METHODS TO ENHANCE CONTENT DISTRIBUTION FOR VERY LARGE SCALE ONLINE COMMUNITIES

JUAN MANUEL TIRADO MARTÍN

Bajo la dirección de:
DR. JESÚS CARRETERO
DR. FLORIN ISAILA

Doctorado en ciencia y tecnología informática
Departamento de informática
Escuela politécnica superior
Universidad Carlos III

Leganés, Diciembre 2012

METHODS TO ENHANCE CONTENT DISTRIBUTION FOR VERY LARGE SCALE ONLINE COMMUNITIES

JUAN MANUEL TIRADO MARTÍN

Under the supervision of:
DR. JESÚS CARRETERO
DR. FLORIN ISAILA

PhD. in computer science and technology
Computer science department
Polytechnic school
Carlos III University

Leganés, December 2012

METHODS TO ENHANCE CONTENT DISTRIBUTION FOR VERY LARGE SCALE ONLINE COMMUNITIES

AUTOR: Juan Manuel Tirado Martín
DIRECTORES: Jesús Carretero Pérez
Florin Isaila

	Nombre y apellidos	Firma
Presidente:		
Vocal:		
Secretario:		

En Leganés, de del 20

AGRADECIMIENTOS

Este ha sido sin lugar a dudas el viaje más largo que nunca he emprendido. He perdido el rumbo muchas veces y he tenido que rodear montañas (a veces por el camino más largo). Muchas veces me he sentado a descansar para acabar abrumado por lo inmensa que es una gota de agua. Irónicamente todo este tiempo ha servido para darme cuenta que la tesis que tienes ante tí no es el destino de este viaje, sino únicamente un hito. El destino final está por decidir. Llegar hasta allí será apasionante. Eso sí, lo que diferencia a un viaje de un gran viaje es la compañía y desde luego en mi caso ha sido inmejorable. Que estas líneas sirvan para expresar mi gratitud a todos aquellos que de alguna forma me han acompañado durante este largo camino. No están todos los que son, ni son todos los que están.

Gracias a mis directores por haberme señalado el camino. A Jesús, gracias por darme la oportunidad de conocer el mundo de la investigación dentro de este magnífico grupo de gente que es ARCOS. A Florin gracias por su buen saber hacer y por todo el tiempo que ha dedicado a dar forma a esta tesis. A mis compañeros de ARCOS por haberme hecho sentir como en casa. Gracias a Javi, Álex, María Cristina, Javi (Doc), Félix, José Daniel, Fran, Sole, Borja, Rubén, Carlos, Alberto, Luismi, Pablo y Gonzalo.

Muchísimas gracias a todos aquellos con los que he podido compartir este tiempo, haya sido dentro o fuera de la universidad. A Dani por haberme aguantado toda una carrera, un máster, un doctorado, tres meses de estancia (comiendo lo que yo cocinaba) y algún que otro viaje. Es difícil encontrar amigos así y por ello te doy las gracias. Gracias a Guille, Jorge, Sergio, Lorena, Chema, Jose, Javi, Andrea, Mario, Pablo y Teresa por haberme ayudado a hacer mas ameno el viaje.

Muchísimas gracias a los que me han ayudado desde más allá de los Pirineos. Muchísimas gracias a Javierito por su conocimiento parisino, sus recomendaciones literarias, excelente sentido del humor y por ser un gran amigo. Gracias a Anne-Marie, François y a todos los chicos de ASAP (Juliene, Kostas, Eleni, Antoine, Armando, Arnaud y Tyler) que me dieron una calurosa acogida en el lugar más lluvioso de toda Francia.

Por último un billón de gracias a Gloria por haberme acompañado durante una etapa tan importante de este viaje.

A mi familia

"People can come up with
statistics to prove anything.
14% of people know that."
Homer Simpson.

"Felix qui potuit rerum cognoscere causas."
Virgil.

"A man can be destroyed but cannot be defeated."
Hemingway. The Old Man and the Sea.

"Ph'nglui mglw'nafh Cthulhu R'lyeh wgah'nagl fhtagn."
H. P. Lovecraft. The Call of Cthulhu.

ABSTRACT

The Internet has experienced an exponential growth in the last years, and its number of users far from decay keeps on growing. Popular Web 2.0 services such as Facebook, YouTube or Twitter among others sum millions of users and employ vast infrastructures deployed worldwide. The size of these infrastructures is getting huge in order to support such a massive number of users. This increment of the infrastructure size has brought new problems regarding scalability, power consumption, cooling, hardware lifetime, underutilization, investment recovery, etc. Owning this kind of infrastructures is not always affordable nor convenient. This could be a major handicap for starting projects with a humble budget whose success is based on reaching a large audience. However, current technologies might permit to deploy vast infrastructures reducing their cost. We refer to peer-to-peer networks and cloud computing.

Peer-to-peer systems permit users to yield their own resources to distributed infrastructures. These systems have demonstrated to be a valuable choice capable of distributing vast amounts of data to large audiences with a minimal starting infrastructure. Nevertheless, aspects such as content availability cannot be controlled in these systems, whereas classic server infrastructures can improve this aspect.

In the recent time, the cloud has been revealed as a promising paradigm for hosting horizontally scalable Web systems. The cloud offers elastic capabilities that permit to save costs by adapting the number of resources to the incoming demand. Additionally, the cloud makes accessible a vast amount of resources that may be employed on peak workloads. However, how to determine the amount of resources to use remains a challenge.

In this thesis, we describe a hierarchical architecture that combines both: peer-to-peer and elastic server infrastructures in order to enhance content distribution. The peer-to-peer infrastructure brings a scalable solution that reduces the workload in the servers, while the server infrastructure assures availability and reduces costs varying its size when necessary.

We propose a distributed collaborative caching infrastructure that employs a cluster-based locality-aware self-organizing P2P system. This system, leverages collaborative data classification in order to improve content locality. Our evaluation demonstrates that incrementing data locality permits to improve data search while reducing traffic.

We explore the utilization of elastic server infrastructures addressing three issues: system sizing, data grouping and content distribution. We propose novel multi-model techniques for hierarchical workload prediction. These predictions are employed to determine the system size and request distribution policies. Additionally, we propose novel techniques for adaptive control that permit to identify inaccurate models and redefine them. Our evaluation using traces extracted from real systems indicate that the utilization of a hierarchy of multiple models increases prediction accuracy. This hierarchy in conjunction with our adaptive control techniques increments the accuracy during unexpected workload variations. Finally, we demonstrate that locality-aware request distribution policies can take advantage of prediction models to adequate content distribution independently of the system size.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

ACF	AutoCorrelation Function
AIC	Akaike Information Criterion
AP2P	Affinity Peer-to-Peer
AR	AutoRegressive
ARIMA	AutoRegressive Integrated Moving Average
BIC	Bayesian Information Criterion
CAN	Content Addressable Network
DHT	Distributed Hash Table
FGC	Fixed Gain Control
HW	Holt-Winters
MA	Moving Average
MAE	Mean Average Error
MAPE	Mean Average Percentage Error
MPC	Model Predictive Control
PACF	Partial AutoCorrelation Function
P2P	Peer-to-peer
RMSE	Root Mean Squared Error
SLA	Service Level Agreement

1 INTRODUCTION

Last years have shown an unprecedented increase in the Internet traffic, boosted by the emergence of Web 2.0 sites such as social networking, wikis and blogs. Applications such as Twitter, YouTube, Flickr or Facebook facilitate the publishing of user-generated content and the creation of social networks of people sharing common interests. This continuous creation of contents has tremendously incremented the amount of information to be stored and processed in data centers. Data management has become one of the most important topics due to its direct impact on performance, elastic scalability, resource sharing, availability, and energy-efficiency. Controlling the data layout has become critical for making feasibility the processing of large data amounts, as the reorganization of large data sets is severely limited by the I/O infrastructure capabilities and must not negatively impact the quality of service [95].

Peer-to-peer (P2P) architectures have been classically used for the design of large scale content-sharing systems. These architectures offer a scalable platform for applications such as file sharing or video streaming with a low cost. The success of these architectures have made P2P solutions very popular. Napster got 26.4 million users [20] before its closure. File-sharing solutions employing the BitTorrent protocol [14] were the second traffic source in Europe during 2011 [78]. Nevertheless, these solutions manage sets of users that join and leave the system unceasingly. This makes difficult to guarantee aspects such as content availability.

In the last years, the advent of cloud computing has been revealed as a suitable match for the variable demand of Internet Web 2.0 applications. Cloud computing enables elastic horizontal scalability of server infrastructures, which allows to dynamically allocate resources depending on demand and to pay only for their utilization. However, efficiently exploiting these dynamic resource allocation mechanisms strongly depends on understanding and controlling the dynamics of workloads and on reducing the data traffic inside the infrastructure. Increasing scale and demand variations poses huge challenges on developing, deploying, and evaluating control mechanisms and policies for an efficient resource allocation.

Current Web platforms require novel approaches to manage highly variable and growing workloads. The economy of scale promise of cloud computing is a great approach to reduce the total cost of ownership of existing infrastructures. However, proactively adapting the resources to the incoming workload requires the utilization of prediction models and control theory. On the other hand, the utilization of P2P architectures might reduce the number of employed resources while providing a scalable content distribution solution. In this thesis, we address the utilization of both P2P and cloud architectures in the development of content distribution applications for very large scale on-line communities.

1.1 MOTIVATION

The popularity of user-generated content systems such as YouTube, Twitter or Facebook among others have demonstrated the capability of users to generate new content. Zhou et al. [103] estimate YouTube to store up to 500 million videos summing up around 5 Petabytes of storage. Twitter claims to have 140 million active users, receiving 340 million tweets per day [90]. The amount of resources employed by these infrastructures has become a major problem in multiple aspects such as hardware, power, cooling or networking. According to Greenberg et al. [42], in a state of the art data center 59% of the energy goes to equipment, 8% is lost in the distribution, and 33% goes to cooling. If we add the problem of highly variable demand we have that the server utilization is between 10% and 50% of the total capacity [11]. This means that 59% of the total energy consumed by the infrastructure is employed to run underutilized machines. Several approaches have been proposed to achieve a more efficient utilization of the resources through power-proportional solutions [11], data center design, more efficient hardware, etc.

The utilization of the cloud relives system designers from the burdens of owning physical infrastructures. The cloud offers access to virtual infinite resources on-demand. However, the demander of resources may not be aware of the underlying hardware, nor the network, but she has to find fulfilled a set of restrictions hired in a service agreement with the cloud provider. One of the most valuable advantages of the cloud is the capability to dynamically modify the amount of employed resources only paying for their utilization. The demand in web systems is known to have variations from periods of intense activity followed by low activity [102, 38, 37]. These variations are even more dramatic under the presence of unexpected events that may exceed the capacity of provisioned resources. However, this should not be a problem for cloud-based solutions that can rapidly scale up and down the employed resources. Scaling resources to the incoming workload permits to have an infrastructure whose cost approximates an ideal demand-proportional solution.

The cloud is a promising approach to provide a robust infrastructure for content distribution systems. However, adding more resources to satisfy users demand increases the cost. Collaborative distributed systems based on P2P systems have been demonstrated to be highly scalable and fault-tolerant solutions to distribute massive amounts of data [43, 28, 59] among large audiences. In an abstract sense, decentralized architectures are similar to social networks, where users try to solve a task using their social links. This suggests that P2P architectures might be employed as a feasible scalable solution for Web content based systems [24, 25, 12, 23].

The cloud and P2P systems are interesting solutions for the problem of content delivery to large communities of users. Both solutions have complimentary facets. The cloud permits to build elastic infrastructures only paying for the employed resources. On the other hand, P2P systems are highly scalable solutions with a low cost. P2P networks do not assure content availability, but can be employed as a distributed cache to reduce the pressure under cloud-based Web solutions. The combination of both architectures can reduce the number of employed resources, and therefore the cost.

In the present thesis, we address the idea of combining both approaches (cloud and decentralized architectures) in the design of a content distribution architecture for large communities of users. In particular, we propose a hierarchical architecture for content distribution that combines both approaches. Afterwards, we explore the design challenges of this architecture and explore possible solutions.

1.2 OBJECTIVES

The major goal of this thesis is to **propose and study a hierarchical content distribution architecture for large scale on-line communities that combines decentralized systems and cloud-based infrastructures**. Additionally, this thesis targets the following objectives:

O1 Explore a decentralized content distribution solution that leverages users preferences and community knowledge, in order to reduce pressure on the back-end infrastructures by improving content locality.

O2 Study the data access patterns of Internet applications, analyze the limits of workload predictability, and investigate solutions that adapt to workload variability.

