Delta_u(i-1)v_{i-1} + \rho(N-i)\Delta_d(i+1)v_{i+1} \quad (3.24)$$

for $i = 1, 2, \ldots, N$, or equivalently (3.15) in matrix form with $\mathbf{v} = (v_1, \ldots, v_N)$. The uniqueness of the solution of (3.15) is shown in Appendix B.1 using assumption (3.13).

The vector $\mathbf{v}$ in (3.15) gives the expected conditional amount of fluid just before jump epochs (up to a multiplicative constant) in stationary state. However, the hit probability p_H in (3.12) is defined in terms of the stationary expected amount of fluid correctly cached at an *arbitrary* epoch. The latter metric can be deduced from the former one by using Palm calculus, through the identity (see e.g. [BB94, Formula (4.3.2)])

$$\mathbb{E}\left[X^{\infty}\right] = \Lambda \mathbb{E}^0 \left[\int_0^{T_1} X(t)dt\right] \quad (3.25)$$

where $\mathbb{E}^0$ denotes the expectation with respect to the Palm distribution, i.e. assuming that a jump occurs at time 0 and that the system is in steady-state at time 0, T_1 denotes the time of the first jump after 0, and Λ denotes the global rate of the Engset model, i.e.

$$\Lambda = \frac{1}{\mathbb{E}^0[T_1]}. \quad (3.26)$$

From now on we assume that the system is in steady-state at time 0. Under the Palm distribution we denote by N_{-1} and Y_{-1} the number of up caches and the amount of correctly cached fluid respectively, just before time 0 (i.e. just before the jump that occurs at time 0).

We first compute $1/\Lambda$. using (3.4)-(3.5) we have

$$\frac{1}{\Lambda} = \sum_{i=0}^{N} \pi_i \, \mathbb{E}^0[T_1 \,|\, N_0 = i] = \frac{1}{\mu} \sum_{i=0}^{N} \frac{\pi_i}{\rho(N-i)+i} = \frac{1+\rho}{2N\rho\mu} \tag{3.27}$$

Let us now determine $\mathbb{E}\,[X^\infty]$. From (3.25), (3.9), (3.27) we find

$$\mathbb{E}\,[X^\infty] = \Lambda \sum_{i=1}^{N} \pi_i \, \mathbb{E}^0 \left[\int_0^{T_1} \left(\eta + (X_0 - \eta)\, e^{-t\sigma/\eta} \right) dt \,|\, N_0 = i \right] \tag{3.28}$$

$$= \Lambda\eta \left[\sum_{i=1}^{N} \pi_i \mathbb{E}^0[T_1 | N_0 = i] + \frac{1}{\sigma} \sum_{i=1}^{N} \pi_i \mathbb{E}^0 \left[(X_0 - \eta) \left(1 - e^{-T_1\sigma/\eta} \right) | N_0 = i \right] \right] \tag{3.29}$$

$$= \Lambda\eta \Bigg[\mathbb{E}^0[T_1] - \pi_0 \mathbb{E}^0[T_0 \,|\, N_0 = 0] + \frac{1}{\sigma} \sum_{i=1}^{N} \pi_i \left(\mathbb{E}^0[X_0 \,|\, N_0 = i] - \eta \right) \times \left(1 - \mathbb{E}^0 \left[e^{-T_1\sigma/\eta} \,|\, N_0 = i \right] \right) \Bigg] \tag{3.30}$$

$$= \Lambda\eta \left[\frac{1}{\Lambda} - \frac{1}{2N\rho\mu\,(1+\rho)^{N-1}} + \frac{1}{\mu} \sum_{i=1}^{N} \pi_i \, \frac{\eta^{-1}\mathbb{E}^0[X_0 \,|\, N_0 = i] - 1}{\rho(N-i) + i + \gamma(1+\alpha)} \right] \tag{3.31}$$

$$= \frac{c}{1+\alpha} \left[1 - \frac{1}{(1+\rho)^N} + \frac{2N\rho}{(1+\rho)} \sum_{i=1}^{N} \pi_i \, \frac{\eta^{-1}\mathbb{E}^0[X_0 \,|\, N_0 = i] - 1}{\rho(N-i) + i + \gamma(1+\alpha)} \right] \tag{3.32}$$

By definition, $\mathbb{E}^0[X_0 \,|\, N_0 = i] = \lim_{n\uparrow\infty} \mathbb{E}[X_n \,|\, N_n = i]$, which has been computed in (3.23). By combining (3.23) and (3.24) we obtain

$$\mathbb{E}^0[X_0 \,|\, N_0 = i] = \eta \, \frac{(\rho\,(N-i) + i + \gamma(1+\alpha))v_i - \gamma(1+\alpha)}{\rho(N-i) + i} \tag{3.33}$$

Plugging this value of $\mathbb{E}^0[X_0 \,|\, N_0 = i]$ into the right hand side of (3.32), and using (3.5), yields after some straightforward algebra

$$\mathbb{E}\,[X^\infty] = \frac{c}{1+\alpha} \left[1 - \frac{1}{(1+\rho)^N} + \frac{N\rho}{(1+\rho)^N} \sum_{i=1}^{N} \binom{N-1}{i-1} \frac{\rho^{i-1}}{i} \, (v_i - 1) \right] \tag{3.34}$$

$$= \frac{c}{(1+\alpha)(1+\rho)^N} \sum_{i=1}^{N} \binom{N}{i} \rho^i v_i \tag{3.35}$$

According to (3.12) it remains to divide both sides of (3.35) by c to get (3.14). This concludes the proof. ■

The set of conditions (3.13) in Proposition 3.3.1 ensure that the system (3.15) has a unique solution (see proof in Appendix B.1). They are satisfied for both winning hashing (since $\Delta_u(i)\Delta_d(i+1) = (i/(i+1))^2$ for $i < N$) and for partition hashing (since $\Delta_u(i)\Delta_d(i+1) = 1/4$ for all $i < N$) schemes (see Section 3.2).

Remark 3.3.1 *Since A is a tridiagonal matrix, (3.15) can be solved in only $\mathcal{O}(N)$ operations, once the mappings Δ_u and Δ_d are specified.*

3.4 Application

In this section we use Proposition 4.1 to analyze cache clusters. First, we show how the result provides qualitative insight on the hit probability. We then use the result to compare the hit probabilities of partition hashing and winning hashing.

3.4.1 Qualitative behavior

For small N, we can compute the hit probability in closed-form. We do this now for winning hashing. For $N = 2$ we have

$$p_H = 2\gamma \frac{\rho}{(1+\rho)^2} \frac{2\gamma\alpha + \rho\gamma\alpha + 2\gamma + \rho\gamma + \rho^2 + 4 + 3\rho}{2\gamma^2 + 4\gamma^2\alpha + 6\gamma + 2\gamma^2\alpha^2 + 6\gamma\alpha + 4 + 2\rho\gamma + 2\rho\gamma\alpha + 3\rho} \tag{3.36}$$

We observe that the hit probability only depends on the parameters α and γ, defined in Section 3.3 (see (3.7)), and ρ. This result actually holds for any value of N since a glance at Proposition 3.3.1 indicates that the components of A and $\mathbf{b}$ depend on the model parameters only through α, ρ and γ. Interestingly enough, the fact that the hit probability for a given rate of change depends on the parameters σ and c only through the ratio σ/c was observed in [WVS+99] in a slightly different context. This is an indication that our fluid model is able to capture some of the main features of the caching system.

Figure 3.2 shows how the hit probability depends on γ, ρ and α for small clusters (i.e. when N is small). The concave shapes on Figure 3.2(a) shows that increasing γ (through σ, for instance) can offer a large performance gain in the smaller range,

in this case when $\gamma \leq 20$. This can be related to empirical observations in [CI97, GB97, WVS+99]: for small client population sizes, the authors found that the hit probability increases in a log-like[1] fashion of the population size. Our model exhibits similar shapes, although the hit probability is a rational function of γ rather than a logarithmic function of γ. Moreover, it explains analytically these properties, and also includes the caches dynamic behavior through μ in the γ definition.

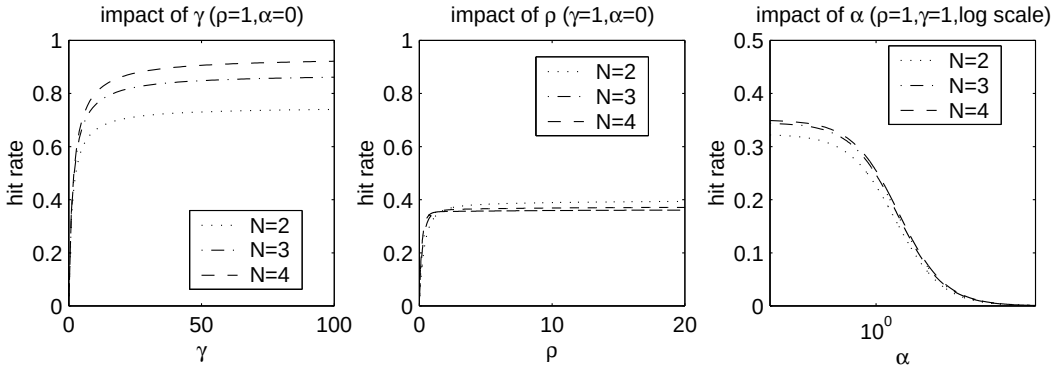

Figure 3.2: Impact of ρ, γ and α on the hit probability for small clusters.

Parameter ρ is fairly new in cache cluster analysis. It typically represents one aspect of the dynamic behavior of the system, as apparent in (3.2) and (3.3).

Figure 3.2(b) represents the hit probability as function of ρ. The hit probability converges rapidly to its maximum value ($\approx 40\%$ in Figure 3.2b) as ρ increases. The curves flatten to become almost constant for high values of ρ; the larger N the quicker the curve flattens. Therefore, except for very small values, in which case the hit probability drops very quickly, ρ has very little influence on the hit probability. This can be easily explained. Indeed, we see from (3.2) that with probability $1/(1+\rho)^N$, all caches are down. Therefore, all caches are down with very high probability when ρ is small, yielding a very low hit probability, as shown in Figure 3.2(b). On the other hand, when ρ is large there is always at least one cache up with a high probability which prevents the hit probability from dropping to zero. Under these circumstances, the limiting factor for the hit probability will be the removal of documents in caches, modeled by parameters γ (as described above) and α.

Figure 3.2(c) shows how α impacts the hit probability for $\rho = 1$ (which is large

[1]The hit probability is either a logarithm or a small power of the population size.

enough to avoid long periods of total unavailability since in this case 50% of the caches are up on the average as shown in (3.3)) and $\gamma = 1$. The curve is obviously decreasing since α is proportional to the rate of change (or expiration) of cached documents. The highest hit probability is therefore obtained with $\alpha = 0$. Also observe that the hit probability drops significantly as α increases.

From Figure 3.2 we infer that the key parameters of the system are γ and α, which almost determine the hit probability as long as ρ is not too close to zero. This can be explained by the fact that for high values of ρ, γ and α capture the main interactions between object population, request rate, document rate of change and cache dynamics — which correspond to document losses and misplacements in the cluster.

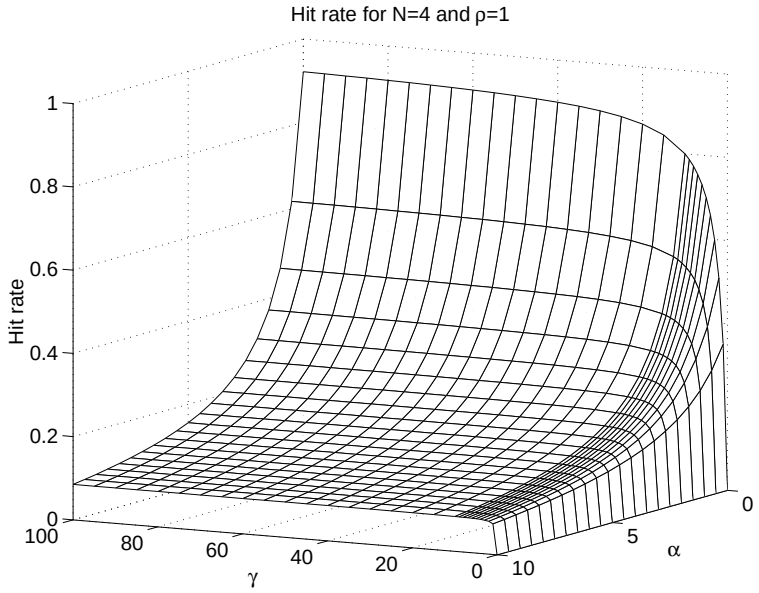

Figure 3.3: p_H as a function of γ and α for $\rho = 1$

Since γ and α are the two only limiting factors for realistic systems (where $P(N_\infty = 0) \approx 0$), we may want to compare their influence on the system. In Figure 3.3 we observe that the domain where the hit probability is high (above 40%) is very small ($\alpha \leq 1$, $\gamma \geq 10$). In fact, γ has a real impact on the hit probability when $\alpha \leq 1$. The concave shape observed in Figure 3.2(a) for $\alpha = 0$ is still present for positive values of α but it is less and less pronounced as α increases. This can be explained analytically from the fact that $X(t) \leq \eta$ for all $t > 0$ (provided that $X_0 < \eta$) as already observed in Section 3.3, which implies that p_H is bounded from above by $1/(1 + \alpha)$.

3.4.2 Comparison of partition hashing and winning hashing

Figures 3.4 and 3.5 compare the hit probability for partition hashing with that of winning hashing when $\alpha = 0$, i.e. when documents do not expire. The performance difference is obvious, especially for small γ and $\rho > 1$: for any set of parameters, winning hashing always exhibits a much higher hit probability than does partition hashing. For instance, at $\rho = 50$ and $\gamma = 1$ (see Figure 3.5), the hit probability for winning hashing is 36%, which is 50% higher than that for partition hashing, i.e., 24%.

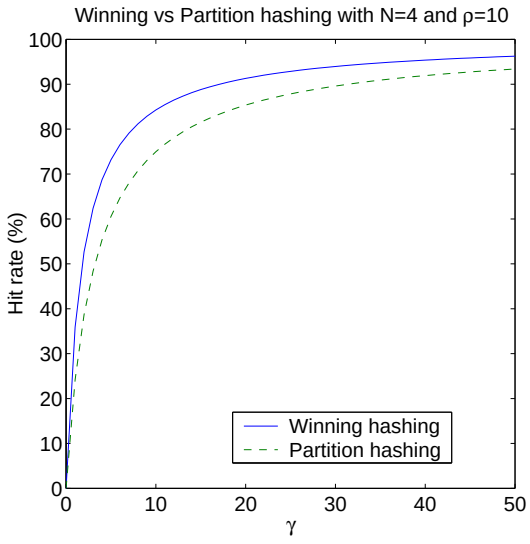

Figure 3.4: Comparison of winning hashing and partition hashing for $N = 4$, $\alpha = 0$ and $\rho = 1$

3.5 Experimental Validation

In this section, we compare quantitatively our macroscopic fluid model with a discrete-event driven simulation of the cache cluster for $N = 10$ caches. Throughout this section we use winning hashing. The CARP hash function [VR97] is implemented in the simulator while the corresponding values of Δ_u and Δ_d are used in the fluid model. The simulation uses the Engset model for cache dynamics and a Poisson process for request arrivals. Object TTLs in caches are assumed to be constant and identical for all objects. The simulation also assumes caches are cleared upon failure. The simulator implements

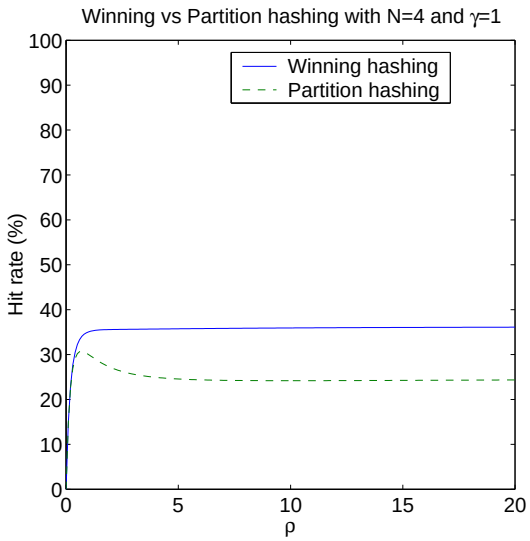

Figure 3.5: Comparison of winning hashing and partition hashing for $N = 4$, $\alpha = 0$ and $\gamma = 1$

the detailed behavior of cache clusters, including concurrent requests (almost simultaneous requests for the same object that will count for two misses if the object is not in cache due to external latency). Experimental results are given with 99% confidence intervals.

The fluid model estimation is computed using Proposition 3.3.1. We begin with a validation of parameter γ. Table 3.2 shows results for various values of the (σ, μ, c) triple with a constant ratio $\gamma = 2$, and $\rho = 1$. Of course, the fluid model provides identical values, i.e., 50.9%, for all experiments (see Section 3.4). We observe that the discrete-event simulation is almost insensitive to variations in the number of documents, request arrival rate, or failure rate when γ is constant: even when they vary by several orders of magnitude, the hit probability remains between 50% and 52%, which is close to the fluid model value. This validates our finding that the system is characterized by parameters ρ and γ. Of course, when σ becomes of the same order of magnitude as μ, which is highly unlikely to happen in real systems, discrete-event simulation does not see enough requests to create reliable statistics.

We now consider the impact of γ on the hit probability. Figure 3.6 displays the hit probability as a function of γ with $\rho = 1$, for two different values of α. We observe that the predictions made by the fluid model agree well with those made by discrete-

Table 3.2: Hit probability (%) for $\gamma = 2$ and $\rho = 1$.

c	2000			20,000		
σ	0.2	2	20	0.2	2	20
μ	5×10^{-5}	5×10^{-4}	5×10^{-3}	5×10^{-6}	5×10^{-5}	5×10^{-4}
Simulation	50.0±1.5	50.8±1.9	51.8±1.7	51.0±1.2	50.3±2.4	50.4±1.9
Fluid Model	50.9	50.9	50.9	50.9	50.9	50.9

event simulation, and therefore mimics the discrete system behaviour very accurately. An important feature appearing in Figure 3.6 is the range of p_H when $\alpha = 0$ and ρ is not too small: p_H increases with γ from zero to almost 1. Although this observation is not true for very small values of N, as shown in Figure 3.2, the upper bound of the hit probability seems to increase with N and is already very close to 1 for $N = 10$. Therefore, for small values of α and $\rho \geq 1$, it is possible to reach almost any desired hit probability by increasing γ accordingly. This validates our finding that γ determines the hit probability of the system when $\alpha = 0$. Also, the curves comparing our fluid model to discrete-event simulation when $\alpha = 1$ clearly show how this second parameter limits the hit probability even for large values of γ, which is rather intuitive. Indeed, α represents the time needed for the system to cache all existing documents (filling time) divided by the time-to-live of the cached documents, while γ is the ratio of the average lifetime of a cache and this filling time. It is clear that if the document modification rate is high with regard to the filling time, fewer documents will become misplaced upon failure events.

Finally, we examine the influence of ρ on the hit probability. Figure 3.7 shows that both the fluid model and the simulation exhibit a steep slope for small values of ρ and an almost flat shape for $\rho \geq 1$. This validates the fact that ρ has very little influence on the hit probability except when it is close to zero.

We conclude that the fluid model provides an accurate approximation for the actual hit probability of the discrete system and more importantly, highlights the key parameters and properties of the system. Furthermore, we also would like to emphasize the computational gain of our fluid model compared to simulation. The simulation C code, though probably not fully optimized, typically runs for several hours on a 2GHz Pentium 4 with 768MB of RAM, even for small clusters as simulated in this section. In comparison, our Maple implementation of Proposition 3.3.1 produces the hit probability almost instantaneously (in less than a second).

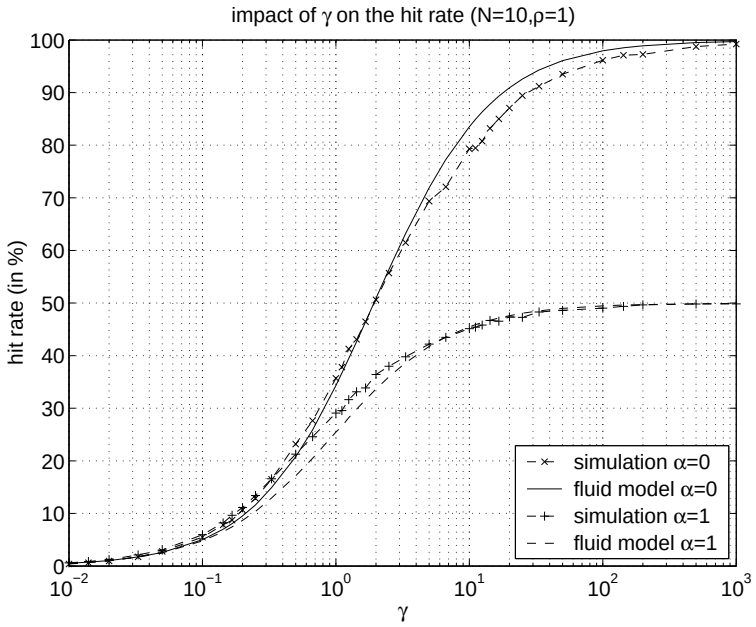

Figure 3.6: Fluid model vs simulation: impact of γ (with $N = 10$ and $\rho = 1$).

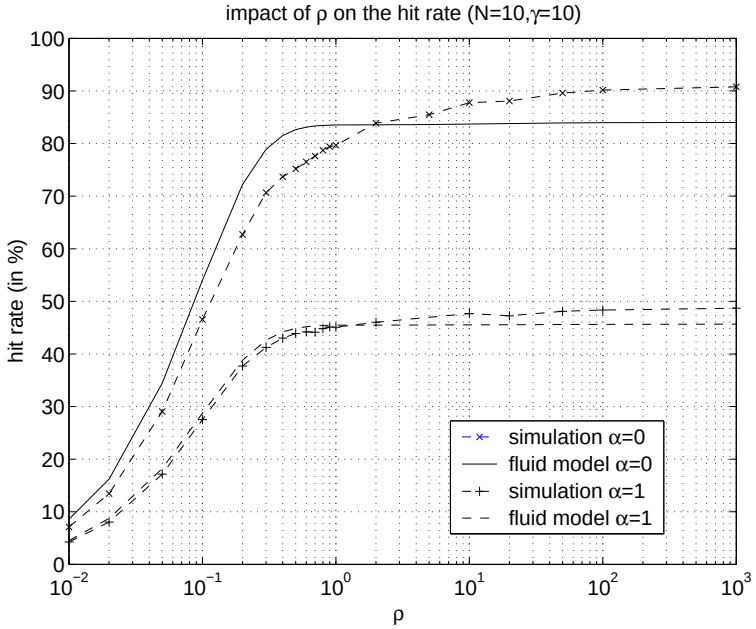

Figure 3.7: Fluid model vs simulation: impact of ρ (with $N = 10$ and $\gamma = 10$).

3.6 Finite Capacity Case

In this section we briefly explore the case where each cache has a limited storage capacity. Let us assume the amount of fluid in each cache cannot exceed a given constant B. More specifically, we assume that $X(t) \leq BN(t)$ for all $t \geq 0$. In this setting the time-evolution of $X(t)$ between two consecutive jump times of the process $\{N(t),\, t \geq 0\}$ is given by

$$X(t) = \min\left(BN_n,\, \eta + (X_n - \eta)\, e^{-(t-T_n)\sigma/\eta}\right) \tag{3.37}$$

for $T_n < t < T_{n+1}$. When $B = \infty$ then the previous equation turns into (3.9).

Unfortunately, when B is finite the computation of $\mathbb{E}\left[Y_n | N_n = i\right]$ introduces a non-linearity due to the minimum operator in (3.37). Therefore, unlike the case when $B = \infty$, it is not possible to find a closed-form expression for the hit probability p_H. An alternative approach to computing p_H is to use a hybrid equation-based/discrete-event simulator that uses (3.37). This can be done as follows. First, run a discrete-event driven simulation of the process $\{(N_n, T_n),\, n \geq 1\}$. Then, use $\{(N_n, T_n),\, n \geq 1\}$ in (3.37) to evaluate $\mathbb{E}[X^{\infty}]$. We expect this solution to be much more time-efficient than a classical discrete-event driven simulation of the entire system since the hybrid approach will only have to simulate events (up/down events) on a slow time-scale. This method is discussed below.

Figure 3.8 compares the results obtained with the equation-based simulator that uses (3.37) with that of a discrete-event driven simulator as a function of the average storage capacity $N_{\text{mean}}B$ for $\gamma = \rho = 1$ and $\alpha = 0$, where $N_{\text{mean}} = N\rho/(1+\rho)$ is the mean number of active caches (see (3.3)).

We observe from Figure 3.8 that when $\alpha = 0$, the hit probability no longer increases when the average storage capacity exceeds a threshold around $1.5c$, where c is the total number of documents (see Table 2.1). This indicates that increasing buffer capacity beyond a certain value does not improve the cache performance, limited by other factors such as cache dynamics. This phenomenon is even more obvious when $\alpha > 0$, because object expirations happen faster than cache filling, and justifies the infinite capacity assumption used in the generic model of Section 2.4.2.

Figure 3.8 shows that the equation-based simulator results not only exhibits the

same shape as the discrete system hit probability, but also provides an accurate numerical approximation. This strengthens the conclusion that the fluid model is able to capture the main features of the discrete system.

Moreover, the equation-based simulator is a much faster tool than the discrete-event driven simulation of the system, especially for large values of request rates. In addition to an obvious efficiency gain, it provides higher accuracy by allowing the simulation of a much larger number of up/down events, thereby approaching more closely the stationary state. Also, the equation-based simulation method can easily be extended to other equations than (3.37), for instance to take into account document popularity as discussed in Section 2.4.2.3.

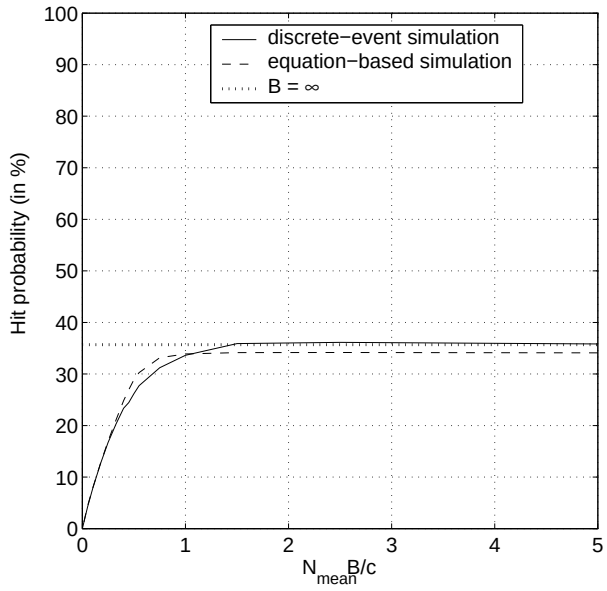

Figure 3.8: Impact of cache size B on the hit probability when $\alpha = 0$.

3.7 Conclusion

In this chapter we have considered a complex caching system consisting of multiple nodes that randomly go up and down and which store new objects arriving randomly from origin servers. The system exhibits randomness on two time scales: object arrivals on a fast time scale, and cache up/down events on a slower time scale. To analyze this complex system, we have approximated the system with a stochastic fluid model using

the framework introduced in Chapter 2, which, though non-trivial, turns out to be mathematically tractable. Comparison with discrete-event simulation has shown that the hit probability provided by the solution to the model is a accurate approximation of the actual hit probability. Also, the solution highlights the key characteristics of the actual system.

Chapter 4

Performance of the Squirrel P2P Caching System

4.1 Introduction

In this chapter we use our stochastic fluid model to investigate the performance of a peer-to-peer caching system, namely, Squirrel (cf. Section 2.2.3.2).

The fluid model specifics for Squirrel are introduced in Section 4.2. We use the resulting model in Section 4.3 to compute the main performance metrics of Squirrel: hit probability and average latency. In particular, we provide a simple expression for the hit probability. We show in Section 4.4 that our model provides substantial insight into performance issues of P2P cooperative Web caches such as Squirrel. Our analysis shows that two key parameters largely determine the performance of the system, especially the ratio of the document expiration rate to the per-document and per-node request rate. In Section 4.5 we compare results obtained with the fluid model to those obtained from a discrete-event simulation of Squirrel. We find that the fluid model is both qualitatively and quantitatively accurate. We conclude in Section 4.6 with possible extensions of our fluid model.

Note that, since the analysis is based on the same model and method as introduced in Chapter 3, the structure of the present chapter will globally resemble that of Chapter 3. However, note that because the Squirrel architecture is very different from CARP, all equations are different and require a specific analysis.

4.2 Specific Model

We first check that the Squirrel system satisfies the main assumptions of our generic model. First, the Pastry substrate (request routing protocol used by Squirrel - see Section 2.2.3.2) provides sufficient load balancing among nodes so that we can assume that the total fluid in the system is equally divided among the nodes. Second, as we did when we analyzed cache clusters (cf. Section 3.2) we consider the fluid to be equal to the number of available documents which are in a unique node in the home-store scheme. This means that we neglect the presence of possible duplicates which may appear when a node goes up and becomes home node for a few objects that are not necessarily removed immediately from their previous home node. Indeed these duplicates do not affect the performance of the system, except when a request hits the client local cache before being sent to the home node. This event is assumed to be rare compared to the global Squirrel system hit events, and is therefore neglected. We validate this claim experimentally in Section 4.5.

The assumption that each node can store an unlimited number of objects has already been explained in Section 2.4.2.1 by the fact that even moderate individual cache sizes are enough to avoid capacity misses. This claim is supported by several trace-based simulations in [IRD02] which show that with an individual storage capacity of 100MB a Squirrel network can achieve a performance similar to that of a sufficiently large centralized cache.

We now specialize our generic fluid model to capture the behavior of a Squirrel network. Similar to the cache cluster case, we assume the behavior of nodes in the network to follow an Engset model of maximum size N, with birth rate λ and death rate μ. We also use $\rho = \lambda/\mu$. The stationary distribution and expectation of this process are already given in Section 3.2 by equations (3.2) and (3.3). We also recall that the stationary distribution of the number of nodes just after a jump is given by π defined in (3.4)-(3.5).

Unlike the cache cluster case in Chapter 3, the request rate now depends on the number of active nodes $N(t)$ since nodes are now both clients and servers. As a result, throughout this chapter we assume a constant request rate σ *for each client*, which gives

$$\sigma(t) = \sigma N(t) \tag{4.1}$$

The hit probability model is again chosen to be the linear function (2.2), i.e. we assume that all objects are all equally popular. Although somewhat unrealistic, this assumption leads to a clear analysis and highlights the effect of different parameters on the system performance. We show how this assumption can be relaxed in Chapter 5. We also denote by p_H the stationary hit probability of the Squirrel system.

The model is complete once we define of $\Delta_u(i)$ and $\Delta_d(i)$. We have seen in Section 2.2.3.2 that join events probably do not affect the performance of the system. On the other hand, we consider all failures (leaves) to be abrupt failures; this assumption is discussed in Section 4.3.3. Therefore, when a node leaves, its share of objects is lost to the system. If we assume that the requests are well balanced across all nodes of the network (a property of the Pastry hashing technique), then a fraction $1/i$ of the total amount of fluid is lost when a leave occurs when i nodes are connected prior to this leave event. This value has been confirmed empirically in [IRD02]. As a result we have:

$$\Delta_u(i) = 1 \text{ and } \Delta_d(i) = (i-1)/i.$$

A glossary of the Squirrel specific parameters is provided in Table 4.1 which can be compared to Table 3.1: note that the definition of σ and the value of $\Delta_u(i)$ differ significantly from those in Chapter 3.

4.3 Analysis

In this section we provide a simple closed-form expression for the hit probability of the Squirrel system. The end-to-end latency reduction offered by the Squirrel system, which might be a more meaningful metric than the hit probability, can easily be derived from the following results as shown in Section 4.3.2. Finally, we discuss the possible sources of inaccuracy of this model in Section 4.3.3 and try to identify remedies where possible.