O3 Propose and study methods of prediction and control theory to enhance data distribution on elastic server infrastructures.

1.3 CONTRIBUTIONS

The study and analysis of the aforementioned objectives has generated the following contributions:

C1 We design a cluster-based locality-aware self-organizing P2P system leveraging collaborative classification and demonstrate that it improves content locality and search latency.

C2 We propose a novel multi-model technique for improving hierarchical workload prediction accuracy at aggregate and group levels and demonstrate their suitability with real workloads.

C3 We propose novel locality-aware data management policies based on predictive models for elastic server infrastructures and demonstrate that they improve content locality.

C4 We propose novel techniques for adaptive control of elastic server infrastructures and explore the under- and over-provisioning of resources simulating a real workload.

Structure of the document
This document is structured as follows:

- Chapter 2 describes the state of the art in data distribution, elastic systems and workload forecasting.
- Chapter 3 overviews the design of a hierarchical architecture that combines P2P networks and cloud infrastructures for content distribution solutions.

- Chapter 4 presents a decentralized collaborative caching infrastructure that employs user information to efficiently allocate and access users data.
- Chapter 5 describes the utilization of automatic models generation for the problem of workload forecasting. In this Chapter, we describe three datasets extracted from real systems and evaluate the proposed prediction solutions using these datasets.
- Chapter 6 presents the methods employed in the design of a system controller for elastic server infrastructures. In this Chapter, we explore the vast design space of elastic Web infrastructures and propose some solutions for adaptive system sizing, data grouping and content distribution.
- Chapter 7 presents the conclusions, describes the future research lines, and the contributions of this thesis.

2 STATE OF THE ART

This Chapter presents the state of the art and the background concepts used in this thesis. This Chapter is organized into four Sections. First, we overview user-generated content distribution systems. Second, we describe the state of the art of decentralized distribution systems. Third, we describe solutions for elastic content distribution systems. Finally, we present various works addressing the problem of workload forecasting for elastic systems.

2.1 USER-GENERATED CONTENT DISTRIBUTION SYSTEMS

Web 2.0 users have been revealed to be powerful content creators. Popular Web 2.0 applications such as YouTube, Flickr, Twitter, Wikipedia, or Vimeo among others continue adding more users and contents. For example, YouTube is known to serve daily more than 4,000 million videos [101], with 60 hours of new videos each day. Similarly, Facebook has impressive statistics with more than 159 million users only in the USA [32].

The infrastructures supporting these applications are composed of hundreds of thousands of machines that aim at delivering content items to users on demand. From the multiple architectures these machines can be organized into, the three-layers architecture is the most common choice [19]. These three layers are: dispatchers, content servers, and storage backend as shown in Figure 2.1. These layers are composed of several machines with homogeneous or heterogeneous features. The dispatcher layer is in charge of redistributing the user requests among the content servers. The content servers act as intermediaries between the storage backend and the final user. Finally, the storage backend stores the available contents.

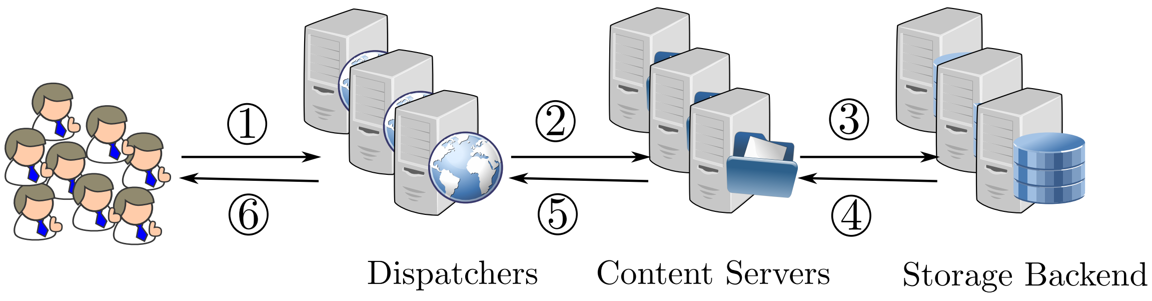


Figure 2.1: Dataflow in a three layers server architecture. First, (1) a user accesses the system dispatcher requesting an item, this (2) redirects the request to a content server. In the case the server does not contain the requested item, (3) it requests a copy to the storage backend. In the case the server does not contain the requested item, (4) it requests a copy to the storage backend. Finally, the request is returned to the user (5, 6).

2.2 DECENTRALIZED DISTRIBUTION SYSTEMS

Decentralized distribution systems can be conceptually designed as multilayer solutions. These layers are combined depending on the application domain. A common approach is to consider two layers: network and semantic or clustering (Figure 2.2). The network layer is in charge of connecting users across the network. The semantic or clustering layer organizes the nodes into semantic clusters regarding the capacity of the nodes to serve requests. A large variety of systems is based on the idea of connecting users with the same tastes into the same cluster. This dichotomy between network and semantic layer makes possible to distinguish between network nodes and neighbors with similar tastes. For example, in Figure 2.2 user 5 knows users 2 and 3 using the network layer, but they are not connected in the semantic layer as they are not relevant.

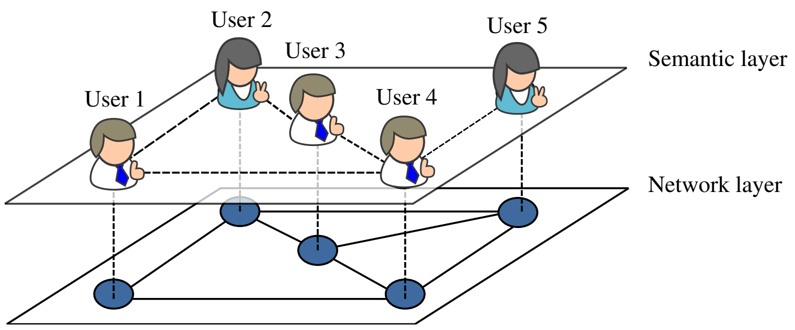


Figure 2.2: Components of a decentralized distribution system.

The network layer is oblivious to the content distributed by the application. This layer only provides methods to maintain the network coherence and the users allocation. The semantic layer may use the operations provided by the network layer to allocate and locate users, or search for content. In the next subSections we present examples of solutions for network and semantic layers.

2.2.1 *Network layers*

According to Androutsellis and Theotokis [4] we can distinguish two major kinds of peer-to-peer network infrastructures: unstructured and structured. In this Section, we briefly describe relevant solutions for both kinds.

2.2.1.1 *Structured networks*

Structured networks are organized into graph topologies such as trees, rings, d-dimensional spaces, etc. These topologies are controlled by using distributed control protocols. These protocols ensure the correctness of the network under users joins and departures. Contents are allocated to the network using an identifier. The identifier permits to locate the content inside the network. Maintaining a structure makes possible to reduce the number of hops when searching content. Below we describe four typical examples of structured networks.

CHORD Chord [83] is a DHT that uses consistent hashing to assign keys to nodes. When a node joins the network, it is assigned a unique identifier and it is placed into a ring topology. The node immediately becomes responsible for the items in the range that goes from the identifier of its previous neighbor to its own identifier. Each content allocated in the network has a unique identifier. This allows the nodes to send requests to the nodes whose identifier is closer to the one of the requested item. This is done by checking the identifier of the neighbors. Chord's search method is demonstrated to locate content in $O(\log N)$ hops, where N is the number of nodes in the network for a steady state. Chord may have load balance issues when resources are not homogeneously distributed throughout the ring, or when some resources are more requested than others.

PASTRY Pastry [77] is a distributed object location and routing schema that uses a tree topology. An object with identifier id is replicated into the k nodes whose identifiers are numerically closer to id. When a node requests an object with identifier id, it sends a message to k nodes. This policy assures load balance by distributing the object request among several nodes and a location complexity of $O(\log_{2^b} N)$ with b the branches of the tree and N the total number of nodes. Nodes in pastry participating in the search of content automatically replicate the searched content. This permits to increase the availability of the most popular content. However, this replication does not take into account the probability of the nodes to be asked for a given content generating unnecessary replicas.

CAN CAN (Content Addressable Network) [74] is designed as a distributed network that uses cartesian coordinates to place and search data in a multidimensional torus. Every node is assigned a portion of the space that is identified by a set of virtual coordinates. When a node is inserted and it falls into a portion of the space already occupied by another node, this space is partitioned and assigned to both nodes. CAN permits searching using several parameters by mapping every parameter onto a space coordinate. The search is done using a greedy approach that looks for the closest node to the desired resource. The search method has an upper bound of $O(dN^{\frac{1}{d}})$ hops, where d is the number of dimensions and N the total number of nodes in the network. One of the main disadvantages of CAN is the possibility of deep partitioning of the search space by continuously inserting nodes in the same space. This can result in poor load balancing and cannot be controlled by the network.

KADEMLIA Kademlia [62] is a decentralized binary tree overlay. Each Kademlia node is assigned a unique identifier. This identifier is used to place the node in a binary tree topology. Kademlia uses the idea of distances to determine the minimal path to search resources. Content search in Kademlia is unidirectional in a such a way that the requests for the same key follow the same path in the network. This is done to permit the utilization of cache techniques, that may reduce the number of hops by replicating popular content. The problem is that the replication is only done in the nodes that are supposed to participate in the search which can lead to load balance problems.

2.2.1.2 *Unstructured networks*

Unstructured topologies are organized into random graphs. Communication between network nodes is typically done by flooding or random walking. These networks are widely used, in part due to the simplicity of their design, and their low cost for self-maintenance. The main disadvantage of these topologies is the lack of scalability of the search methods. The most typical example of unstructured network is Gnutella.

GNUTELLA Gnutella [18] is a fully-decentralized peer-to-peer network whose users are placed in a flat hierarchy. The nodes exchange messages using a flooding communication protocol in order to find resources. Each request is associated with a time to live (TTL) attribute. When a node receives a request message it forwards it to its neighbors until the TTL reaches 0. This flooding mechanism generates redundant messages, as the same node can receive several times the same request. The utilization of a TTL creates a virtual horizon that delimits the search [75]. This phenomenon limits the scalability of Gnutella.

2.2.1.3 *Comparison of network topologies*

Table 2.1 shows a summary table comparing the main features of the aforementioned topologies. All the approaches except CAN use a single identifier in order to locate content. This simplifies the design, but it also reduces the possibility of creating complex queries. The lowest search complexities are obtained by Chord and Kademlia. In the case of Pastry, the search complexity can be tuned by setting the number of branches of the tree (parameter b), and in the case of CAN the complexity varies according to the number of dimensions (parameter d). As expected, the worst value is obtained by Gnutella and its exponential complexity. Nevertheless, the simplicity of its design makes it a good candidate for a wide number of applications.

	Chord	**Pastry**	**CAN**	**Kademlia**	**Gnutella**
Overlay	Structured	Structured	Structured	Structured	Unstructured
Topology	Ring	Tree	d-dimensions	Binary tree	Random
Content replication	No	No	No	Yes	Yes
Search mechanism	Logarithmic key	Logarithmic key	Greedy	XOR	Flooding
Search complexity	$O(\log N)$	$O(\log_{2^b} N)$	$O(dN^{1/d})$	$O(\log N)$	$N^{\lvert TTL \rvert}$

Table 2.1: Comparison of fully-decentralized network topologies.

One of the major limitations of structured networks is that they are designed to support exact match lookups and group-based organizations such as clusters. This reduces their applicability to scenarios with complex queries. Furthermore, the utilization of keys for distributing content destroys content locality. Content should be allocated to enhance browsing or searching contents. Additionally, the application level information is lost. The data used in the supported applications is discarded by these networks.

2.2.2 *Semantic P2P networks*

The absence of centralized knowledge indicating where the content can be found, makes necessary to develop alternative methods to access and find content. The semantic layer offers search capabilities which can also include solutions to improve system performance. How a semantic layer works depends on the application domain and the employed network layer, thus we can find several examples of different semantic layers. In the case of solutions with semantic layers deployed on top of on unstructured network, a generalized approach is to apply similarity metrics that permit to identify groups of similar nodes. The nodes are placed together increasing the probabilities of finding the content inside the group a node belongs to, and therefore diminishing the number of hops.

In [30], Crespo and García propose a semantic cluster-based overlay network that based on a classifier determines how to reorganize users in clusters in order to maximize the resources they share. For instance, users share music files that are classified following a predefined hierarchy of music types. Users are clustered according to the number of resources they own in a certain category, and can belong to several clusters. Finally, when a cluster contains a small number of nodes it is consolidated with other clusters. The experiments carried out in this work reveal that placing the users in clusters of interests reduces the number of hops. However, the results may significantly vary depending on the employed classification hierarchy. The reorganization of the users and the consolidation of clusters is done off-line using a third entity that identifies the best configuration of clusters. P2PDating [68] extends the ideas presented by Crespo and García enhancing the overlay network by providing full autonomy to the nodes. The nodes decide autonomously which cluster to join according to different metrics such as the degree of overlapping between the documents they own, the history, level of trust, etc. Raftopolou et al. [72] propose iClusters, a method to connect similar nodes with different interests in a system that support text search. In iCluster, the nodes are clustered according to the type of documents they store, and create additional links to connect with other clusters they might be interested in. The search process is done by forwarding a query containing one or several words to the nodes with the highest similarity. Finally, the node with the similarity beyond a certain threshold computes a list of possible documents that may answer the query. Iamnitchi et al. propose the utilization of data sharing graphs [47] to capture common user interest. These graphs can be used to dynamically organize files into clusters of interest [45].

The previous approaches suppose that the data a node owns is representative for the kind of requests the node will do in the short term. Other solutions differentiate between the nature of the stored data and the nature of the performed queries. REMINDIN' [86] proposes nodes to store the address of those nodes that answered correctly previous queries. This permits to improve the efficiency of future queries by adaptive semantic-based routing. Similarly, Rostami et al. [76] propose to employ ontologies to identify the most relevant nodes to answer a query. Their evaluation on top of Chord shows a relevant reduction of the generated traffic.

The aforementioned solutions employed metrics based on documents and shared elements. However, the recent interest in social networks has extended the definition of similarity metrics to include social aspects such as personal interests, relationships, tastes, etc. Anwar et al. [5] propose to leverage common user interests to reduce latency in group communication. Pouwelse et al. [69] propose a social P2P network in which communities are built based on similar peers interests. Voulgaris et al. [96] propose a proactive gossip-based management network that uses a semantic proximity function to define the semantically closest neighbors of a node. Nodes periodically try to find new neighbors that may improve this function in order to increase the similarity between nodes. Isaacman et al. [48] propose a fully distributed recommendation system that uses the content-rating of other neighbors to identify contents of interests. Bertier et al. [13] propose Gossple, a fully decentralized anonymous collaborative network that employs similarity metrics to identify clusters of users "socially close". Their solution works on top of an unstructured network that employs gossiping protocols to explore the neighbors of a user. Afterwards, the collected information is employed to determine which users have a higher similarity. Linking users with a higher similarity increments the probabilities of finding relevant content.

2.3 ELASTIC DISTRIBUTION SYSTEMS

By elastic system we mean an infrastructure that can dynamically vary the number of employed resources. This can be achieved by server infrastructures turning on and off machines. However, cloud-based solutions facilitate this task. According to Armbrust et al. [9] cloud computing refers to both the applications delivered as services over the Internet and the hardware and systems software in the data centers that provide those services. Foster et al. [34] provide four key points to define what is cloud computing. The cloud is massively scalable (1), can be encapsulated as an abstract entity that delivers different levels of services to customers (2), it is driven by economies of scale (3), and the services can be dynamically configured and delivered on demand (4).

The cloud itself enables elastic horizontal scalability, which allows to provision resources in order to adapt the system size to the incoming demand. This permits to reduce the costs of the infrastructure modeling the system regarding the system demand. Additionally, the possibility of adding a virtually infinite amount of resources from public clouds makes possible to deal with large peaks of demand.

2.3.1 *System sizing techniques*

The utilization of servers in current data centers is known to be between 10% and 50% of their total capacity [11], while workload peaks may incur in poor performance [42] due to the lack of available resources. This problem has led some authors to propose enhancements and extensions to address the problems of under- and overutilization by dynamically modifying the system size. Control theory has been recognized to be a good fit addressing changes in resource allocations [51]. Nevertheless, many system designers still employ ad-hoc solutions of feedback control loops [104]. Furthermore, the fact that many subsystems are not designed from scratch to be controllable complicates the development of scalable controllable systems. Patikirikorala and Colman [85] describe four types of standard controlling techniques relevant for computing systems: fixed gain control, model predictive control, adaptive control, and reconfiguring control. Fixed gain control (FCG) is based on off-line tuning of control model parameters, which remain fixed during the controller life. FCG is not considered appropriate for highly changing workload conditions. Model predictive control (MPC) allows for proactive decisions, as it employs predictive models for forecasting the future system behavior. As in the case of FGC, the off-line estimation of parameters makes MPC inappropriate for highly changing workload conditions. Adaptive control addresses the limitations of FGC and MPC by allowing for on-line modification of control model parameters based on well-studied methods such as recursive least squares [52]. Reconfiguring control allows for on-line changing of both models and model parameters.

The system sizing solutions we present below are based on the utilization of a control loop. However, there are significant differences among the presented solutions. We group them according to the utilization of reactive and proactive solutions. Proactive solutions are generally more complex than reactive solutions as they work with analytical models to estimate the future state of the system. This permits to take mid-term and long-term decisions. On the other hand, reactive solutions permit to take fast decisions by observing the current system state.

2.3.1.1 *Reactive system sizing*

Reactive system solutions monitor the system status and compare it with a reference model. This model indicates the upper and lower bounds for a set of observable system variables. When the observations fall out of the defined bounds, the system reacts to correct the deviation in the variable. Below we describe some examples of reactive solutions.

- Zhang et al. [102] present a hybrid cloud solution for content distribution. During normal traffic periods, requests are served from a private cloud (base zone), while traffic peaks are redirected to instances deployed in an external cloud (trespassing zone). They employ a predefined workload model that indicates the normal number of requests to be served per unit of time. They compare the current requests with the requests indicated by the model. If the current requests are larger than the requests in the model, then the exceeding requests are redirected to the trespassing zone.

- Maurer et al. [61] propose a system that automatically identifies the thresholds used by the control loop in order to trigger system size modifications. They use three sets: variables observations, thresholds and actions (adding resources, removing resources...). A control system stores the result of performing an action for a given set of observations. This information is employed to identify how the service quality changes after modifying the system size. And therefore, determining the thresholds to use. This approach is case-based which implies that in order to identify a situation and to act accordingly, the same situation must have been observed before.

- In [73] Rao et al. propose to employ a distributed reinforcement learning algorithm for provisioning virtual machines. Each virtual machine runs an agent that determines if the machine needs more resources, if so it will request these resources (CPU, memory, bandwidth and disk) to a central entity. Their approach analyzes the CPU, bandwidth and memory in order to determine the machine performance. This approach is oriented to the consolidation of applications in virtual machines, by modifying features such as memory size, CPU and bandwidth. This solution may be appropriate in a private cloud, but in a public one this would imply additional costs because the resources employed by running instances cannot be modified. In public clouds where the payment is done by hours this would imply additional costs.

- Amazon auto-scaling [3] is a service provided by Amazon to automatically vary the number of running instances deployed in their public cloud. This service is a clear example of FCG where a set of metrics is defined in a monitor system with their minimum and maximum values. When the maximum threshold is exceeded the controller launches a new instance.

The utilization of FCG solutions is a common practice. Examples are found in [102, 3]. Nevertheless, this static approach is not recommended for highly variable workloads, as pre-defined scenarios might not correctly adapt to unexpected situations. The solution presented by Maurer et al. [61] could be considered as an MPC solution, but future system states are not identified. The solution presented by Rao et al. [73] employs models that could provide the system with a proactive solution. Nevertheless, their approach only considers normal traffic and exceeding traffic.

2.3.1.2 *Proactive system sizing*

Proactive techniques are based on the capacity to take decisions speculating about the future of the system after observing it. The idea is to provide the system with the resources it will need before they are demanded. These approaches rely on the utilization of prediction models for estimating the incoming workload, or what the system size should be during the next period of time.

- MUSE [27] uses a control strategy that adds, removes, shuts down or reassigns servers to maximize energy efficiency subject to SLA constraints and quality of service for each application. At pre-defined points in time, an allocator module determines what resources can be added or removed for an application. In order to do this, MUSE assumes that each application has a utility function that identifies the monetary value of adding or removing resources. Each node is expected to have a determined load. If the load is lower than a defined threshold the resources can be released, otherwise an estimation of the future workload is done using an exponential filter.

- Kusic et al. [54] address the problem of maximizing the revenue of virtual clusters by minimizing the amount of employed resources while maintaining the quality of service. The authors differentiate between two different levels of quality (silver and gold) and try to reduce the number of machines while maintaining the service quality for each level. They monitor different variables of the virtual machines in the system such as CPU, memory, queue length, etc. The authors propose a two layer control infrastructure, one to identify the ratio of CPU to use by the virtual machine, and a second layer that identifies the number of machines to allocate the application into. This solution is oriented to infrastructures that rent their resources while maximizing the revenue. However, the possibility of switching on and off machines is not explored.

- Bodík et al. [16] propose to dynamically modify the number of instances in a private cloud by using a machine learning algorithm to estimate workload models. The solution is composed of a controller that firstly collects a dataset that contains the number of machines, workload and response time of the system. This dataset is used to define a correlation model that identifies the number of machines to use for a given workload in order to not exceed a defined response time. In particular, the authors use a curve smoothing model to predict the workload during the next 5 minutes based on the previous 15 observations.

- Santana et al. [79] propose the utilization of a proactive workload prediction model to dynamically modify the CPU frequency and voltage of machines. Their solution forecast decays in the number of incoming requests to first reduce the CPU voltage, and second to shutdown the machine. Nevertheless, this approach is not valid for private clouds where the user does not have direct control of the infrastructure as the CPU voltage cannot be modified.

- Krioukov et al. [53] present the design of a power-proportional cluster, in which a cluster manager runs a knapsack algorithm in order to provision an optimal number of servers. This approach employs a statically parametrized prediction model that addresses the number of requests to be served. A scheduler uses the energy consumption profile of the servers to distribute traffic among the available servers. This solution is oriented to private clouds, and it assumes that the energy profiles are known.

All the presented works employ MPC control. Chase et al. employ off-line defined models [27], while the remainder works use models whose parameters may vary through time [54, 16, 79, 53]. Nevertheless, the mentioned solutions only employ a single kind of models. These works do not explore how the results may vary depending on the employed prediction model. Furthermore, it remains uncertain how would these models work in the case of sudden workload variations or unexpected workloads.

2.4 WORKLOAD MODELING

Events taking place in the real world are mapped into the workload of web applications [100, 89]. From the sociological point of view, this fact converts web systems into a niche for studies that may help us to understand the habits of users, their interests, explore commercial utilities, etc. From the system point of view, the comprehension of the workload in a web application is particularly relevant for two reasons. First, it permits to understand how the users interact with the system. Second, a better comprehension of workload dynamics makes possible to offer a better service while reducing costs.

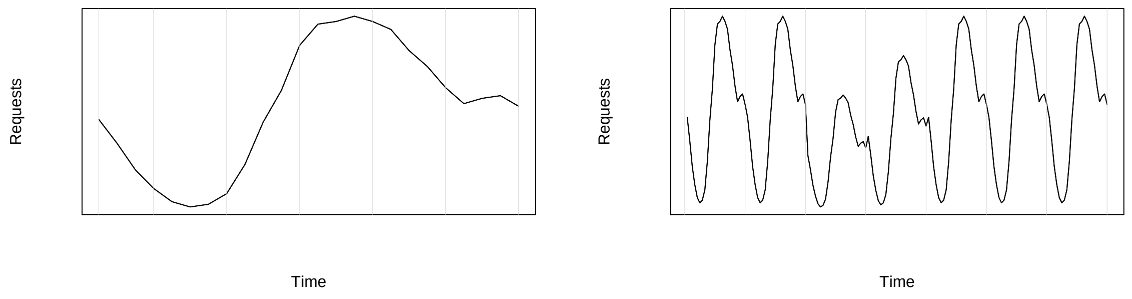
In 1997 Arlitt an Williamson [8] analyzed the workload of six different web servers. Ten years later in [98] they repeated the analysis finding that their earlier conclusions still were valid. Bodik et al. [17] provide a detailed comparison of workloads taking into account the dynamics of content popularity. These three studies have common claims. Below we enumerate the most relevant ideas for this thesis.

- Web systems demand is highly volatile. The workload presents periods of low activity, followed by periods of intense activity that may multiply several times the workload during low activity periods during the same day. This pattern repeats daily with a lower amount of traffic on certain days, typically weekends.

- Content popularity follows a power-law distribution. The busiest 10% of files account for 80-90% of the requests.

- Unexpected workload peaks are due to variations in the popularity of a small number of items.