Table 4.1: System Parameters

N	Maximum number of nodes
λ	Birth rate of each Squirrel node
μ	Death rate of each Squirrel node
ρ	λ/μ
π	Stationary distribution of $\{N_n\}$
σ	Request rate *per client*
$\Delta_d(i)$	$(i-1)/i$
$\Delta_u(i)$	1
$\mathbb{P}[\text{hit}\|i,x]$	x/c
p_H	stationary hit probability of the Squirrel network

4.3.1 Hit probability analysis

Our first task is to characterize the fluid process $\{X(t)\}$. The fluid process is defined as follows (see Section 4.2): between two consecutive jumps (T_n, T_{n+1}) of $\{N(t)\}$ the fluid increases at rate

$$\frac{d}{dt}X(t) = \sigma N_n \left(1 - \frac{X(t)}{c}\right) - \theta X(t) \tag{4.2}$$

provided that $N_n > 0$. Note that unlike the cache cluster case in Chapter 3, the evolution of the fluid between two jumps now depends on N_n. Integrating (4.2) gives

$$X(t) = \frac{\sigma N_n}{\frac{\sigma N_n}{c} + \theta} + \left(X_n - \frac{\sigma N_n}{\frac{\sigma N_n}{c} + \theta}\right) e^{-(t-T_n)(\theta + \frac{\sigma N_n}{c})} \tag{4.3}$$

for $T_n \le t < T_{n+1}$ provided that $N_n > 0$. If $N_n = 0$ then $X(t) = 0$ for $T_n \le t < T_{n+1}$.

We now reuse some parameters introduced in Chapter 3. While the analytic expressions of these parameters do not change, note that the new definition of parameter σ (individual request rate instead of global request rate) changes the physical meaning of these parameters:

$$\alpha = \frac{\theta c}{\sigma} \quad \text{and} \quad \gamma = \frac{\sigma}{\mu c} \tag{4.4}$$

We now introduce a new parameter η_i, which is analog to η (see Section 3.3) that now depends on the number of active nodes:

$$\eta_i \stackrel{\text{def}}{=} \frac{c}{1 + \frac{\theta c}{i\sigma}}, \quad 1 \le i \le N. \tag{4.5}$$

We can now re-write the solution of (4.2) as

$$X(t) = \eta_{N_n} + (X_n - \eta_{N_n})\, e^{-(t-T_n)\frac{\sigma N_n}{\eta_{N_n}}},\ T_n \leq t < T_{n+1} \tag{4.6}$$

Similar to Chapter 3, by definition if T_n corresponds to a node leave or join event then the amount of cached fluid changes by $X_n = \Delta_d(N_n)X(T_n-)$ and $X_n = \Delta_u(N_n)X(T_n-)$ respectively. Therefore, $\{X(t)\}$ is again a piecewise (exponential) process, with randomness at jump times $\{T_n\}$, but with different evolution parameters than in Chapter 3. A sample path of the process $\{(N(t), X(t))\}$ is represented on Figure 4.1.

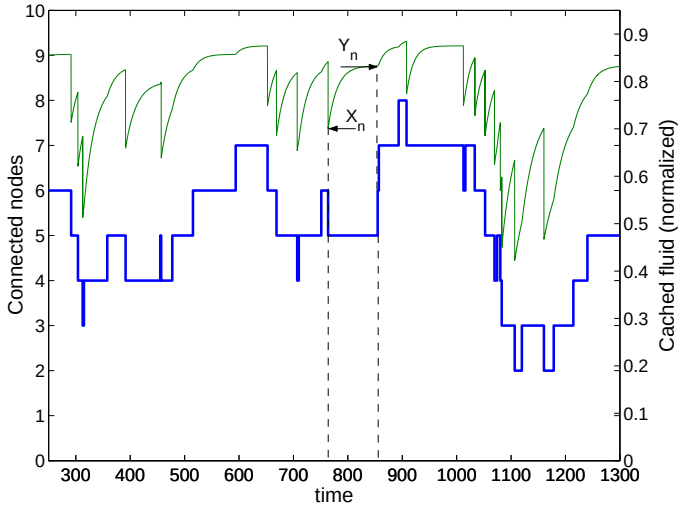

Figure 4.1: Sample path of $\{(N(t), X(t))\}$.

As in the previous chapter, the process $\{(N(t), X(t))\}$ is an irreducible Markov process on the set $\{0, 0\} \cup \{\{1, 2, \ldots, N\} \times [0, c)\}$ and we denote by X^∞ the stationary regime of $\{X(t)\}$.

Recall that under the assumption that all objects are equally popular, the steady-state hit probability p_H is defined as

$$p_H = \frac{\mathbb{E}\left[X^\infty\right]}{c} \tag{4.7}$$

We give a simple formula for the Squirrel value of p_H in Proposition 4.3.1.

Proposition 4.3.1 *Assuming that for* $i=0,...,N-1$,

$$0 \leq \Delta_u(i)\Delta_d(i+1) \leq 1, \tag{4.8}$$

the hit probability p_H *is given by*

$$p_H = \frac{1}{(1+\rho)^N} \sum_{i=1}^{N} \binom{N}{i} \rho^i \, v_i \tag{4.9}$$

where the vector $\mathbf{v} = (v_1, \ldots, v_N)^T$ *is the unique solution of the linear equation*

$$A\,\mathbf{v} = \mathbf{b} \tag{4.10}$$

with $\mathbf{b} = (b_1, \ldots, b_N)^T$ *a vector whose components are given by* $b_i = \gamma i$ *for* $1 \leq i \leq N$, *and* $A = [a_{i,j}]_{1 \leq i,j \leq N}$ *a* $N \times N$ *tridiagonal matrix whose non-zero elements are*

$$a_{i,i} = \alpha\gamma + (\gamma+1)i + \rho(N-i), \qquad 1 \leq i \leq N \tag{4.11a}$$

$$a_{i,i-1} = -i\Delta_u(i-1), \qquad 2 \leq i \leq N \tag{4.11b}$$

$$a_{i,i+1} = -\rho(N-i)\Delta_d(i+1), \qquad 1 \leq i \leq N-1. \tag{4.11c}$$

◇

Proof. As with Proposition 3.3.1, we first compute the expected amount of cached fluid just before a jump in the process $\{N(t)\}$ and then use Palm calculus to deduce the expected amount of cached fluid at any time. Therefore, we also use Y_n as the amount of cached fluid just before the $(n+1)$-th jump in the process $\{N(t)\}$:

$$Y_n = X_{T_{n+1}^-} \tag{4.12}$$

We first compute a new parameter v_i which is slightly different from the v_i defined in Chapter 3:

$$v_i \stackrel{\text{def}}{=} \lim_{n\to\infty} (1/c)\mathbb{E}\left[Y_n \,|\, N_n = i\right] \tag{4.13}$$

for $1 \leq i \leq N$. The vector $\mathbf{v} = (v_1, \ldots, v_N)^T$ also gives the conditional stationary expected amount of cached fluid just before jump epochs, up to a multiplicative constant that is now simply the total number of existing documents. We show in Appendix C that v_i satisfies the following recursive equation:

$$v_i\left(\rho(N-i) + \alpha\gamma + (\gamma+1)i\right) = i\Delta_u(i-1)v_{i-1} + \rho(N-i)\Delta_d(i+1)v_{i+1} + i\gamma \tag{4.14}$$

for $i = 1, 2, \ldots, N$, or equivalently (4.10) in matrix form with $\mathbf{v} = (v_1, \ldots, v_N)$. The uniqueness of the solution of (4.10) is shown in Appendix B.2.

We now compute the hit probability p_H in (4.7) in terms of the stationary expected amount of cached fluid at arbitrary epochs. To this end we use Palm calculus as in Chapter 3, with

$$\mathbb{E}\left[X^{\infty}\right] = \Lambda \mathbb{E}^0 \left[\int_0^{T_1} X(t)\, dt\right] \tag{4.15}$$

where we recall that $\mathbb{E}^0$ denotes the expectation with respect to the Palm distribution (the Palm distribution is the distribution of the process $\{X(t)\}$ assuming that a jump occurs at time 0 and that the system is in steady-state at time 0), T_1 denotes the time of the first jump after 0, and that Λ denotes the global rate of the Engset model, i.e.

$$\Lambda = \frac{1}{\mathbb{E}^0[T_1]} \tag{4.16}$$

From now on we assume that the system is in steady-state at time 0. Under the Palm distribution we denote by N_{-1} and Y_{-1} the number of connected nodes and the amount of cached fluid respectively, just before time 0 as in Chapter 3.

Since Λ is the global rate of the Engset model its expression is given by (3.27). We then show in Appendix D that

$$\mathbb{E}\left[X^{\infty}\right] = \frac{c}{(1+\rho)^N} \sum_{i=1}^{N} \binom{N}{i} \rho^i v_i \tag{4.17}$$

Dividing both sides of (4.17) by c, we get (4.9), which concludes the proof. ■

The set of conditions (4.8) in Proposition 4.3.1 ensures that the system (4.10) has a unique solution (see Appendix B.2). It is satisfied for the home store scheme since $\Delta_u(i)\Delta_d(i+1) = i/(i+1)$.

Remark 4.3.1 *A tridiagonal $N \times N$ linear system can be solved in only $\mathcal{O}(N)$ operations. We see from (4.11) that matrix A is tridiagonal, so that (4.10) can be solved in $\mathcal{O}(N)$ operations once the mappings Δ_u and Δ_d are specified.*

4.3.2 Latency reduction

In this section we show how to estimate the latency seen by clients based on the hit probability given by Proposition 4.3.1.

The latency can be divided into two quantities, the external latency T_e and the internal latency T_i, which are defined as follows. The external latency represents the average delay between a proxy server of a corporate network, and the originating Web server. Therefore, this latency is only seen in case of a cache miss. This external latency is caused by network bottlenecks and Web server delays outside the organization. The internal latency is intrinsic to the local network and can be for instance the average delay between a proxy cache and a client, or in the case of Squirrel, the average latency induced by the network between two randomly chosen clients. The internal latency has to be small since it is experienced by clients even in case of a cache (home node) hit. Typically, the external latency T_e accounts for most of the total retrieval delay in the absence of caching (e.g. 77%, and up to 88% for a geographically located network [KLM97]).

The expected delay to fetch a document can easily be derived from the hit probability as follows. The total expected delay T with Squirrel is

$$\mathbb{E}[T] = T_i\, p_H + (T_i + T_e)\,(1 - p_H) \tag{4.18}$$

The Squirrel cache system reduces the average delay by saving the external latency whenever there is a hit. The relative latency reduction observed with Squirrel is thus

$$\frac{T_i + T_e - \mathbb{E}[T]}{T_i + T_e} = p_H \frac{T_e}{T_i + T_e} \tag{4.19}$$

4.3.3 Discussion and extensions

We now discuss some specific features that were not explicitly taken into account in the analysis of Section 4.3.1, apart from the popularity of documents.

The first remark is that the model assumes that every requested object is saved in the cooperative cache when downloaded a first time from the origin server. However, a non-negligible fraction (around 28%, cf. [CK01a]) of the requested objects is in

practice non-cacheable (mainly, expiration date before current date, but also explicitly non-cacheable). We can take into account the uncacheability in our model as follows: let u be the fraction of objects that are uncacheable. So far, we have considered that the fluid increases after each miss, thereby implicitly assuming that all objects are cacheable. The uncacheability can be incorporated in our model by considering that only a fraction $1-u$ of misses will yield a fluid increase. This gives rise to the following equation

$$\frac{d}{dt}X(t) = (1-u)\,\sigma N_n \left(1 - \frac{X(t)}{c}\right) - \theta X(t) \tag{4.20}$$

$$= (1-u)\,\sigma N_n - \left(\frac{(1-u)\sigma N_n}{c} + \theta\right) X(t) \tag{4.21}$$

for $T_n < t < T_{n+1}$ and $N_n \in \{1, 2, \ldots, N\}$, since only requests for cacheable objects will lead to a fluid increase. Therefore, uncacheable objects can be added to the model simply by modifying the request rate accordingly.

Second, the impact of node join and leave events, modeled through the mappings Δ_u and Δ_d, may differ slightly from the values described in Section 4.2. Indeed, in the two following cases we need to re-estimate these factors. Though Proposition 4.3.1 provides an expression for general values of $\Delta_d(i)$, we need to ensure that condition (4.8) is still satisfied in both cases:

- Some nodes may announce their intention to disconnect, thereby avoiding a performance degradation (see Section 2.2.3.2). This requires a change to $\Delta_d(i)$, which may reach unity if all nodes are able to announce their departures. If $\Delta_d(i) = 1$ and if $\Delta_u(i) < 1$ is unchanged, then (4.8) is still satisfied.
- The individual Squirrel caches may be stored either on disk or in memory. In the first case, the local cache may not be erased when a node i goes down or disconnects. When node i goes back up, it may therefore join the system with a set of previously stored documents. This can possibly add fluid into the network, if the three following conditions are satisfied simultaneously: node i has not announced its last departure, the corresponding objects have not been retrieved by the system while i was down, and node i is still home node for these documents. If this happens, the problem is not only how to re-estimate $\Delta_u(i)$, but also that $\Delta_u(i)$ might be greater than one, making condition (4.8) more difficult to verify. However, we expect that node i will stay down for a minimum time that will be

orders of magnitude greater than request inter-arrival times (the reboot time is typically a few minutes). Meanwhile, most of the objects stored in node i will be requested again and added to their new home nodes. As a result, when node i goes back up it will probably not add any fluid in the system, thereby ensuring $\Delta_u(i) \leq 1$ and the validity of (4.8).

Finally, formula (4.9) involves binomial coefficients $\binom{N}{i}$ and an exponential in N. Therefore, computing p_H accurately for very large values of N may prove difficult. Nonetheless, we would like first to mention that though we have occasionally encountered such problems, Proposition 4.3.1 is tractable for an order of magnitude of several thousands of nodes, where a simulation would be untractable for high-confidence results. In addition, for much larger values of N, the node dynamics can be approximated by an $M/M/\infty$ model instead of an Engset model. This extension is presented in Chapter 5.

4.4 Qualitative insight in the Squirrel system

Proposition 4.3.1 shows that the performance of the Squirrel system exhibits only four degrees of freedom: N, ρ, γ, and α while our model introduced six parameters: N, λ, μ, σ, θ, and c. We now examine the relative importance of these new parameters and how they characterize the Squirrel system behavior.

We first examine the influence of ρ on the hit probability. Figure 4.2 shows that while there is a sharp drop of the hit probability for very small values of ρ (smaller that one), the performance is almost constant when ρ exceeds one. Therefore, except when it is close to zero, ρ has very little influence on the performance of the Squirrel system. It is unlikely that ρ will be really small, since it would mean that the event that all nodes are down would occur with nonnegligible probability. In this circumstance, the limiting factors for the hit probability will be parameters N, γ and α.

In Figure 4.3 we examine the influence of γ and α on the hit probability. We find that, for fixed α, the hit probability is a concave function of γ, and can reach almost one when $\alpha = 0$. This is consistent with our observation that ρ does not limit the hit probability when it is greater or equal to one. Recall that $\gamma = \sigma/(\mu c)$ where σ is the individual request rate of the nodes. This concave shape in γ reminds us of the

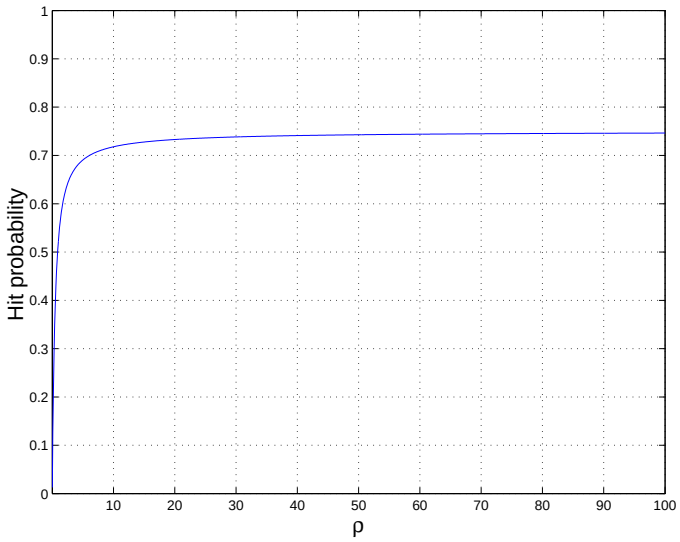

Figure 4.2: Impact of ρ (with $N = 3$, $\alpha = 1$ and $\gamma = 2$).

log-like (i.e., the hit rate is either a logarithm or a small power of the global request rate) performance of a centralized Web cache (or Web cache cluster) as described in [WVS+99, GB97, DMF97].

However, we observe that the hit probability is high only when $\alpha \leq 1$, $\gamma \geq 10$. Indeed, α has a strong impact on the hit probability, hence γ has a significant impact on the performance of the Squirrel system only when α is small.

These observations suggest possible methods to improve the performance of the Squirrel system. The best possible improvement would be to reduce parameter $\alpha = \theta c/\sigma$. Since the total number of existing objects, c, cannot be modified, there are two options:

- Reduce the expiration rate θ as much as possible: increase the default value of the maximum allowed value (denoted by CONF_MAX) in the freshness calculation heuristic for example (see Section 2.2) especially since most cGET requests (e.g. 90%) are responded with Not-Modified message [CK01a]. Another solution can be the refreshment policy proposed by Cohen and Kaplan in [CK01b].
- Increase the request rate σ, for instance by using prefetching techniques. We believe that prefetching can be incorporated into the fluid model, which will allow us to quantify the gain of using it. Intuitively, although increasing the request

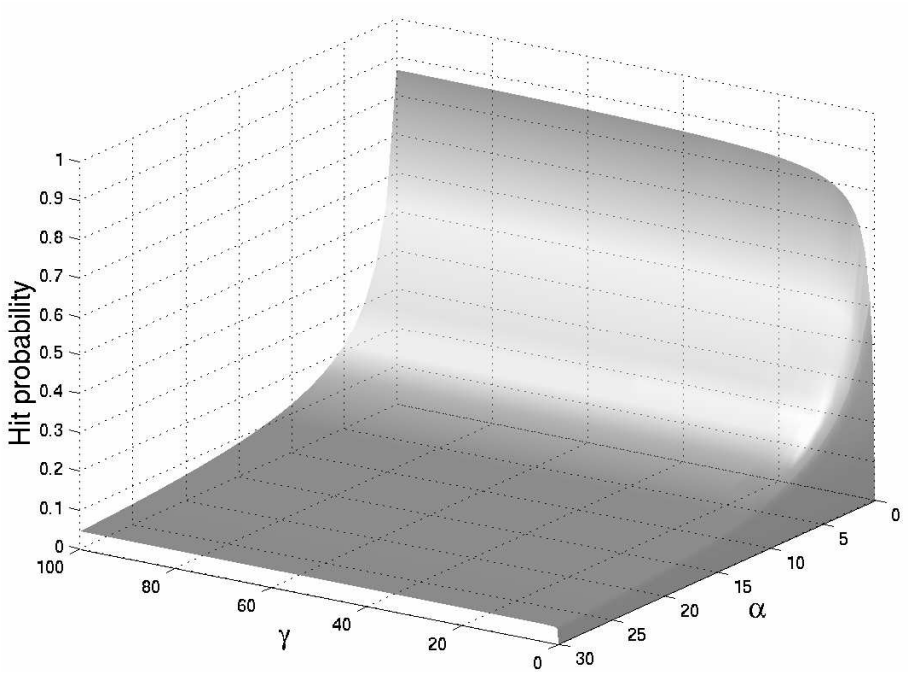

Figure 4.3: Impact of γ and α on the hit probability (with $N = 3$ and $\rho = 1$). (Note that α is decreasing.)

rate will increase the load in the system, it will also increase the rate at which objects are retrieved to the Squirrel network. This phenomenon is already known in the context of centralized caches [DMF97].

Finally, if the global shape of the hit probability does not depend on N, the optimal values of γ and α vary with N. As a result any optimization of the system requires a realistic estimation of the maximum number of nodes in the network.

4.5 Experimental Validation

In this section we compare our macroscopic fluid model with a discrete-event driven simulation of the Squirrel home-store system. Request arrivals are Poisson and object time-to-live are taken to be all constant and all identical. We also assume that nodes follow the same time-evolution as in the fluid model, i.e. an Engset model. The external latency is taken into account whereas the internal latency is considered to be

zero (corresponding to instantaneous internal transfers). Simulation results are given with 99% confidence intervals.

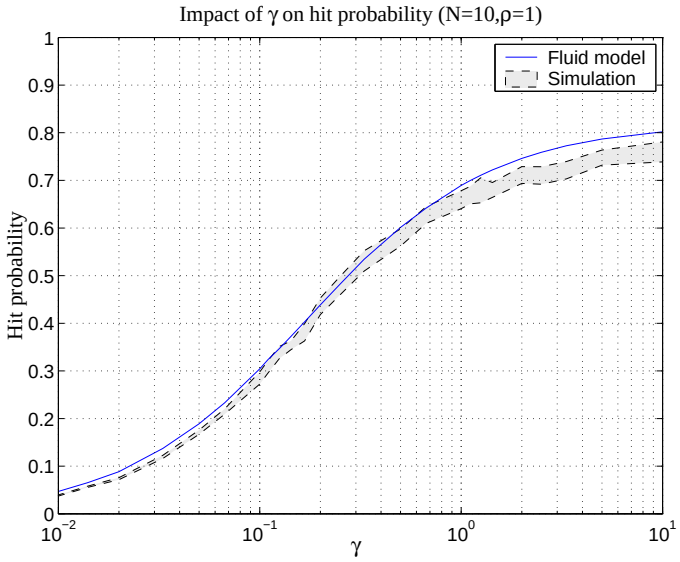

Figure 4.4: Fluid model vs discrete-event simulation. ($N = 10$, $\rho = 1$ and $\alpha = 1$).

Figure 4.4 displays the hit probability as a function of γ with $\rho = 1$ and $\alpha = 1$. We observe that the fluid model curves closely follow the same shapes as the discrete-event simulations and therefore mimics the simulated system behavior very accurately. We conclude that the model is robust to assumptions such as the request rate distribution (which we assumed constant in Section 4.3.1), and although microscopic features such as objects replication and local hits (requests not forwarded to home node) are being ignored, the fluid model provides an accurate approximation for the actual performance of the Squirrel system.

Moreover, as for cache clusters the discrete-event simulation (implemented in C) of the Squirrel system is very slow and limited to very small network sizes. Even with the restrictions mentioned in Section 4.3.3, Proposition 4.3.1 provides an efficient estimation of the Squirrel hit probability up to the order of 10,000 nodes, and even provides an immediate result for smaller network sizes.

We show in Figure 4.5 how the hit probability would look like for large networks, since simulation of such systems would be either too slow or statistically irrelevant. Since Figure 4.4 validated the accuracy of our model for small network sizes, we expect the results for large networks to be as relevant – though we do not have simulation re-

sults to demonstrate it. We observe the same shape as in Figure 4.4, though on a larger

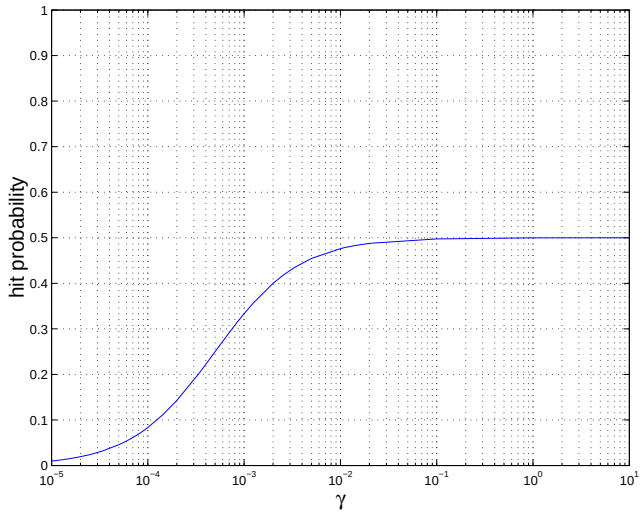

Figure 4.5: Hit probability for large networks ($N = 2000$ and $\alpha = 1000$).

range (thanks to the low complexity of the model), which suggests that Squirrel scales with the same type of behavior, and that the characteristics observed in Section 4.4 should be valid for large networks.

4.6 Conclusion

In this chapter we used our stochastic fluid model to analyze the performance of the Squirrel cooperative cache system. Our resulting stochastic fluid model turns out to be mathematically tractable, and has allowed us to provide a simple and very low-complexity procedure for computing the hit probability. Moreover, the analysis has emphasized the key characteristics of the Squirrel system and allows a better understanding of its performance. Comparison with simulation results has shown that the hit probability provided by the solution to the model is an accurate approximation of the actual hit probability and has validated the qualitative conclusions driven by the model results.

It is worth observing that our analysis is not strictly limited to Squirrel, but can also be applied to other P2P systems based on distributed hash tables such as

Chord, CAN or Tapestry ([SMK+01, RFH+01, ZKJ00]). The necessary conditions are the load balancing (provided by Pastry), and above all the absence of replication that characterizes the home-store scheme.

Future work will focus on extending the model to handle prefetching techniques. We also address larger populations of peers and quantify the accuracy of the approach for the Zipf-like popularity model in the next chapter.

Chapter 5

Extension to Large Networks and Zipf-Like Popularity

5.1 Introduction

In Chapter 4 we developed a model for the quantitative analysis of Squirrel using the stochastic fluid framework introduced in Chapter 2. We derived the hit probability under the assumption that all documents are equally popular. The node dynamics were modeled by a Engset process, a N-state Markov process, where N is the number of nodes in the Squirrel network. As to the request process, we assumed that each active Squirrel node generated requests at a constant rate. We then showed that the total number of available documents in the Squirrel network was accurately modeled by a piecewise deterministic fluid process.

The aim of the present chapter is to extend the analysis of Chapter 4 in two directions. First, we replace the Engset model by an infinite-state Markov process, the M/M/∞ queuing model (see Section 5.2), which yields a dramatic decrease in the complexity of computing the hit probability. Indeed, solving the Engset model requires a computational effort that grows exponentially with N, the size of the network, due

to the presence of binomial coefficients and exponentials in the hit probability formula (4.9). This restricted the performance analysis to the order of 10,000 nodes. Our new M/M/∞ model allows us to easily handle real size networks (e.g., a million nodes for a large corporate network). Second, we relax the assumption made in Chapter 4 that all documents are equally popular, and provide an efficient method for computing the hit probability in realistic situations (i.e. with Zipf-like document popularity distribution). Numerical comparisons with discrete-event simulations validate these extensions.

The rest of the chapter is organized as follows. In Section 5.2 we introduce the new model for the node dynamics and recall the document model used in Chapter 4. We then show in Section 5.3 how to compute the hit probability at a constant cost in the number of nodes under the assumption that all objects are equally popular. The latter assumption is relaxed in Section 5.4, where we incorporate a Zipf-like object popularity distribution in our model, and show how to compute the hit probability in this more general setting. These models are used in Section 5.5 to make a number of qualitative observations on Squirrel performance, related to the impact of unequal document popularity and of announced/unannounced departures (see 5.5.3) on Squirrel performance. Section 5.6 is devoted to the experimental validation of our approach, and concluding remarks are given in Section 5.7.

5.2 A M/M/∞-Modulated Fluid Model

In Chapter 4 we modeled the node dynamics by a finite-state birth and death process, with birth (resp. death) rate $\lambda(N-i)$ (resp. μi) when there are $i = 0, 1, \ldots, N$ nodes up, where parameters N (the number of nodes), λ and μ are given. In the literature this Markov process is referred to as the Engset model. This model has two main problems. First, it requires the existence of a bound on the number of nodes which can simultaneously be active (the parameter N). In general there does not exist such a bound and, if it did, it would be very difficult to determine. Second, the calculation of the hit rate induced by the Engset model poses serious computational complexity issues as N becomes very large. As an illustration, it took more than one day to compute the hit probability (given in Proposition 4.3.1, using a realistic value of $\rho = 100$) on a 2GHz Pentium 4 with 768MB RAM for 10,000 nodes, a relatively small population for a corporate network.

To overcome the shortcomings of using the Engset model (i.e. the need to have a bound on the number of users and the scalability issue), in this chapter we model the node dynamics by a M/M/∞ queuing system [Kle75, p. 101]. In the M/M/∞ setting, nodes become active according to a Poisson process with intensity λ (referred to as the arrival process) and each node remains active for an exponentially distributed amount of time, with mean $1/\mu$. It is a natural model since it assumes nodes join the system at arbitrary times, independently of each other. At the end of its activity period a node disappears, an event which corresponds to a departure in the M/M/∞ queue. Node activity periods are assumed to be mutually independent, and furthermore independent of the arrival process. Therefore, in our stochastic framework introduced in Chapter 2, the M/M/∞ model corresponds to $\lambda_i = \lambda$ and $\mu_i = i\mu$ for $i \in \mathbb{N}$.

Remark 5.2.1 *The M/M/∞ queuing system can be seen as a limit case of the Engset model, in the sense that their steady-state distributions are equivalent when the mean number of nodes goes to infinity (cf. Appendix F).*

We now look at the stationary distribution of this new $\{N(t)\}$ process. We redefine parameter ρ as:

$$\rho \stackrel{\text{def}}{=} \frac{\lambda}{\mu} \tag{5.1}$$

where λ now denotes the total birth rate and no longer the birth rate of an individual node. It is known that N^∞ has a Poisson distribution with parameter ρ [Kle75, p. 101], namely

$$\mathbb{P}[N^\infty = i] = \frac{\rho^i}{i!} e^{-\rho}, \quad i \geq 0. \tag{5.2}$$

In particular, the expected number of active nodes in steady-state is given by

$$\mathbb{E}\left[N^\infty\right] = \rho \tag{5.3}$$

which now gives a very intuitive meaning to parameter ρ. Recall that π_i is the steady-state probability that there are i nodes active just after a jump (see Chapter 2). We show in Appendix A.2 that

$$\pi_0 = \frac{e^{-\rho}}{2} \tag{5.4}$$

$$\pi_i = \frac{i+\rho}{i!}\,\rho^{i-1}\,\frac{e^{-\rho}}{2}, \quad i \geq 1. \tag{5.5}$$

Apart from the M/M/∞ node process, the rest of the model is identical to the model in Chapter 4. We now recall the main parameters of the fluid model. As in Chapter 4 we assume that each active node produces a continuous and deterministic stream of requests with rate σ, so that σN_n is the total request rate in (T_n, T_{n+1}). Hence, in the time-interval (T_n, T_{n+1}), $X(t)$ satisfies the following first-order differential equation

$$\frac{d}{dt}X(t) = \sigma N_n(1 - \mathbb{P}[\text{hit}|N_n, X(t)]) - \theta X(t) \tag{5.6}$$

if $N_n > 0$. In this chapter we will compute the hit probability for the two possible expressions (2.2) and (2.3) of $\mathbb{P}[\text{hit}|N_n, X(t)]$ depending on whether or not documents are equally popular (see Chapter 2).

We now provide the values of $\Delta_u(i)$ and $\Delta_d(i)$ for the Squirrel system. As discussed in Chapter 4 there is no loss of content when a new node joins Squirrel. Also, we assume that a node joining Squirrel does not bring any document with it (see Section 4.3.3). This again gives $\Delta_u(i) = 1$ for $i \in \mathbb{N}$. On the other hand, there is no loss of content if a departure is announced, so that $\Delta_d(i) = 1$ $(i \geq 2)$ when such an event occurs. In the case of an abrupt departure the content of the departing node is totally lost, which was assumed in Chapter 4 then we have $\Delta_d(i) = (i-1)/i$ for $i \geq 1$. In the following we will analyze both the situations where $\Delta_d(i) = 1$ and $\Delta_d(i) = (i-1)/i$, with $\Delta_u(i) = 1$ in both cases.

5.3 Hit Probability: Uniform Popularity Case

In this section we assume that all objects are equally popular, which implies that the probability that a given object o is requested is $1/c$. This assumption is relaxed in Section 5.4, where a more realistic Zipf-like popularity distribution is considered. Under the uniform document popularity assumption, the (conditional) hit probability at time t, $\mathbb{P}[\text{hit}|N_n, X(t)]$, is a simple linear function of $X(t)$, given by (2.2). Therefore, as in Chapter 4 the fluid evolution is given by (4.2) and its solution (4.3) in the interval (T_n, T_{n+1}). We also use the same parameters α and γ as in Chapter 4 (given in (4.4)). A first expression for the hit probability p_H is derived in the following proposition.

Proposition 5.3.1 *The hit probability is given by*

$$p_H = e^{-\rho} \sum_{i=1}^{\infty} \frac{\rho^i}{i!} v_i \tag{5.7}$$

where the constants $v_1, v_2, \ldots$ satisfy the infinite linear recursion

$$(\rho + \alpha\gamma + (\gamma+1)i)\, v_i = \gamma i + i\Delta_u(i-1)v_{i-1} + \rho\Delta_d(i+1)v_{i+1}, \quad i \geq 1, \tag{5.8}$$

with $v_0 = 0$. ◇

Proof. The proof of Proposition 5.3.1 is given in appendix E. It is shown in this proof that v_i is the stationary hit probability just before a *jump epoch* given that i nodes are active.