The ideas above have also been found in analysis of large Internet applications. Studies of YouTube traffic workloads [23, 37, 29, 28, 65] show that 10% of the top popular videos account for 80% of the total requests. However, some videos in the long tail of the popularity distribution seem to be active and are periodically requested. Similar results can be found in Wikipedia [91]. This study suggests that independently of the popularity of the item, this will be requested. Studied popularity distributions have demonstrated to show temporal patterns [41, 103]. A study of the video on demand system deployed during the 2008 Beijing Olympics [100] shows that popularity changes occur frequently. Furthermore, this study claims that unexpected events may dramatically modify the popularity distribution, redirecting the traffic to a small number of items.



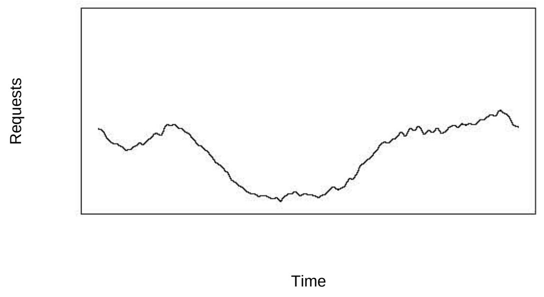
(a) Example of a generic daily workload pattern (b) Example of a one week workload pattern

Figure 2.3: Examples of basic workload patterns.

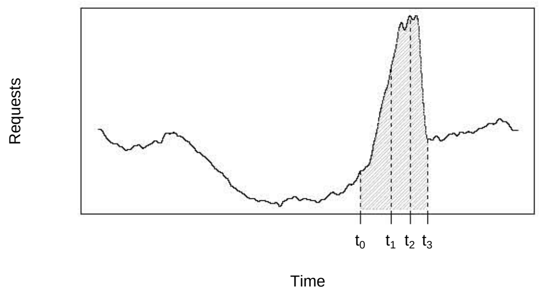
Some of the aforementioned studies indicate that Web systems workloads follow a predictable pattern. In Figure 2.3a we present the workload per hour observed in [37] after the analysis of the requests to YouTube in a university campus. In the same day, the workload significantly varies from a valley of low activity to progressively grow until the maximum load is reached and then decreases again. This pattern can be observed during the week with variations in the total number of requests achieved as shown in Figure 2.3b. This phenomenon can also be found in traces extracted from Wikipedia [91], 1998 World Cup [7] or LastFM [58].

2.4.1 *Events and peaks*

We define an event as a situation that modifies what we can consider the normal system workload. The effects produced by an event may impact the system for a large period of time or not. These events, expected or unexpected may exceed the available system resources. In the case of expected events, the system can be previously prepared. Nevertheless, in the case of unexpected events the system must react to overcome the situation without any previous preparation. For example, after Michael Jacksons' 22% of all the content generated in Twitter mentioned him [88]. Similarly, 5% of the whole traffic in the Wikipedia was redirected to the Michael Jackson's article [35]. Another significant case was the inauguration of president Obama where the number of tweets per second was five times higher than a regular day in Twitter [89]. From these examples, we can distin-



(a) Original workload.



(b) Workload after and event generates a spike.

Figure 2.4: Example of how an event generates a spike for a regular workload.

guish the case of Michael Jackson as an unexpected event, while Obama's inauguration was an expected event.

During an event, the system workload can be modified in two ways: varying the volume of requests, or modifying the popularity distribution. Both of them can be combined in the same event. For example, in the video on demand application studied in [100] certain events reduced the traffic of the top ten videos from 80% to 10%. Events analyzed during the World Cup 1998 [16] reveal that 5% of the items multiplied by 5 the total number of requests during events.

Some works have studied the nature of unexpected events. Parikh and Sundaresan [67] analyze events occurred in queries inside the EBay system. They propose a method to identify events based on analyzing changes in the volume of queries. Lassnig et al. [56] study the events in a scientific infrastructure and propose a method to identify them. Eriksson et al. [31] propose a framework to detect network events that may result in anomalies based on previous network variables using experts knowledge. Similarly, Carter et al. [21] propose to analyze unexpected events with a set of clusters defined by experts to identify possible events. Nevertheless, none of the previous studies address the problem of how to deal with these events or even how to predict them.

We find the work done by Bodík et al. [17] the most complete workload events analysis in Web systems. The authors study several workloads from real systems and propose a methodology to characterize events and how they impact the system. This permits to synthetically generate events based on predefined parameters. According to their methodology an event can be defined by the tuple (t_0, t_1, t_2, t_3, M, V, H). The values t_0, t_1, t_2, and t_3 indicate the times when the event starts, when it reaches its peak of demand, the end of the peak period, and the end of the event, respectively. M is the magnitude of the peak compared with the baseline workload. V is the variability of the popularity of the hotspot items during the spike compared to regular workload. This value is defined by the statistical distribution of the popularities for the hotspot items. Finally, H is the number of hotspot items that are responsible for the workload increment.

Figure 2.4 shows how according to the previous methodology, a regular workload can be modified to include a synthetically generated spike. This figure only shows the variations in the volume of requests. The shadowed area in Figure 2.4b indicates the duration of the spike, which goes from t_0 to t_3.

2.4.2 *Workload forecasting*

We have mentioned that several studies have observed periodic patterns in server workloads. This means that using past observations it would be possible to predict the future workload. The workload forecasting problem can be addressed as a time series forecasting. The time series forecasting problem has been studied during the last 25 years [40] developing a wide theoretical background. The main advances in this area include the development of generic prediction models that can be applied to several scenarios. Nevertheless, not all the existing solutions address the problem of workload forecasting using already designed generic models nor from the perspective of time series forecasting. Below we briefly describe a selected sample of proposed solutions.

- In [26] the authors propose a system to efficiently place applications in a servers infrastructure. Their solution contains a predictor module that addresses the problem of proactively identifying the resources demanded by applications, in order to allocate them in servers while improving the performance.

  The authors address the problem employing two models: the arrival rate model and the service demand model. The arrival rate model identifies the rate of incoming requests per unit of time. The authors model the arrival rate using a statically defined model for the time series of previous arrival rates. This model returns the expected arrival times for a predefined time window. The service demand is calculated by computing the probability of demanding resources such as CPU or memory per request. Their model is designed to assure a minimal utilization of those CPUs shared among applications, while avoiding overload. The experimental evaluation is conducted with synthetic traces and traces from the World Cup 1998 [7]. The results, indicate that the solution is able to model the workload in order to avoid overloads. The utilization of a single statically defined forecasting model, does not permit to modify the predictions in the case the workload includes modifiers such as trends or bursts.

- Gmach et al. [38] propose a workload prediction model based on access patterns and trend analysis. Their approach assumes that the same workload patterns repeat over time. Based on this assumption, they analyze the workload in the system and look for periods of time with similar workloads. Repeating this process, they gather a set of possible workload patterns in the system. Then, they try to identify the current workload pattern among one of the observed patterns. If they find a similar pattern in the past, they identify the amount of resources that should have been provisioned in the past. Their experimental evaluation using synthetic traces demonstrates that employing a 5 weeks dataset can be enough to plan the capacity during 1 week. However, the authors do not mention how the pattern recognition system would react in case of unexpected events.

- Gong et al. [39] propose a light-weight online resource demand prediction scheme for cyclic and non-cyclic workloads. The solution looks for similar workload patterns over time and compare them using the Pearson coefficient. This process is called signature analysis. This signature analysis is combined with a state analysis, which is employed in a transition probability matrix among states. A state is a discrete interval of the resources employed in the system. In summary the prediction is done analyzing workload signatures and identifying the next state (or number of resources needed) the system will be in. This work has some similarities with [38]. As in [38] there is no evaluation of what would happen the first time a workload pattern is detected.

- The work presented in [16] proposes to use statistical machine learning techniques for modeling the workload of Internet applications. The predictions are done using two models. One is a simple autoregressive model that predicts the number of requests for the next five minutes. The second model indicates the service time according to the system size. The system stores tuples containing the workload, the system size, and the system performance. The system performance is measured as the time needed to satisfy 95% of the incoming requests. This table serves as input for a linear regression model that estimates the system size according to the predicted workload in order to get a certain performance. Additionally, the authors propose to redefine the models by analyzing the error distribution. Experimental evaluations indicate that only 0.52% of the requests violated a pre-defined SLA. The authors do not clarify what would happen in the case that the predictions where incorrect during one of the 5 minutes prediction intervals. Another interesting issue would be to address the problem of employing linear regression to identify the number of machines when unexpected events require the utilization of a larger number of machines for a short time.

- Urgaonkar et al. [92] design an analytical method based on queuing theory for web infrastructures composed of multiple tiers. Their model permits to estimate the rate of incoming requests for each of the tiers, and the time needed to answer these requests. The experimental evaluation does not describe the employed workload and thus there is not a clear idea of what kind of workloads this solution is oriented to. However, the utilization of different queuing models between layers might permit to identify bottlenecks between them.

- Lassnig et al. [56] evaluate the behavior of several prediction methods under the presence of non-expected events. These events are translated into an increment of the workload during a short period of time, but they do not present any periodic component to be detected. The employed datasets are taken from real scientific computing applications, but the evaluation of several prediction models demonstrate their lack of adaptability to non expected events. The authors propose a method to identify the bursts produced by non expected events. However, this method does not predict bursts and it only detects them when they have already occurred.

- Casolari and Colajanni [22] propose a short-term prediction algorithm in conjunction with signal filtering for web-based systems. This approach does not exploit periodic elements in the workload nor use any knowledge base. Actually, in the evaluation the authors bring reasonably good predictions employing a few number of observations. This would make this solution feasible to address unexpected workload variations. However, no predictions are given for mid-term predictions.

- Vercauteren et al. [94] follow a hierarchical approach that divides the prediction in long term and short term components. The long-term component is modeled using frequency analysis capturing seasonal effects, while the residual short-term process is an autoregressive model. Their experimental evaluation shows an improvement of previously proposed approaches. However, the dataset used during the evaluation shows strong periodic patterns, and no evaluation of the behavior under workload variations is proposed.

- In [6] the authors propose an architecture to improve data placement in geographically distributed cloud environments. In order to provision resources with enough anticipation, they employ long-term and short-term predictions simultaneously. For the long-term they use exponential smoothing solution. In the case of the short-term, they combine EWMA (Exponential Weighted Moving Average) with signal filtering claiming that this solution is particularly appropriate for spiky signals. Their evaluation does not directly indicate the accuracy of the prediction methods. However, we can extract that the accuracy must be appropriated taking into account that the methods based on their predictions work properly.

The presented works address the workload prediction problem from significantly different points of view. The lack of a common workload for evaluating the accuracy of the prediction models makes difficult the comparison of the obtained results. The utilization of previous knowledge methods that look for matching patterns seems to have a lack of adaptability to new situations. In this sense, other solutions such as exponential models seem more suitable as they can extrapolate information from a set of past observations. Additionally, these models are general solutions which simplifies their utilization.

2.5 SUMMARY

This Chapter presents the state of the art and background concepts used in this thesis. First, we introduce the concept of user-generated content distribution systems. Afterwards, we describe solutions for content distribution based on decentralized and elastic server infrastructures. Finally, we present some works dealing with workload modeling and forecasting.

In decentralized distribution systems we observe an effort to develop solutions that leverage user data to improve content search and network configuration. However, most of the presented solutions are based on unstructured network topologies. Solutions based on these networks are easy to implement and flexible but have a limited scalability.

There is a vast work in elastic server infrastructures. However, there is not a common evaluation framework that permits to compare different infrastructures. We classify these infrastructures according to the manner they determine the system size: reactively or proactively. Reactive approaches are a common solution as they are easy to design. However, they have to employ security margins in order to have enough time to configure the system. In this sense, proactive solutions permit to anticipate the configuration to be employed. However, the described infrastructures do not emphasize the analysis of different prediction models and how these models work under unexpected workload variations.

Finally, we enumerate works in workload forecasting that propose methods to forecast the incoming workload of the system. As it happens with elastic server infrastructures, there is a wide variety of solutions. It exists a trend to design ad-hoc methods adapting or modifying queue-theory models, autoregressive models, etc. However, it is difficult to compare this ad hoc solutions with general purpose ones as there is no common dataset used in the evaluation. In general, we can conclude that employing methods leveraging exponential equations obtain a better accuracy in short term predictions. The described works do not explore the behavior of their methods during unexpected workloads, although some of them try to identify unexpected workloads after they occur.

3 HIERARCHICAL CONTENT DISTRIBUTION ARCHITECTURE

3.1 INTRODUCTION

The major goal of this thesis is to propose and study a hierarchical content distribution architecture for large scale on-line communities that combines decentralized systems and cloud-based infrastructures. This hierarchical approach permits to combine two highly scalable solutions for content distribution: P2P file-sharing systems and cloud-based infrastructures. Our goal is to integrate these two solutions into a complementary approach, in order to reduce the cost and increase scalability.

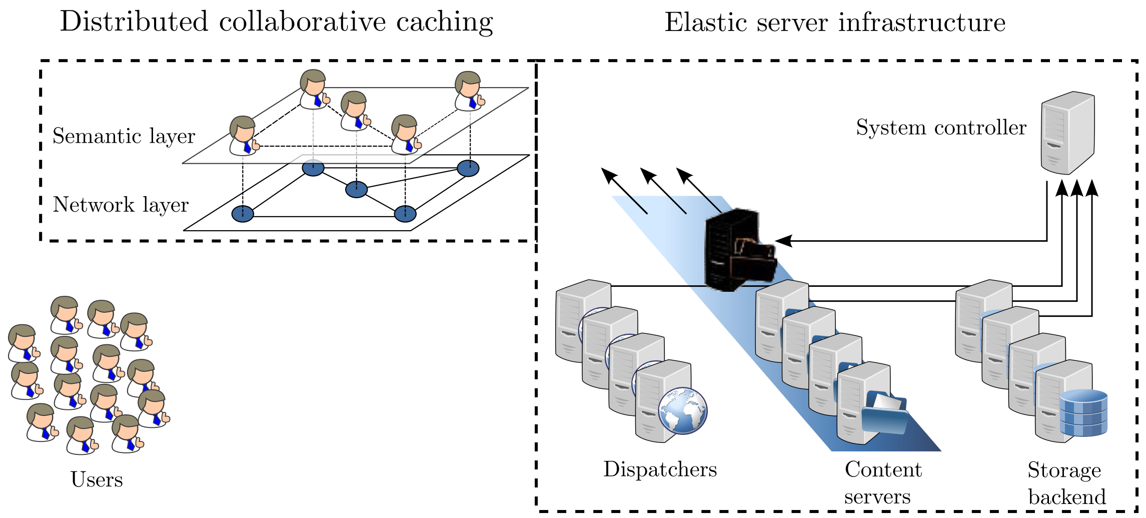


Figure 3.1: Proposed hierarchical architecture for content distribution.

In this Chapter, we overview the components of this architecture: a distributed collaborative caching system based on a P2P file-sharing solution and an elastic server infrastructure as shown in Figure 3.1. We structure this Chapter as follows. First, we describe the steps carried out to join the system and retrieve content. Second, we describe the distributed collaborative caching and their design challenges. Third, we describe the elastic server infrastructure and the challenges of designing an efficient system controller.

3.2 DATA FLOW

In this Section we present an overview of the actions occurring in our hierarchical architecture when a user accesses one item. This process is shown in Figure 3.2. In this figure we assume that the user is going to join the decentralized system. First the red user joins the distributed collaborative caching infrastructure (1). Once the user is part of the network, she needs to retrieve the content object o_i. The user starts a search employing the methods provided by the distributed infrastructure. This method returns a list of potential candidates storing o_i (2) and then the user can retrieve o_i from these candidates (3).

It is possible that the requested content is not found in the P2P system. In this case, the request is redirected to one of the available dispatchers (4). The dispatcher redirects the request to one server according to its requests distribution policy (5). The content server checks whether o_i is available in its local memory. If it is available, then the request is answered (8, 5), otherwise the content server requests o_i to the storage backend (6, 7) and finally returns the content (8, 9). Now the object is available at one of the users of the P2P system, and therefore can be accessed by the other users. In the case that a user does not join the decentralized system, she can directly send a request to the dispatchers following the data flow (from 4 to 9).

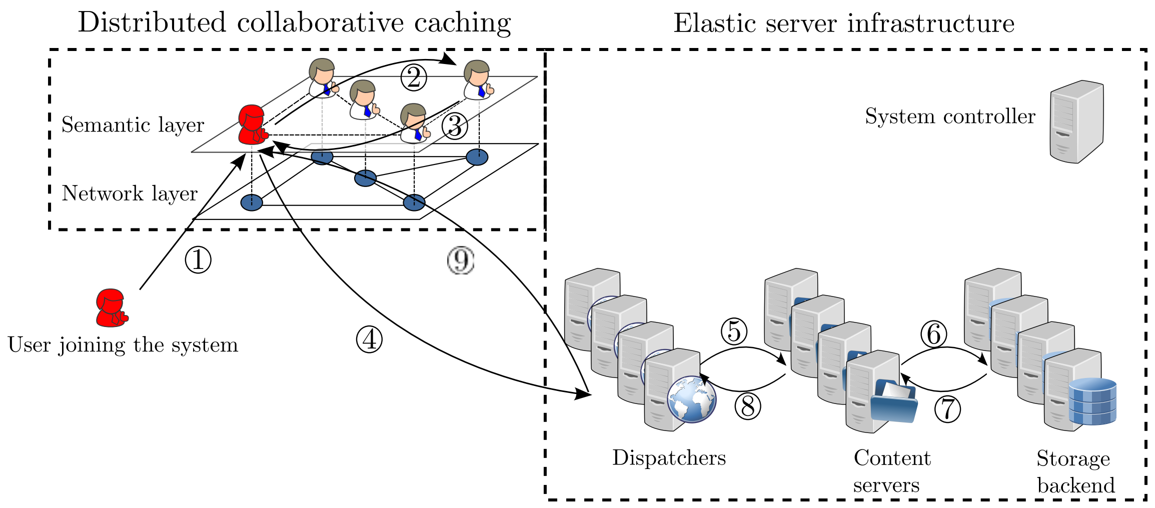


Figure 3.2: Steps to follow in order to retrieve content from our proposed hierarchical content distribution architecture.

3.3 DISTRIBUTED COLLABORATIVE CACHING

P2P file-sharing solutions have been employed to support large infrastructures with a low cost [18, 43, 59]. This is possible because users yield their resources to the community in a self-organizing network. In this manner P2P solutions can serve large amounts of content for a large number of users. Having a scalable solution with a low cost fulfill our idea of what a distributive collaborative caching infrastructure shall be.

The distributed collaborative caching infrastructure is designed to mitigate the pressure in the elastic server infrastructure. When a user requests content from the servers infrastructure, she makes this content available to other users. The more popular is the content, the higher probability to be found in the caching infrastructure exists. This can significantly reduce the traffic redirected to the server infrastructure as popular elements consume most of the traffic of the server infrastructure. In the case of elements with low popularity they will probably be requested to the servers infrastructure.

The design of this decentralized collaborative caching infrastructure involves several challenges to be addressed. Nodes join and departure constantly. The time a user remains in the infrastructure depends on the application domain. For the video on demand application described in [100] 70% of the user sessions take less than 5 minutes, for Facebook this percentage rises up to 90% [80]. This volatility of the user sessions makes necessary to employ robust self-organizing network topologies. In this sense, unstructured network topologies (Section 2.2.1) can deal with bursts of users joining or leaving the network. However, they do not seem to be the most appropriate solution as they are known to limit the scalability. On contrary, structured topologies have demonstrated to be scalable solutions.

The utilization of structured topologies may permit to have a scalable and self-organizing solution. However, employing a structure reduces their flexibility. In particular, queries have to be simple typically employing a single identifier. There is not support for content replicas as the content identifier is unique. And the classification of users into clusters is not usually supported. These limitations make structured network based solutions to be extremely dependent on how these topologies perform queries. One of the keys for the success of our caching infrastructure is to offer an efficient content search mechanism in terms of latency and recall. In this sense, previous works have demonstrated the benefits of employing clusters or groups of interests to increase content locality in P2P solutions [30, 70, 13]. Incrementing content locality reduces the search latency while improving content recall. However, these works only present solutions employing unstructured topologies as clusters and complex queries are not supported by structured topologies.

The design of our distributed collaborative caching infrastructure has to be based on a P2P file-sharing solution. This would permit to have a scalable infrastructure with a low cost. However, we have to additionally facilitate content search to make it attractive to users. One approach is to combine structured network topologies with the utilization of clusters of interests. However, as we have mentioned before this structured networks limit the search of content to simple solutions. We particularly focus our design on the modification of existing structured topologies in order to support clusters of interesting.

3.4 ELASTIC SERVER INFRASTRUCTURE

The elastic server infrastructure follows a classic three-layers architecture. The cloud facilitates to dynamically scale the number of employed system resources. This is particularly suitable for horizontally scalable architectures such as the one we use. However, efficiently exploiting the dynamic resources allocation mechanisms offered by clouds strongly depends on understanding and controlling the dynamics of workloads and on reducing the data traffic inside the infrastructure. Increasing scale and demand variations pose huge challenges on developing, deploying, and evaluating control mechanisms and policies for efficient resources allocation.

The three tiers of our architecture are managed by a system controller. This controller monitors and configures the three tiers in order to adequate the system to the incoming demand. This configuration has to guarantee enough resources in order to serve the incoming demand. Additionally, the configuration has to reduce the traffic generated between the content servers and the storage backend in order to avoid bottlenecks. Then, this configuration has to indicate the number of machines to employ, the content distribution among the servers and the redirection of requests to these servers. Defining the content distribution among the servers has to be done in order the employ the machines independently of system size variations. Additionally, an intelligent distribution of the requests among the servers might improve the content locality reducing the accesses to the storage backend. This aspect is particularly important as accessing to the storage backend increases the time to serve a request.

The design space for the system controller is vast and difficult to explore. Although the three tiers of the architecture are scalable, we focus on the scalability of the content server tiers. We understand that this is the most important layer in terms of applying elastic methods. The storage backend can also be elastically scaled but it implies to take into account how data is stored, the different existing file systems and how they work, etc. These aspects are outside the scope of this thesis. On the other hand, the pressure in the content dispatchers is small compared with the traffic moved by the content servers. For these reasons, we study and explore the applicability of elastic solutions for the content servers layer. In order to do this, we propose a modular system controller that permits the combination of different methods and strategies facilitating the exploration of several solutions. Content distribution is a complex problem that involves the definition of several policies. We consider necessary to explore combinations of these policies as components of a complete solution.

Our system controller is designed to analyze the interplay of the techniques mentioned below:

- First, building prediction models of workload variations at various granularities. Prediction models can provide the system with an anticipated vision of its future status permitting to have an adequate configuration. However, defining these models requires a background knowledge of fields such as statistics or time series analysis. We assume that the users of any elastic server infrastructure do not have enough background in these areas. For this reason, we address the utilization of automatically generated prediction models to abstract the user away from the model definition process.

- Second, the system has to adapt to both permanent changes in workload patterns and unexpected workload spikes. Prediction models has to provide accurate predictions while adapting to unexpected workload changes. Unexpected workload spikes are impossible to predict. For this reason, we propose to employ adaptive mechanisms that redefine inaccurate models based on the obtained accuracy.

- Third, determine the system size based on demand variations. This can be done by employing prediction models and calculating the amount of resources needed to serve the predicted demand.

- Fourth, employ techniques for grouping data. Using data groups has been demonstrated to be more efficient that employing single items [46]. By defining groups which members are highly probable to be accessed in a short time, we can improve content locality. Additionally, defining groups can reduce the overhead of control structures that may have to manage millions of single elements.

- Fifth, defining content distribution policies to reduce the number of accesses to the storage backend. Redirecting user requests to servers trying to maximize the hit rate in the content servers reduces the accesses to the storage backend. However, in an elastic system the variations in the number of available servers needs of additional techniques in order to maintain content locality independently of the system size.

3.4.1 *Summary*

In this Section, we describe a hierarchical content distribution architecture aimed at providing high scalability while reducing costs. This architecture combines a P2P file-sharing and cloud-based infrastructures. We employ the P2P file-sharing system as a distributed collaborative caching infrastructure to reduce the pressure in the server infrastructures. Additionally, we propose to employ an elastic server infrastructure that configures accordingly to the incoming demand. This configuration varies the number of employed resources and the data distribution policies in order to maximize the resources utilization while reducing the traffic in the storage backend. Finally, we overview some of the design challenges to address for each of the components of this proposed hierarchy.

4 DISTRIBUTED COLLABORATIVE CACHING

One of the three main objectives of this thesis is to explore a decentralized content distribution solution that leverages users preferences and community knowledge. In the previous Chapter we have presented our hierarchical content distribution architecture and their components. One of these components is a distributed collaborative caching solution designed to reduce the pressure in backend infrastructures while improving content locality.