The expression in (5.7) is not amenable to efficient computation, since it involves the solution of an infinite system of linear equations and the computation of an infinite series. Building on Proposition 5.3.1, the next result provides an alternative expression for the hit probability, which will turn out to be more amenable to numerical computation than (5.7). This is done for the cases (i) $\Delta_d(i) = (i-1)/i$, $\Delta_u(i) = 1$ and (ii) $\Delta_d(i) = \Delta_u(i) = 1$.

Proposition 5.3.2 *Assume that $\Delta_u(i) = 1$ (no loss of content at node arrival).*

If node departures are not announced (i.e. $\Delta_d(i) = (i-1)/i$) then

$$p_H = e^{-\frac{\gamma\rho}{\gamma+1}} \gamma^{-(1+\kappa)} \int_{\frac{1}{\gamma+1}}^{1} \gamma\rho e^{\frac{\gamma\rho t}{\gamma+1}} \left(t(\gamma+1)-1\right)^{\kappa} dt \tag{5.9}$$

where $\kappa \stackrel{\text{def}}{=} \gamma(\alpha(\gamma+1)+\rho)/(\gamma+1)^2$.

If node departures are announced (i.e. $\Delta_d(i) = 1$) then

$$p_H = \rho e^{-\frac{\rho\gamma}{\gamma+1}} \gamma^{-\nu} \int_{\frac{1}{\gamma+1}}^{1} (\gamma t e^{\rho t} - v_1) e^{-\frac{\rho t}{\gamma+1}} \left((\gamma+1)t-1\right)^{\nu-1} dt \tag{5.10}$$

with $v_1 \stackrel{\text{def}}{=} \dfrac{\int_0^{1/(\gamma+1)} \gamma t e^{\frac{\rho\gamma t}{\gamma+1}} (1-(\gamma+1)t)^{\nu-1}\, dt}{\int_0^{1/(\gamma+1)} e^{\frac{\rho t}{\gamma+1}} (1-(\gamma+1)t)^{\nu-1}\, dt}$ and $\nu \stackrel{\text{def}}{=} \frac{\alpha\gamma(\gamma+1)+\rho}{(\gamma+1)^2}$. ◇

Proof. For $0 \leq z \leq 1$ introduce the generating function

$$F(z) = \sum_{i=1}^{\infty} \frac{\rho^i}{i!} v_i z^i. \tag{5.11}$$

Observe that $0 \leq F(z) \leq \exp(\rho z)$ since $0 \leq v_i \leq 1$ for all $i \geq 1$. With (5.11) the hit probability p_H given in (5.7) can be rewritten as

$$p_H = e^{-\rho} F(1) \tag{5.12}$$

It remains to determine $F(1)$. Assume first that $\Delta_d(i) = (i-1)/i$. Multiplying both sides of (5.8) by $\rho^i z^i / i!$ and summing the resulting equation over all values of $i \geq 1$ yields, after easy algebra,

$$\left(\rho + \alpha\gamma - \rho z + \frac{1}{z}\right) F(z) + ((\gamma+1)z - 1)\, \frac{d}{dz} F(z) = \gamma\rho z e^{\rho z}, \quad \text{for } z \in (0,1) \tag{5.13}$$

Equation (5.13) defines an ordinary differential equation for $F(z)$, with the initial condition $F(0) = 0$. Letting $z = 1/(\gamma+1)$ in (5.13) we see that necessarily

$$F\left(\frac{1}{\gamma+1}\right) = \frac{\gamma\, \rho\, e^{\,\rho/(\gamma+1)}}{(\gamma+1)(\gamma+1+\alpha\gamma+\gamma\rho/(\gamma+1))} \tag{5.14}$$

Since we only need to compute $F(1)$ (see (5.12)), it is enough to solve (5.13) for $z \in (1/(\gamma+1), 1]$, with the initial condition (5.14), and then to use the continuity of the function $F(z)$ at point $z = 1/(\gamma+1)$.

We first solve the standard homogeneous equation, then use the method of variation of constant. The homogeneous equation writes

$$\begin{aligned} \frac{d}{dz} F(z) &= \frac{\rho + \alpha\gamma - \rho z + \frac{1}{z}}{1 - (\gamma+1)z} F(z) && (5.15) \\ &= \left[\frac{\rho}{\gamma+1} + \frac{1}{z} - \frac{(\gamma+1)^2 + \alpha\gamma(\gamma+1) + \gamma\rho}{(\gamma+1)\,(z(\gamma+1)-1)}\right] F(z) && (5.16) \end{aligned}$$

Its solution is

$$F(z) = C e^{\frac{\rho z}{\gamma+1}}\, z\, (z(\gamma+1)-1)^{-(1+\kappa)} \tag{5.17}$$

where κ is defined in the statement of the proposition, and where C is an integration constant. Considering C as a function of z, we routinely find from (5.13) and (5.17) that $C = C(z)$ satisfies the equation

$$\frac{d}{dz} C(z) = \gamma\rho e^{\frac{\gamma\rho}{\gamma+1} z}\, (z(\gamma+1)-1)^{\kappa}$$

Solving for $C(z)$ gives

$$C(z) = \int_{\frac{1}{\gamma+1}}^{z} \gamma\rho e^{\frac{\gamma\rho t}{\gamma+1}} \left(t(\gamma+1)-1\right)^{\kappa}\, dt + C_0 \tag{5.18}$$

where C_0 is a constant to be determined from the initial condition (5.14). Since the exponent $-(1+\kappa)$ of $(z(\gamma+1)-1)$ in (5.17) is strictly negative, and since $F(1/(\gamma+1))$ is finite from (5.14), we conclude that necessarily $C(1/\gamma+1))=0$, which implies that the constant C_0 in (5.18) must be equal to zero. Therefore, given (5.17) and (5.18), we get

$$F(z) = e^{\frac{\rho z}{\gamma+1}} z \left(z(\gamma+1)-1\right)^{-(1+\kappa)} \int_{\frac{1}{\gamma+1}}^{z} \gamma\rho e^{\frac{\gamma\rho t}{\gamma+1}} \left(t(\gamma+1)-1\right)^{\kappa}\, dt \tag{5.19}$$

for $z \in (1/(\gamma+1), 1)$. Letting $z \to 1$ in (5.19) and using (5.12) finally gives (5.9).

Assume now that $\Delta_d(i) = 1$. In this case $F(z)$ satisfies the ordinary differential equation

$$(\rho(1-z)+\alpha\gamma)F(z) + ((\gamma+1)z-1)\frac{d}{dz}F(z) = \rho(ze^{\rho z} - v_1), \quad \text{for } z \in (0,1) \tag{5.20}$$

We only sketch the derivation of $F(z)$ as it does not offer any difficulty. The first step is to solve (5.20) separately for $z \in (0, 1/(\gamma+1))$ and for $z \in (1/(\gamma+1), 1)$, with the initial condition $F(0) = 0$ and $F(1/(\gamma+1)) = \rho(e^{-\rho/(\gamma+1)}/(\gamma+1) - v_1)/(\rho+\alpha\gamma)$, respectively (the latter condition is obtained by setting $z = 1/(\gamma+1)$ in (5.20)). The second and last step is to use the continuity of $F(z)$ at point $z = 1/(\gamma+1)$, which gives a linear equation to be satisfied by v_1, from which we find v_1 and ultimately (5.10). This concludes the proof. ■

Proposition 5.3.2 provides a low-complexity formula for the computation of p_H. The only difficulty lies in the evaluation of the various exponentials, especially when ρ is large or equivalently (see (5.3)) when the expected number of active nodes is large. In this case, a good accuracy can be achieved by rewriting p_H in the form $p_H = \int_{1/\gamma+1}^{1} e^{f(t,\rho,\alpha,\kappa)}dt$, where the mapping f can easily be identified from (5.9) (resp. (5.10)). Using this method, the average CPU time needed to compute the hit probability using (5.9) or (5.10) is typically less than a second with an Intel 4 2GHz/768Mo workstation, even for networks as large as a million nodes.

5.4 Hit Probability: Zipf-like Popularity Case

We now relax the assumption that all objects have equal popularity. Following [BCF+99] we assume that the popularity of documents follows a Zipf-like distribution. This implies that the probability ψ_n that the n-th most popular object is requested, is given by

$$\psi_n = \frac{\Omega}{n^\beta} \quad \text{for } n = 1, \ldots, c \tag{5.21}$$

with $0 < \beta \leq 1$, where $\Omega \stackrel{\text{def}}{=} 1/\sum_{i=1}^{c} i^{-\beta}$ is a normalization factor. When $\beta = 1$ then we have the Zipf's law. (For the sake of comparison, note that $\psi_n = 1/c$ under the homogeneous popularity assumption – see analysis in Section 5.3.)

The next step is to replace (2.2) by an expression that takes into account the popularity of the documents. We now use a concave hit probability model as suggested by (2.3) in Chapter 2, or even a more refined model using (5.21). If we assume that the $X(t)$ cached objects at time t are the most popular ones, then using the approximation $\sum_{i=1}^{\lfloor x \rfloor} i^{-\beta} \approx \int_1^x t^{-\beta} dt = (x^{1-\beta} - 1)/(1-\beta)$ for $x \geq 1$, a natural choice for $\mathbb{P}[\text{hit}|N_n, X(t)]$ is (with $\lfloor x \rfloor$ the largest integer less than or equal to c)

$$\mathbb{P}[\text{hit}|N_n, X(t)] = \sum_{i=1}^{\lfloor X(t) \rfloor} \frac{\Omega}{i^\beta} \approx \frac{X(t)^{1-\beta} - 1}{c^{1-\beta} - 1} \tag{5.22}$$

Unfortunately, with this hit probability function equation (5.6) has no closed-form solution, which does not allow us to develop the same kind of analysis as in Section 5.3. Instead, we approximate the hit probability by dividing the set of c documents into K popularity classes of size c_k, $1 \leq k \leq K$ ($\sum_{k=1}^{K} c_k = c$) and to assume that documents belonging to the same class have the same popularity. By doing this, the hit probability within each class can be computed by using Proposition 5.3.2. This approximation is validated in Section 5.6.2.

More specifically, assume that the K classes are ordered according to the popularity of their documents, with class 1 containing the most popular documents, class 2 the second most popular documents, etc. We define the global hit rate p_H as a weighted sum of the intra-class hit probabilities, that is,

$$p_H = \sum_{k=1}^{K} q_k \, p_H^k \tag{5.23}$$

with p_H^k the hit rate for documents of class k, and q_k the probability that a document of class k is requested. From (5.21) we see that

$$q_k = \mathbb{P}[\text{request for class } k] = \sum_{i=1}^{c_K} \frac{\Omega}{(\sum_{l=1}^{k-1} c_l + i)^\beta}, \quad k = 1, 2, \ldots, K. \tag{5.24}$$

This formula is obtained by summing the popularities of all documents in class k, with $\Omega/(\sum_{l=1}^{k-1} c_l + i)^\beta$ the popularity of the i-th most popular document of class k.

The intra-class hit probability p_H^k is obtained from Proposition 5.3.1 by replacing the parameters α and γ in (5.9) and (5.10) by $\alpha_k = \theta c_k/(\sigma q_k)$ and $\gamma_k = \sigma q_k/(\mu c_k)$, respectively.

It remains to specify how to choose the number of classes K and the number of objects assigned to each class. We first select the number of classes K. This number has to be low enough for computational efficiency, but large enough to capture the effect of the skew factor β on the hit probability. Clearly, the accuracy of this approximation will only increase with the number of classes. As a result, we simply choose the highest value of K that leads to an affordable computation. In Section 5.6.2 we will search for an acceptable number of classes through a comparison with a simulation of the real system.

Once K is chosen, we need to calculate the number of objects c_k assigned to each class k, $1 \leq k \leq K$. This is a classical clustering problem that can be solved with a scalar quantization algorithm (see e.g. [GG92]), which also readily provides the q_k coefficients. Given the initial popularity vector $(\psi_1, ..., \psi_c)$, the vector quantization algorithm aims at finding the class vector $(\phi_1, ..., \phi_K)$ that minimizes

$$E = \sum_{n=1}^{c} d(\psi_n, Q(\psi_n)) \tag{5.25}$$

where $d(.)$ is a distance measure (in our case the Euclidean distance) and $Q(\psi_n)$ the quantified version of ψ_n in the set $\{\phi_1, ..., \phi_K\}$, namely,

$$Q(\psi_n) = \arg\min_{\phi_k} d(\psi_n, \phi_k) \tag{5.26}$$

The quantity ϕ_k can be understood as the average popularity of documents in class k. Therefore the q_k coefficients are given by

$$q_k = c_k \phi_k, \quad 1 \leq k \leq K. \tag{5.27}$$

In order to determine the set $\{\phi_1, ..., \phi_K\}$ we used the Lloyd algorithm [GG92, page 189] that can be seen as an application of the Expectation-Maximization (EM) algorithm (cf. [Bil98]). This algorithm is composed of the following four steps:

S1: Initialize $(\phi, \ldots, \phi_K)$ (for example by using random sampling);

S2: For $n = 1, ...N$, estimate $Q(\psi_n)$ from (5.26): for each ϕ_k we obtain c_k corresponding objects;

S3: For $k = 1, 2, \ldots, K$, re-estimate ϕ_k: $\phi_k = (1/c_k) \sum_{n:Q(\psi_n)=\phi_k} \psi_n$;

S4: Go back to step 2 (S2) until convergence.

Since this algorithm is based on EM, the error will decrease at each iteration so that the set $\{\phi_1, ..., \phi_K\}$ will converge to a local optimum. In practice, this algorithm provides the optimal vector $(\phi_1, ..., \phi_K)$ along with the corresponding $(c_1, ..., c_K)$ values.

5.5 Application to Qualitative and Quantitative problems

In this section we investigate the impact on the hit probability of the document popularity distribution (Section 5.5.2) and of announced/unannounced departures (Section 5.5.3).

5.5.1 Experimental setup

We used Matlab to compute the hit probability from (5.9), (5.10) and (5.23) with the following parameters

$$\begin{cases} c &= 10^7 \text{ files} \\ \sigma &= 10^{-3} \text{ requests per second and per user} \\ \theta &= 10^{-6} \text{ s}^{-1} \text{ (corresponding to a 11-day TTL)} \\ \mu &= 10^{-7} \text{ s}^{-1} \text{ (corresponding to 3 failures/departures per year and per user)} \end{cases}$$

With the above values, we see from the definition of γ (4.4) that

$$\begin{cases} \gamma &= 10^{-3} \\ \alpha &= 10^4. \end{cases}$$

For the Zipf-like distribution we used $\beta = 0.7$ (cf. [BCF$^+$99]) and an approximation of $K = 10$ classes for 10^7 documents (cf. Section 5.6.2 for a discussion on the choice of K).

We also investigate the role of the mean online time on the hit probability in Section 5.5.3 by setting $\rho = 10^5$ and varying μ instead. This case will be explicitly mentioned.

5.5.2 Impact of the popularity distribution on the performance

Using our M/M/∞ model, we provide in Figure 5.1 the hit probability for the Squirrel system with unannounced departures as a function of the expected number of active nodes ρ, for uniform and Zipf-like document popularity distributions. In both cases, the hit probability is an increasing function of the size of the network (i.e. ρ), which reflects the self-scaling nature (and therefore the scalability) of the Squirrel system.

We can see from Figure 5.1 that the document probability distribution has an important impact on the hit probability. More specifically, the Zipf-like document popularity distribution generates a higher hit probability than the uniform popularity for small and medium-sized networks (say up to 10^4-10^5 active nodes on average). This is rather intuitive since when the popularity is skewed, many requests can be served with only a few popular cached documents. From this, we conclude that the document probability distribution is a crucial performance factor, which must be carefully modeled.

One can also use Figure 5.1 to determine the minimum network size necessary for an acceptable performance. For instance, with the experimental setting in Section 5.5.1, 8000 nodes must be active on the average with the Zipf-like distribution if one wants the hit probability to exceed 1/2.

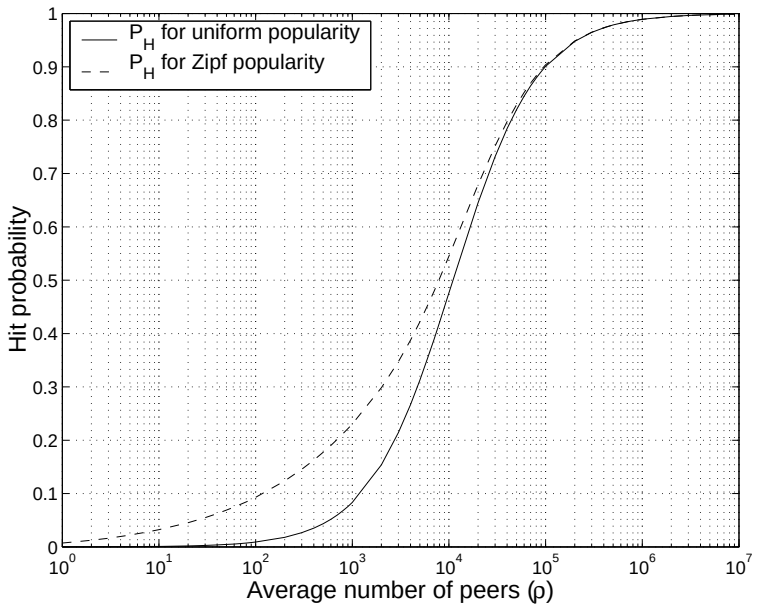

Figure 5.1: Hit probability of Squirrel for various document popularity distributions.

5.5.3 Utility of announced departures

In this section we evaluate the benefit of announcing departures on Squirrel performance. We compare the hit probability of the Squirrel system in the case of abrupt failures and announced departures. We do this for the uniform popularity case, using (5.9)-(5.10).

In Figure 5.2 we show the hit probability as a function the the network size. As expected (cf. Section 2.2.3.2 and 5.2), the hit probability is improved when users are able to announce their departure. However, the improvement that this feature brings is rather small, typically a 5% improvement over the abrupt failure case. Therefore, the benefit of announcing departures has to be balanced against the overhead cost that this feature induces, due to departing nodes transferring their content to their neighbors.

We can expect this tradeoff to depend strongly on the mean online time of peers $1/\mu$. In particular, if peers disconnect much more often than 3 times a year as assumed in Figure 5.2, the cost of not announcing departures may be much more important - as well as the overhead cost. In Figure 5.3 we compare the hit probability of the Squirrel system for announced departures and abrupt failures for various departure

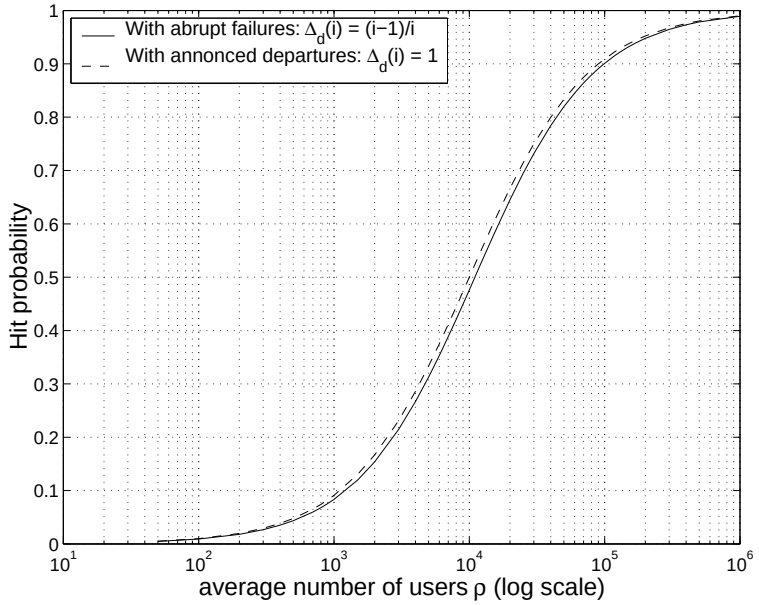

Figure 5.2: Hit probability of Squirrel for announced and unannounced node departures as a function of the network size.

rates, ranging from 10^{-7} (3 departures per year) to 1^{-5} (around 1 departure per day). For this experiment we used a network size of $\rho = 10^5$ nodes and $\theta = 10^{-5}$ (24 hours TTL).

We observe that the performance of the system does not depend on γ (i.e., on μ in this experiment) when nodes are able to announce their departure. While this property is not directly visible from the expression in (5.10), it is fairly intuitive since an announced departure does not generate performance degradation, unlike abrupt failures. We also observe that the performance degradation due to abrupt failures only becomes significant for $\gamma \leq 10^{-4}$, corresponding to $\mu \geq 10^{-6}$, or a mean online time of 11 days at most.

5.6 Experimental Validation

The goal of this section is to validate the fluid model approximation of requests, as well as the clustering approximation of document popularity, against a discrete-event simulation of the Squirrel system.

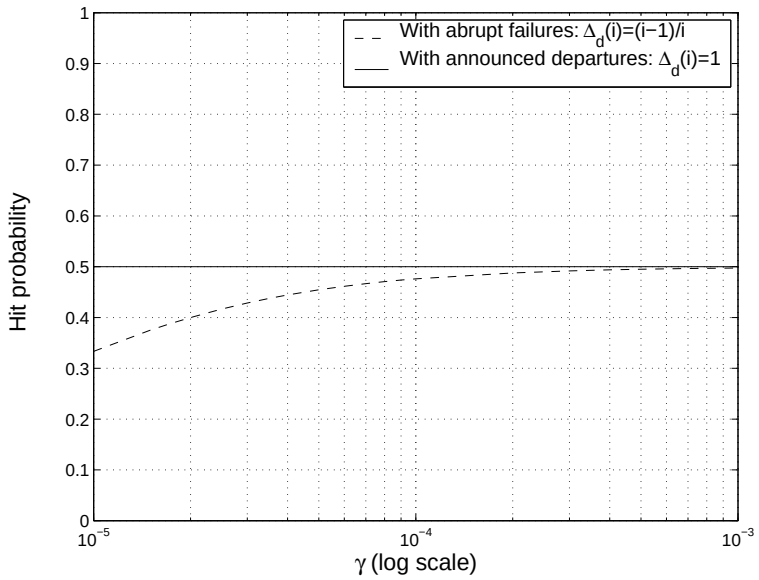

Figure 5.3: Hit probability of Squirrel for announced and unannounced node departures as a function of the mean online time. (network of 100,000 nodes.)

5.6.1 Uniform popularity case

We compared the hit probability when the node dynamics are modeled as the number of customers in a M/M/∞ queuing system, given in (5.9), to the corresponding formula (4.9) when node dynamics are represented by the Engset model. To do so we fixed the mean number of active nodes to the same value in both models and varied it between 1 and 11000 nodes (range of tractability of the hit probability obtained via the Engset model). The hit probability with the Engset model was computed using Maple V.

We found that both models consistently predict the same hit probability over all range of loads (i.e. mean number of active nodes), even for very small networks. The relative error was always smaller than 10^{-4}. Therefore, we can expect both models to describe the Squirrel system with the same accuracy.

In Chapter 4, we compared the theoretical results obtained via the Engset model to a discrete-event simulation of the Squirrel system with uniform popularity distribution. The simulation validates the fluid model approximation by using Poisson arrivals for requests and by allowing concurrent requests. We found that the theoretical hit probability was remarkably close to the hit probability obtained through simulations

(see Chapter 4 for details).

We can therefore safely conclude from the above that the hit probability computed via the M/M/∞ model offers the same accuracy as the one obtained via the Engset model, at least in the uniform popularity case (the analysis in Chapter 4 was only carried out for uniformly popular objects). In particular, we can reasonably extrapolate that the model developed in this chapter is a good approximation of the Squirrel behavior when deployed on large networks (say larger than 10,000 users), a situation where both discrete-event simulations and the model in Chapter 4 fail to work.

5.6.2 Zipf-like popularity

In Figure 5.4 we compare our multiclass approach (see 5.4) to a discrete-event simulation of the Squirrel system with a Zipf-like popularity distribution. The parameters were

$$c = 40,000 \text{ files}, \quad \rho = 9.99 \text{ nodes}, \quad \theta = 10^{-3} \text{ s},^{-1} \quad \mu = 10^{-7} \text{ s},^{-1} \quad \beta = 0.7$$

and we varied the request rate σ. Simulation results are subject to a 99% confidence interval of width 0.2%.

Figure 5.4 shows that our multiclass model is able to approximate very closely the hit probability of the simulated system: with 10 classes the curve follows already closely the same shape as the curve obtained by simulation, and with 100 classes the relative error amounts to 1%. We conclude that the combination of the M/M/∞ model for node dynamics and of the multiclass approach for modeling the different document popularities provides a very accurate estimation of Squirrel behavior and performance.

5.7 Conclusion

In this chapter, we modeled the Squirrel peer-to-peer cooperative caching system with a new stochastic fluid model that is tractable for very large networks (i.e., the order of a million nodes). This model, based on M/M/∞ node dynamics, can be viewed as a scalable extension of our previous Engset-based fluid model. The new model turns out to be tractable for any network size and is also more convenient than our previous

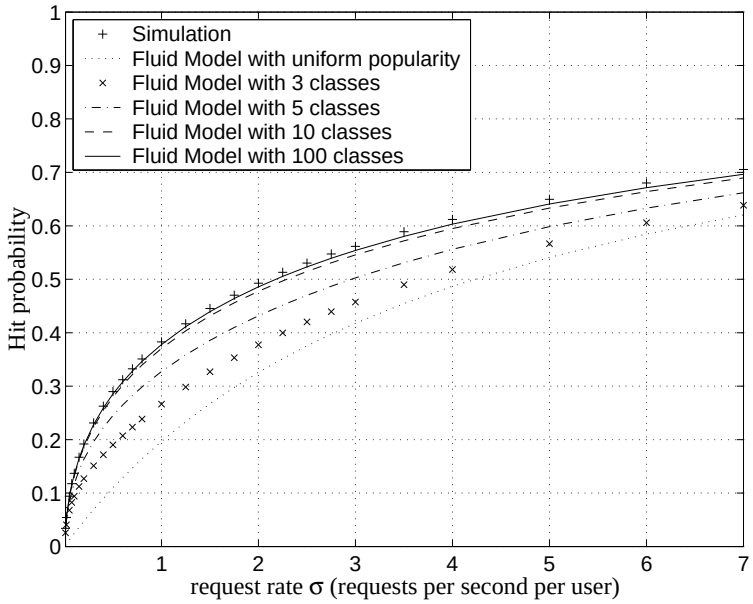

Figure 5.4: Comparison of the multiclass M/M/∞ model with a discrete-event simulation of Squirrel under a Zipf-like popularity distribution.

model. In addition, the model also allowed us to study the effect of nodes announcing their departure on the resulting hit probability.

Furthermore, we proposed, implemented and evaluated a multiclass approach to take variable object popularity into account. We found that this method gives accurate results even with a small number of classes.

Part II

A Client-Based Fluid Model

Chapter 6

A Multiclass Model for P2P Networks

6.1 Introduction

In a traditional client-server content distribution system, a large number of clients download content from a single server. If the single server cannot keep up with the demand from all the clients, the load can potentially be handled by replacing the server with a server farm and increasing the access bandwidth from the server farm. Although it is possible in theory to match any demand with a sufficient number of servers and sufficiently wide access pipes, the cost can easily become prohibitive.

BitTorrent is a content-distribution booster which enables a content provider to distribute popular content to a large number of clients without the need of large server farms and expensive high-speed Internet connections. The idea in essence is to split the file into small chunks, distribute different chunks to different downloading peers, and then have the peers obtain their missing chunks from each other. In this manner, the clients become servers, each of them contributing bandwidth to the content-distribution system. This approach has proved to be a highly successful mechanism to distribute popular content at low cost. In BitTorrent terminology, the servers that make available the entire file are called "seeds". The clients that are collecting and sharing chunks are

called "leechers". Once a leecher has downloaded the entire file, it becomes a seed for as long as it continues to distribute chunks to other clients. The BitTorrent protocol includes a "tit-for-tat" mechanism to ensure that leechers not only download content but also upload content [Coh]. BitTorrent is a peer-to-peer system since clients (peers) upload chunks directly to each other.

Qiu and Srikant [QS04] developed a tractable fluid model for BitTorrent-like content distribution systems. The model sheds insight on throughput, average download times, and stability of these systems. Although the model is elegant and tractable, it has limited applicability. First, the model assumes that all peers are homogeneous, with all peers having the same upload and download capacity. In reality, peers have diverse bandwidth characteristics, including dial-up modem access, broadband access (cable and ADSL), and high-speed Ethernet access. Second, the model does not allow for the exploration of distribution systems that provide application-layer differentiated services. Indeed, it is natural to conceive of a BitTorrent-like system in which there are, say, first-class peers and second-class peers. The first-class peers pay more (in some sense) and should receive better service – that is, shorter average download times – than the second-class peers. This is a form of "application-layer differentiated-service" as the service differentiation would be provided by the BitTorrent-like application rather than by the core of the Internet. Intuitively, BitTorrent-like systems could provide differentiated service by having the seeds and leechers allocate more of their upload bandwidth to first-class peers.

In this chapter we propose a deterministic multiclass fluid model for BitTorrent-like content distribution systems. The new fluid model can model both heterogeneous peer access and multiple differentiated service classes. Our multiclass fluid model results in a system of differential equations which generalize the single-class equations in [QS04]. The equations are significantly more complex and difficult to solve, as they explicitly distinguish between the various classes. The system of differential equations are so-called "linear switched systems" which are nonlinear differential equations with special structure (see e.g. [Lib03]). Nevertheless, for a number of important special cases, we explicitly solve the equations, obtaining closed-form solutions for average download times for each of the classes.

In particular, we consider the special case where downloaders leave the system immediately after completing their download. This is a worst-case scenario since altruistic seeds could instead stay in the system when they have completed their download,

contributing bandwidth and providing any missing chunk to other peers. For the service differentiation problem we prove that the system of differential equations governing the system dynamics admits a unique stable equilibrium that we compute in closed-form. From this result, we find the average download time for each class of peers and show how this result can be used to achieve service differentiation among the peers. We also indicate to what extent our results remain valid when seeds stay in the system for a non-negligible amount of time.

In the second part of the chapter, we address the bandwidth diversity problem. We show that the system of differential equations has a stable stationary state that may depend on the initial conditions. We identify all stationary solutions and compute the average download time associated with each solution. Last, we minimize the maximum average download time of both classes, regardless of the initial conditions.

The chapter is organized as follows. In Section 6.3 we introduce the multiclass model and derive the equations governing the system dynamics. Sections 6.4 and 6.5 provide results for the service differentiation problem and bandwidth diversity problem, respectively. Section 6.6 concludes the chapter.

6.2 Related Work

Peer-to-peer systems, like other content distribution systems, have been the object of few performance studies, perhaps because of their relative novelty, their constantly changing technology and popularity, as well as their intrinsic complexity. Among the few works that address performance issues of peer-to-peer systems, many studies rely on traffic measurement [IUKB+04, PGES04, SGP04, SGG02] and simulation [Qur04, FB04].

One of the pioneer works in peer-to-peer analytical modeling was [GFJ+03]. The authors propose a closed queing system which is sufficiently general to be able to model various P2P architectures such as distributed hash tables (DHT), flooding architectures and central index schemes, and to study the effect of various parameters such as the number of peers, the presence of freeloaders, or the request rates, by using an approximate numerical resolution of the model. In [BRF04], Biersack et al. propose a deterministic model of peer-to-peer systems for various peer organization topologies

(chain, tree and a more complex architecture connecting several trees). They derive bounds on the service capacity.

Finally, several models of BitTorrent-like systems have been successively proposed, beginning with [YD04] where both the transient and steady-state regime have been analyzed using respectively branching processes and numerically-solved Markov chains. Then, inspired by [YD04], [QS04] proposed a simple fluid model, as described in Section 6.1, and obtained closed-form expressions of the downloading delay instead of requiring numerical resolutions. More recently, Massoulié and Vojnović proposed a large population asymptotic analysis of BitTorrent systems [MV05]. They considered open and closed systems, the latter being appropriate for the flash crowd transient behavior. Their paper shows many interesting properties such as the stability conditions of BitTorrent systems as well as an explicit expression of equilibrium points. In particular, they show that the performance of BitTorrent systems is not critically dependent on the goodwill of users to stay in the system after completing their download.

A first multiclass fluid model of BitTorrent-like networks based on [QS04] was proposed in [LNB04]. The authors study the specific bandwidth diversity problem through a comparison with the single-class homogeneous model in [QS04]. This is done in the case of symmetric access links and focuses on parallel download, using max-min fairness to numerically compute connexion rates. Besides these detailed assumptions which make the work [LNB04] very different from this chapter, we can outline the following important differences. In this chapter, we propose a generic framework designed to study, for instance, the resource allocation problem at individual peers. We then apply our general model to two important problems, including bandwidth diversity, but from an optimization point of view. Second, as a result of these different objectives, the theoretical aspect of our work is very different. In particular, our search for a generic model leads us to address stability issues for each problem, which is not the case in [LNB04]. We also study carefully the boundaries between the working regions defined by the system and show that in some cases the steady-state bottleneck depends on initial conditions.