We base the distributed collaborative caching design on a P2P file-sharing solution. P2P solutions have been used for designing large scale content-sharing systems [45, 18, 30, 83, 64]. However, classic P2P solutions do not leverage for the exhaustive metadata provided by Web 2.0 applications, nor take into account the capabilities of users to categorize and distribute data. As mentioned in the previous Chapter existing structured network topologies provide solutions with high scalability. However, their structure limits the design of solutions employing social-oriented features that might be an important improvement in terms of content locality, search latency and recall.

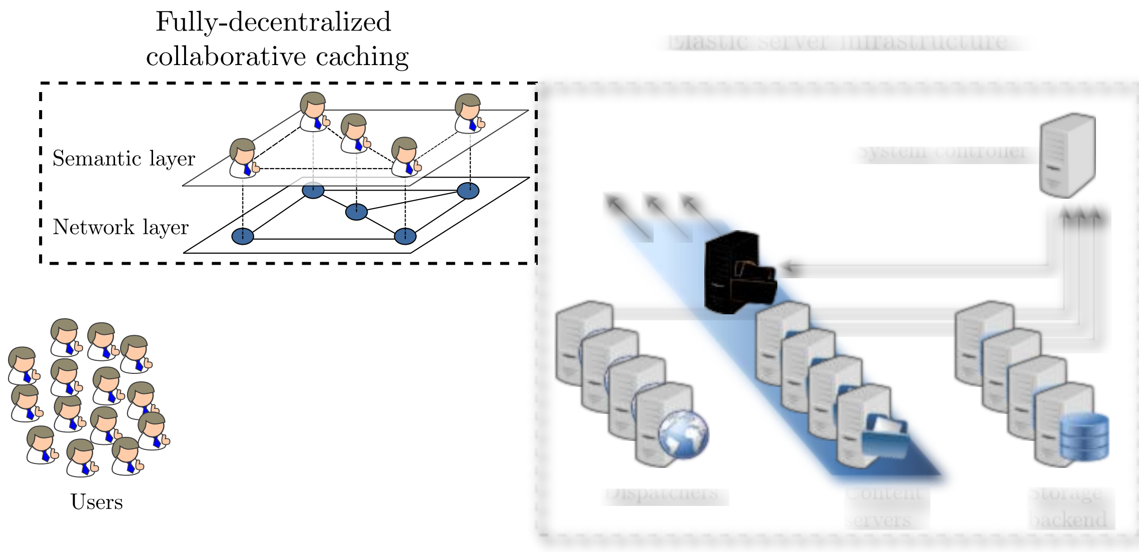


Figure 4.1: Decentralized collaborative caching infrastructure inside the hierarchical architecture.

In this Chapter, we describe our distributed collaborative caching solution as part of our proposed hierarchical architecture for content distribution (Figure 4.1). The remainder of this Chapter is organized as follows. First, we overview the P2P file-sharing solution our caching infrastructure is based on which is called AP2P (Affinity P2P). Second, we present the AP2P architecture. Third, we describe the mechanisms offered by AP2P. Finally, we present an experimental evaluation.

4.1 AP2P OVERVIEW

Our distributed collaborative caching solution is based on AP2P. AP2P is a distributed file-sharing cluster-based locally-aware self-organizing system that leverages collaborative classifications in order to self-organize. We propose self-organization to increase content locality to achieve a high data recall with low latency and low network traffic through the utilization of clusters of interest. Furthermore, we define methods for efficient content retrieval and determine the most adequate clusters for users. Our distributed collaborative has the following major features:

- We employ a Chord-like [83] overlay with three main differences when compared to Chord. First, our logical nodes correspond to Chord nodes, with the difference that Chord identifiers are generated through uniform hashing, while our identifiers are a concatenation of a cluster identifier and a node identifier generated through uniform hashing. While Chord generates a global uniform distribution of node identifiers, we locally distribute node identifiers uniformly inside each cluster. Second, resource placement is not done based on node identifiers: each peer stores the resources it shares, as in many real applications scenarios. Third, interest-based clusters of peers are supported by extending the identifier space with a cluster identifier field. All the other overlay maintenance operations of Chord are preserved with the same functionality.

- We use labeled clusters of interests. Each cluster is labeled with a category. Labeling a cluster with a category does not mean that all the resources of its peers are classified into the same category, but that the peers have a sufficient number of resources classified in that category. This permits users to join clusters when their interest in that category is growing.

- Each peer joins at least one primary cluster, corresponding to the category to which the highest number of its resources are classified. Additionally, a peer may join one or several secondary clusters, if a sufficiently large number of its resources are classified in the category corresponding to the secondary clusters. Peers automatically change their primary and secondary clusters over time, adapting to changes in the resources they make available.

- The semantic closeness between any two clusters is estimated by an affinity matrix. The affinity matrix is leveraged in order to place clusters with close semantic content close to each other in the overlay. The affinity matrix is not an exact global measure of content distribution, but rather an approximation periodically updated.

- Our lookup algorithm localizes first the cluster of interest for the search and, subsequently, performs a fast parallel logarithmic flooding based on a recursive doubling strategy. The parallelism of the algorithm assures that if a node fails, the search can continue on alternative parallel branches.

- Periodically, clusters are reorganized in the overlay based on the changing affinities of the peers. This approach ensures that affine clusters are placed close to each other, thus increasing the locality of related content.

We offer a general solution in several respects. First, content can be classified into more than one category. Second, the classification is not limited to Web 2.0 based collaborative classifications: any clustering algorithm can be used in order to classify resources. Third, the categories can be split into subcategories without affecting the correctness of the proposed methods.

4.2 AP2P ARCHITECTURE

The AP2P architecture is composed of a network layer and a semantic layer [1, 13] as shown in Figure 4.2. The network layer offers basic interconnection services such as joining or leaving a cluster. The semantic layer provides the management of content-based information and offers high level services to the applications such as strategic node joins and efficient content-based searches.

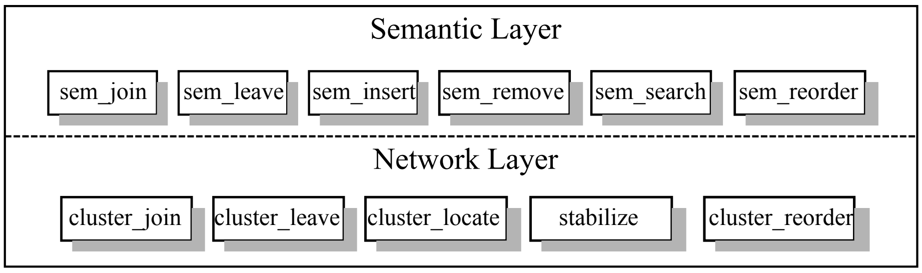


Figure 4.2: Distributed collaborative caching architecture description.

Figure 4.3 shows an example of the connections inside AP2P. Users are identified by a number e.g. in this case n_{42}, n_1 and n_{27}. This number identifies a *physical node*. A *physical node* is a user contributing with a collection of contents to the network. We define a content item or resource as a persistent object that can be stored by any user characterized by a set of attributes, including a unique identifier. In the general case, one *physical node* shares resources that belong to different clusters. These clusters group users with affine interests. Clusters are denoted by C_k, where $k = 0, 1, ..., (c-1)$, where c is the total number of clusters. In the figure above, clusters are denoted by identifiers C_3, C_5 and C_9 and correspond to contents classified into Music, Movies and Games respectively. Users may join several clusters at the same time as *logical nodes*. A *logical node* has an identifier denoted n_{kj}, whose value is the concatenation of the cluster identifier and the node identifier $n_{kj} = [C_k|n_j]$. Using this nomenclature, the *physical node* n_1 joins clusters C_3, C_5 and C_9 with logical nodes $n_{1,3}$, $n_{1,5}$ and $n_{1,9}$ respectively.

A node n_j can contribute with resources to a cluster k associated to a certain category. The number of resources user n_j contributes to cluster C_k is denoted by r_{kj}. Consequently, a cluster C_k can be formally defined as a set of nodes $\{n_{kj}\}$ such that n_j is a physical node sharing r_{kj} resources with $r_{kj} > 0$. In AP2P a node can join several clusters. However, we distinguish between primary and secondary clusters. The primary cluster corresponds to the category to which the user contributes with more resources. Secondary clusters are determined according to Algorithm 1 using the affinity matrix.

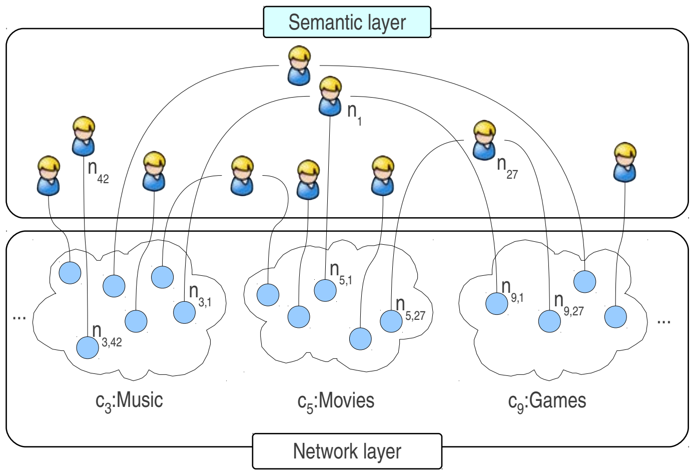


Figure 4.3: Example of AP2P two-layers architecture.

4.2.1 *Affinity matrix*

One goal of AP2P is to achieve high locality for related resources. In a real world scenario a node may share resources classified into different categories, given that a user may have different interests. The *affinity matrix* approximates the level of interest sharing among all pairs of clusters in the network and therefore, the probability that a user from a cluster may share content classified in the category of another cluster. This global approximation of the network content can be used to dynamically adapt the system configuration to maximize the content locality. More formally, the affinity matrix (A) is a square $(c-1) \times (c-1)$ matrix of the form:

$$\mathbf{A} = \begin{bmatrix} a_{00} & \cdots & a_{0(c-1)} \\ \vdots & \ddots & \vdots \\ a_{(c-1)0} & \cdots & a_{(c-1)(c-1)} \end{bmatrix} \tag{4.1}$$

where each element a_{kt} is computed as:

$$a_{kt} = \frac{R_{kt} + R_{tk}}{\sum_{s=0}^{c-1}(R_{ks} + R_{sk})} \tag{4.2}$$

where R_{kt} is the number of resources accessible through cluster C_k and classified in the category of cluster C_t. The a_{kt} values represent the sum of the resources stored by the nodes with primary cluster C_k and classified in the category of cluster C_t and the resources stored in nodes with primary cluster C_t and classified in the category of cluster C_k divided by the total number of resources accessible on cluster C_k. The affinity matrix is used to determine the placement of the clusters in the overlay: affine clusters should be placed near each other in order to improve cross-cluster locality. This algorithm is described in Section 4.4.4. Placing clusters with affine content close to each other is a heuristic to minimize the distance between them. Searches from one cluster to another are with high probability targeted to affine clusters; in this situation the latency to locate the target cluster is reduced.

The affinity matrix does not accurately reflect the system state at each instant. Actually, it works as an approximation of the current system state. In order to calculate R_{kt} we employ the monitoring services used by the elastic server infrastructure. However, it is possible to get these values in a distributed manner by spawning agents as proposed in [33]. The agents are spawn periodically from nodes bordering the clusters in clockwise directions. Each agent travels using the successor links and sums up the contribution of each node to all clusters (R_{kt}) values. These values are an approximation, as nodes may leave and join the network, while the agent traverses the overlay. In the second pass, the agent distributes the R_{kt} values to all nodes, which can compute the new affinity matrix. New nodes joining the system receive the current affinity matrix from their neighbors.

4.2.2 *Chord-based network structure*

Users allocated in clusters together with other users sharing similar interests increases the probability to find relevant content. Additionally, this reduces the search space improving search performance. However, structured P2P networks are not designed to work with clusters nor replicated content. For this reason, we propose a modified version of the Chord [83] DHT that supports clusters. This DHT is employed to locate clusters inside the network rapidly. Once a cluster has been found, the semantic layer can search the requested content.

Each logical node maintains one *finger table* of size $2 \times m$, where m is the number of table entries. The entry i in the finger table points to the logical node at distance 2^i in both clockwise and counterclockwise directions, where distance is the path length between two nodes following exclusively ring edges. Unlike in Chord where the i^{th} entry in the finger table of node identified by n points to the first node that succeeds $n + 2^i$, in AP2P the i^{th} entry points to the logical node at distance 2^i. Figure 4.4 shows an example of a clockwise finger table with 4 entries. Entry i in the finger table points to the node at distance 2^i. Note that fingers are cluster-agnostic. For instance, the last finger of node $n_{3,4}$ from cluster C_3 points to a node in cluster C_5.