6.3 Multiclass Model

In this chapter we consider a BitTorrent-like system with two classes of peers, with the classes denoted by $i = 1$ and $i = 2$. All peers in both classes want to obtain the single file F. Without loss of generality, we take the file size to be equal to 1. Each class has seeds and downloaders (leechers). Seeds have all of the file, whereas downloaders have only portions of the file. When a downloader obtains the whole file, it immediately becomes a seed. Let $y_i(t)$ and $x_i(t)$ denote the number of seeds and downloaders, respectively, for class-i peers at time t. Since we consider a deterministic fluid model as in [QS04], $y_i(t)$ and $x_i(t)$ are continuous variables. In this chapter, we are particularly interested in the steady-state behavior of y_i and x_i, $i = 1, 2$. We need to also define the following:

- Let λ_i be the constant rate at which new class-i downloaders arrive. Whenever a new class-i downloader arrives, x_i is incremented by 1.
- Let μ_i be the upload bandwidth of a peer from class i.
- Let c_i be the download bandwidth of a peer from class i. We make the realistic assumption that $c_i \geq \mu_i$, which is consistent with the current access technologies. Whenever a class-i peer has fully downloaded the file, x_i is decremented by 1 and y_i is incremented by 1.
- As in [QS04], we allow downloaders to abort downloading before fully obtaining the file. Let θ_i be the rate at which class-i downloaders abort. Whenever a class-i downloader aborts, x_i is decremented by 1.
- Let γ_i denote the rate at which class-i seeds leave the system. Whenever a class-i seed leaves the system, y_i is decremented by 1.
- Let $\eta_i \in (0, 1)$ denote the efficiency of class-i downloaders, which is the average amount of a downloader's upload bandwidth that is being used for content distribution. This parameter was first introduced in [YD04] in a Markov chain model, then used in [QS04] in the single-class case.

We now discuss the resource allocation policy. A peer (seed or downloader) will upload chunks to multiple peers simultaneously. The aggregate rate at which a class-i seed peer uploads is μ_i; the aggregate rate at which a class-i downloader peer uploads

is $\eta_i \mu_i$. A peer will allocate its upload rate between the two classes of peers. For a class-i peer, let $\alpha_i(x_1, x_2)$ (resp. $1 - \alpha_i(x_1, x_2)$) be the fraction of its upload rate that is allocated to class-i peers, that is, to peers in its own class (resp. to peers in the other class) when there are x_1 class-1 downloaders present and x_2 class-2 downloaders present. Thus, $\alpha_i(x_1, x_2)$ lies between 0 and 1. We refer to $(\alpha_1(x_1, x_2), \alpha_2(x_1, x_2))$ as a **dynamic allocation policy**. To implement such a resource allocation, peers only need to know to which class the other peers belong, and also the population in each class for the dynamic policy. This information may be provided, for instance, by the tracker server which is used in BitTorrent as a bootstrap to help incoming peers discover seeds and other downloaders.

In this chapter we limit our attention to **static allocation policies**, namely, policies of the form $\alpha_i(x_1, x_2) = \alpha_i$ for all x_1 and x_2 for $i = 1, 2$.

Our deterministic model of the two-class multiclass P2P network is now complete. Figure 6.1 summarizes the states and rates in the system.

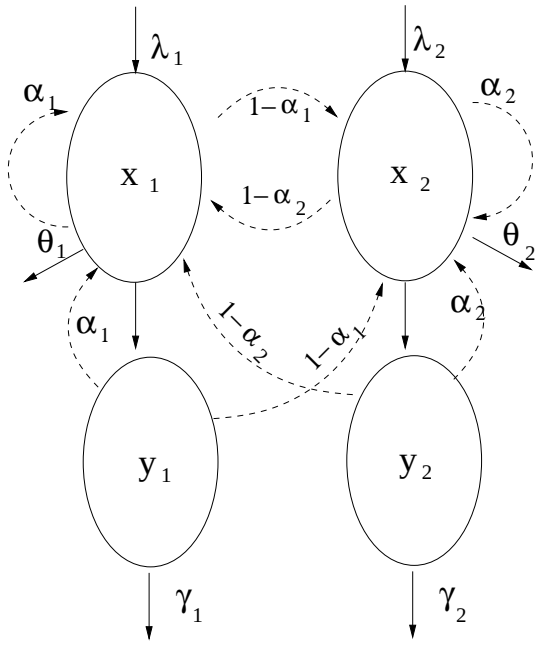

Figure 6.1: General model for a two-class P2P file dissemination system. Solid arcs represent migrations of users (connections, migrations from downloaders to seeds, disconnections). Dashed arcs represent the fraction of allocated upload bandwidth from users of one class to downloaders of another class.

We now develop a system of differential equations for the fluid-version of the above multiclass model. At time t, the total upload rate provided by class-i peers to peers of class i is

$$\alpha_i \mu_i (\eta_i x_i(t) + y_i(t)) \tag{6.1}$$

and to peers of the other class

$$(1 - \alpha_i) \mu_i (\eta_i x_i(t) + y_i(t)) \tag{6.2}$$

Therefore, the total upload rate provided by class-i peers is

$$\mu_i (\eta_i x_i(t) + y_i(t)) \tag{6.3}$$

Let $k = 3 - i$, $i = 1, 2$ designate the other class. The total download rate provided to peers of class i cannot exceed $c_i x_i(t)$ so that the total flow rate out of state $x_i(t)$ is

$$\min(c_i x_i(t), \alpha_i \mu_i (\eta_i x_i(t) + y_i(t)) + (1 - \alpha_k) \mu_k (\eta_k x_k(t) + y_k(t))) \tag{6.4}$$

to which we must add $\theta_i x_i(t)$, the total flow rate at which downloaders leave the system without having downloaded the entire file. By definition, the flow rate into state $x_i(t)$ is λ_i. Hence, the time-evolution of $(x_1(t), x_2(t))$ is governed by the following differential equations

$$\begin{aligned} \frac{dx_i(t)}{dt} &= \lambda_i - \theta_i x_i(t) - \min \Big(c_i x_i(t),\ \alpha_i \mu_i (\eta_i x_i(t) + y_i(t)) \\ &\quad + (1 - \alpha_k) \mu_k (\eta_k x_k(t) + y_k(t)) \Big) \end{aligned} \tag{6.5}$$

for $i = 1, 2$ and $k = 3 - i$.

Similarly, we find that the total flow rate into state $y_i(t)$ is given by the total rate at which downloaders become seeds, namely $\mu_i(\eta_i x_i(t) + y_i(t)) + (1 - \alpha_k)\mu_k(\eta_k x_k(t) + y_k(t))$ as explained above, while the total flow rate out of state $y_i(t)$ is simply $\gamma_i y_i(t)$. This gives the following equations for the time-evolution of $(y_1(t), y_2(t))$

$$\begin{aligned} \frac{dy_i(t)}{dt} &= \min \Big(c_i x_i(t), \alpha_i \mu_i (\eta_i x_i(t) + y_i(t)) \\ &\quad + (1 - \alpha_k) \mu_k (\eta_k x_k(t) + y_k(t)) \Big) - \gamma_i y_i(t) \end{aligned} \tag{6.6}$$

for $i = 1, 2$ and $k = 3 - i$. Equations (6.5)-(6.6) fully define the system dynamics.

We will be particularly interested in the case where downloaders leave the system at once when they have completed their download, namely $1/\gamma_1 = 1/\gamma_2 = 0$. There

are two reasons why we consider this situation. First, it will yield much more tractable equations, as shown next. Second, this case represents a worst-case situation, where peers are not willing to cooperate, and leave the system just when they are the most useful to the system, being able to provide chunks and bandwidth without having to consume resources. In this case, they never become seeds, which implies $y_i(t) = 0$ for all $t > 0$. As a result, system (6.5) reduces to

$$\frac{dx_i(t)}{dt} = \lambda_i - \theta_i x_i(t) - \min\left(c_i x_i(t),\, \alpha_i \beta_i x_i(t) + (1-\alpha_k)\beta_k x_k(t)\right) \tag{6.7}$$

for $i = 1, 2$ and $k = 3 - i$, where

$$\beta_i \stackrel{\text{def}}{=} \mu_i \eta_i. \tag{6.8}$$

Note that

$$c_i > \beta_i, \quad i = 1, 2 \tag{6.9}$$

since we have assumed that $c \geq \mu_i$ and $0 < \eta_i < 1$. In matrix form (6.7) writes

$$\dot{\mathbf{x}}(t) = A_{\sigma(\mathbf{x}(t))}\, \mathbf{x}(t) + \mathbf{b} \tag{6.10}$$

with $\mathbf{x}(t) \stackrel{\text{def}}{=} (x_1(t), x_2(t))^T$ and $\mathbf{b} = (-\lambda_1, -\lambda_2)^T$ (as usual $\mathbf{v}^T$ denotes the transpose vector of the vector $\mathbf{v}$, and $\dot{\mathbf{x}}(t)$ denotes the derivative of vector $\mathbf{x}(t)$ with regard to t). In (6.10) σ is an integer-value mapping, taking values in $\sigma \in \{1, 2, 3, 4\}$, given by

$$\begin{aligned} \sigma(\mathbf{x}) \;=\; & 1 + 2 \times \mathbb{1}(c_1 x_1 \geq \alpha_1 \beta_1 x_1 + (1-\alpha_2)\beta_2 \eta_2 x_2) \\ & + \mathbb{1}(c_2 x_2 \geq \alpha_2 \beta_2 \eta_2 x_2 + (1-\alpha_1)\beta_1 \eta_1 x_1) \end{aligned} \tag{6.11}$$

for $\mathbf{x} = (x_1, x_2)$, where $\mathbb{1}(A)$ denotes the indicator function of the event A (i.e. $\mathbb{1}(A) = 1$ if A holds and zero otherwise). The mapping σ is called a *switching condition* and a system like (6.10) is called a *switched system* [Mor97, Lib03]. The 2-by-2 matrices A_i, $i = 1, \ldots, 4$, can easily be identified from (6.7).

The model where $1/\gamma_1 = 1/\gamma_2 = 0$ will be referred to as the *no-seed* model. A natural question is the following one: how do downloaders ever get any chunk if there are no seeds? Here, we make a distinction between two notions of seeds. A BitTorrent-like system needs, at startup time, at least one seed, for as long as it needs to upload (at least) a complete copy of the file. Though this bootstrap seed is mandatory to make the file available, it may leave long before the system reaches a steady-state. Therefore, its role is limited to starting the torrent, and is negligible on the long-term. Note that the general system (6.5)-(6.6), as well as the single-class model in [QS04], also neglect

this bootstrap seed, since the system may have a nonzero solution even if $y_i(0) = 0$ for $i = 1, 2$. Downloaders which have a complete copy of the file, on the other hand, will have an impact on the steady-state since they belong to the long-term dynamics of the system. These regular seeds are considered in (6.5)-(6.6), whereas the no-seed model assumes they leave the system immediately. Though the BitTorrent system kindly asks its users to stay online as long as possible when they become seeds, the system is kept alive by the downloaders only, since the protocol really incites them to exchange chunks to each other. It has been shown in [QS04] and [MV05] that when $\eta > 0$ the system does not die, no matter how short a time seeds stay in the system.

We conclude this section by introducing the cost functions that we will consider throughout the chapter. Let ϕ_i be the *download cost* of peers of class i, which is defined as the expected download time given that the peer completes the download. An analytic expression for ϕ_i can easily be derived as follows. Assume that $x_i(t)$ has a stationary regime, denoted by $\bar{x}_i$. By Little's formula, the expected download time T_i for peers of class i is given by $T_i = \bar{x}_i/\lambda_i$. On the other hand, the stationary probability p_i that a class-i peer completes its download is $p_i = (\lambda_i - \theta_i \bar{x}_i)/\lambda_i$. Therefore, the download cost for peers of class i takes the form

$$\phi_i = \frac{\bar{x}_i}{\lambda_i - \theta_i \bar{x}_i}, \quad \text{for } i = 1, 2. \tag{6.12}$$

In the next two sections we shall address two different problems corresponding to different subsets of (static) allocation policies: $(\alpha_1, \alpha_2) = (\alpha, 1 - \alpha)$, referred to as the service differentiation problem (Section 6.4), and $(\alpha_1, \alpha_2) = (\alpha, \alpha)$, referred to as the bandwidth diversity problem (Section 6.5). Both problems will be considered for no-seed models.

6.4 Resource Allocation Policy for Service Differentiation

In this section we address the service differentiation problem for the no-seed model (unless otherwise mentioned). For the sake of simplicity we further restrict the analysis to the case where all peers have the same download/upload bandwidths and the same efficiency parameters. In other words, we assume that $1/\gamma_i = 0$, $c_i = c$, $\mu_i = \mu$ and $\eta_i = \eta$ for $i = 1, 2$. We define $\beta \stackrel{\text{def}}{=} \mu\eta$.

We recall that the service differentiation problem corresponds to the situation where $\alpha_1 = 1 - \alpha_2 = \alpha$ (see end of Section 6.3). With these assumptions the generic model described in Figure 6.1 is now simplified to the two-dimensional model (i.e. with only two variables x_1 and x_2) represented in Figure 6.2.

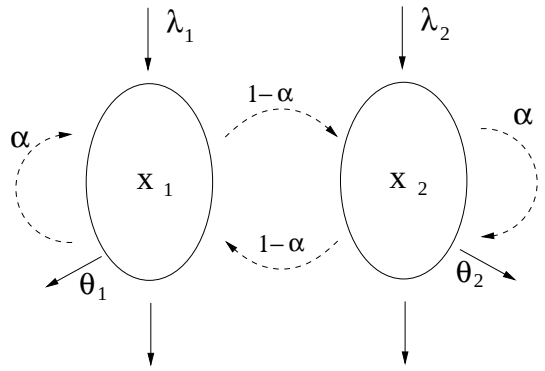

Figure 6.2: Two-class deterministic model for service differentiation in BitTorrent-like networks. Solid arcs represent migration rates of users. Dashed arcs represent the fraction of allocated bandwidth.

Our goal is to solve the resulting system of differential equations (see below) and determine the download cost (defined in (6.12)) of the two classes of peers. In particular, we shall show that differential service can indeed be provided to the two classes of peers via the allocation parameter α.

6.4.1 Equilibrium

Under the above assumptions the system of differential equations (6.7) governing the dynamics of $(x_1(t), x_2(t))$ simplifies to

$$\frac{dx_1(t)}{dt} = \lambda_1 - \theta_1 x_1(t) - \min\big(cx_1(t),\, \alpha\beta(x_1(t) + x_2(t))\big) \tag{6.13}$$

$$\frac{dx_2(t)}{dt} = \lambda_2 - \theta_2 x_2(t) - \min\big(cx_2(t),\, (1-\alpha)\beta(x_1(t) + x_2(t))\big) \tag{6.14}$$

In matrix notation this system is given by (6.10) with the switching condition

$$\sigma(\mathbf{x}) = 1 + 2 \times \mathbb{1}(cx_1 \geq \alpha\beta(x_1 + x_2)) + \mathbb{1}(cx_2 \geq (1-\alpha)\beta(x_1 + x_2)) \tag{6.15}$$

We introduce the new parameters

$$a_1 \stackrel{\text{def}}{=} \max\left(0, 1 - \frac{c\lambda_2(\theta_1+\beta)}{D}\right) \tag{6.16}$$

$$a_2 \stackrel{\text{def}}{=} \min\left(1, \frac{c\lambda_1(\theta_2+\beta)}{D}\right) \tag{6.17}$$

with

$$D \stackrel{\text{def}}{=} \beta(\lambda_1(\theta_2+c) + \lambda_2(\theta_1+c)) \tag{6.18}$$

Proposition 6.4.1 below computes the equilibrium point of the switched system (6.13)-(6.14).

Proposition 6.4.1 (Equilibrium point for service differentiation) *Regardless of the initial condition* $\mathbf{x}(0)$*, the system of equations (6.13)-(6.14) has a unique equilibrium point* $\bar{\mathbf{x}}$ *given by*

$$\bar{\mathbf{x}}^T = \begin{cases} \left(\dfrac{\lambda_1 - \alpha\frac{\lambda_2\beta}{\theta_2+c}}{\theta_1+\alpha\beta}, \dfrac{\lambda_2}{\theta_2+c}\right) & \text{if } 0 \le \alpha < a_1 \\ \left(\dfrac{\lambda_1(\theta_2+(1-\alpha)\beta) - \lambda_2\alpha\beta}{\theta_2(\theta_1+\alpha\beta)+\theta_1(1-\alpha)\beta}, \dfrac{\lambda_2(\theta_1+\alpha\beta) - \lambda_1(1-\alpha)\beta}{\theta_2(\theta_1+\alpha\beta)+\theta_1(1-\alpha)\beta}\right) & \text{if } a_1 \le \alpha \le a_2 \\ \left(\dfrac{\lambda_1}{\theta_1+c}, \dfrac{\lambda_2 - (1-\alpha)\frac{\lambda_1\beta}{\theta_1+c}}{\theta_2+(1-\alpha)\beta}\right) & \text{if } a_2 < \alpha \le 1. \end{cases} \tag{6.19}$$

◇

Proof. We first check that if $\lim_{t\uparrow\infty}\mathbf{x}(t)$ exists, then it is given by (6.19).

Assume that $\lim_{t\uparrow\infty}\mathbf{x}(t) = \bar{\mathbf{x}}$. Letting $t \to \infty$ in (6.10) yields

$$A_{\sigma(\bar{\mathbf{x}})}\,\bar{\mathbf{x}} + \mathbf{b} = 0 \tag{6.20}$$

where σ is given in (6.15). We consider separately the four systems of linear equations obtained from (6.20) when (a) $\sigma(\bar{\mathbf{x}}) = 1$, (b) $\sigma(\bar{\mathbf{x}}) = 2$, (c) $\sigma(\bar{\mathbf{x}}) = 3$ and (d) $\sigma(\bar{\mathbf{x}}) = 4$.

(a) When $\sigma(\bar{\mathbf{x}}) = 1$ or equivalently $c\bar{x}_1 < \alpha\beta(\bar{x}_1+\bar{x}_2)$ and $c\bar{x}_2 < (1-\alpha)\beta(\bar{x}_1+\bar{x}_2)$:

the download rate is the bottleneck for both classes of peers. We find

$$\bar{\mathbf{x}}^T = \left(\frac{\lambda_1}{\theta_1 + c}, \frac{\lambda_2}{\theta_2 + c} \right) \tag{6.21}$$

(b) When $\sigma(\bar{\mathbf{x}}) = 2$ or equivalently $c\bar{x}_1 < \alpha\beta(\bar{x}_1 + \bar{x}_2)$ and $c\bar{x}_2 \geq (1-\alpha)\beta(\bar{x}_1 + \bar{x}_2)$:

the bottleneck is the download rate for class-1 peers and the upload rate for class-2 peers. We find

$$\bar{\mathbf{x}}^T = \left(\frac{\lambda_1}{\theta_1 + c}, \frac{\lambda_2 - (1-\alpha)\frac{\lambda_1\beta}{\theta_1 + c}}{\theta_2 + (1-\alpha)\beta} \right) \tag{6.22}$$

(c) When $\sigma(\bar{\mathbf{x}}) = 3$ or equivalently $c\bar{x}_1 \geq \alpha\beta(\bar{x}_1 + \bar{x}_2)$ and $c\bar{x}_2 < (1-\alpha)\beta(\bar{x}_1 + \bar{x}_2)$:

the bottleneck is the download rate for peers of class 2 and the upload rate for peers of class 1. In this case

$$\bar{\mathbf{x}}^T = \left(\frac{\lambda_1 - \alpha\frac{\lambda_2\beta}{\theta_2 + c}}{\theta_1 + \alpha\beta}, \frac{\lambda_2}{\theta_2 + c} \right) \tag{6.23}$$

(d) When $\sigma(\bar{\mathbf{x}}) = 4$ or equivalently $c\bar{x}_1 \geq \alpha\beta(\bar{x}_1 + \bar{x}_2)$ and $c\bar{x}_2 \geq (1-\alpha)\beta(\bar{x}_1 + \bar{x}_2)$:

the bottleneck is the download rate for both classes of peers. The equilibrium point is

$$\bar{\mathbf{x}}^T = \left(\frac{\lambda_1(\theta_2 + (1-\alpha)\beta) - \lambda_2\alpha\beta}{\theta_2(\theta_1 + \alpha\beta) + \theta_1(1-\alpha)\beta}, \frac{\lambda_2(\theta_1 + \alpha\beta) - \lambda_1(1-\alpha)\beta}{\theta_2(\theta_1 + \alpha\beta) + \theta_1(1-\alpha)\beta} \right) \tag{6.24}$$

In the following, we call "type-i equilibrium" the equilibrium found when $\sigma(\bar{\mathbf{x}}) = i$.

The next step is to check if a type-i equilibrium may exist, namely, if $\sigma(\bar{\mathbf{x}}) = 1$ (resp. $\sigma(\bar{\mathbf{x}}) = 2$, $\sigma(\bar{\mathbf{x}}) = 3$, $\sigma(\bar{\mathbf{x}}) = 4$) when $\bar{\mathbf{x}}$ is given by (6.21) (resp. (6.22), (6.23), (6.24)).

It is easily seen that a type-1 equilibrium may only exist if $c \leq \beta$. Since this condition is never met (use (6.9) with $c_i = c$ and $\beta_i = \beta$) we conclude that there is no type-1 equilibrium. Recall that $0 \leq \alpha \leq 1$. We prove in Appendix G that a type-2

equilibrium may only exist if $a_2 < \alpha \leq 1$. The same type of analysis shows that a type-3 equilibrium may only exist if $0 \leq \alpha < a_1$, and that a type-4 equilibrium may only exist if $a_1 \leq \alpha \leq a_2$. This concludes the proof that, if $\lim_{t\uparrow\infty} \bar{\mathbf{x}}(t) = \bar{\mathbf{x}}$ exists, then $\bar{\mathbf{x}}$ is given by (6.19) (regardless of the initial condition).

We now turn to the proof that $\lim_{t\uparrow\infty} \bar{\mathbf{x}}(t)$ exists. To this end, we investigate the nature of the equilibrium of each of the linear systems $\dot{\mathbf{x}}(t) = A_i\,\mathbf{x}(t) + \mathbf{b}$, for $i = 2, 3, 4$, with

$$A_2 = \begin{pmatrix} -(\theta_1 + c) & 0 \\ -(1-\alpha)\beta & -(\theta_2 + (1-\alpha)\beta) \end{pmatrix} \tag{6.25}$$

$$A_3 = \begin{pmatrix} -(\theta_1 + \alpha\beta) & -\alpha\beta \\ 0 & -(\theta_2 + c) \end{pmatrix} \tag{6.26}$$

$$A_4 = \begin{pmatrix} -(\theta_1 + \alpha\beta) & -\alpha\beta \\ -(1-\alpha)\beta & -(\theta_2 + (1-\alpha)\beta) \end{pmatrix} \tag{6.27}$$

Recall that the equilibrium of the system $\dot{\mathbf{x}}(t) = A_i\,\mathbf{x}(t) + \mathbf{b}$ is stable if and only if all eigenvalues of the matrix A_i have strictly negative real parts [Kha92]. It is easily seen that A_2 and A_3 have two strictly negative eigenvalues, given by $(-(\theta_1 + c), -(\theta_2 + (1-\alpha)\beta))$ and $(-(\theta_1 + \alpha\beta), -(\theta_2 + c))$, respectively. The same property holds for A_4. To see this, denote by $D(c, r)$ the closed disc of center c and radius r in the complex plane. Recall that Geršgorin circle theorem [HJ85, p. 344] states that every eigenvalue of a square matrix $A = (a_{i,j}), 1 \leq i, j \leq n$ lie in at least one of n the Geršgorin circles $D(a_{i,i}, r_j)$ $(1 \leq i \leq n)$ with center $a_{i,i}$ and with radius r_i equal to the sum of the modulus of all the i-th line elements except the diagonal element: $r_i = \sum_{j=1 j\neq i}^{n} |a_{i,j}|$. The direct application of this theorem gives that both eigenvalues of A_4 lie in the region $D(-\theta_1 - \alpha\beta, \alpha\beta) \cup D(-\theta_2 - (1-\alpha)\beta, (1-\alpha)\beta)$, from which the result follows.

We have now proved the local stability of the equilibrium of each linear subsystem of (6.13)-(6.14). However, up to now we have not yet been able to prove the *global* stability of (6.13)-(6.14). The interested reader can refer to [Lib03] for the stability of linear switched systems.

In summary, we have shown that for a given value of α, a unique equilibrium exists, is given in (6.19), and is stable. This completes the proof. ■

6.4.2 How can we achieve a target QoS ratio k?

It is now possible to achieve service differentiation using parameter α as follows. The goal is to differentiate the download costs ϕ_1 and ϕ_2 of class-1 and class-2 peers, respectively. These costs are given in the next proposition.

Proposition 6.4.2 (Download costs for service differentiation)
In a no-seed model, the download cost ϕ_i of class-i peers in the service differentiation problem is given by:

$$\begin{array}{lll} \phi_1 = \dfrac{\lambda_1(\theta_2+c)-\alpha\lambda_2\beta}{\alpha\beta(\lambda_2\theta_1+\lambda_1(\theta_2+c))}, & \phi_2 = \dfrac{1}{c} & \text{if } 0 \le \alpha < a_1 \\ \phi_1 = \dfrac{\lambda_1(\theta_2+\beta)-\alpha\beta(\lambda_1+\lambda_2)}{\alpha\beta(\lambda_2\theta_1+\lambda_1\theta_2)}, & \phi_2 = \dfrac{\lambda_2\theta_1-\lambda_1\beta+\alpha\beta(\lambda_1+\lambda_2)}{(1-\alpha)\beta(\lambda_2\theta_1+\lambda_1\theta_2)} & \text{if } a_1 \le \alpha \le a_2 \\ \phi_1 = \dfrac{1}{c}, & \phi_2 = \dfrac{\lambda_2(\theta_1+c)-\lambda_1\beta+\alpha\lambda_1\beta}{(1-\alpha)\beta(\theta_2\lambda_1+\lambda_2(\theta_1+c))} & \text{if } a_2 < \alpha \le 1. \end{array}$$

◇

First, note that in the service differentiation problem, we considered the static allocation policy $(\alpha, 1-\alpha)$. Since the two classes have the same bandwidth characteristics (i.e. $c_1 = c_2$, $\mu_1 = \mu_2$) and the same efficiency parameters ($\eta_1 = \eta_2$), this policy results in a download cost tradeoff governed by α. This tradeoff is represented in Figure 6.3.

There are at least two ways to achieve service differentiation. The first one is to guarantee a subscribed download cost for one class (e.g. $\phi_1 = \Phi$ for peers of class 1) with no constraint on the download cost of the other class. This can be done by assigning to the parameter α the (unique) root in $[0,1]$ of the linear mapping $\alpha \to \phi_1 - \Phi$, where ϕ_1 is given in Proposition 6.4.2.

The second one is to achieve a target download cost ratio k between first- and second-class peers, namely

$$\frac{\phi_2}{\phi_1} = k. \tag{6.28}$$

The parameter α is then obtained as the (unique) root in $[0,1]$ of the (either linear or quadratic) mapping $\alpha \to \phi_2/\phi_1 - k$. For a given set of parameters (see caption), Figure 6.4 reports the value of α that satisfies (6.28) as a function of k, for $k \in [1, 300]$.

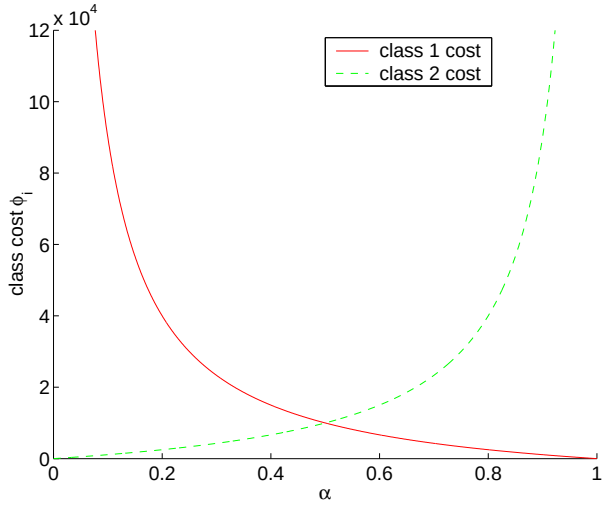

Figure 6.3: Download cost tradeoff ($\lambda_1 = \lambda_2 = 10^{-1}$, $\theta_1 = \theta_2 = \beta = 10^{-4}$, $c = 10^{-3}$)

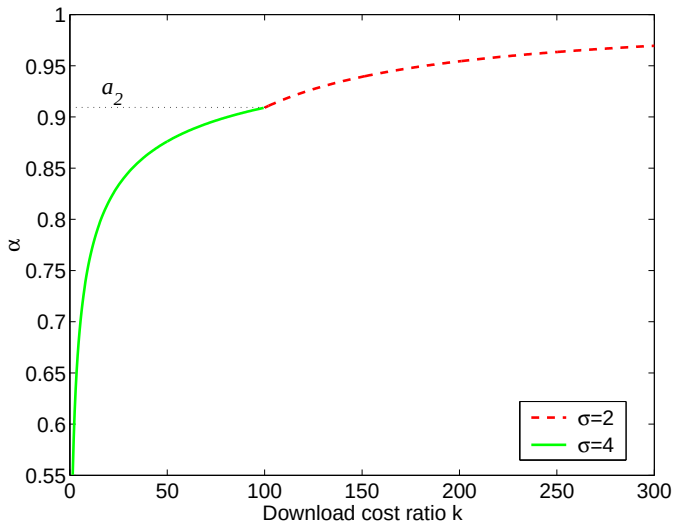

Figure 6.4: Selection of α for a target cost ratio k ($\lambda_1 = \lambda_2 = 10^{-1}$, $\theta_1 = \theta_2 = \beta = 10^{-4}$, $c = 10^{-3}$)

We conclude that service differentiation in BitTorrent-like networks can easily be achieved through the single parameter α.

6.4.3 What if users stay connected after the download?

All the results obtained so far in this section have been derived under the assumption that there are no seeds in the system. As already observed this case can be seen as a worst-case scenario, where peers are selfish and leave the system as soon as they have downloaded the file.

In this section, we relax the no-seed assumption. In other words, we assume that downloaders do not leave the system immediately after they have downloaded the file, but continue to upload chunks to the other peers for some time of average duration $1/\gamma_i > 0$ for class-i peers.

In this more general setting the time-evolution of the system is given by the system of differential equations (6.5)-(6.6), with $(\alpha_1, \alpha_2) = (\alpha, 1-\alpha)$. We still assume that $\mu_1 = \mu_2$, $c_1 = c_2$ and $\eta_1 = \eta_2$ (these assumptions could be relaxed). The analysis of this system is much more complex than that of the no-seed model. While it is still easy to compute the stationary solutions of (6.5)-(6.6) in explicit form, it is much more complex to study the existence and stability of these solutions. However, there is no difficulty to numerically compute the steady-state of these equations once numerical values have been assigned to the system parameters.

This has been done for the following set of parameters: $\lambda_1 = \lambda_2 = 10^{-1}$ peers/s, $\theta_1 = \theta_2 = \mu = 10^{-4}s^{-1}, c = 10^{-3}s^{-1}, \eta_1 = \eta_2 = 0.9$. These parameters are rounded values of typical values estimated using the statistics in [IUKB+04] in particular. We also assumed $\gamma_1 = \gamma_2 = \gamma$.

For given values of γ and $\alpha \in (0,1)$ we have computed the ratio of download costs $R = \phi_2/\phi_1$ for the seed model and the ratio of download costs $r = \phi_2/\phi_1$ for the no-seed model. We have found that for $\gamma = c$, the relative error $|R-r|/R$ averages 1%. For $\gamma \geq c$, this relative error rapidly decreases, making the no-seed model very-well suited for the service differentiation problem. This is consistent with the conclusions in [MV05] and [QS04] which indicate that altruism of users is not critical to the performance of BitTorrent-like systems. For $\gamma < c$, the relative error rapidly increases, making a

numerical estimation of α necessary, using (6.5)-(6.6).

6.5 Bandwidth Diversity

We now address the bandwidth diversity problem for the no-seed model ($1/\gamma_i = 0$ for $i = 1, 2$). We consider two classes of peers with different bandwidths (e.g., ADSL users and corporate users). Recall that the bandwidth diversity problem we consider is characterized by $\alpha_1 = \alpha_2 = \alpha$ (see Section 6.3).