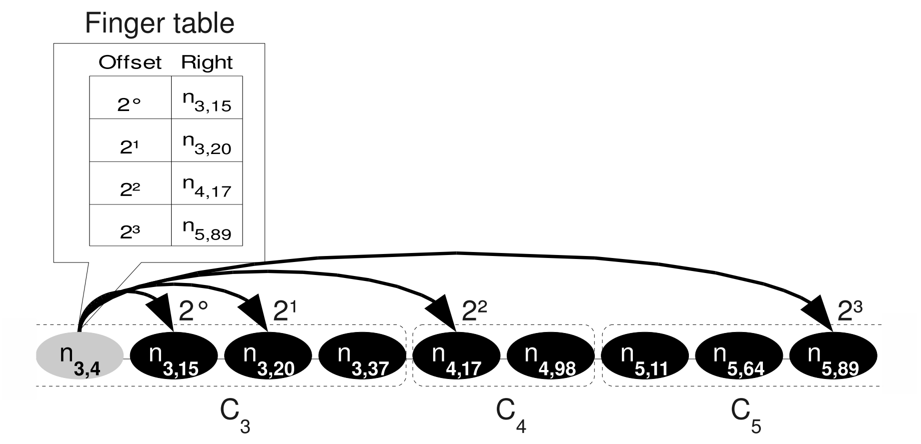


Figure 4.4: Example of a finger table with 4 entries.

4.3 NETWORK LAYER

The network layer provides mechanisms for the construction and maintenance of the overlay network. This layer is not semantics-aware. Below we describe the main operations of this layer:

- *cluster_join*(n_{kj},n_i) receives as parameters a logical node n_{kj} and an arbitrary physical node n_i and attaches the physical node n_j to the cluster C_k through the intermediation of n_i. Subsequently, the successor and predecessor nodes are informed and the Chord-like stabilize method is called in order to update the corresponding finger tables. This operation is called by the upper content-aware join operation after deciding the primary and secondary clusters of a physical node.

- *cluster_leave*(n_{kj}) receives as parameter a logical node n_{kj} and removes the physical node n_j from the cluster C_k. When leaving, the successor and predecessor nodes are informed and the Chord-like stabilize function is called in order to update the finger tables of all involved nodes.

- *cluster_locate*(C_k) locates and returns an arbitrary node from the cluster C_k. This operation works similarly to the key retrieve algorithm from Chord. The invoking node looks at its finger table for an entry from the target cluster. If the node does not find the target cluster, the locate operation consults the cluster placement order table and delegates the locate operation to a logical node closer to the target cluster. This operation is used by join and stabilize operations from the network layer and the search operation from the semantic layer.

- *cluster_reorder*(C_0, C_1, C_2,..., C_{c-1}) is called by the upper layer for changing the order of cluster placement on the overlay. The parameter is an ordered list of clusters. If the parameter order is different from the current order, each involved cluster is cut from the present location and reinserted to the new location. This is an expensive operation, as it may involve a substantial stabilization process. However, it is not expected to be invoked frequently, as cluster cross-locality evolves slowly, given that the interests of users do not change fast. Additionally, when a new placement is necessary, it likely involves a small number of cluster reorderings.

- *stabilize()* is similar to the method with the same name from Chord: brings the finger table of invoking nodes to a coherent state, i.e., the entries point clockwise and counterclockwise to 2^i hops away.

4.4 SEMANTIC LAYER

The semantic layer contains operations that leverage collaborative classifications in order to improve content locality via a self-configuring topology. These operations may use the support offered by the network layer.

4.4.1 *Node join*

The *sem_join*(n_j,n_i) operation connects the physical node n_j to the network through the intermediation of an arbitrary physical node n_i, as described in Algorithm 1. First, n_j

contacts a known node n_i and receives the affinity matrix A. Second, the node computes and joins its *primary cluster*. Finally, the node joins its *secondary clusters*. The intuition behind the algorithm for choosing secondary clusters can be summarized as follows: a node is replicated in secondary clusters only if it stores enough resources classified in the category of the secondary clusters. The number of such resources should be larger than an absolute and a relative thresholds. Both thresholds are necessary in order to avoid nodes with a negligible number of relevant resources, increasing the overhead with practically no benefit. The absolute threshold is computed as the multiplication of an absolute factor of logical node presence (α) and an estimation of the total number of resources classified to the category corresponding to cluster C_k, denoted T_k. The relative threshold is given by the affinity matrix values (a_{kt}) multiplied by a relative factor of logical node presence (β). Both α and β values are network wide constant parameters and can be chosen empirically based on application requirements.

Algorithm 1 sem_join(n_j,n_i)

```
1: Request and receive affinity matrix $A$ from $n_i$
2: $p_j = t$ such that $r_{tj} = \max_{k=0}^{c-1}(r_{kj})$ /* Determine the primary cluster */
3: cluster_join($n_{p_j j}$,$n_i$) /* Join the primary cluster */
4: for $k \in \{0, 1, ..., c-1\}$ do
5:    /* $T_k$: total number of resources classified in
6:    the category of cluster $C_k$ */
7:    $T_k = \sum_{s=0}^{c-1} R_{ks}$
8:    if $r_{jk} > \alpha \times T_k$ then
9:       /* $r_{jk}$ documents stored in the node $n_j$ and
10:      classified to the category of cluster $C_k$ */
11:      $ratio \leftarrow r_{jk} / \sum_{t=0}^{c-1} r_{tk}$
12:      if $ratio > \beta \times a_{pk}$ then
13:         /* Join secondary clusters */
14:         cluster_join($n_{kj}$,$n_i$)
15:      end if
16:   end if
17: end for
```

4.4.2 *Resource insertion*

The *sem_insert(*$resource$*)* function adds a resource to the local library of a physical node. Given that the physical node can have several points of logical presence in the network, the inserted resource becomes visible in the primary and all secondary clusters. Before insertion, a resource must be classified in at least one category.

The insertion of resources may cause changes in the relative contribution of a node to different clusters. The node decides autonomously to join a new secondary cluster, to leave a secondary cluster or to choose a different primary cluster, by using the same procedure described in Algorithm 1.

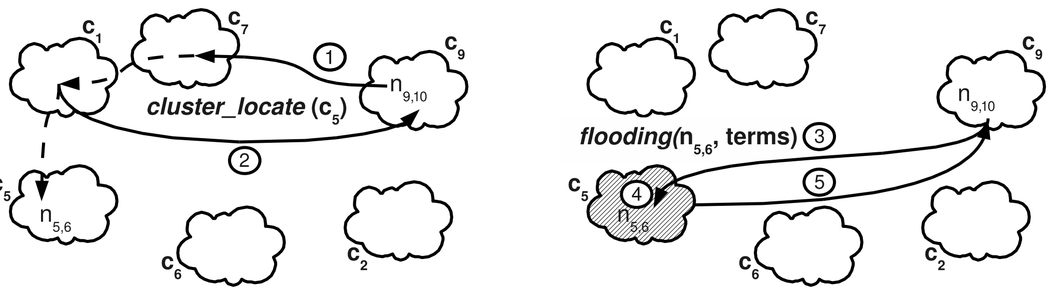
4.4.3 *Resource search*

The *sem_search(*$\mathtt{search\ terms}$*,* C_k*)* operation (Algorithm 2) receives a set of terms and the cluster a resource is classified into. The set of terms can contain wildcards, exact filenames, etc. The search consists of two phases. First, a node initiates the search, using the network layer operation `cluster_locate` to find any node in the target cluster C_k. Second, a parallel flooding algorithm is used in the target cluster C_k in order to locate the nodes that store the searched resources.

Algorithm 2 sem_search(search terms, C_k)

```
1: /* Locate the target cluster */
2: $n_{kj} \leftarrow$ cluster_locate($C_k$)
3: /* Initiate parallel flooding */
4: $r \leftarrow 0$
5: while $r <$ max_relaunches and insufficient results returned do
6:     flooding($n_{kj}$, search terms)
7:     $n_{kj} \leftarrow$ last node of the previous flooding
8:     $r \leftarrow r + 1$
9: end while
```

Figure 4.5 shows a simplified search operation in five steps. In step 1 node $n_{9,10}$ invokes `cluster_locate` for finding a node in cluster C_5. A pointer to node $n_{5,6}$ is found in cluster C_1 and is returned to $n_{9,10}$ in step 2. In step 3, $n_{9,10}$ contacts $n_{5,6}$ which starts the flooding search (step 4). Finally, the nodes with matching resources answer directly to the source node $n_{9,10}$ (step 5).

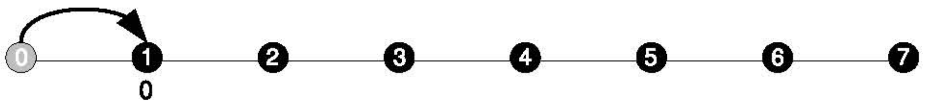


(a) Node $n_{9,10}$ locates a node ($n_{5,6}$) in cluster C_5 using `cluster_locate` operation. The `cluster_locate` identifies in the finger table of node $n_{9,10}$ a node in a cluster closer to destination according to the cluster order table. The operation is repeated in subsequent nodes until cluster C_5 is reached.

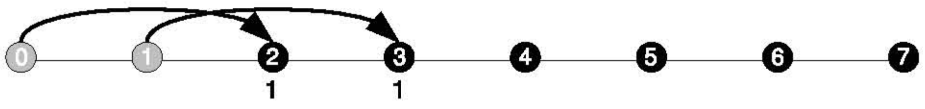
(b) Node $n_{5,6}$ starts flooding search in cluster C_5. Those nodes having the requested resource reply to $n_{9,10}$. In case the flooding is not successful, the negative answer is returned by the last node reached by flooding.

Figure 4.5: Different phases involved in the search process.

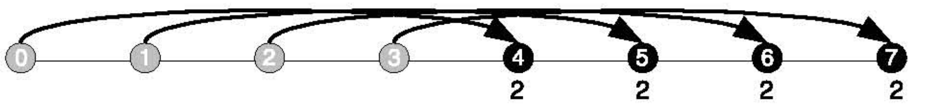
Unstructured network systems such as Gnutella [75], use flooding for content location. One of the main disadvantages of Gnutella flooding is the generated redundant communication, which reduces the scalability of the system. AP2P employs a parallel flooding algorithm based on a recursive doubling strategy. We define the flooding horizon (FH) as the maximum distance from the source to the farthest node reached by a flooding message. In our algorithm, at step i, all the peers that have received a search message forward it in parallel to peers at distance 2^i. Therefore, the flooding horizon is reached in $\log_2 FH$ parallel steps. Each node that receives a search message and has a match replies to the source. Negative responses are reported to the source only from the nodes reaching the boundary of the flooding horizon. The source can subsequently decide to relaunch the search beyond the current flooding horizon in order to locate more replicas.



(a) Flooding step 0.



(b) Flooding step 1.



(c) Flooding step 2.

Figure 4.6: Example of a flooding search with $FH = 8$ on 8 neighbors on the ring.

Figure 4.6 shows an example of a flooding search for FH=8. For clarity, the figure does not show replies from visited nodes. In the initial step (Figure 4.6a) node 0 sends a message to node 1 (finger table entry 0^{th}). In step 1 (Figure 4.6b), 0 and 1 send messages in parallel to nodes 2 and 3 (finger table entry 1). In step 2 (Figure 4.6c) nodes 0, 1, 2 and 3 send messages in parallel to 4, 5, 6 and 7 (finger table entry 2). Nodes 4, 5, 6 and 7 reach the flooding horizon and the search is stopped. This flooding generates no redundant messages, visiting 7 nodes and requiring 7 messages.

For reaching the flooding horizon, the flooding algorithm needs $s = \log_2 FH$ steps and generates $1 + 2 + 2^2 + ... + 2^{s-1} = FH - 1$ requests followed either by a maximum number of $FH - 1$ replies or $FH/2$ negative acknowledgements. Therefore, assuming that $FH > 1$, the upper bound on number of messages involved in the search can be calculated in the following way:

$$N_{max} = FH - 1 + max\left(FH - 1, \frac{FH}{2}\right) = 2 \times FH - 2 \tag{4.3}$$

Given that the search is started simultaneously clockwise and counterclockwise, the upper bound is $4 \times FH - 4$. Equation (4.4) gives the total number of messages generated in the worst case for AP2P. We assume a limited FH, therefore relaunching searches is needed to cover the whole cluster.

$$msg_{AP2P} = effective\ msg + relaunch\ msg = |C_k| + \frac{|C_k|}{FH} \tag{4.4}$$

4.4.4 *Cluster reordering*

AP2P self-organizes the placement of the clusters on the ring topology in order to improve the inter-cluster locality. The reorganization of clusters is infrequent, as it is not likely that the interests of users change very quickly. The function *sem_reorder()* is automatically invoked by any node of the system at regular intervals in order to check if cluster reorderings may improve locality and to effectively perform the reordering.