Our first objective is to determine the download cost for each class of peers. Then, we will find a static allocation policy (α, α) that minimizes the maximum download cost of both classes. With a slight abuse of notation, a static allocation policy (α, α) will simply be referred to as an allocation α from now on.

Under the aforementioned assumptions the system of differential equations (6.7) becomes

$$\frac{dx_1}{dt} = \lambda_1 - \theta_1 x_1 - \min\left(c_1 x_1,\, \alpha\beta_1 x_1 + (1-\alpha)\beta_2 x_2\right) \tag{6.29}$$

$$\frac{dx_2}{dt} = \lambda_2 - \theta_2 x_2 - \min\left(c_2 x_2,\, (1-\alpha)\beta_1 x_1 + \alpha\beta_2 x_2\right) \tag{6.30}$$

In matrix notation the system (6.29)-(6.30) is given by (6.10), with the switching condition

$$\sigma(\mathbf{x}) = 1 + 2 \times \mathbb{1}(cx_1 \geq \alpha\beta_1 x_1 + (1-\alpha)\beta_2 x_2) + \mathbb{1}(cx_2 \geq (1-\alpha)\beta_1 x_1 + \alpha\beta_2 x_2) \tag{6.31}$$

For the sake of compactness we introduce the new parameters

$$a_3 \stackrel{\text{def}}{=} \frac{\lambda_2\beta_2(\theta_1 + c_1) - \lambda_1(c_1\theta_2 + \beta_1\beta_2)}{\lambda_2\beta_2(\theta_1 + c_1) - \lambda_1(\beta_1\theta_2 + 2\beta_1\beta_2 - c_1\beta_2)} \tag{6.32a}$$

$$a_4 \stackrel{\text{def}}{=} \frac{\lambda_1\beta_1(\theta_2 + c_2) - c_2\lambda_2(\theta_1 + c_1)}{\lambda_1\beta_1(\theta_2 + c_2) - \beta_2\lambda_2(\theta_1 + c_1)} \tag{6.32b}$$

$$a_5 \stackrel{\text{def}}{=} \frac{\lambda_1\beta_1(\theta_2 + c_2) - \lambda_2(c_2\theta_1 + \beta_2\beta_1)}{\lambda_1\beta_1(\theta_2 + c_2) - \lambda_2(\beta_2\theta_1 + 2\beta_2\beta_1 - c_2\beta_1)} \tag{6.32c}$$

$$a_6 \stackrel{\text{def}}{=} \frac{\lambda_2\beta_2(\theta_1 + c_1) - c_1\lambda_1(\theta_2 + c_2)}{\lambda_2\beta_2(\theta_1 + c_1) - \beta_1\lambda_1(\theta_2 + c_2)} \tag{6.32d}$$

$$d \stackrel{\text{def}}{=} (\theta_2 + \alpha\beta_2)(\theta_1 + \alpha\beta_1) - (1-\alpha)^2\beta_1\beta_2 \tag{6.32e}$$

We also define the elementary conditions

$$(\mathcal{C}1): \quad \lambda_1(c_1\theta_2+\beta_1\beta_2) \le \lambda_2\beta_2(\theta_1+c_1) \quad \text{and} \quad 0 \le \alpha < a_3 \tag{6.33}$$

$$(\mathcal{C}2): \quad c_2\lambda_2(\theta_1+c_1) \ge \lambda_1\beta_1(\theta_2+c_2) \quad \text{or} \quad a_4 \le \alpha \le 1 \tag{6.34}$$

$$(\mathcal{C}3): \quad \lambda_2(c_2\theta_1+\beta_2\beta_1) \le \lambda_1\beta_1(\theta_2+c_2) \quad \text{and} \quad 0 \le \alpha < a_5 \tag{6.35}$$

$$(\mathcal{C}4): \quad c_1\lambda_1(\theta_2+c_2) \ge \lambda_2\beta_2(\theta_1+c_1) \quad \text{or} \quad a_6 \le \alpha \le 1. \tag{6.36}$$

Furthermore, let us define the following set of conditions

$$(\mathcal{D}2) = (\mathcal{C}1) \cap (\mathcal{C}2) \tag{6.37a}$$

$$(\mathcal{D}3) = (\mathcal{C}3) \cap (\mathcal{C}4) \tag{6.37b}$$

$$(\mathcal{D}4) = (\text{not } (\mathcal{C}1)) \cap (\text{ not } (\mathcal{C}3)). \tag{6.37c}$$

The above definitions imply that $(\mathcal{D}4) \cap (\mathcal{D}2) = (\mathcal{D}4) \cap (\mathcal{D}3) = \emptyset$, where $\emptyset$ denotes the empty set. However, $(\mathcal{D}2) \cap (\mathcal{D}3)$ is not necessarily empty, so that $(\mathcal{D}2)$ and $(\mathcal{D}3)$ may hold simultaneously for some sets of parameters. Finally, we define the two-dimensional vectors $\mathbf{x}_i$, $i = 2, 3, 4$, by

$$\mathbf{x}_2 = \left(\frac{\lambda_1}{c_1+\theta_1}, \frac{\lambda_2 - (1-\alpha)\beta_1 \frac{\lambda_1}{c_1+\theta_1}}{\theta_2+\alpha\beta_2} \right) \tag{6.38a}$$

$$\mathbf{x}_3 = \left(\frac{\lambda_1 - (1-\alpha)\beta_2 \frac{\lambda_2}{c_2+\theta_2}}{\theta_1+\alpha\beta_1}, \frac{\lambda_2}{\theta_2+c_2} \right) \tag{6.38b}$$

$$\mathbf{x}_4 = \left(\frac{\lambda_1(\theta_2+\alpha\beta_2) - (1-\alpha)\lambda_2\beta_2}{d}, \frac{\lambda_2(\theta_1+\alpha\beta_1) - (1-\alpha)\lambda_1\beta_1}{d} \right) \tag{6.38c}$$

where d is defined in (6.32e). Proposition 6.5.1 below investigates the steady-state behavior of the switched system (6.29)-(6.30).

Proposition 6.5.1 (Equilibrium for bandwidth diversity) *The system of differential equations (6.29)-(6.30) has a unique equilibrium point* $\bar{\mathbf{x}}$ *given by*

$$\bar{\mathbf{x}} = \begin{cases} \mathbf{x}_2^T & \textit{regardless of } \mathbf{x}(0), \textit{ if } (\mathcal{D}2) \textit{ holds and } (\mathcal{D}3) \textit{ does not hold} \\ \mathbf{x}_3^T & \textit{regardless of } \mathbf{x}(0), \textit{ if } (\mathcal{D}3) \textit{ holds and } (\mathcal{D}2) \textit{ does not hold} \\ \mathbf{x}_4^T & \textit{regardless of } \mathbf{x}(0), \textit{ if } (\mathcal{D}4) \textit{ holds} \\ \mathbf{x}_2^T \textit{ or } \mathbf{x}_3^T & \textit{depending on } \mathbf{x}(0), \textit{ if } (\mathcal{D}2) \textit{ and } (\mathcal{D}3) \textit{ hold simultaneously.} \end{cases} \tag{6.39}$$

$\diamond$

Proof. As in Section 6.4, we first assume that $\lim_{t\uparrow\infty} \mathbf{x}(t)$ exists and check that it is given by (6.39). Letting $t \to \infty$ in (6.10) yields (6.20), where σ is now given by (6.31).

We consider separately the four systems of linear equations obtained from (6.20) when (a) $\sigma(\bar{\mathbf{x}}) = 1$, (b) $\sigma(\bar{\mathbf{x}}) = 2$, (c) $\sigma(\bar{\mathbf{x}}) = 3$ and (d) $\sigma(\bar{\mathbf{x}}) = 4$.

(a) When $\sigma(\bar{\mathbf{x}}) = 1$ or equivalently $c_1\bar{x}_1 < \alpha\beta_1\bar{x}_1 + (1-\alpha)\beta_2\bar{x}_2$ and $c_2\bar{x}_2 < (1-\alpha)\beta_1\bar{x}_1 + \alpha\beta_2\bar{x}_2$:

the download rate is the bottleneck for both classes of peers. We find

$$\bar{\mathbf{x}}^T = \left(\frac{\lambda_1}{\theta_1 + c_1}, \frac{\lambda_2}{\theta_2 + c_2}\right) \tag{6.40}$$

(b) When $\sigma(\bar{\mathbf{x}}) = 2$ or equivalently $c_1\bar{x}_1 < \alpha\beta_1\bar{x}_1 + (1-\alpha)\beta_2\bar{x}_2$ and $c_2\bar{x}_2 \geq (1-\alpha)\beta_1\bar{x}_1 + \alpha\beta_2\bar{x}_2$:

the bottleneck is the download rate for class-1 peers and the upload rate for class-2 peers. We find

$$\bar{\mathbf{x}}^T = \left(\frac{\lambda_1}{\theta_1 + c_1}, \frac{\lambda_2 - (1-\alpha)\beta_1 \frac{\lambda_1}{c_1+\theta_1}}{\theta_2 + \alpha\beta_2}\right) \tag{6.41}$$

(c) When $\sigma(\bar{\mathbf{x}}) = 3$ or equivalently $c_1\bar{x}_1 \geq \alpha\beta_1\bar{x}_1 + (1-\alpha)\beta_2\bar{x}_2$ and $c_2\bar{x}_2 < (1-\alpha)\beta_1\bar{x}_1 + \alpha\beta_2\bar{x}_2$:

the bottleneck is the download rate for peers of class 2 and the upload rate for peers of class 1. In this case

$$\bar{\mathbf{x}}^T = \left(\frac{\lambda_1 - (1-\alpha)\beta_2 \frac{\lambda_2}{c_2+\theta_2}}{\theta_1 + \alpha\beta_1}, \frac{\lambda_2}{\theta_2 + c_2}\right) \tag{6.42}$$

(d) When $\sigma(\bar{\mathbf{x}}) = 4$ or equivalently $c_1\bar{x}_1 \geq \alpha\beta_1\bar{x}_1 + (1-\alpha)\beta_2\bar{x}_2$ and $c_2\bar{x}_2 \geq (1-\alpha)\beta_1\bar{x}_1 + \alpha\beta_2\bar{x}_2$:

the bottleneck is the download rate for both classes of peers. The stationary solution is

$$\bar{\mathbf{x}}^T = \left(\frac{\lambda_1(\theta_2 + \alpha\beta_2) - (1-\alpha)\lambda_2\beta_2}{d}, \frac{\lambda_2(\theta_1 + \alpha\beta_1) - (1-\alpha)\lambda_1\beta_1}{d}\right) \tag{6.43}$$

where d is defined in (6.32e).

The next step is to check if a type-i equilibrium may exist, namely, if $\sigma(\bar{\mathbf{x}}) = 1$ (resp. $\sigma(\bar{\mathbf{x}}) = 2$, $\sigma(\bar{\mathbf{x}}) = 3$, $\sigma(\bar{\mathbf{x}}) = 4$) when $\bar{\mathbf{x}}$ is given by (6.40) (resp. (6.41), (6.42), (6.43)).

It is easily seen that a type-1 equilibrium may only exist if $c_1\lambda_1 + c_2\lambda_2 \leq \beta_1\lambda_1 + \beta_2\lambda_2$. Since $c_i > \beta_i$ for $i = 1, 2$ (see (6.9)) we conclude that there is no type-1 equilibrium, where both classes would saturate their download capacity. A simple analysis, similar to that in Appendix G, shows that a type-2 equilibrium only exists if (6.33) and (6.34) are true, and that a type-3 equilibrium only exists if (6.35) and (6.36) are true. For the existence conditions of a type-4 equilibrium, we also use the stability condition (6.49) below, in addition to $\sigma(\bar{\mathbf{x}}) = 4$, to get the following condition: (not (6.33)) and (not (6.35)). It happens that conditions for $\sigma = 2$ and $\sigma = 3$ are not mutually exclusive. When they are simultaneously satisfied (i.e., $(\mathcal{D}2) \cap (\mathcal{D}3)$ holds) then the steady-state is given either by (6.41) or by (6.42) depending on the initial conditions.

We now turn to the proof that $\lim_{t\uparrow\infty} \bar{\mathbf{x}}(t)$ exists. Namely, we investigate the nature of the equilibrium of each of the linear systems $\dot{\mathbf{x}}(t) = A_i\,\mathbf{x}(t) + \mathbf{b}$, for $i = 2, 3, 4$, with

$$A_2 = \begin{pmatrix} -\theta_1 - c_1 & 0 \\ -(1-\alpha)\beta_1 & -\theta_2 - \alpha\beta_2 \end{pmatrix} \tag{6.44}$$

$$A_3 = \begin{pmatrix} -\theta_1 - \alpha\beta_1 & -(1-\alpha)\beta_2 \\ 0 & -\theta_2 - c_2 \end{pmatrix} \tag{6.45}$$

and

$$A_4 = \begin{pmatrix} -\theta_1 - \alpha\beta_1 & -(1-\alpha)\beta_2 \\ -(1-\alpha)\beta_1 & -\theta_2 - \alpha\beta_2 \end{pmatrix} \tag{6.46}$$

It is easily seen that the matrices A_2 and A_3 have two strictly negative eigenvalues. The eigenvalues of the matrix A_4 are the roots in λ of the polynomial

$$\begin{aligned} det(A_4 - \lambda I) &= (\theta_1 + \alpha\beta_1 + \lambda)(\theta_2 + \alpha\beta_2 + \lambda) - (1-\alpha)^2\beta_1\beta_2 && (6.47) \\ &= \lambda^2 + \lambda(\theta_1 + \alpha\beta_1 + \theta_2 + \alpha\beta_2) + d && (6.48) \end{aligned}$$

where I denotes the 2×2 identity matrix. The roots of this polynomial have strictly negative real parts if and only if their product is strictly positive and their sum is

strictly negative, which is equivalent to

$$d > 0 \tag{6.49}$$

This shows that all equilibria are stable, which concludes the proof. ■

We now compute the download costs ϕ_1 and ϕ_2 associated with each equilibrium point found in Proposition 6.5.1. In order to simplify the notation, we introduce the following two-dimensional vectors

$$\varphi_2 = \left(\frac{1}{c_1}, \quad \frac{\lambda_2(\theta_1 + c_1) - (1-\alpha)\lambda_1\beta_1}{\theta_2\lambda_1\beta_1 + \alpha(\lambda_2\beta_2(\theta_1 + c_1) - \lambda_1\beta_1\theta_2)}\right) \tag{6.50a}$$

$$\varphi_3 = \left(\frac{\lambda_1(\theta_2 + c_2) - (1-\alpha)\lambda_2\beta_2}{\theta_1\lambda_2\beta_2 + \alpha(\lambda_1\beta_1(\theta_2 + c_2) - \theta_1\lambda_2\beta_2)}, \quad \frac{1}{c_2}\right) \tag{6.50b}$$

$$\varphi_4 = \left(\frac{(\lambda_1\theta_2 - \lambda_2\beta_2 + \alpha\beta_2(\lambda_1 + \lambda_2))}{\beta_2(\lambda_2\theta_1 - \lambda_1\beta_1) + \alpha(\lambda_1\beta_1(\theta_2 + 2\beta_2) - \theta_1\lambda_2\beta_2)}, \right. \tag{6.50c}$$

$$\left. \frac{\lambda_2\theta_1 - \lambda_1\beta_1 + \alpha\beta_1(\lambda_1 + \lambda_2)}{\beta_1(\lambda_1\theta_2 - \lambda_2\beta_2) + \alpha(\lambda_2\beta_2(\theta_1 + 2\beta_1) - \theta_2\lambda_1\beta_1)}\right) \tag{6.50d}$$

The next proposition partially characterizes the download costs ϕ_1 and ϕ_2.

Proposition 6.5.2 (Download costs for bandwidth diversity) *In a no-seed model, the vector of download costs (ϕ_1, ϕ_2) in the bandwidth diversity problem is given by*

$$(\phi_1, \phi_2) = \begin{cases} \varphi_2 & \text{regardless of } \mathbf{x}(0), \text{ if } (\mathcal{D}2) \text{ holds and } (\mathcal{D}3) \text{ does not hold} \\ \varphi_3 & \text{regardless of } \mathbf{x}(0), \text{ if } (\mathcal{D}3) \text{ holds and } (\mathcal{D}2) \text{ does not hold} \\ \varphi_4 & \text{regardless of } \mathbf{x}(0), \text{ if } (\mathcal{D}4) \text{ holds} \\ \varphi_2 \text{ or } \varphi_3 & \text{depending on } \mathbf{x}(0), \text{ if } (\mathcal{D}2) \text{ and } (\mathcal{D}3) \text{ hold simultaneously.} \end{cases}$$

◇

Proof. The proof directly follows from Proposition 6.5.1 and (6.12). ■

6.5.1 How can we minimize the highest download cost?

In the bandwidth diversity problem, several optimization problems could be considered. For instance, one may wish to find an allocation α that yields the same download costs.

Another objective could be to minimize a linear combination of the download costs. However, as shown in Proposition 6.5.2, it is difficult to analytically determine ϕ_1 and ϕ_2 whenever $(\mathcal{D}2) \cap (\mathcal{D}3) \neq \emptyset$, and thereby to solve the above optimization problems.

Instead, we will seek to minimize the maximum download cost over all initial states and over all classes. To this end, we introduce the mapping $\alpha \to E(\alpha)$, called the *envelope function*, defined by

$$E(\alpha) = \max_{\sigma \in \{2,3,4\}} \max_{i \in \{1,2\}} \phi_i \tag{6.51}$$

Our objective is to minimize the envelope function as a function of α.

We now use Proposition 6.5.2 to calculate the value of α that minimizes $E(\alpha)$. In Figures 6.5 and 6.6, the envelope function is represented along with the possible values of (ϕ_1, ϕ_2) for α in $(0, 1)$, for two different set of physical parameters.

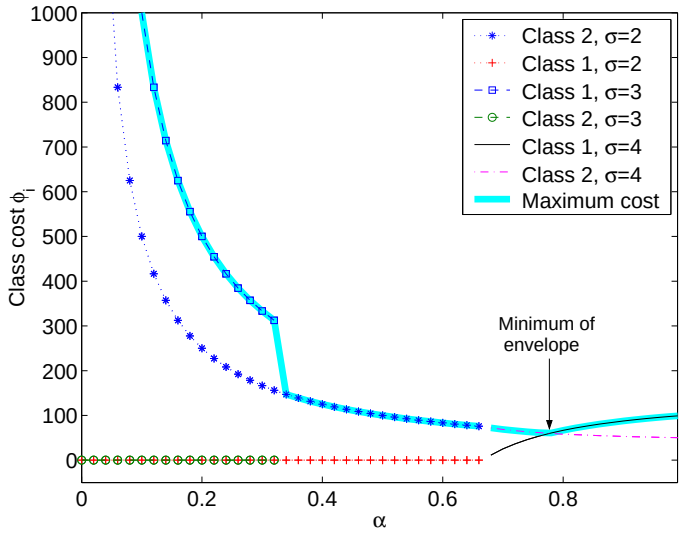

Figure 6.5: Minimum of maximum download cost achieved for $\alpha \approx 0.78$. $(\lambda_1 = \frac{\lambda_2}{2} = 10, \beta_1 = \frac{\beta_2}{2} = 10^{-2}, c_1 = \frac{c_2}{2} = 400, \theta_1 = \theta_2 = 10^{-5})$

In Figure 6.5, we observe that $E(\alpha)$ is minimal for a single value of α, when $\sigma = 4$ and $\phi_1 = \phi_2$. In this case, the exact value of α that minimizes the maximum download cost can be found by solving $\phi_1 = \phi_2$ using Proposition 6.5.2. Note that in Figure 6.5, we have both type-2 and 3 equilibria for $\alpha \leq 0.32$. The steady-state is then determined by initial conditions.

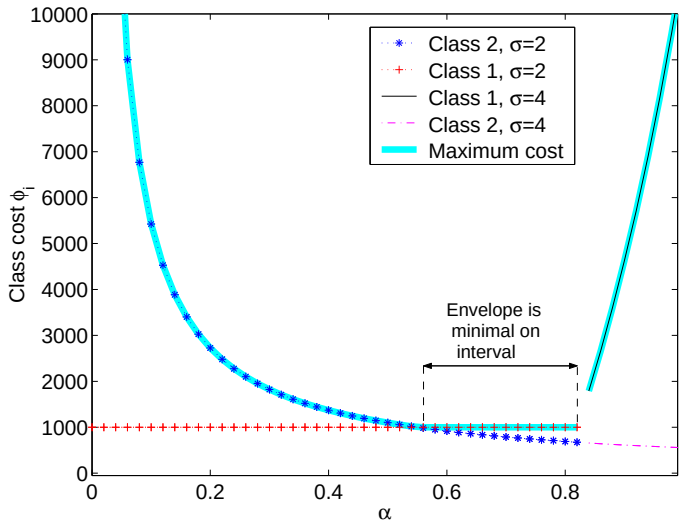

Figure 6.6: Minimum of maximum download cost achieved for a whole interval $[0.5502, 0.8207]$. $(\lambda_1 = \frac{\lambda_2}{4} = 10^{-1}$, $\theta_1 = 2\theta_2 = \beta_1 = \frac{\beta_2}{20} = 10^{-4}$, $c_1 = \frac{c_2}{2} = 10^{-3})$

In Figure 6.6, $E(\alpha)$ is minimal on a whole interval on which it is equal to the constant download cost ϕ_1 when $\sigma = 2$. In this case, the interval can also be determined using Proposition 6.5.2, by solving $\phi_1 = \phi_2$ for $\sigma = 2$ for the lower bound, and by determining the maximum value of α that satisfies (6.33) and (6.34) for the upper bound. Note that in Figure 6.6, condition (6.35) and (6.36) are never satisfied simultaneously with this set of physical parameters, since we do not have a type-3 equilibrium.

In any case, finding the value of α that minimizes the worst download cost, amounts to solve a linear or quadratic equation $\phi_1 = \phi_2$ using the appropriate expression in Proposition 6.5.2. We conclude that for a given physical set of parameters, it is possible to account for bandwidth diversity in BitTorrent-like networks through parameter α.

6.6 Conclusions and Perspectives

In this chapter we presented a simple multiclass fluid model for BitTorrent-like distribution systems. We successfully applied this model to account for two specific problems: service differentiation and bandwidth diversity. We mainly focused our attention to the special case where peers selfishly leave the system immediately after their down-

load ("no-seed case"). For both the service differentiation and bandwidth diversity problems, we have defined a single parameter α that defines a resource allocation strategy. We showed how this parameter affects the steady-state of the system and provided closed-form expressions for the successful download time in each case. In addition, we showed how this parameter α can be chosen so as to achieve a target quality of service ratio (download time ratio) for the service differentiation problem. We also quantified the impact of the no-seed assumption on this result through a numerical resolution of the general problemand showed that when users stay for a reasonably short time in the system, the no-seed model gives accurate results and the performance of the system is not affected by the altruism of users. This last property is consistent with the findings in [MV05] and [QS04]. For the bandwidth diversity problem, we also showed how it is possible to choose parameter α so as to minimize the highest download time between two classes of peers.

Many open problems remain. For instance, though we have proved the local stability of each equilibrium, the global stability of the system has been observed rather than proved. Also, though the fluid approximation was experimentally validated in [QS04], we intend to compare the results of our multiclass model to a simulation of a real P2P file dissemination system. Another problem for further research is the study of dynamic resource allocation, where α would depend on the class population.

Chapter 7

Conclusion

7.1 Summary and Contributions

In this thesis we proposed tractable fluid models for analyzing the performance of several content distribution systems. The fluid approximation reduces drastically the complexity of these highly dynamic systems and offers simple means to accurately estimate their performance. It also provides significant insight of the qualitative behavior of these systems.

In the first part of the thesis we considered distributed caching systems. We proposed in Chapter 2 a generic stochastic fluid framework which replaces the total cached content by a quantity of fluid, under the main assumption that each document is cached at a single place in the system. This framework is able to model various architectures of distributed caching systems.

We then applied this model in Chapter 3 to analyze the performance of CARP-like cache clusters. We modeled the number of active caches by an Engset model. We found an explicit expression of the hit probability in the case of homogeneous document popularity. Our formula proves very accurate when compared to discrete-

event simulation and is much more tractable than simulation (i.e. several orders of magnitude faster to compute). Our analytical model is able to capture the key tradeoffs of the cache cluster system, in particular the utmost importance of the time-to-live and number of documents versus the total request rate. The model also allowed us to compare quantitatively two direction policies of the caching system.

In Chapter 4 we used the fluid framework introduced in Chapter 2 to study a peer-to-peer caching system called Squirrel. Although similar in spirit to the mathematical analysis in Chapter 3, the calculation of the hit probability is different for two main reasons: the total request rate now depends on the number of active nodes, and the total amount of cached fluid is not degraded when a node goes up. We provided an efficient mean for computing the hit probability in the case of a uniform document popularity distribution, within a limit of 10,000 nodes. Similarly to Chapter 3 we identified two critical parameters which represent the major tradeoffs for Squirrel performance. We also validated the accuracy of the model as well as the observed degrees of freedom through a comparison with discrete-even simulation

In Chapter 5 we extended our analysis of Squirrel to relax the 10,000 nodes tractability restriction on the size of the network and to account for a more realistic popularity distribution of objects. We also considered the possibility of announced departures and not only abrupt failures. To this end, we first replaced the Engset model by an $M/M/\infty$ queuing model for the number of active nodes. This allowed us to find a closed-form expression for the hit probability which is now tractable (and immediate) even for very large networks (e.g., a million nodes). This modified model also allowed us to evaluate quantitatively the benefit of announcing departures. Then, we relaxed the uniform popularity assumption and used a more realistic Zipf-like popularity distribution. Since this distribution makes the evolution equation nonlinear, we opted for a clustering approximation where the set of documents is divided into a number of popularity classes. We explained how to dimension these classes and validated this approximation through a comparison with discrete-event simulation.

In the second part of this thesis, we used a second fluid model which relaxes the assumption that a document is not replicated in several locations of the content distribution system. This new model does not replace documents with fluid but instead replaces users (nodes) with fluid.

Based on the fluid model in [QS04], in Chapter 6 we proposed a multiclass fluid

approximation which is designed to model peer-to-peer file sharing systems like BitTorrent. Our multiclass model is a complex extension of [QS04] since the evolution equations are nonlinear and may admit several stationary solutions. Due to the complexity of these models we considered a practical worst-case in which users leave the system immediately upon download completion, instead of staying alive and contributing additional data and bandwidth to the system. We used this model to address two practical problems: service differentiation and bandwith diversity, for a static bandwidth allocation strategy. In the first problem we showed the existence of a unique stable equilibrium and gave a closed-form expression of the expected download time. Our model also gives a simple means to compute the bandwidth allocation strategy which achieves a target service differentiation ratio. Furthermore we have quantified the impact of our worst-case approximation and found that under reasonable circumstances the altruism of users does not modify the performance of the system. For the bandwidth diversity problem, we showed that the equilibrium may depend on initial conditions. We provided closed-form expressions of the possible values of this equilibrium and showed that the bandwidth allocation parameter can be chosen so as to minimize the worst download time.

These contributions have led to the following publications:

[CNR05b] F. Clévenot, P. Nain and K.W. Ross. Stochastic fluid models for cache clusters. *P*erformance Evaluation, 59(1):1-18, January 2005.

[CN04] F. Clévenot and P. Nain. A simple model for the analysis of the Squirrel peer-to-peer caching system. In *P*roceedings of IEEE Infocom 2004, Hong-Kong, March 2004.

[CN05] F. Clévenot-Perronnin and P. Nain. Stochastic fluid model for P2P caching evaluation. I *P*roceedings of the 10th International Workshop on Web Content Caching and Distribution (WCW'05), Sophia Antipolis, September 2005. To appear.

[CNR05a] F. Clévenot-Perronnin, P. Nain and K.W. Ross. Multiclass P2P networks: static resource allocation for service differentiation and bandwidth diversity. In *P*roceedings of the 24th International Symposium on Computer performance, Modeling, Measurements and Evaluation (Performance 2005), Juan-les-Pins, October 2005. To appear.

7.2 Perpectives

We already outlined several possible extensions and improvements of our work at the end of Chapters 3 to 6. We now propose more general research directions.

Our work focused on two types of CDS, namely, caching systems and peer-to-peer file sharing systems. A natural question is whether simple macroscopic models, as used in this thesis, would apply to the third type of CDS described in Chapter 1, the content distribution networks (CDNs), and what insight they could bring on their performance. Unlike caching systems studied in the first part of this thesis, CDNs rely on document replication at strategic locations worldwide. This makes the model of Cha[pter 2 unsuited for these systems. Nor can the client-based fluid model used in Part II be directly applied since a CDN differentiates between clients and servers (unlike P2P file sharing), so the server dynamics also need to be taken into account. As a result, CDNs would require a specific model. Their instrinsic complexity pleads for simple macroscopic models, since detailed models would soon become untractable.

However, CDNs exhibit novel complexity issues and performance factors. For instance, the mapping of requests to content servers depends on several criteria such as network conditions, server load, topological location and content of the server [DMP+02]. Several measurement and simulation studies of CDNs have evaluated their performance regarding the hit ratio [SGD+02] and TCP connexion times [JCDK01, KWZ01]. However, in [KWZ01] the authors show that, though CDNs can significantly reduce reponse times compared to origin servers, the additional DNS redirection latencies introduced by these systems are significant and may introduce noticeable performance degradation. These DNS costs are due to several factors and are inherent to distributed systems. First, the embedded objects of a single HTML page may be stored at different locations. One reason for this is the frequent inclusion of one dynamic (and uncacheable) object while all other objects are static and may be stored in the system [DMP+02]. This results in additional DNS queries and may overload DNS servers. Note that this DNS bottleneck problem has also been observed in the deployment of Squirrel [Rod04]. Another cause of DNS overhead is the load balancing feature. For instance, Akamai uses DNS servers to redirect requests when the load on a given server reaches a threshold [DMP+02]. Since these DNS costs may actually degrade the performance of CDNs [KWZ01], an interesting problem would be to incorporate the DNS costs with a specific model to the analysis of CDNs hit rate or reponse time.

On a more general perspective, we believe our methodology may also be applied to cooperative systems other than CDS. In particular, the Grid is a natural candidate for this type of large-scale analysis since it involves large number of cooperative computers and exhibits performance issues which may be similar to P2P systems [LSSH03]. In particular, grid computing not only involves the pooling of computational resources, but focuses on the large-scale sharing of various resources, including storage space and sets of files.

Appendix A

Stationary Distribution of the Node Process at Jump Times

A.1 Stationary Distribution π of the Engset Model at Jump Times

In this section we compute the limiting distribution of the Markov chain $\{N_n, n \geq 1\}$. Let $P = [p_{i,j}]_{0 \leq i,j \leq N}$ be its transition probability matrix. We have $p_{i,i+1} = \rho(N - i)/(\rho(N-i)+i)$ for $i = 0, 1, \ldots, N-1$, $p_{i,i-1} = i/(\rho(N-i)+i)$ for $i = 1, 2, \ldots, N$ and $p_{i,j} = 0$ otherwise.

Since this Markov chain[1] has a finite-state space and is irreducible, it is positive recurrent Therefore, it possesses a unique stationary distribution $\pi = (\pi_0, \cdots, \pi_N)$ given by the (unique) solution of the equation $\pi P = \pi$ such that $\sum_{i=0}^{N} \pi_i = 1$ [GS92, page 208].

We proceed by induction to compute π. From the equation $\pi P = \pi$ we find that

[1] Note that this chain is period (with period 2).

$\pi_1 = (\rho(N-1)+1)\pi_0$ and $\pi_2 = \frac{\rho(N-2)+2}{2}\rho(N-1)\pi_0$. This suggests that

$$\pi_j = \frac{\rho(N-j)+j}{j}\frac{\rho^{j-1}}{(j-1)!}\frac{(N-1)!}{(N-j)!}\pi_0 \tag{A.1}$$

for $j = 1, 2, \ldots, N$. Let us assume that (A.1) holds for $j = 1, 2, \ldots, i < N-1$. Let us show that it still holds for $j = i+1$. We have

$$\pi_{i+1} = \frac{\rho(N-(i+1))+i+1}{i+1}\left(\pi_i - \frac{\rho(N-(i-1))}{\rho(N-(i-1))+i-1}\pi_{i-1}\right) \tag{A.2}$$

$$= \frac{\rho(N-(i+1))+i+1}{i+1}\left(\frac{(\rho(N-i)+i)\,\rho^{i-1}}{i!(N-i)!}\right.$$

$$\left. -\frac{\rho(N-(i-1))}{\rho(N-(i-1))+i-1}\frac{(i-1+\rho(N-i+1))\,\rho^{i-2}}{(i-1)(i-2)!(N-i+1)!}\right)(N-1)!\pi_0 \tag{A.3}$$

$$= \frac{\rho(N-(i+1))+i+1}{i+1}\frac{\rho^i(N-1)!}{i!(N-(i+1))!}\pi_0$$

where (A.3) follows from the induction hypothesis. The constant π_0 is computed by using the normalizing condition $\sum_{i=0}^{N}\pi_i = 1$; we find $\pi_0 = 1/(2(1+\rho)^{N-1})$ as announced in (3.4). Plugging this value of π_0 into (A.1) gives (3.5). ■

A.2 Stationary Distribution π of the M/M/∞ Model at Jump Times

In this section we compute the invariant distribution of the Markov chain $\{N_n, n \geq 1\}$. Let $P = [p_{i,j}]_{i,j\geq 0}$ be its transition probability matrix. We have : $p_{i,i+1} = \rho/(\rho+i)$ for $i \geq 0$, $p_{i,i-1} = i/(\rho+i)$ for $i \geq 1$ and $p_{i,j} = 0$ if $|j-i| \neq 1$.

Note that this chain is periodic with period 2. Since this Markov chain has a finite-state space and is irreducible, it is positive recurrent [Ç75, Cor. 5.3.19, 5.3.22]. Therefore, it possesses a unique stationary distribution $\pi = (\pi_0, \pi_1 \cdots)$ given by the (unique) solution of the equation $\pi P = \pi$ such that $\sum_{i=0}^{\infty}\pi_i = 1$ [GS92, page 208].

We proceed by induction to compute π. From the equation $\pi P = \pi$ we find that $\pi_1 = \frac{1}{1+\rho}\pi_0$ and $\pi_2 = \frac{\rho+2}{2}\rho\pi_0$. This suggests that

$$\pi_j = \frac{\rho+j}{j!}\rho^{j-1}\pi_0 \tag{A.4}$$

for $j = 1, 2, \ldots, N$. Let us assume that (A.4) holds for $j = 1, 2, \ldots, i$, with $i \geq 2$. We now show that it still holds for $j = i + 1$. We have

$$\begin{aligned} \pi_{i+1} &= \frac{\rho + i + 1}{i + 1} \left(\pi_i - \frac{\rho}{\rho + i - 1} \pi_{i-1} \right) && \text{(A.5)} \\ &= \frac{\rho + i + 1}{i + 1} \left(\frac{\rho + i}{i!} \rho^{i-1} - \frac{\rho}{\rho + i - 1} \times \frac{i - 1 + \rho}{(i-1)!} \rho^{i-2} \right) \pi_0 && \text{(A.6)} \\ &= \frac{\rho + i + 1}{i + 1} \frac{\rho^i}{i!} \pi_0 && \text{(A.7)} \end{aligned}$$

where (A.6) follows from the induction hypothesis. The constant π_0 is computed by using the normalizing condition $\sum_{i=0}^{N} \pi_i = 1$; we find $\pi_0 = e^{-\rho}/2$ as announced in (5.4). Plugging this value of π_0 into (A.4) gives (5.5). ■

Appendix B

Uniqueness of the solution of the tridiagonal systems (3.15) and (4.10)

B.1 Uniqueness of the solution of (3.15)

The linear system (3.15) defined in Proposition 3.3.1 admits a unique solution if and only if $\det(A) \neq 0$. Since A is a tridiagonal matrix we can use the LU decomposition [HJ85, Sec. 3.5] $A = LU$ with

$$L = \begin{pmatrix} l_1 & 0 & \cdots & 0 \\ \beta_2 & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & \beta_n & l_n \end{pmatrix} \qquad U = \begin{pmatrix} 1 & u_1 & \cdots & 0 \\ 0 & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & u_{n-1} \\ 0 & 0 & \cdots & 1 \end{pmatrix} \tag{B.1}$$

where l_i's and u_i's are defined as follows:

$$\begin{aligned} a_{1,1} &= l_1 \\ a_{i,i} &= l_i + a_{i,i-1}u_{i-1}, \quad i = 2, \ldots, N \\ l_i u_i &= a_{i,i+1}, \quad i = 1, \ldots, N-1. \end{aligned} \tag{B.2}$$

Both matrices L and U being bidiagonal matrices it follows that $\det(A) \neq 0$ if and only if $l_i \neq 0$ for $i = 1, 2, \ldots, N$.

We use an induction argument to show that $l_i \neq 0$ for $i = 1, 2, \ldots, N$. We have $l_1 = \gamma + \rho(N-1) + 1$. Assume that $l_i > \gamma + \rho(N-i)$ for $i = 1, 2, \ldots, n < N-1$ and let us show that $l_{n+1} > \gamma + \rho(N-n-1)$. We have

$$\begin{aligned} l_{n+1} &= a_{n+1,n+1} - \frac{a_{n+1,n}a_{n,n+1}}{l_n} && \text{(B.3)} \\ &= \gamma + \rho(N-n-1) + (n+1)\frac{l_n - \rho(N-n)\Delta_u(n)\Delta_d(n+1)}{l_n} && \text{(B.4)} \\ &> \gamma + \rho(N-n-1) && \text{(B.5)} \end{aligned}$$

by using the induction hypothesis along with the fact that $0 \leq \Delta_u(n)\Delta_d(n+1) \leq 1$. ■

B.2 Uniqueness of the solution of (4.10)

We use an induction argument to show that $l_i \neq 0$ for $i = 1, 2, \ldots, N$. We have $l_1 = \rho(N-1)+1+\gamma(1+\alpha)$. We assume that $l_i > \gamma(i+\alpha)+\rho(N-i)$ for $i = 1, 2, \ldots, n < N-1$. Let us show that $l_{n+1} > \gamma(n+1+\alpha) + \rho(N-n-1)$. We have

$$\begin{aligned} l_{n+1} &= a_{n+1,n+1} - \frac{a_{n+1,n}a_{n,n+1}}{l_n} && \text{(B.6)} \\ &= \gamma(n+1+\alpha) + \rho(N-n-1) && \text{(B.7)} \\ &\quad +(n+1)\frac{l_n - \rho(N-n)\Delta_u(n)\Delta_d(n+1)}{l_n} && \text{(B.8)} \\ &> \gamma(n+1+\alpha) + \rho(N-n-1) && \text{(B.9)} \end{aligned}$$

by using the induction hypothesis along with the fact that $0 \leq \Delta_u(n)\Delta_d(n+1) \leq 1$. ■

Appendix C

Proof of equation (4.14)

Recall that $v_i = \lim_{n\to\infty} \dfrac{\mathbb{E}\left[Y_n \,|\, N_n = i\right]}{c}$ for $1 \le i \le N$. With (4.6) we have

$$
\begin{aligned}
v_i &= \frac{1}{c} \lim_{n\to\infty} \mathbb{E}\left[Y_n \,|\, N_n = i\right] && \text{(C.1)}\\
&= \frac{1}{c} \lim_{n\to\infty} \mathbb{E}\left[\eta_i + (X_n - \eta_i)\, e^{-(T_{n+1}-T_n)\frac{\sigma i}{\eta_i}} | N_n = i\right] && \text{(C.2)}\\
&= \frac{1}{c} \lim_{n\to\infty} \left(\eta_i \times \frac{\alpha\gamma + \gamma i}{\alpha\gamma + \gamma i + \rho(N-i) + i} \frac{(\rho(N-i)+i)\mathbb{E}\left[X_n \,|\, N_n = i\right]}{\rho(N-i) + i + \alpha\gamma + \gamma i}\right) && \text{(C.3)}
\end{aligned}
$$

To derive (C.3) we have used the fact that, given $N_n = i$, the random variables X_n and $T_{n+1} - T_n$ are independent, and $T_{n+1} - T_n$ is exponentially distributed with parameter $(N-i)\lambda + \mu i$.

Let us now evaluate $\lim_{n\to\infty} \mathbb{E}[X_n \,|\, N_n = i]$ for $1 \le i \le N$. Conditioning on N_{n-1}

we have

$$\lim_{n\to\infty} \mathbb{E}[X_n \,|\, N_n = i] = \lim_{n\to\infty} \mathbb{E}\left[X_n | N_n = i, N_{n-1} = i-1\right] \mathbb{P}[N_{n-1} = i-1 | N_n = i]$$
$$+ \lim_{n\to\infty} \mathbb{E}\left[X_n \,|\, N_n = i, N_{n-1} = i+1\right] \mathbb{P}[N_{n-1} = i+1 \,|\, N_n = i]\, \mathbb{1}_{[i<N]} \quad \text{(C.4)}$$

$$= \Delta_u(i-1) \lim_{n\to\infty} \mathbb{E}\left[Y_{n-1} | N_{n-1} = i-1\right] \frac{\pi_{i-1}}{\pi_i} \frac{\rho(N-i+1)}{\rho(N-i+1)+i-1}$$
$$+ \Delta_d(i+1) \lim_{n\to\infty} \mathbb{E}\left[Y_{n-1} | N_{n-1} = i+1\right] \frac{\pi_{i+1}}{\pi_i} \frac{i+1}{\rho(N-i-1)+i+1} \mathbb{1}_{[i<N]} \quad \text{(C.5)}$$

$$= c \frac{i\Delta_u(i-1)v_{i-1} + \Delta_d(i+1)v_{i+1}\rho(N-i)}{\rho(N-i)+i} \quad \text{(C.6)}$$

by using (3.5) and the definition of v_i. Finally, dividing both sides by c and introducing (C.6) into (C.3) yields (4.14). ■

Appendix D

Proof of equation (4.17)

Let us determine $\mathbb{E}[X^\infty]$ from the v_is. We use the Palm formula and condition on the value of N_0. From (4.15), (4.6), (3.27) we find

$$\mathbb{E}[X^\infty] = \Lambda \sum_{i=1}^{N} \pi_i \mathbb{E}^0 \left[\int_0^{T_1} \left(\eta_i + (X_0 - \eta_i) e^{-t\frac{\sigma i}{\eta_i}} \right) dt | N_0 = i \right] \tag{D.1}$$

$$= \Lambda \left[\sum_{i=1}^{N} \pi_i \eta_i \mathbb{E}^0[T_1 | N_0 = i] + \sum_{i=1}^{N} \frac{\pi_i \eta_i}{\sigma i} \mathbb{E}^0 \left[(X_0 - \eta_i) \left(1 - e^{-T_1 \frac{\sigma i}{\eta_i}} \right) | N_0 = i \right] \right] \tag{D.2}$$

$$= \Lambda \left[\sum_{i=1}^{N} \pi_i \eta_i \frac{1}{\lambda(N-i) + \mu i} \right.$$
$$\left. + \sum_{i=1}^{N} \frac{\pi_i \eta_i}{\sigma i} \left(\mathbb{E}^0[X_0 | N_0 = i] - \eta_i \right) \left(1 - \mathbb{E}^0 \left[e^{-T_1 \frac{\sigma i}{\eta_i}} | N_0 = i \right] \right) \right] \tag{D.3}$$

$$= \frac{\Lambda}{\mu} \left[\sum_{i=1}^{N} \pi_i \eta_i \frac{1}{\rho(N-i) + i} \sum_{i=1}^{N} \pi_i \frac{\mathbb{E}^0[X_0 \,|\, N_0 = i] - \eta_i}{\rho(N-i) + i + \alpha\gamma + \gamma i} \right] \tag{D.4}$$

$$= \frac{2N\rho}{1+\rho} \sum_{i=1}^{N} \pi_i \left[\eta_i \frac{1}{\rho(N-i) + i} \frac{\mathbb{E}^0[X_0 \,|\, N_0 = i] - \eta_i}{\rho(N-i) + \alpha\gamma + (\gamma+1)i} \right] \tag{D.5}$$

By definition, $\mathbb{E}^0[X_0 \,|\, N_0 = i] = \lim_{n\to\infty} \mathbb{E}[X_n \,|\, N_n = i]$, which has been computed in (C.6). By combining (C.6) and (4.14) we obtain

$$\mathbb{E}^0[X_0 \,|\, N_0 = i] = c\,\frac{(\rho(N-i) + i + \alpha\gamma + i\gamma)v_i - i\frac{\sigma}{\mu}}{\rho(N-i) + i} \tag{D.6}$$

Plugging this value of $\mathbb{E}^0[X_0 \,|\, N_0 = i]$ into the r.h.s. of (D.5), and using (3.5), yields after some straightforward algebra:

$$\mathbb{E}\,[X^\infty] = \frac{c}{(1+\rho)^N} \sum_{i=1}^{N} \binom{N}{i} \rho^i v_i \tag{D.7}$$

which is nothing but (4.17). ■

Appendix E

Proof of Proposition 5.3.1

We compute the stationary hit probability which is given by (4.7). As in Chapters 3 and 4 the idea of the proof is to first compute the expected amount of cached fluid just before a jump in the process $\{N(t)\}$ conditioned on the value of $N(t)$ just before this jump, and then to invoke Palm calculus to deduce the expected amount of cached fluid at any time. Therefore we use Y_n again as the amount of cached fluid just before the $(n+1)$-st jump in the process $\{N(t)\}$

$$Y_n = X(T_{n+1}-) \tag{E.1}$$

We first compute v_i as defined by (4.13) (i.e. $v_i = \lim_{n\to\infty}(1/c)\mathbb{E}\left[Y_n \,|\, N_n = i\right]$ for $i \geq 1$). We use the fact that, given $N_n = i$, the random variables X_n and $T_{n+1} - T_n$ are independent, and $T_{n+1} - T_n$ is exponentially distributed with parameter $\lambda + \mu i$:

$$\begin{aligned}
\lim_{n\uparrow\infty} \mathbb{E}\left[Y_n | N_n = i\right] &= \lim_{n\uparrow\infty} \mathbb{E}\Bigg[\frac{\sigma N_n}{\frac{\sigma N_n}{c} + \theta} \\
&\quad + \left(X_n - \frac{\sigma N_n}{\frac{\sigma N_n}{c} + \theta} \right) e^{-(T_{n+1}-T_n)(\theta + \sigma N_n/c)} | N_n = i \Bigg] \qquad \text{(E.2)} \\
&= \lim_{n\uparrow\infty} \frac{\left(\frac{\sigma i}{\frac{\sigma i}{c}+\theta}\right)\left(\frac{\theta}{\mu} + \frac{\sigma i}{\mu c}\right)}{\frac{\theta}{\mu} + \frac{\sigma i}{\mu c} + \rho + i} + \frac{(\rho + i)\,\mathbb{E}\left[X_n \,|\, N_n = i\right]}{\rho + i + \frac{\theta}{\mu} + \frac{\sigma i}{\mu c}} \qquad \text{(E.3)}
\end{aligned}$$

We now evaluate $\lim_{n\uparrow\infty} \mathbb{E}[X_n \,|\, N_n = i]$ for $i \geq 1$. Conditioning on N_{n-1} we have

$$\begin{aligned}
\lim_{n\uparrow\infty} \mathbb{E}[X_n|N_n = i] &= \lim_{n\uparrow\infty} \mathbb{E}\left[X_n|N_n = i, N_{n-1} = i-1\right] P\left(N_{n-1} = i-1|N_n = i\right) \\
&\quad + \lim_{n\uparrow\infty} \mathbb{E}\left[X_n \,|\, N_n = i, N_{n-1} = i+1\right] P\left(N_{n-1} = i+1 \,|\, N_n = i\right) \qquad \text{(E.4)} \\
&= \Delta_u(i-1) \lim_{n\uparrow\infty} \mathbb{E}[Y_{n-1} \,|\, N_{n-1} = i-1] \frac{\pi_{i-1}}{\pi_i} \frac{\rho}{\rho+i-1} \\
&\quad + \Delta_d(i+1) \lim_{n\uparrow\infty} \mathbb{E}[Y_{n-1}, \,|\, N_{n-1} = i+1] \frac{\pi_{i+1}}{\pi_i} \frac{i+1}{\rho+i+1} \qquad \text{(E.5)} \\
&= c \frac{i\Delta_u(i-1)v_{i-1} + \Delta_d(i+1)v_{i+1}\,\rho}{\rho+i} \qquad \text{(E.6)}
\end{aligned}$$

by using (5.5) and the definition of v_i. Finally, dividing both sides by c and introducing (E.6) into (E.3) yields

$$(\rho + i + \frac{\theta}{\mu} + \frac{\sigma i}{\mu c})\, v_i = \frac{\sigma i}{c\mu} + i\Delta_u(i-1)v_{i-1} + \rho\Delta_d(i+1)v_{i+1} \qquad \text{(E.7)}$$

for $i \geq 1$, or equivalently (5.8) by using (4.4).

Equation (5.8) gives the conditional stationary expected amount of fluid correctly cached just before jump epochs. Similarly to Chapters 3 and 4 we use Palm calculus to deduce the expected amount of fluid in stationary state at arbitrary instants. Recall that it is given by

$$\mathbb{E}\left[X^\infty\right] = \Lambda \mathbb{E}^0 \left[\int_0^{T_1} X(t)dt\right] \qquad \text{(E.8)}$$

where $\mathbb{E}^0$ denotes the expectation with respect to the Palm distribution, T_1 denotes the time of the first jump after 0, and where Λ denotes the global rate of the M/M/∞ model:

$$\Lambda = \frac{1}{\mathbb{E}^0[T_1]}. \qquad \text{(E.9)}$$

From now on we assume that the system is in steady-state at time 0. Under the Palm distribution we denote by N_{-1} and Y_{-1} the number of up caches and the amount of correctly cached fluid respectively, just before time 0 (i.e. just before the jump to occur at time 0).

We first compute $1/\Lambda$ for the M/M/∞ model. We have

$$\frac{1}{\Lambda} = \sum_{i=0}^{\infty} \pi_i\, \mathbb{E}^0[T_1 \,|\, N_0 = i] = \frac{1}{\mu} \sum_{i=0}^{\infty} \frac{\pi_i}{\rho+i} = \frac{1}{2\rho\mu} \qquad \text{(E.10)}$$

by using (5.4)-(5.5).

We now determine $\mathbb{E}[X^\infty]$. From (E.8), (4.3), (E.10) we find

$$\begin{aligned}
\mathbb{E}[X^\infty] &= \Lambda \sum_{i=1}^{N} \pi_i \mathbb{E}^0 \left[\int_0^{T_1} \left(\frac{\sigma N_0}{\frac{\sigma N_0}{c} + \theta} + \left(X_0 - \frac{\sigma N_0}{\frac{\sigma N_0}{c} + \theta} \right) e^{-t(\theta + \sigma N_0/c)} \right) dt | N_0 = i \right] \\
&= \Lambda \left[\sum_{i=1}^{\infty} \pi_i \frac{\sigma i}{\frac{\sigma i}{c} + \theta} \mathbb{E}^0[T_1 \,|\, N_0 = i] \right. \\
&\quad \left. + \sum_{i=1}^{\infty} \frac{\pi_i}{\theta + \sigma i/c} \mathbb{E}^0 \left[\left(X_0 - \frac{\sigma i}{\frac{\sigma i}{c} + \theta} \right) \left(1 - e^{-T_1(\theta + \sigma i/c)} \right) \,|\, N_0 = i \right] \right] \qquad \text{(E.11)} \\
&= \Lambda \left[\sum_{i=1}^{\infty} \pi_i \frac{\sigma i}{\theta + \frac{\sigma i}{c}} \frac{1}{\lambda + \mu i} + \sum_{i=1}^{\infty} \frac{\pi_i}{\theta + \sigma i/c} \left(\mathbb{E}^0[X_0 \,|\, N_0 = i] - \frac{\sigma i}{\frac{\sigma i}{c} + \theta} \right) \right. \\
&\quad \left. \times \left(1 - \mathbb{E}^0 \left[e^{-T_1(\theta + \sigma i/c)} \,|\, N_0 = i \right] \right) \right] \qquad \text{(E.12)} \\
&= \frac{\Lambda}{\mu} \left[\sum_{i=1}^{\infty} \pi_i \frac{\sigma i}{\theta + \frac{\sigma i}{c}} \frac{1}{\rho + i} + \sum_{i=1}^{\infty} \pi_i \frac{\mathbb{E}^0[X_0 \,|\, N_0 = i] - \frac{\sigma i}{\theta + \frac{\sigma i}{c}}}{\rho + i + \frac{\theta}{\mu} + \frac{\sigma i}{\mu c}} \right] \qquad \text{(E.13)} \\
&= 2\rho \sum_{i=1}^{\infty} \pi_i \left[\frac{\sigma i}{\theta + \frac{\sigma i}{c}} \frac{1}{\rho + i} + \frac{\mathbb{E}^0[X_0 \,|\, N_0 = i] - \frac{\sigma i}{\theta + \frac{\sigma i}{c}}}{\rho + i + \frac{\theta}{\mu} + \frac{\sigma i}{\mu c}} \right] \qquad \text{(E.14)}
\end{aligned}$$

By definition, $\mathbb{E}^0[X_0 \,|\, N_0 = i] = \lim_{n \uparrow \infty} \mathbb{E}[X_n \,|\, N_n = i]$, which has been computed in (E.6). By combining (E.6) and (5.8) we obtain

$$\mathbb{E}^0[X_0 \,|\, N_0 = i] = \frac{(\rho + i + \frac{\theta}{\mu} + i\frac{\sigma}{\mu c}) c v_i - i\frac{\sigma}{\mu}}{\rho + i} \qquad \text{(E.15)}$$

Plugging this value of $\mathbb{E}^0[X_0 \,|\, N_0 = i]$ into the r.h.s. of (E.14), and using (5.5), yields after some straightforward algebra

$$\mathbb{E}[X^\infty] = c e^{-\rho} \sum_{i=1}^{\infty} \frac{\rho^i}{i!} v_i \qquad \text{(E.16)}$$

According to (4.7) it remains to divide both sides of (E.16) by c to get (5.7). This concludes the proof. ■

Appendix F

Engset and M/M/∞ Models

In the Engset model, every user independently goes down (resp. up) after an exponentially distributed time with parameter μ (resp. λ_E). The total number of users (connected or not) is N. Let us denote by $N_E(t)$ the state of the Engset model at time t. Observe that this Engset model and the M/M/∞ model introduced in Section 5.2 have the same death rate in state $i \in \mathbb{N}$, given by $i\mu$. Let us define $\rho_E = \lambda_E/\mu$.

We have for the Engset model [Kel79]:

$$P(N_E^\infty = i) = \binom{N}{i} \frac{\rho_E^i}{(1+\rho_E)^N}, \quad 1 \le i \le N \tag{F.1}$$

$$\mathbb{E}\left[N_E^\infty\right] = N \frac{\rho_E}{1+\rho_E} \tag{F.2}$$

as opposed to (5.2) and (5.3), respectively, for the M/M/∞ model.

In the following, for any mappings f and g, the shorthand

$$f(N) \sim g(N)$$

will stand for

$$\lim_{N\to\infty} f(N)/g(N) = 1$$

Proposition F.0.1 *Assuming that the mean number of active users in the M/M/∞ model and in the Engset model with N nodes are the same, namely,*

$$\rho = \frac{N\rho_E}{1+\rho_E}. \tag{F.3}$$

Then, as $N \to \infty$, the stationary distribution of the Engset model defined in (F.1) is equivalent to that of the M/M/∞ model given in (5.2), namely

$$P(N_E^\infty = i) \sim \frac{\rho^i}{i!} e^{-\rho} \tag{F.4}$$

for any $i \in \mathbb{N}$.

Proof. For any fixed $i \in \mathbb{N}$ we have from (F.1) and (F.3)

$$P(N_E^\infty = i) = \frac{N!}{i!(N-i)!} \frac{(\frac{\rho}{N-\rho})^i}{(1+\frac{\rho}{N-\rho})^N}. \tag{F.5}$$

Using the Stirling formula $N! \underset{N\to\infty}{\sim} (N/e)^N \sqrt{2\pi N}$ in (F.5) yields

$$P(N_E^\infty = i) \underset{N\to\infty}{\sim} e^{-i} \left(\frac{N}{N-i}\right)^N (N-i)^i \, \frac{\rho^i}{i!} \frac{1}{(N-\rho)^i} \left(\frac{N-\rho}{N}\right)^N \tag{F.6}$$

$$\underset{N\to\infty}{\sim} e^{-i} \frac{\rho^i}{i!} \frac{1}{(1-\frac{i}{N})^N} \left(1-\frac{\rho}{N}\right)^N. \tag{F.7}$$

With the identity $\lim_{N\to\infty} (1+x/N)^N = e^x$ applied to the last equation, we find (F.4), which completes the proof. ■

Appendix G

Service Differentiation in BitTorrent-like networks: Type-2 Equilibrium

In this appendix we show that a type-2 equilibrium exists for $\alpha \in [0,1]$ if and only if $a_2 < \alpha \leq 1$, where a_2 is defined in Section 6.4.

By definition, a type-2 equilibrium exists if $\bar{\mathbf{x}} = (\bar{x}_1, \bar{x}_2)$ given in (6.22) is such that $\sigma(\bar{\mathbf{x}}) = 2$, to which we need to add the condition that $\bar{x}_2 \geq 0$ (note that $\bar{x}_1$ is always nonnegative). Equivalently, we need to find the values of α in $[0,1]$ such that

$$
\begin{aligned}
c\xi &< \alpha\beta\left(\xi + \frac{\lambda_2 - (1-\alpha)\beta\xi}{\theta_2 + (1-\alpha)\beta}\right) && \text{(G.1)}\\
c\frac{\lambda_2 - (1-\alpha)\beta\xi}{\theta_2 + (1-\alpha)\beta} &\geq (1-\alpha)\beta\left(\xi + \frac{\lambda_2 - (1-\alpha)\beta\xi}{\theta_2 + (1-\alpha)\beta}\right) && \text{(G.2)}\\
\lambda_2 - (1-\alpha)\beta\xi &\geq 0 && \text{(G.3)}
\end{aligned}
$$

where we have set $\xi \stackrel{\text{def}}{=} \lambda_1/(\theta_1 + c)$. The first two conditions express the identity $\sigma(\bar{\mathbf{x}}) = 2$ and the third condition expresses the constraint $\bar{x}_2 \geq 0$.

Straightforward algebra shows that these conditions are simultaneously met for

$\alpha \in [0,1]$ if and only if

$$\alpha > \frac{c\lambda_1(\theta_2+\beta)}{D} \quad \text{and} \quad \alpha \geq \max\left(1-\frac{c\lambda_2(\theta_1+c)}{D}, 1-\frac{\lambda_2(\theta_1+c)}{\lambda_1\beta}\right) \tag{G.4}$$

where we recall that $D = \beta(\lambda_1(\theta_2+c)+\lambda_2(\theta_1+c))$.

Let us first compare $c\lambda_1(\theta_2+\beta))/D$ to $1-c\lambda_2(\theta_1+c))/D$. We have

$$\frac{c\lambda_1(\theta_2+\beta)}{D} - \left(1-\frac{c\lambda_2(\theta_1+c)}{D}\right) = \frac{1}{D}(c\lambda_1(\theta_2+\beta)+c\lambda_2(\theta_1+c)-D) \tag{G.5}$$

$$= \frac{1}{D}\left(c\left(\lambda_1(\theta_2+\beta)+\lambda_2(\theta_1+c)\right)-\beta(\lambda_1(\theta_2+c)+\lambda_2(\theta_1+c))\right) \tag{G.6}$$

$$= \frac{1}{D}(c-\beta)(\lambda_1\theta_2+\lambda_2(\theta_1+c)). \tag{G.7}$$

We have observed earlier in the proof of Proposition 6.4.1 that $c > \beta$, which shows that $c\lambda_1(\theta_2+\beta))/D > 1-c\lambda_2(\theta_1+c))/D$.

We now compare $c\lambda_1(\theta_2+\beta))/D$ to $1-\lambda_2(\theta_1+c)/(\lambda_1\beta)$. We have

$$\frac{c\lambda_1(\theta_2+\beta)}{D} - \left(1-\frac{\lambda_2(\theta_1+c)}{\lambda_1\beta}\right)$$

$$= \frac{1}{D\lambda_1\beta}\left(c\lambda_1^2\beta(\theta_2+\beta)-D\lambda_1\beta+\lambda_2(\theta_1+c)D\right) \tag{G.8}$$

$$= \frac{1}{D\lambda_1\beta}\left(\lambda_1\beta(c\lambda_1(\theta_2+\beta)-\beta\lambda_1(\theta_2+c)-\beta\lambda_2(\theta_1+c)+\lambda_2(\theta_1+c)(\theta_2+c)\right) \tag{G.9}$$

$$+\beta\lambda_2^2(\theta_1+c)^2\big) \tag{G.10}$$

$$= \frac{1}{D\lambda_1}\left(\lambda_1(\lambda_1\theta_2(c-\beta)+\lambda_2(\theta_1+c)(\theta_2+c-\beta))+\lambda_2^2(\theta_1+c)^2\right) > 0 \tag{G.11}$$

since $c > \beta$.

In summary we have shown that the conditions $\sigma(\bar{\mathbf{x}}) = 2$ and $\bar{x}_2 \geq 0$ will simultaneously hold for $\alpha \in [0,1]$ if and only if $\alpha > \min(1, c\lambda_1(\theta_2+\beta))/D) = a_2$, which is the announced result. ■

Appendix H

Présentation des Travaux de Thèse

H.1 Introduction

H.1.1 Systèmes de distribution de contenu

Considérons un ensemble de documents multimédia tels que des pages HTML, des images, musiques ou clips vidéo, proposés par un ensemble de serveurs Web une population de clients intéressés dans un réseau. Les systèmes de distribution de contenu (CDS) peuvent tre définis comme l'ensemble des systèmes qui facilitent la distribution de ces documents aux clients intéressés, par rapport une mesure de performance choisie. Les serveurs Web d'origine sont parfois également considérés comme des CDS [SGD+02]. Cependant, d'après la définition proposée ci-dessus nous limitons la classe des CDS aux seuls intermédiaires logiques entre les clients et serveurs Web.

Notons que cette notion d'intermédiaire est purement logique. Dans leur réalité physique, les CDS peuvent tre implémentés directement dans les clients eux-mmes, comme par exemple dans les réseaux peer-to-peer (P2P) tels que [Kaz] et Gnutella [Gnu], de mme que l'on peut les retrouver au niveau des serveurs comme dans certains réseaux de distribution de contenu comme Akamai [Aka, DMP+02]. Ils peuvent

également former un ensemble dédié de serveurs un niveau intermédiaire entre clients et serveurs, comme c'est le cas dans le principe des caches web. Ainsi, les systèmes de distribution de contenu dépassent le concept client-serveur qui a servi construire de nombreuses applications Internet (FTP, Telnet, navigation sur le Web...).

Après avoir défini le concept des systèmes de distribution de contenu, nous nous intéressons maintenant leur classification. Il existe actuellement trois grands types d'architectures conues pour faciliter la diffusion de contenu.

La première classe de CDS est la classe des systèmes de caches Web. Ces systèmes sont largement utilisés et peuvent aisément tre mis en œuvre sur les serveurs proxy de pratiquement n'importe quel réseau privé ou institutionnel. Ces systèmes sont fondés sur la simple observation suivante : un document récemment demandé a de fortes chances d'tre nouveau demandé dans un futur proche, en particulier étant donné le biais de la distribution de popularité des documents du Web [BCF+99]. Les serveurs de cache Web sont typiquement placés physiquement entre les utilisateurs finaux et les serveurs Web. Ces serveurs conservent une copie de chaque fichier demandé afin de pouvoir répondre directement les futures requtes pour ces mmes fichier, et ainsi économiser aux utilisateurs le temps de contacter le serveur d'origine.

Une seconde classe de CDS est la classe des systèmes d'échanges de fichiers. L'idée principale est qu'un fichier populaire téléchargé par un client c_i peut également intéresser un autre client c_j du même réseau local. Si c_j peut obtenir le fihcier directement de c_i, le délai perçu est fortement réduit tout en réduisant également la charge sur le serveur d'origine. C'est l'idée essentielle du concept pair-à-pair, également appelé "peer-to-peer" (P2P), où les clients servent également de serveurs pour les noeuds voisins. Dans ce cas, le CDS appartient physiquement au réseau client. Ces réseaux peer-to-peer sont récemment devenus la principale source de trafic sur Internet (cf. notamment [AG04, KBB+04]), principalement en rendant aisément disponible des contenus multimédia très populaires comme des fichiers musicaux ou des films vidéo. Dans les systèmes peer-to-peer, chaque noeud (peer) maintient un certain nombre de documents à la disponibilité des autres noeuds. Ces objets peuvent être localisés par différentes techniques, comme la diffusion de requêtes comme dans Gnutella [Gnu], l'utilisation de tables de hashage comme dans Chord [SMK+01] par exemple, ou même par la consultation d'un serveur centralisé comme dans la première version de Napster (cf. notamment [SGG03] pour une description de cette architecture).

La troisième et dernière catégorie de CDS est la classe des réseaux de distribution de contenu (CDN). Ces réseaux sont conçus pour accélérer la distribution de contenu et pour diminuer la charge sur les serveurs Web, en dupliquant leur contenu et en le rendant ainsi accessible sur d'autres serveurs. Ce principe est assez différent de celui des caches Web en particulier sur les deux points suivants. Premièrement, les CDN sont des réseaux privés qui offrent des services aux serveurs Web, tandis qu'un système de cache est typiquement administré par le réseaux LAN client ou le réseau du serveur Web. Le service CDN typique comprend notamment une répartition géographique à des positions stratégiques, la disponibilité des serveurs et la gestion de contenu dynamique, tandis que les systèmes de caches n'offrent qu'un service local et un nombre limité de type de documents éligibles pour le cache. Deuxièmement, le contenu peut être envoyé aux réplicas CDN sur l'initiative du serveur Web, tandis que dans le principe du cache la copie est généralement faite sur requête d'un client. Les CDN peuvent être un réseau mondial de serveurs partagés, auquel cas leur localisation physique reflète leur rôle logique entre les clients et les serveurs, ou bien ils peuvent constituer une colleciton de serveurs localisés sur le réseau du serveur Web, auquel cas ils appartiennent physiquement au réseau du serveur bien que leur rôle logique soit inchangé.

H.1.2 Analyse de Performance

L'analyse de performance de ces CDS est cruciale pour de nombreuse raisons. Premièrement, en ce qui concerne les technologies émergentes comme de nouvelles architectures P2P par exemple, il est essentiel d'évaluer la performance et le passage à l'échelle au début du processus de développement, sous peine de déployer des systèmes inadaptés. Cela permet également d'anticiper les causes possibles de délais ou de surcoût de signalisation. L'analyse de performance permet également d'identifier les principaux compromis et de dimensionner ces systèmes efficacement. Enfin, l'analyse de performance de systèmes existants sert également pour concevoir de nouveaux services ou des systèmes concurrents pouvant apporter d'importantes améliorations. Elle peut également servir pour les problèmes de détermination de prix des services.

Cependant, les CDS présentent une complexité intrinsèque qui fait de l'analyse de leur performance un problème complexe. En effet, ces systèmes mettent en oeuvre un nombre croissant d'utilisateurs, de serveurs et de documents, qui en outre sont hétérogènes et fortement dynamiques. L'ordre de grandeur de la dimension d'un CDS

peut être évalué à l'aide de quelques chiffres clés. Par exemple, des caches institutionnels doivent pouvoir servir des dizaines de milliers d'utilisateurs pour un taux de requêtes total allant de 12 à 178 requêtes par seconde dans de grands systèmes [WVS+99, DMF97], choisis dans un Web qui contient des milliards de documents (environ 8 milliards de pages référencées par Google en juin 2005). Les CDN, eux, sont utilisés par une fraction significative des sites Web les plus populaires [KWZ01] et doivent donc faire face à de forts taux de requêtes pour des documents très volatiles. Quant aux systèmes P2P, ils mettent généralement en jeu des millions d'utilisateurs (statistiques sur [Edo, Sly, IUKB+04]) qui interrompent et reprennent fréquemment leur téléchargement[IUKB+04]. Le trafic total gèneré par ces systèmes représente plus de la moitié du trafic internet total [AG04]. En outre, les noeuds peuvent se déconnecter et revenir en service, ce qui peut modifier à la fois la population des caches, des clients et des serveurs, à des fréquences non négligeables: d'après [LMG95], de nombreux postes restent connectés une semaine et reviennent en service après une courte déconnexion. Ces taux de volatilité sont même plus élevés dans les systèmes P2P où les utilisateurs se connectent et se déconnectent plusieurs fois par jour [BSV03].

Pour toutes ces raisons, les outils classiques d'analyse de performance comme les modèles Markoviens ou la simulation à événements discrets souffrent d'un trop grand espace détat et nécessitent souvent de coûteuses méthodes de résolution numériques ou des simulations de modèles [ZA03, GFJ+03].

S'inspirant des travaux fondateurs de Anick, Mitra et Sondhi en 1982 [AMS82] et du succès des modèles fluides pour les réseaux de paquets (cf. notamment [EM92, EM93, KM01a, LZTK97, BBLO00, RRR02, LFG+01]), l'axe central de cette thèse est de proposer une approche fluide pour modéliser les systèmes de distribution de contenu, l'approximation fluide permettant de réduire l'espace d'état de ces systèmes.

H.1.3 Organisation et contributions de la thèse

La thèse se décompose en deux parties. Dans la première partie, composée des chapitres 2 à 5, nous proposons de modéliser les systèmes de caches distribués en remplaçant le contenu des caches par une quantité de fluide. En particulier, nous proposons un modèle générique pour ces systèmes, que nous appliquons à deux systèmes de caches différents. Nous montrons en particulier quelles informations qualitatives ce modèle peut apporter sur ces systèmes. Dans la deuxième partie, constituée du chapitre 6, nous proposons de

modéliser des systèmes P2P de partage de fichiers en remplaçant cette fois les utilisateurs par une quantité de fluide, ce qui permet de prendre en compte la duplication des fichiers dans ce type de systèmes. En particulier, nous montrons comment ce modèle permet de concevoir une offre de service différencié pour les utilisateurs, ou encore de tenir compte de l'hétérogénéité des utilisateurs en terme de bande passante. Dans les sections suivantes nous présentons plus en détail les principaux résultats de chacun des chapitres de cette thèse.

H.2 Un modèle fluide générique pour les caches distribués

H.2.1 Etat de l'art des systèmes de caches distribués

Dans le chapitre 2 nous commençons par présenter les systèmes de caches Web distribués et leurs principales caractéristiques. Notamment, nous montrons qu'il existe trois grands types d'architectures:

- les grappes de caches ("cache clusters"), qui sont des ensembles de serveurs dédiés à cette fonction dans un réseau institutionnel. Ces caches communiquent par un protocole ICP, auquel cas il peuvent stocker les mêmes fichiers, ou bien les documents à prendre en charge sont partitionnés entre les caches de manière à optimiser l'espace de stockage et à minimiser la signalisation, comme dans CARP [VR97].

- les systèmes hiérarchiques comme Harvest [CDN+96], dans lesquels un niveau de cache s'adresse au niveau supérieur lorsqu'il n'est pas en mesure de satisfaire une requête, et dans lesquels chaque document demandé est dupliqué à chaque niveau de cache qui voit passer la requête.

- Les systèmes de caches pair-à-pair (P2P), dans lequel chaque utilisateur partage son cache individuel pour former un système global de cache coopératif. Un exemple de tel système est Squirrel [IRD02].

H.2.2 Modèle fluide générique

En raison de leur complexité intrinsèque, peu d'études de performance de ces systèmes proposent un modèle analytique. En particulier, nous montrons que parmi les études analytiques existantes, aucune ne tient compte des incidents de connexion et déconnexions des usagers et des serveurs. Nous proposons donc un modèle fluide qui tient compte de cet aspect important. Les principales hypothèses de notre modèle générique sont les suivantes:

- le système permet une bonne répartition de la charge de requêtes entre les servers
- chaque document est présent à un seul noieud du système distribué. En particulier, cette hypothèse exclut les architectures commes ICP dans lesquelles plusieurs serveurs différents peuvent répondre à une même requête.

L'idée principale du moèle est de remplacer les dcuments cachés (i.e., stockés dans le cache) par une quantité globale de fluide. Cette quantité $X(t)$ représente le nombre total de documents stockés par le système de cache en fonction du temps.

Cette quantité de fluide évolue de la façon suivante. Tant que la population de serveurs de cache est inchangée, le nombre de documents stockés augmente au fur et à mesure des requêtes insatisfaites, i.e. de manière proportionnelle au taux global de requêtes $\sigma(t)$ et au taux de "miss" $1 - p_H(X(t))$ où $p_H(X(t))$ est le taux de "hit" (requêtes satisfaites par le cache distribués), mesure de performance du système et dépendant bien sûr de la quantité de fluide présente. Cette quantité de fluide diminue parallèlement, à mesure que les documents stockés arrivent à expiration. (Ces documents ont une durée de vie limitée, typiquement 24 heures, pour éviter de stocker et de servir des documents ayant éé modifiés depuis sur le serveur Web d'origine.) Notons par θ le taux de départ de ces documents.

Par ailleurs, cette quantité de fluide est modulée par le processus dévolution des serveurs de cache. Nous supposons que ces derniers suivent un processus de naissance et de mort. Nous notons $N(t)$ le nombre de noeuds actifs du cache à l'instant t. Lorsqu'un cache se déconnecte, la quantité de fluide qu'il contenait est perdue pour le système global. Ainsi, lors d'une déconnexion à un instant T_n, la quantité de fluide est réduite d'un facteur $\Delta_d(N_{T_n})$ compris entre 0 et 1. Ce facteur peut valoir 1 (aucune diminution de fluide) si les noeuds ont la faculté de prévoir leurs déconnexion et de transférer leur

contenu aux autres noeuds avant leur départ. De même, lorsqu'un cache se connecte, il peut devenir responsable d'un certain nombre de documents qu'il ne possède pas encore en cache, mais qui étaient déjà stockés dans les noeuds qui en étaient les anciens responsables. Ainsi lors d'une déconnexion à T_M, la quantité de fluide est réduite d'un facteur $\Delta_U(N_{T_M})$ compris entre 0 et 1.

Ainsi, emtre deux incidents de connexion ou déconnexion de caches, le fluide évolue de façon continue selon l'équation différentielle du premier ordre suivante:

$$\frac{dX}{dt} = \sigma(t)(1 - p_H(X(t)) - \theta X(t) \tag{H.1}$$

Lors des connexions et déconnexions, la quantité de fluide évolue de manière discontinue selon les facteurs Δ_u et Δ_d.

Ce modèle est utilisé dans les Chapitres 3 à 5 pour étudier différents types de caches distribués.

H.3 Application aux Grappes de Caches

H.3.1 Spécialisation du modèle

Dans le Chapitre 3 nous appliquons le modèle du Chapitre 2 pour étudier les grappes de caches (clusters) basées sur CARP. Nous supposons que les noeuds du cluster de caches suivent un processus d'Engset d'une taille maximale N, de taux de naissance individuel *lambda* et de taux de mort individuel μ. Notons $\rho = \lambda/\mu$. Dans tout ce chapitre nous supposons que la popularité des documents est uniforme, i.e., la probabilité de hit est simplement proportionnelle à la quantité de fluide: $p_H = \frac{X_t}{c}$ où c est le nombre total de documents existant dans l'univers.

Nous montrons que la probabilité de hit stationnaire du système est donnée par la formule suivante:

$$p_H = \frac{1}{(1+\alpha)(1+\rho)^N} \sum_{i=1}^{N} \binom{N}{i} \rho^i \, v_i \tag{H.2}$$

sous la condition

$$0 \leq \Delta_u(i)\Delta_d(i+1) \leq 1, \quad \text{pour } i = 0, 1, \ldots, N-1, \tag{H.3}$$

et où le vecteur $\mathbf{v} = (v_1, \ldots, v_N)^T$ est l'unique solution du système linéaire

$$A\,\mathbf{v} = \mathbf{b} \tag{H.4}$$

avec $\mathbf{b} = (b_1, \ldots, b_N)^T$ un vecteur dont les composantes sont données par $b_i = \gamma(1+\alpha)$ pour i=1,2,...,N, et $A = [a_{i,j}]_{1 \le i,j \le N}$ une matrice tridiagonale $N \times N$ dont les éléments non-nuls sont donnés par:

$$a_{i,i} = \gamma(1+\alpha) + i + \rho(N-i), \qquad 1 \le i \le N \tag{H.5a}$$

$$a_{i,i-1} = -i\Delta_u(i-1), \qquad 2 \le i \le N \tag{H.5b}$$

$$a_{i,i+1} = -\rho(N-i)\Delta_d(i+1), \qquad 1 \le i \le N-1. \tag{H.5c}$$

et où $\alpha = \frac{\theta c}{\sigma}$ et $\gamma = \frac{\sigma}{\mu c}$.

Nous remarquons que ce résultat analytique montre déjà que les performances du systèmes ne dépendent plus que de 4 degrés de liberté: N, ρ, α, γ au lieu des six paramètres initiaux $N, \mu, \lambda, \sigma, \theta, c$. C'est l'un des premiers résultats qualitatifs de notre modèle.

H.3.2 Résultats Expérimentaux

H.3.2.1 Résultats qualitatifs

Le modèle permet de mieux comprendre le fonctionnement intrinsèque du système. Outre le nombre restreint de degrés de liberté identifié à la section précédente, nous montrons en particulier que le paramètre crucial est α comme le montre la figure H.1. Cela implique en particulier que le facteur déterminant de performance est la durée de vie des documents dans le cache $1/\theta$. Enfin, nous comparons deux politiques de direction des requêtes (manière de répartir les requêtes entre les noeuds actifs) et nous montrons que l'une des deux permet d'obtenir de largement meilleures performances que l'autre.

H.3.2.2 Validation du modèle

Afin de valider les conclusions de notre modèle nous comparons ses prédictions avec une simulation à événements discrets du système. La figure H.2 nous montre clairement que

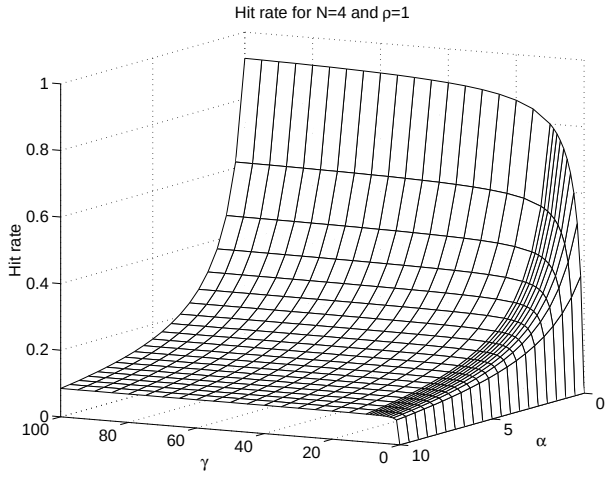

Figure H.1: Probabilité de hit d'une grappe de caches en fonction de γ et α ($\rho = 1$)

le modèle offre une estimation très précise de la probabilité de hit. L'un des principaux

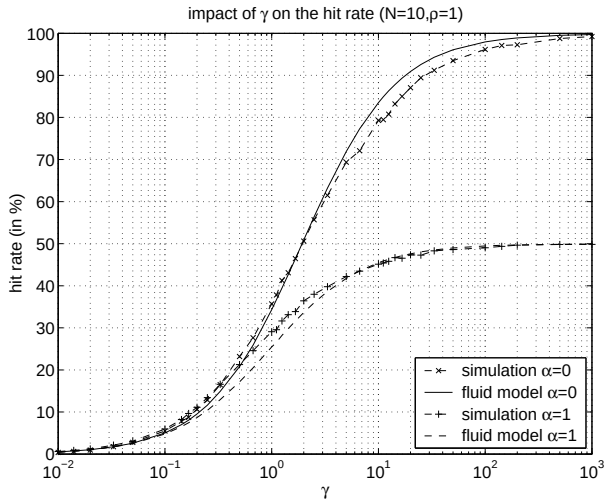

Figure H.2: Validation du modèle fluide par simulation: probabilité de hit en fonction de γ ($N = 10$ et $\rho = 1$).

intérêts du modèle est son faible coût de calcul: pour obtenir la figure H.2, la simlation de 10 noeuds peut prendre plusieurs heures, tandis que les résultats du modèles sont obtenus de façon instantanée.

H.4 Application au système Squirrel de cache P2P

H.4.1 Spécialisation du modèle

Dans le Chapitre 4 nous appliquons le modèle du Chapitre 2 pour étudier un système de cache P2P appelé Squirrel [IRD02]. Nous supposons que les noeuds du réseau (qui sont à la fois clients et serveurs) suivent également un processus d'Engset. Nous faisons également l'hypothèse d'une distribution de popularité uniforme des documents, et utilisons donc le modèle linéaire de probabilité de hit. Cette hypothèse sera levée au Chapitre 5.

La différence essentielle par rapport au chapitre précédent est que le taux global de requêtes dépend désormais de la population de noeuds actifs, ce qui modifie les équations d'évolution du système.

Nous montrons que la probabilité de hit stationnaire du système est donnée par la formule suivante:

$$p_H = \frac{1}{(1+\rho)^N} \sum_{i=1}^{N} \binom{N}{i} \rho^i \, v_i \tag{H.6}$$

sous la condition:

$$0 \leq \Delta_u(i)\Delta_d(i+1) \leq 1, \text{pour } i = 0, 1, \ldots, N-1, \tag{H.7}$$

et où le vecteur $\mathbf{v} = (v_1, \ldots, v_N)^T$ est l'unique solution du système linéaire

$$A\,\mathbf{v} = \mathbf{b} \tag{H.8}$$

avec $\mathbf{b} = (b_1, \ldots, b_N)^T$ un vecteur dont les composantes sont données par $b_i = \gamma i$ pour $1 \leq i \leq N$, et $A = [a_{i,j}]_{1 \leq i,j \leq N}$ une matrice tridiagonale $N \times N$ dont les éléments non-nuls sont donnés par:

$$a_{i,i} = \alpha\gamma + (\gamma+1)i + \rho(N-i), \qquad 1 \leq i \leq N \tag{H.9a}$$

$$a_{i,i-1} = -i\Delta_u(i-1), \qquad 2 \leq i \leq N \tag{H.9b}$$

$$a_{i,i+1} = -\rho(N-i)\Delta_d(i+1), \qquad 1 \leq i \leq N-1. \tag{H.9c}$$

et où $\alpha = \frac{\theta c}{\sigma}$ et $\gamma = \frac{\sigma}{\mu c}$.

De même qu'au chapitre précédent nous observons la réduction du nombre de degrés de liberté du système.

H.4.2 Résultats Expérimentaux

H.4.2.1 Résultats qualitatifs

Le modèle permet de mieux comprendre le fonctionnement intrinsèque du système Squirrel. Outre le nombre restreint de degrés de liberté identifié à la section précédente, nous montrons en particulier que le paramètre ρ exerce une très faible influence sur la probabilité de hit, excepté lorsqu'il est trés proche de zéro - ce qui en pratique est peu réaliste. Cette observation permet de réduire à 3 le nombre de réels degrés de liberté du système: N, α, γ.

Par ailleurs, la figure H.3 montre que le paramètre crucial est α. Cela implique en particulier que le facteur déterminant de performance est la durée de vie des documents dans le cache $1/\theta$.

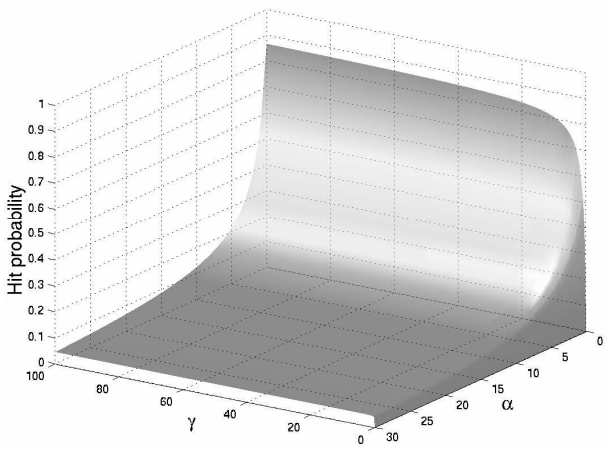

Figure H.3: Probabilité de hit du système Squirrel en fonction de γ et $\alpha(N = 3$ and $\rho = 1)$.

H.4.2.2 Validation du modèle

Afin de valider les conclusions de notre modèle nous comparons ses prédictions avec une simulation à événements discrets du système avec 10 noeuds. La figure H.4 nous montre clairement que le modèle offre une estimation très précise de la probabilité de hit. Là encore, l'un des principaux intérêts du modèle est son faible coût de calcul: pour

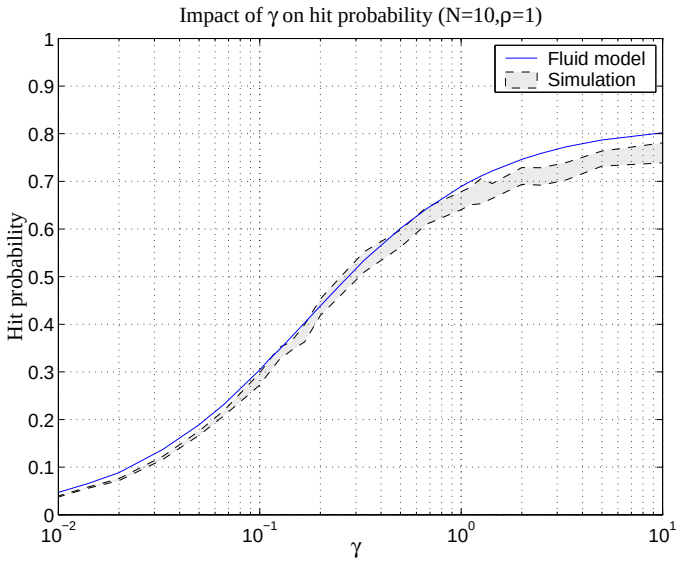

Figure H.4: Validation du modèle fluide de Squirrel par simulation: probabilité de hit en fonction de γ ($N = 10$ et $\rho = 1$)

obtenir la figure H.4, la simlation de 10 noeuds peut prendre plusieurs jours, tandis que les résultats du modèle sont obtenus de façon quasi-instantanée. Notons que cet ordre de grandeur est valable pour de petits réseaux. Lorsque la population augmente, le modèle continue de fournir une estimation efficace de la probabilité de hit pour des réseaux jusqu'à 10 000 noeuds. Au-delà, la complexité du modèle lié à la présence de coefficients binomiaux et d'exponentielles dans la formule de la probabilité de hit nécessite une adaptation du modèle. C'est l'un des sujets du chapitre suivant.

H.5 Extension aux grands réseaux et à d'autres distributions de popularité

Dans le Chapitre 5 nous proposons une variante du modèle de Squirrel permettant le passage à l'échelle en nombre de noeuds. Nous proposons aussi une méthode pour prendre en compte la popularité différenciée des documents.

H.5.1 Adaptation du modèle

Nous remplaçons le modèle d'Engset choisi au Chapitre 4 par un modèle M/M/∞. La motivation pour ce changement est qu'il n'est alors plus besoin de définir un nombre maximum N de noeuds dans le réseau, ce dernier pouvant grandir jusqu'à n'importe quelle taille. Ce modèle permet également, comme nous le verrons, de calculer aisément la probabilité de hit pour des réseaux pouvant aller jusqu'au million de noeuds.

Les noeuds sont donc modélisés par un processus M/M/∞ ayant pour taux de naissance global λ et pour taux de mort individuel μ. Notons que $\rho = \lambda/\mu$ a une signification différente du paramètre ρ utilisé dans les deux chapitres précédents.

H.5.1.1 Popularité uniforme

Sous l'hypothèse d'une popularité uniforme des documents (levée dans la section suivante), nous obtenons les formules closes suivantes pour la probabilité de hit :

Supposons que $\Delta_u(i) = 1$ (aucune perte de contenu lors de l'arrivée d'un noeud).

Si les noeuds ne sont pas en mesure d'annoncer leur départ (i.e. $\Delta_d(i) = (i-1)/i$) alors

$$p_H = e^{-\frac{\gamma\rho}{\gamma+1}} \gamma^{-(1+\kappa)} \int_{\frac{1}{\gamma+1}}^{1} \gamma\rho e^{\frac{\gamma\rho t}{\gamma+1}} \left(t(\gamma+1)-1\right)^{\kappa} \, dt \tag{H.10}$$

où $\kappa \stackrel{\text{def}}{=} \gamma(\alpha(\gamma+1)+\rho)/(\gamma+1)^2$.

Si les noeuds sont en mesure d'annoncer leur départ (i.e. $\Delta_d(i) = 1$) alors

$$p_H = \rho e^{-\frac{\rho\gamma}{\gamma+1}} \gamma^{-\nu} \int_{\frac{1}{\gamma+1}}^{1} (\gamma t e^{\rho t} - v_1) e^{-\frac{\rho t}{\gamma+1}} \left((\gamma+1)t-1\right)^{\nu-1} \, dt \tag{H.11}$$

avec $v_1 \stackrel{\text{def}}{=} \dfrac{\int_0^{1/(\gamma+1)} \gamma t e^{\frac{\rho\gamma t}{\gamma+1}} \left(1-(\gamma+1)t\right)^{\nu-1} \, dt}{\int_0^{1/(\gamma+1)} e^{\frac{\rho t}{\gamma+1}} \left(1-(\gamma+1)t\right)^{\nu-1} \, dt}$ et $\nu \stackrel{\text{def}}{=} \frac{\alpha\gamma(\gamma+1)+\rho}{(\gamma+1)^2}$.

H.5.1.2 Popularité de Zipf

En réalité, les documents Web n'ont pas tous la même popularité. La distribution de popularité est connue pour suivre une loi de type Zipf [BCF+99]. Cette distribution revient à modéliser la probabilité de hit par une fonction concave de type $p_H(t) = \left(\frac{X_t}{c}\right)^{\beta}$ où β est un coefficient typiquement compris entre 0 et 1.

Une telle expression de p_H rend malheureusement l'équation d'état:

$$\frac{dX}{dt} = \sigma N_t(1 - p_H(X(t)) - \theta X(t) \tag{H.12}$$

non linéaire, et qui plus est sans solution connue.

Nous proposons donc la méthode suivante. Nous divison l'ensemble des c documents existants en un nombre K de classes de popularité. A l'intérieur d'une même classe tous les documents sont supposés avoir la même popularité. La méthode de popularité linéaire peut donc être appliquée à chacune de ces classes.

La répartition des documents entre les classes est un problème classique de classification, et se résoud à l'aide d'un algorithme de Lloyd [GG92, page 189].

H.5.2 Résultats Expérimentaux

H.5.2.1 Résultats qualitatifs

Nous utilisons les formules obtenues par le modèle pour étudier le comportement du système selon certains paramètres. Tout d'abord nous montrons que l'impact de la popularité de Zipf est très important. Cette dernière augmente considérablement la probabilité de hit par rapport à une popularité uniforme.

Par ailleurs nous étudions en figure H.5 le gain de performance obtenu lorsque les noeuds sont capables d'annoncer leur départ. Nous constatons que ce gain existe bien mais qu'il reste relativement faible (environ 5% pour les paramètres utilisés). En particulier, ce gain est à comparer au coût de signalisation et de transfert induit par le fait d'annoncer un départ. Nous montrons par exemple que la dégradation de

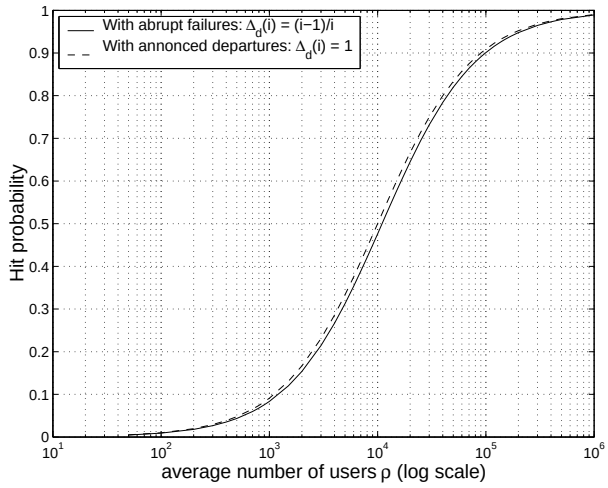

Figure H.5: Gain de performance entre départs annoncés et départs imprévus pour Squirrel.

performance dans le cas de départs imprévus a principalement lieu pour des durées de connexion inférieures à une dizaine de jours pour les paramètres considérés.

H.6 Un Modèle Multiclasses pour les Réseaux P2P

Dans le Chapitre 6, nous nous intéressons à la modélisation d'un autre type de systèmes, les systèmes P2P de partage de fichiers, et tout particulièrement ceux de type BitTorrent [Coh] qui sont conçus pour faire face au succès soudain de certains fichiers.

H.6.1 Présentation de BitTorrent

Le principe de ces systèmes est le suivant. Un fichier est découpé en un grand nombre N fragments de petite taille. Une source initiale répond aux requête des clients en diffusant les différents fragments aléatoirement aux clients, qui ensuite s'échangent leurs fragments directement entre eux. Ce principe permet d'utiliser la bande passante passante des clients eux-mêmes pour faire face à la demande, et donc d'obtenir un service dont la capacité s'accroît avec la demande. Les clients sont mis en contact les uns les autres par l'utilisation d'un serveur central de mise en relation appelé "tracker"

qui fait le suivi des différents téléchargements. Afin d'éviter que des clients obtiennent des fragments sans partager ceux qu'ils possèdent déjà, un mécanisme de réciprocité est mis en œuvre : chaque utilisateur envoie des fragments en priorité aux quatre pairs qui lui fournissent des fragments avec le meilleur débit. Ainsi, un client qui voudrait ne rien transmettre aurait beaucoup de mal à obtenir des fragments.

H.6.2 Modèle Multiclasses

S'appuyant sur le modèle de Qiu et Srikant [QS04], nous proposons le modèle fluide multiclasses suivant.

Il y a deux classes d'utilisateurs (classe de bande passante par exemple, ou de qualité de service). Dans chaque classe i, les clients possédant entre 0 et $N-1$ fragments sont appelés "leechers" et sont représentés par une quantité de fluide $x_i(t)$; les clients possédants N fragments sont appelés "seeds" et sont modélisés par une quantité de fluide $y_i(t)$. Les leechers rejoignent le système à un taux constant λ_i. Les seeds le quittent à un taux γ_i, i.e., il restent un temps $1/\gamma_i$ après avoir fini de télécharger le fichier complet. Typiquement, γ_i est très élevé si les utilisateurs sont individualistes. Les leechers abandonnent leur téléchargement à un taux θ_i qui peutêtre élevé [IUKB$^+$04]. Enfin, chaque utilisateur est connecté avec un débit ascendant μ_i et un débit descendant c_i.

Les clients de la classe i consacrent une fraction α_i de leur débit ascendant aux clients de leur propre classe et une fraction $1-\alpha_i$ aux clients de l'autre classe. Le paramètre η_i représente l'efficacité du système, soit la fraction de bande passante totale des leechers qui contribue à la diffusion des fragments. Idéalement ce paramètre doit être proche de 1.

L'évolution du fluide est la suivante. Les leechers de la classe i deviennent seeds à mesure qu'ils terminent leur téléchargement: ce téléchargement se fait aux taux

$$\max(c_i, \alpha_i\mu_i(\eta_i x_i + y_i) + (1-\alpha_j)\mu_j(\eta_j x_j + y_j)) \tag{H.13}$$

H.6.3 Différentiation de service

Le modèle ci-desus peut-être utilisés pour différents problèmes d'allocation de bande passante (détermination des paramètres α_1 et α_2).

Nous proposons d'étudier comment offrir un service différencié de téléchargement, i.e., par exmple, que la classe 1 puisse télécharger le document dans un temps k fois plus court que la classe 2.

Nous faisons donc les simplifications suivantes. Chaque classe favorise la classe 1 donc $\alpha_1 = 1 - \alpha_2 = \alpha$. Par ailleurs, nous faisons l'hypothèse (légèrement pessimiste) que les seeds quittent immédiatement le système, donc à tout instant $y_i = 0$, $i = 1, 2$. Nous considérons aussi que tous les clients ont la même bande passante, soit $\mu_1 = \mu_2$ et $c_1 = c_2$.

Nous montrons que sous la condition réaliste $\mu_i \leq c_i$, le système admet un unique équilibre stable quel que soit les conditions initiales du système, et que le temps de téléchargement complet moyen ϕ_i est donné par les formules closes suivantes:

$$\begin{array}{lll} \phi_1 = \dfrac{\lambda_1(\theta_2+c) - \alpha\lambda_2\beta}{\alpha\beta(\lambda_2\theta_1 + \lambda_1(\theta_2+c))}, & \phi_2 = \dfrac{1}{c} & \text{si } 0 \leq \alpha < a_1 \\ \phi_1 = \dfrac{\lambda_1(\theta_2+\beta) - \alpha\beta(\lambda_1+\lambda_2)}{\alpha\beta(\lambda_2\theta_1 + \lambda_1\theta_2)}, & \phi_2 = \dfrac{\lambda_2\theta_1 - \lambda_1\beta + \alpha\beta(\lambda_1+\lambda_2)}{(1-\alpha)\beta(\lambda_2\theta_1 + \lambda_1\theta_2)} & \text{si } a_1 \leq \alpha \leq a_2 \\ \phi_1 = \dfrac{1}{c}, & \phi_2 = \dfrac{\lambda_2(\theta_1+c) - \lambda_1\beta + \alpha\lambda_1\beta}{(1-\alpha)\beta(\theta_2\lambda_1 + \lambda_2(\theta_1+c))} & \text{si } a_2 < \alpha \leq 1. \end{array} \tag{H.14}$$

où a_1 et a_2 sont des constantes dépendant des paramètres du systèmes, données par les équations (6.16)-(6.17).

Pour atteindre la différentiation de service voulue, il suffit alors de résoudre en α l'équation $\phi_2/\phi_1 - k = 0$.

Nous calculons aussi numériquement l'impact de l'hypothèse de départ immédiat des seeds sur le ratio de temps de téléchargement entre les deux classes.

H.6.4 Accès hétérogènes

Dans cette section nous utilisons le modèle multiclasse pour déterminer la politique idéale d'allocation de bande passante (α_1, α_2) lorsque les clients souscrivent la même qualité de service, d'où $\alpha_1 = \alpha_2 = \alpha$ (noter que α n'a pas la même définition que dans la section précédente) mais que leurs débits d'accès sont hétérogènes: $\mu_1 \neq \mu_2$ et $c_1 \neq c_2$. Nous supposons à nouveau que $\mu_i \leq c_i$, $i = 1, 2$.

Notre objectif dans cette section est de minimiser le pire temps moyen de téléchargement parmi les deux classes.

Les résultats de cette section sont plus complexes car l'équilibre atteint par le système peut parfois dépendre des conditions initiales. Nous définissons donc un ensemble de conditions $(\mathcal{D}2), (\mathcal{D}3)$ et $(\mathcal{D}4)$ sur les paramètres. Ces définitions impliquent que $(\mathcal{D}4) \cap (\mathcal{D}2) = (\mathcal{D}4) \cap (\mathcal{D}3) = \emptyset$, mais que $(\mathcal{D}2) \cap (\mathcal{D}3)$ n'est pas nécéssairement vide.

Nous montrons que le système admet un unique équilibre stable qui est connu lorsque l'un des conditions $(\mathcal{D}2), (\mathcal{D}3)$ ou $(\mathcal{D}4)$ est satisfaite à l'exlusion des deux autres. Lorsque $(\mathcal{D}2)$ et $(\mathcal{D}3)$ sont satisfaite simultanément alors l'équilibre atteint dépend des conditions initiales du système.

En raison de cette incertitude sur le point d'équilibre atteint, nous proposons de choisir l'allocation α qui minimise l'enveloppe du temps de téléchargement, i.e. le maximum du temps ,moyen ϕ_i observé sur les deux classes et pour chaque équilibre possible selon les paramètres.

Notons σ le type déquilibre possible, défini ainsi:

- $\sigma = 2$ si la classe 1 est saturée en voie descendante et la classe 2 en voie ascendante
- $\sigma = 3$ si la classe 1 est saturée en voie ascendante et la classe 2 en voie descendante
- $\sigma = 3$ si les deux classes sont saturées en voie ascendante.

Notons que le cas $\sigma = 1$ où les deux classes seraient saturée en voie descendante n'admet pas d'équilibre stable en raison de l'hypothèse $\mu_i \leq c_i$.

Notre méthode est illustrée en figure H.6. La figure représente les temps de téléchargement des deux classes selon l'équilibre atteint, et trace la fonction enveloppe. Le minimum est ainsi atteint pour $\alpha \approx 0,78$ pour les paramètres choisis.

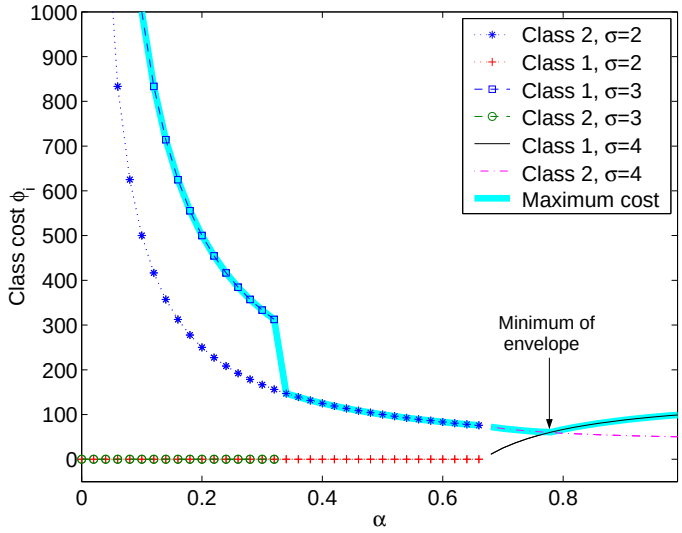

Figure H.6: Illustration de la méthode de l'enveloppe pour minimiser le plus grand temps moyen de téléchargement.

H.7 Conclusion et Perspectives

Dans cette thèse nous avons proposé des modèles fluides efficaces pour analyser différents systèmes de distribution de contenu. L'approximation fluide réduit largement la complexité de ces systèmes fortement dynamiques et permettent une estimation simple et précise de leur performances. Ces modèles offrent aussi une compréhension qualitative des paramètres clés de ces systèmes.

Dans la première partie de cette thèse nous nous sommes intéressés aux systèmes de caches distribués. Nous avons proposé en Chapitre 2 un modèle fluide générique remplaçant le nombre global de documents stockés par une quantité de fluide, sous l'hypothèse que chaque document ne soit stocké qu'en un seul exemplaire. Ce modèle générique est capable de représenter différentes architectures de caches distribués.

Nous avons ensuite appliqué ce modèle aux grappes de caches dans le Chapitre 3.

L'évolution des noeuds fut représentée par un modèle d'Engset. Nous avons trouvé une formule explicite de la probabilité de hit dans le cas d'une popularité homogène des documents. Notre formule donne des résultats très proches de la simulation du système, pour un temps de calcul très inférieur (de plusieurs ordres de grandeur). Notre modèle analytique permet aussi d'identifier les principaux compromis du système, en particulier l'importance cruciale de la durée de vie des documents dans le cache. Ce modèle nous a également permis de comparer quantitativement deux politiques de direction des requêtes.

Dans le Chapitre 4 nous avons appliqué le modèle fluide du Chapitre 2 à un système P2P de cache Web coopératif appelé Squirrel. Bien que l'analyse suive la même méthodologie qu'au Chapitre 3, les calculs de la probabilité de hit sont différents pour les raisons suivantes: le taux de requêtes dépend désormais du nombre de noeuds actifs, et la quantité total de fluide dans le cache n'est pas dégradée quand un noeud se connecte au système. Nous avons donné une expression permettant de calculer efficacement la probabilité de hit dans le cas d'une distribution uniforme de popularité des documemts, pour des réseaux pouvant aller jusqu'à l'ordre de 10 000 noeuds. De même qu'au Chapitre 3, nous avons identifié deux paramètres critiques qui représentent les compromis essentiels pour la performance de Squirrel. Nous avons également validé la pertinence du modèle en le comparant à une simulation à événements discrets.

Dans le Chapitre 5 nous avons étendu l'analyse de Squirrel pour dépasser la limite de 10 000 noeuds imposée par la complexité du premier modèle, ainsi que pour permettre de prendre en compte une distribution de popularité des objets plus réaliste. Nous avons également considéré la possibilité de départs annoncés et non plus simplements les pannes imprévues. Pour ce faire, nous avons remplacé le modèle d'Engset par une modèle $M/M/\infty$ pour représenter la population des noeuds actifs. Nous avons ainsi trouvé une formule close pour la probabilité de hit, ce qui nous permet désormais de calculer instantanément cette probabilité même pour des réseaux allant jusqu'au million de noeuds. Ce modèle nous a également permis de quantifier le gain obtenu lorsque les noeuds sont en mesure d'annoncer leur départ. Nous avons ensuite levé l'hypothèse de distribution uniforme de popularité des objets en considérant une popularité de Zipf, plus réaliste. Étant donné que ce modèle rend l'équation d'évolution non linéaire et sans solution connue, nous avons choisi une approximation multiclasse, dans laquelle l'ensemble des documents est divisé en un certain nombre de classes de popularité. Nous avons montré comment dimensionner ces classes et validé cette approximation en la comparant à une simulation de Squirrel utilisant une popularité de Zipf.

Dans la seconde partie de la thèse, nous avons utilisé un second modèle fluide qui permet cette fois de tenir compte de la duplication des documents dans le système. Ce nouveau modèle remplace cette fois les utilisateurs (et non les documents) par une quantité de fluide.

Le Chapitre 6 propose un modèle fluide multiclasses pour l'analyse de systèmes de partage de fichiers comme BitTorrent. Notre modle est une extension complexe du modèle de [QS04], car l'aspect multiclasse rend les équations d'évolution non linéaires et pouvant admettre plusieurs solutions stationnaires. En raison de la complexité de ces modèles nous avons considéré un cas pessimiste et réaliste dans lequel les utilisateurs quittent immédiatement le système après avoir terminé leur téléchargement, au lieu de rester connectés et de participer aux ressources du système. Nous avons utilisé ce modèle pour résoudre de manière statique deux problèmes pratiques: la différenciation de service et la diversité de bande passante, avec deux classes d'utilisateurs. Dans le premier problème nous avons montré l'existence d'un unique équilibre stable et donné un formule close pour le temps de téléchargement des deux classes. Notre modèle permet ainsi de calculer simplement la stratégie d'allocation de bande passante qui réalise un ratio prédéfini de qualité de service entre les deux classes. De plus nous avons quantifé l'impact de l'hypothèse pessimiste et montré que dans des cas réalistes l'altruisme des utilisateurs avait peu d'impact sur les performances du système. Pour le problème de diversité de bande passante nous avons montré que l'équilibre peut parfois dépendre des conditions initiales du système. Nous avons donné les formules closes des valeurs possibles de ces équilibres et nous avons montré comment choisir le paramètre d'allocation de bande passante de manière à minimiser le temps moyen de téléchargement le plus long entre les deux classes.

Plusieurs extensions possible de nos travaux ont été soulignées dans les conclusions des Chapitres 3 à 6. Nous proposons maintenant des directions de recherche plus générales.

Nos travaux se sont concentrés sur deux types de CDS: les systèmes de caches Web et les réseaux de partage de fichiers P2P. Il est donc naturel de se demander si des modèles macrocopiques simples comme ceux utiliés dans cette thèse pourrait permettre de modéliser le troisième type de CDS, à savoir les réseaux de distribution de contenu (CDN). À la différence des systèmes de cache étudiés dans la première partie de la thèse, les CDN s'appuient sur la duplication des documents en des emplacements stratégiques à l'échelle mondiale, ce qui rend le modèle du Chapitre 2 inadapté pour

ce type de systèmes. Le modèle de la partie II ne peut pas non plus être appliqué tel quel car les CDN différencient les clients des serveurs (contrairement au principe P2P), ce qui impose de tenir compte des dynamiques des serveurs. Un modèle spécifique devrait donc être défini. La complexité intrinsèque des CDN plaide en faveur de moèles macroscopiques car des modèles détaillés deviendraient vite trop complexe à calculer.

Or, les CDN mettent en œuvre de nouveaux facteurs de complexité et de nouveaux critères de performance. Notamment, la redirection des requêtes vers les serveurs de contenu dépend de plusieurs critères comme l'état du réseau, la charge des serveurs, la localisation topologique et le contenu du serveur. Plusieurs études de mesures et de simulation de CDN se sont intéressées au taux de hit [SGD+02] et au temps de connexion TCP [JCDK01, KWZ01]. Cependant, dans [KWZ01] les auteurs montrent que, bien que les CDN réduisent considérablement le temps de réponse par comparaison aux serveurs d'origine, les délais supplémentaires de redirection DNS qu'ils induisent sont significatifs et peuvent provoquer d'importantes dégradations de performance. Ces coûts DNS sont induits par différents facteurs et sont inhérents aux systèmes distribués. Premièrement les objects inclus dans une seule page HTML peuvent être stockés à différents endroits. En effet, il arrive souvent que l'un des objets inclus soit dynamique (et donc incachable) tandis que tous les autres objets sont statiques et peuvent être stockés dans le système [DMP+02]. Cela provoque donc des requêtres DNS additionnelles et risque de surcharger les serveurs DNS. Remarquons que ce goulot d'étranglement DNs a aussi été observé lors du déploiement de Squirrel [Rod04]. Une autre cause de délai DNS est la fonction de répartition de charge. Notamment, Akamai utilise des serveurs DNS pour rediriger des requêtes lorsque la charge d'un serveur donné dépasse un seuil [DMP+02]. Puisque ces coûts DNS peuvent réellement affecter la performance des CDN, un problème intéressant serait d'incorporer ces coûts DNS dans un modèle spécifique des CDN.

D'un point de vue plus général nous pensons que notre méthode peut être appliquée à d'autres systèmes coopératifs que les CDS. En particulier, les grilles de calcul sont un candidat naturel pour ce type d'analyse à grande échelle, car elles impliquent un grand nombre de machines coopérant et présentent des problèmes de performance qui peuvent être similaires aux systèmes P2P [LSSH03]. En effet, non seulement les grilles de calcul impliquent la mise en commun de ressources de calcul, mais elles posent le problème du partage à grande échelle de différentes ressources comme l'espace de stockage et des ensembles de fichiers.

List of Abbreviations

BT	BitTorrent
CARP	Cache Array Routing Protocol
CDS	Content Distribution System
CDN	Content Distribution Network
DHT	Distributed Hash Table
DNS	Domain Name Server
FIFO	First In First Out
HTTP	Hypertext Transfer Protocol
ICP	Internet Caching Protocol
IP	Internet Protocol
ISP	Internet Service Provider
LAN	Local Area Network
LFU	Least Frequently Used
LRU	Least Recently Used
MAESTRO	Models for Performance Analysis and Control of Networks
PAC	Proxy Automatic Configuration
P2P	Peer-to-Peer
QoS	Quality of Service
TCP	Transmission Control Protocol
TTL	Time-to-Live
UDP	User Datagram Protocol
URL	Uniform Resource Locator
WWW	World Wide Web

Bibliography

[AAB00] E. Altman, K. Avrachenkov, and C. Barakat. A stochastic model of TCP/IP with stationary random losses. In *Proceedings of ACM SIGCOMM*, Stockholm, Sweden, August 2000.

[AG04] N. Ben Azzouna and F. Guillemin. Impact of peer-to-peer applications on wide area network traffic: an experimental approach. In *Proceedings of Globecom 2004*, Dallas, Texas, November 29 - December 3 2004.

[Aka] Akamai. http://www.akamai.com/.

[AMS82] D. Anick, D. Mitra, and M. M. Sondhi. Stochastic theory of data-handling systems with multiple sources. *Bell Systems Technical Journal*, 61:1871–1894, 1982.

[BB94] F. Baccelli and P. Brémaud. *Elements of Queueing Theory: Palm-Martingale Calculus and Stochastic Recurrences*. Springer Verlag, 1994.

[BBLO00] R. Boorstyn, A. Burchard, J. Liebeherr, and C. Oottamakorn. Statistical service assurances for traffic scheduling algorithms. *IEEE Journal on Selected Areas in Communications, Special Issue on Internet QoS*, 18:2651–2664, December 2000.

[BCF+99] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web caching and Zipf-like distributions: Evidence and implications. In *Proceedings of IEEE INFOCOM '99*, pages 126–134, New York, 1999.

[Bil98] J.A. Bilmes. A gentle tutorial of the em algorithm and its application to parameter estimation for gaussian mixture and hidden markov models. Technical Report TR-97-021, UC Berkeley, April 1998.

[BRF04] E. Biersack, P. Rodriguez, and P. Felber. Performance analysis of peer-to-peer networks for file distribution. In *Proceedings of the 5th International Workshop on Quality of future Internet Services (QofIS'04)*, 2004.

[BSV03] R. Bhagwan, S. Savage, and G.M. Voelker. Understanding availability. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS)*, pages 256–267, Berkeley, California, February 2003.

[Ç75] E. Çinlar. *Introduction to Stochastic Processes*. Prentice Hall, 1975.

[CDN+96] A. Chankhunthod, P. B. Danzig, C. Neerdaels, M F. Schwartz, and J. Worrell. A hierarchical internet object cache. In *Proceedings of the USENIX Annual Technical Conference*, January 1996.

[CI97] P. Cao and S. Irani. Cost-aware WWW proxy caching algorithms. In *Proceedings of USENIX Symposium on Internet Technologies and Systems*, pages 193–206, Monterey, California, 1997.

[CK01a] E. Cohen and H. Kaplan. The age penalty and its effect on cache performance. In *Proceedings of the 3rd USENIX Symposium on Internet Technologies and Systems (USITS)*, pages 73–84, San Francisco, California, 2001.

[CK01b] E. Cohen and H. Kaplan. Refreshment policies for web content caches. In *Proceedings of IEEE INFOCOM'01*, Anchorage, Alaska, April 2001.

[CN04] F. Clévenot and P. Nain. A simple model for the analysis of the Squirrel peer-to-peer caching system. In *Proceedings of IEEE INFOCOM 2004*, Hong Kong, March 2004.

[CN05] F. Clévenot and P. Nain. Stochastic fluid model for P2P caching evaluation. In *Proceedings of the 10th International Workshop on Web Content Caching and Distribution (WCW'05)*, Sophia Antipolis, September 2005. To appear.

[CNR05a] F. Clévenot, P. Nain, and K. W. Ross. Multiclass p2p networks: Static resource allocation for service differentiation and bandwidth diversity. In *Proceedings of the 24th International Symposium on Computer Performance, Modeling, Measurements and Evaluation (Performance 2005)*, Juan-Les-Pins, October 2005. To appear.

[CNR05b] F. Clévenot, P. Nain, and K. W. Ross. Stochastic fluid models for cache clusters. *Performance Evaluation*, 59(1):1–18, January 2005.

[Coh] B. Cohen. BitTorrent. http://www.bitconjurer.org/BitTorrent/.

[Dav99] B. Davison. A survey of proxy cache evaluation techniques. In *Proceedings of the 4th International Web Caching Workshop*, San Diego, California, March 31 - April 2 1999.

[DFKM97] F. Douglis, A. Feldmann, B. Krishnamurthy, and J.C. Mogul. Rate of change and other metrics: a live study of the world wide web. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, Monterey, California, December 1997.

[DMF97] B. Duska, D. Marwood, and M. Feeley. The measured access characteristics of World Wide Web client proxy caches. In *Proceedings of USENIX Symposium on Internet Technologies and Systems*, pages 23–35, Monterey, California, 1997.

[DMP+02] J. Dilley, B. Maggs, J. Parikh, H. Prokop, R. Sitaraman, and B. Weihl. Globally distributed content delivery. *IEEE Internet Computing*, 6(5):50–58, September 2002.

[Edo] Edonkey. http://www.edonkey.com/.

[EM92] A. Elwalid and D. Mitra. Fluid models for the analysis and design of statistical multiplexing with loss priorities on multiple classes of bursty traffic. In *Proceedings of IEEE INFOCOM'92*, pages 415–425, Florence, Italy, 1992.

[EM93] A. Elwalid and D. Mitra. Effective bandwidth of general markovian traffic sources and admission control of high speed networks. *IEEE/ACM Transactions on Networking*, 1:329–343, June 1993.

[FB04] P. Felber and E.W. Biersack. Self-scaling networks for content distribution. In *Proceedings of the International Workshop on Self-* Properties in Complex Information Systems (Self-*)*, Bertinoro, Italy, May-June 2004.

[FCAB98] L. Fan, P. Cao, J. Almeida, and A.Z. Broder. Summary cache: A scalable wide-area web cache sharing protocol. In *Proceedings of the ACM SIGCOMM '98 conference on Applications, Technologies, Architectures,*

and Protocols for Computer Communication, pages 254–265, Vancouver, British Columbia, Canada, September 1998.

[FCD+99] A. Feldmann, R. Cáceres, F. Douglis, G. Glass, and M. Rabinovich. Performance of web proxy caching in heterogeneous bandwidth environments. In *Proceedings of IEEE INFOCOM'99*, New York, NY, March 1999.

[FGM+99] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext transfer protocol – – HTTP/1.1. RFC 2616, June 1999.

[GB97] S. Gribble and E. Brewer. System design issues for internet middleware services: Deductions from a large client trace. In *Proceedings of USENIX Symposium on Internet Technologies and Systems*, pages 207–218, Monterey, California, 1997.

[GFJ+03] Z. Ge, D. Figueiredo, S. Jaiswal, J. Kurose, and D. Towsley. Modeling peer-peer file sharing systems. In *Proceedings of IEEE INFOCOM 2003*, San Francisco, California, April 2003.

[GG92] Allen Gersho and Robert M. Gray. *Vector Quantization and Signal Compression*. Kluwer Academic Publishers, 1992.

[Gnu] Gnutella. http://www.gnutella.com/.

[GRC97] S. Gadde, M. Rabinovich, and J. Chase. Reduce, reuse, recycle: An approach to building large internet caches. In *Proceedings of the 6th Workshop on Hot Topics in Operating Systems (HotOS-VI)*, page 93, Cape Cod, Massachusetts, May 1997. http://www.cs.duke.edu/ari/cisi/crisp/crisp-recycle/.

[Gro98] The Relais Group. Relais: cooperative caches for the world-wide web. http://www-sor.inria.fr/projects/relais/, 1998.

[GS92] G. Grimmett and D. Stirzaker. *Probability and Random Processes*. Oxford University Press, 1992.

[HJ85] R. A. Horn and C. R. Johnson. *Matrix Analysis*. Cambridge University Press, 1985.

[IRD02] S. Iyer, A. Rowstron, and P. Druschel. Squirrel: A decentralized, peer-to-peer Web cache. In *Proceedings of of ACM Symposium on Principles of*

Distributed Computing (PODC 2002), pages 213–222, Monterey, California, 2002.

[IUKB+04] M. Izal, G. Urvoy-Keller, E.W. Biersack, P.A. Felber, A. Al Hamra, and L. Garcés-Erice. Dissecting bittorrent: Five months in a torrent's lifetime. In *Proceedings of Passive and Active Measurement Workshop (PAM2004)*, Antibes, France, April 2004.

[JCDK01] K.L. Johnson, J.F. Carr, M.S. Day, and M.F. Kaashoek. The measured performance of content distribution networks. *Computer Communication*, 24(2):202–206, February 2001.

[Kaz] Kazaa. http://www.kazaa.com.

[KBB+04] T. Karagiannis, A. Broido, N. Brownlee, K. Claffy, and M. Faloutsos. Is p2p dying or just hiding? In *Proceedings of Globecom 2004*, Dallas, Texas, November 29 - December 3 2004.

[Kel79] F. P. Kelly. *Reversibility and Stochastic Networks*. Wiley, Chichester, 1979.

[KG97] K. Kong and D. Ghosal. Pseudo-serving: A user-responsible paradigm for internet access. In *Proceedings of of the Sixth International World Wide Web Conference*, pages 546–557, Santa Clara, California, April 1997.

[KG99] K. Kong and D. Ghosal. Mitigating server-side congestion in the internet through pseudoserving. *IEEE/ACM Transactions on Networking*, 7(4):530–544, August 1999.

[Kha92] Hassan K. Khalil. *Nonlinear systems*. MacMillan, 1992.

[Kle75] L. Kleinrock. *Queueing systems*, volume 1. J. Wiley and sons, 1975.

[KLM97] T. Kroeger, Darrell D. E. Long, and Jeffrey C. Mogul. Exploring the bounds of web latency reduction from caching and prefetching. In *Proceedings of USENIX Symposium on Internet Technologies and Systems*, pages 13–22, Monterey, California, 1997.

[KM01a] K. Kumaran and M. Mandjes. Multiplexing regulated traffic streams: design and performance. In *Proceedings of IEEE INFOCOM 2001*, pages 527–536, Anchorage, Alaska, April 2001.

[KM01b] K. Kumaran and D. Mitra. Performance and fluid simulations of a novel shared buffer management system. *ACM Transactions on Modeling and Computer Simulation*, 11(1):43–75, January 2001.

[KSB+99] D. Karger, A. Sherman, A. Berkheimer, B. Bogstad, R. Dhanidina, K. Iwamoto, B. Kim, L. Matkins, and Y. Yerushalmi. Web caching with consistent hashing. In *8th International WWW Conference*, Toronto, May 1999.

[KSCK96] G. Kesidis, A. Singh, D. Cheung, and W.W. Kwok. Feasibility of fluid event-driven simulation for atm networks. In *Proceedings of IEEE GLOBECOM'96*, London,UK, November 1996.

[KWZ01] B. Krishnamurthy, C.E. Wills, and Y. Zhang. On the use and performance of content distribution networks. In *Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement*, pages 169–182, San Francisco, California, USA, November 1-2 2001.

[LFG+01] B. Liu, D. Figueiredo, Y. Guo, J. Kurose, and D. Towsley. A study of networks simulation efficiency: Fluid simulation vs. packet-level simulation. In *Proceedings of IEEE INFOCOM 2001*, Anchorage, Alaska, April 2001.

[LGB03] P. Linga, I. Gupta, and K. Birman. A churn-resistant peer-to-peer web caching system. In *Proceedings of the 1st ACM Workshop on Survivable and Self-Regenerative Systems*, Fairfax, Virginia, October 2003.

[Lib03] Daniel Liberzon. *Switching in systems and control*. Birkhäuser, 2003.

[LMG95] D. Long, A. Muir, and R. Golding. A longitudinal survey of internet host reliability. Technical Report UCSC-CRL-95-16, University of California, Santa Cruz, 1995.

[LNB04] Francesca Lo Piccolo, Giovanni Neglia, and Giuseppe Bianchi. The effect of heterogeneous link capacities in bittorrent-like file sharing systems. In *Proceedings of of the Int. Workshop on Hot Topics in Peer-to-Peer Systems (HOT-P2P 2004)*, pages 40–47, Volendam, The Nederlands, Oct. 2004. In conjunction with MASCOTS 2004.

[LSSH03] J. Ledlie, J. Shneidman, M. Seltzer, and J. Huth. Scooped, again. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS 2003)*, Berkeley, California, February 2003.

[LZTK97] F. Lo Presti, Z. Zhang, D. Towsley, and J. Kurose. Source time scale and optimal buffer/bandwidth trade-off for regulated traffic in an ATM node. In *Proceedings of IEEE INFOCOM '97*, pages 676–683, Kobe, Japan, 1997.

[Moh01] C. Mohan. Caching technologies for web applications. Tutorial at the 27th International Conference on Very Large Data Bases (VLDB'01), September 2001. Rome, Italy.

[Mor97] A. S. Morse, editor. *Control Using Logic-Based Switching*. London: Springer-Verlag, 1997.

[MV05] L. Massoulié and M. Vojnović. Coupon replication systems. In *Proceedings of ACM Sigmetrics*, Banff, Alberta, Canada, June 2005.

[PGES04] J. Pouwelse, P. Garbacki, D. Epema, and H. Sips. A measurement study of the bittorrent peer-to-peer file-sharing system. Technical Report 2004-003, Delft University of Technology Parallel and Distributed Systems Report Series, April 2004.

[PH97] D. Povey and J. Harrison. A distributed internet cache. In *Proceedings of the 20th Australian Computer Science Conference*, Sydney, Australia, February 1997.

[QS04] D. Qiu and R. Srikant. Modeling and performance analysis of bittorrent-like peer-to-peer networks. In *Proceedings of ACM Sigcomm*, Portland, OR, Aug 2004.

[Qur04] A. Qureshi. Exploring proximity based peer selection in a bittorrent-like protocol. MIT 6.824 Student Project Report, May 2004. http://pdos.csail.mit.edu/6.824-2004/reports/asfandyar.pdf.

[RD01] A. Rowstron and P. Druschel. Pastry: Scalable distributed object location and routing for large-scale peer-to-peer systems. In *Proceedings of Int. Conf. on Distributed Systems Platforms (Middleware)*, Heideberger, Germany, November 2001.

[RFH+01] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Proceedings of ACM SIGCOMM 2001*, San Diego, California, August 2001.

[Rod04] P. Rodriguez. Personal communication, May 2004.

[Ros97] K. W. Ross. Hash-routing for collections of shared Web caches. *IEEE Network Magazine*, 11:37–45, Nov.-Dec 1997.

[RRR02] M. Reisslein, K. W. Ross, and S. Rajagopal. A framework for guaranteeing statistical QoS. *IEEE/ACM Transactions on Networking*, 10(1):27–42, February 2002.

[RS02] M. Rabinovich and O. Spatscheck. *Web Caching and Replication*. Addison-Wesley, 2002.

[RSB01] P. Rodriguez, C. Spanner, and E. Biersack. Analysis of web caching architectures: Hierarchical and distributed caching. *IEEE ACM Transactions on Networking*, 9(4):404–418, august 2001.

[RW98] A. Rousskov and D. Wessels. Cache digests. *Computer Networks and ISDN Systems*, 30(22-23):2155 – 2168, November 1998. Selected papers of the 3rd international WWW caching workshop.

[SGD+02] S. Saroiu, K. P. Gummadi, R. J. Dunn, S. D. Gribble, and H. M. Levy. An analysis of internet content delivery systems. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI '02)*, Boston, MA, December 9-11 2002.

[SGG02] S. Saroiu, P. K. Gummadi, and S. D. Gribble. A measurement study of peer-to-peer file sharing systems. In *Proceedings of Multimedia Computing and Networking 2002 (MMCN'02)*, San Jose, California, January 2002.

[SGG03] S. Saroiu, P. K. Gummadi, , and S. D. Gribble. Measuring and analyzing the characteristics of napster and gnutella hosts. *ACM Multimedia Systems Journal*, 9(2):170–184, August 2003.

[SGP04] K.A. Skevik, V. Goebel, and T. Plagemann. Analysis of bittorrent and its use for the design of a p2p based streaming protocol for a hybrid CDN. Technical report, Delft University of Technology Parallel and Distributed Systems Report Series, June 2004.

[Sly] Slyck. http://www.slyck.com/stats.php.

[SMB02] T. Stading, P. Maniatis, and M. Baker. Peer-to-peer caching schemes to address flash crowds. In *Proceedings of of the 1st International Workshop on Peer-to-Peer Systems (IPTPS 2002)*, Cambridge, MA, USA, March 2002.

[SMK+01] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of ACM SIGCOMM 2001*, pages 149–160, San Diego, California, August 2001.

[TDVK99] R. Tewari, M. Dahlin, H. M. Vin, and J. S. Kay. Design considerations for distributed caching on the internet. In *19th IEEE International Conference on Distributed Computing Systems*, pages 273–284, Austin, Texas, May 1999.

[VR97] V. Valloppillil and K. W. Ross. Cache array routing protocol (CARP). Internet Draft, June 1997.

[VR02] D. Villela and D. Rubenstein. A queuing analysis of server sharing collectives for content distribution. Technical Report EE200412-1, Columbia University, April 2002.

[Wan99] J. Wang. A survey of web caching schemes for the internet. *ACM Computer Communication Review*, 29(5):36–46, October 1999.

[WC97a] Z. Wang and J. Crowcroft. Cachemesh: A distributed cache system for world wide web. In *NLANR Web Cache Workshop*, Boulder, Colorado, June 1997. Extended Abstract.

[WC97b] D. Wessels and K. Claffy. Application of internet cache protocol (icp), version 2. RFC 2187, September 1997.

[WC97c] D. Wessels and K. Claffy. Internet cache protocol (icp), version 2. RFC 2186, September 1997.

[Wes98] D. Wessels. Squid internet object cache. http://www.squid-cache.org/, 1998.

[WNO+02] X. Wang, W. Ng, B. Ooi, K.-L. Tan, and A. Zhou. Buddyweb: A p2p-based collaborative web caching system. In *Revised Papers from the NETWORKING 2002 Workshops on Web Engineering and Peer-to-Peer Computing*, pages 247–251, Pisa, Italy, May 2002.

[WVS+99] A. Wolman, G. Voelker, N. Sharma, N. Cardwell, A. Karlin, and H. Levy. On the scale and performance of cooperative Web proxy caching. In *17th ACM Symposium on Operating Systems Principles (SOSP '99)*, pages 16–31, Kiawah Island, South Carolina, 1999.

[XZX02] L. Xiao, X. Zhang, and Z. Xu. On reliable and scalable peer-to-peer web document sharing. In *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'02)*, Fort Lauderdale, Florida, April 2002.

[YD04] X. Yang and G. De Veciana. Service capacity of peer-to-peer networks. In *Proceedings of IEEE INFOCOM 2004*, Hong Kong, March 2004.

[ZA03] L. Zou and M. Ammar. A file-centric model for peer-to-peer file sharing systems. In *Proceedings of ICNP 03*, Atlanta, Georgia, USA, November 2003.

[ZKJ00] B. Y. Zhao, J. Kubiatowicz, and A. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, UCB, April 2000.