Algorithm 3 sem_reorder()

```
1: /* Greedy */
2: cluster_order_table ← greedy_order(A)
3: /* Reorder clusters */
4: cluster_reorder(cluster_order_table)
```

The reorder operation consists of two steps. In the first step, the greedy algorithm described below is used for computing a new cluster placement order table. The second step reorders the clusters based on the result of the greedy algorithm.

The greedy algorithm (Algorithm 4) works as follows. First, it computes the maximum non-diagonal element of the affinity matrix $a_{row,col}$. The row and col values are used for placing the first cluster C_{row} and the second cluster C_{col} to the right of C_{row}. Second, the value of the column col becomes the new row; the next cluster is decided by choosing the maximum value of the affinity matrix on the row, given that the column does not correspond to a cluster, which has already been chosen. The second step of the Algorithm 4 is repeated until all the clusters are placed. We call this algorithm GREEDY-MAX, as it greedily targets to maximize the total affinity as a sum of affinities of all pairs of neighbors.

Algorithm 4 greedy_order(A)

```
1: for $k \leftarrow 0$ to $c-1$ do
2:     cluster_order_table[k] ← not_assigned
3: end for
4: (row, col) ← (k,t) such that $a_{kt} = \max a_{ij}$, where $0 \leqslant i, j < c$ and $i \neq j$
5: $cluster\_order\_table[0]$ ← row
6: for $k \leftarrow 1$ to $c-1$ do
7:     cluster_order_table[k] ← col
8:     row ← col
9:     col ← j such that $a_{row,col} = \max a_{row,j}$, where $0 \leqslant j < c$ and $j \notin$
       $cluster\_order\_table[h]$ for $h = 0, 1, ..., k$
10: end for
11: return cluster_order_table
```

For instance, for the affinity matrix in Figure 4.7, the GREEDY-MAX algorithm returns the following cluster order: 0, 3, 1, 4, 2. The total affinity is 0.25 + 0.20 + 0.20 + 0.15 +0.15 = 0.95. If we apply the same greedy algorithm, but choose the minimum affinity between pairs of clusters (GREEDY-MIN), a possible cluster order is 3, 4, 2, 0, 1 with a total affinity of 0.75, representing 78% of the total affinity returned by GREEDY-MAX.

$$\mathbf{A} = \begin{bmatrix} 0.30 & 0.10 & 0.15 & 0.25 & 0.20 \\ 0.10 & 0.35 & 0.15 & 0.20 & 0.20 \\ 0.15 & 0.15 & 0.35 & 0.20 & 0.15 \\ 0.25 & 0.20 & 0.20 & 0.30 & 0.05 \\ 0.20 & 0.20 & 0.15 & 0.05 & 0.40 \end{bmatrix} \tag{4.5}$$

Figure 4.7: Example of affinity matrix for 5 clusters.

Finally, in the second step of the reordering algorithm (Algorithm 3), the solution of the GREEDY-MAX algorithm is used in order to update the cluster placement order in the overlay network.

As a final observation, the accuracy of affinity matrix and the placement order of the clusters on the ring do not affect the correctness of the search, but only its performance in terms of latency reaching the target cluster. Therefore, the update of the affinity matrix and the reorder frequency can be relaxed, trading off reorganization overhead for search efficiency.

4.4.5 *Other operations*

Apart from the operations previously described, there are other two operations worth mentioning: *sem_leave*(n_j) and *sem_remove*(resource_id). The *sem_leave*(n_j) operation disconnects the logical node from the primary and secondary clusters by using *cluster_leave* operation. The *sem_remove*(resource_id) operation removes the resource identified by resource_id from the node. As in the case of insert, this operation may cause a node to change its primary cluster, or to leave or join secondary clusters.

4.5 EXPERIMENTAL RESULTS

We experimentally evaluate the performance of AP2P using real traces complemented with information extracted from YouTube. First, we present the methodology used for data collection and workload characterization. Second, we describe our evaluation setup. Third, we evaluate the small world properties presented in the clusters defined in AP2P. Fourth, we analyze the sensitivity of clusters placement. Fifth, we evaluate the effects of replication on search latency. Sixth, we analyze the sensitivity of AP2P to the network size. Finally, we evaluate content locality.

4.5.1 *Workload characterization*

AP2P experimental results are based on a set of real traces collected in [87], that contain information about YouTube traffic captured from June 2007 to March 2008 on a campus network. Using the video identifiers and the YouTube public API, we collect information about the user who uploaded each video, and all the videos uploaded by that user until June 2008. We also extract the number of views per video and the category of each video. The videos appearing in [87] but not available at the moment of harvesting were discarded. The obtained traces contain 23,076 users sharing a total of 952,718 distinct videos.

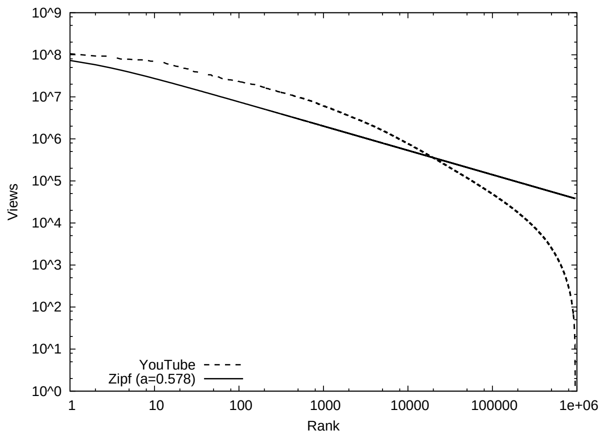
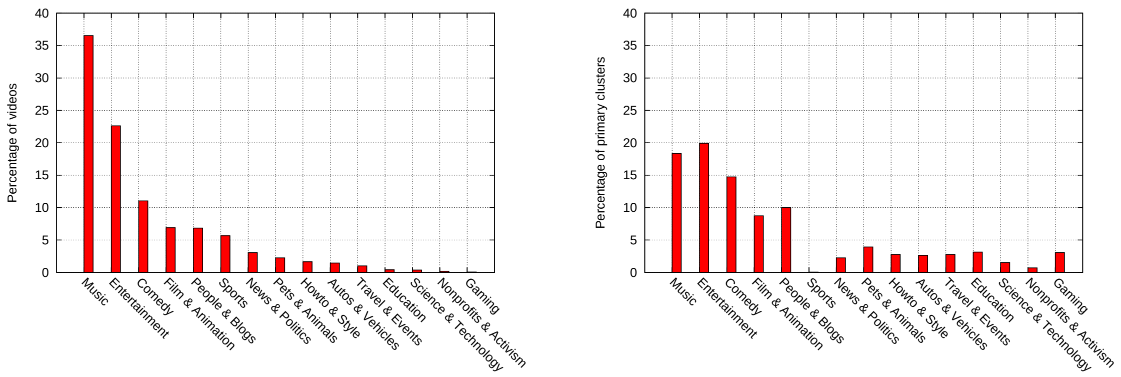


Figure 4.8: YouTube videos rank ordered by number of views.

Figure 4.8 shows the video popularity distribution of the YouTube trace. First we note that this distribution is similar to those obtained in other studies [28, 23]. From our trace, only 0.08% of the videos have more than a million views, and 50% of the videos have less than 2300 views. We try to model the resulting distribution using a Zipf distribution, but this distribution overestimates the number of views for 98% of the videos and underestimates the remaining 2%. This situation is also discussed in [28], where the authors note that the Zipf distribution does not predict the existence of many popular videos. Additionally, we observe that the top 2% most popular videos aggregate 35,603 million views, which represent 70% of the total number of views.



(a) Distribution of videos per category. For instance, 37% of videos from the trace are classified in the music category.

(b) Classification of nodes based on their maximum contribution. For instance, 20% of the node have the majority of their videos classified in the Entertainment category.

Figure 4.9: Distribution of users and videos according to YouTube categories.

YouTube predefines a set of 15 existing categories. In our trace each video was classified by the user who uploaded it into exactly one of the 15 video categories. Figure 4.9a shows the percentage of videos classified in each category. Videos classified in *Music* and

Entertainment represent more than 50% of the sample. Other categories such as *Education*, *Gaming* or *Nonprofits & Activism* have a small representation. These values confirm the recreational character of YouTube, as shown in other studies [28, 23] where we find similar distributions.

Figure 4.9b shows the primary cluster distribution calculated for the YouTube trace. We note that the distribution of videos per category does not directly correspond to the distribution of primary clusters: e.g., music videos represents 37% of the videos in the system, but only 18% of the users joined Music as their primary cluster. This phenomenon corresponds to the fact that the Music videos are popular to the large majority of users, including users having their primary interests in other clusters.

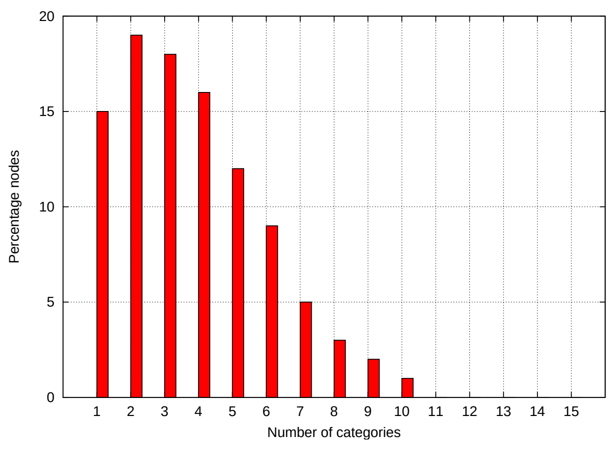


Figure 4.10: Number of categories a user is interested in. For instance, 15% of the users have videos classified only in one category.

Users may be interested in one or several categories. Figure 4.10 shows the percentage of nodes that share videos in one or more categories. We notice that most users are interested in more than three categories, with an average of 3.3. Interest in more than 5 categories is unusual, whereas users interested in just one category represent 15% of the total.

4.5.2 *Experimental setup*

We experimentally evaluate AP2P simulating the network and user behaviors using the aforementioned YouTube workload. We simulate different networks composed of physical nodes that correspond to one of the 23,076 users contained by the traces. These nodes join primary and secondary clusters as explained in Section 4.4.1, and then search for different contents. The simulations are carried out in a Java multi-threaded discrete event simulator. The simulations were performed on two Sun SPARC Enterprise T1000 with 8 GB of RAM and 8 multi-threaded cores.

We use three metrics in this evaluation: the rate of successful searches, latency and recall. The rate of successful searches measures the ratio of searches returning at least one resource. Latency is defined as the number of hops until the first replica of a resource is encountered. Recall is the number of replicas returned by the search divided by the total number of replicas in the network.

A summary of parameter values used in the experiment is shown in Table 4.1.

Parameter	**Values**
Physical network size	3000-5000 nodes
Logical network size	8897-14718 nodes
Number of clusters (c)	15
Finger table size (m)	6
α	0
β	0.5
FH	64

Table 4.1: Parameter values used in the experiments

We set the flooding horizon value with value $FH = 64$, i.e., 6 entries from the finger table were utilized. The search was performed simultaneously in clockwise and counter-clockwise directions, without relaunches. The number of logical nodes in the network depends on the number of secondary clusters the nodes decide to use. We control how the nodes join to secondary clusters with the parameters α and β used in the join operation described in Algorithm 1. In particular, we set $\alpha = 0$ (no restriction) and $\beta = 0.5$.

The construction of the simulated network consists of the following steps. First, n users and their shared resources are read from the trace. Each user corresponds to a new physical node. Second, each video is replicated on the physical nodes according to one of the popularity distributions shown in Figure 4.8. The nodes on which replicas are placed, are chosen randomly with a uniform distribution from the nodes already sharing resources in the same category as the candidate video. The number of replicas is proportional with video popularity as depicted in Figure 4.11 for the three replica distributions used for evaluation: rd1, rd2 and rd3. The shape of these distributions is similar to the popularity distribution depending on the video age showed by Cha et al. [23]. Third, the affinity matrix is computed according to Equation 4.2. Fourth, the physical nodes are inserted one by one as logical nodes in the AP2P network by calling Algorithm 1. Fifth, the affinity matrix is updated to reflect the new network state.

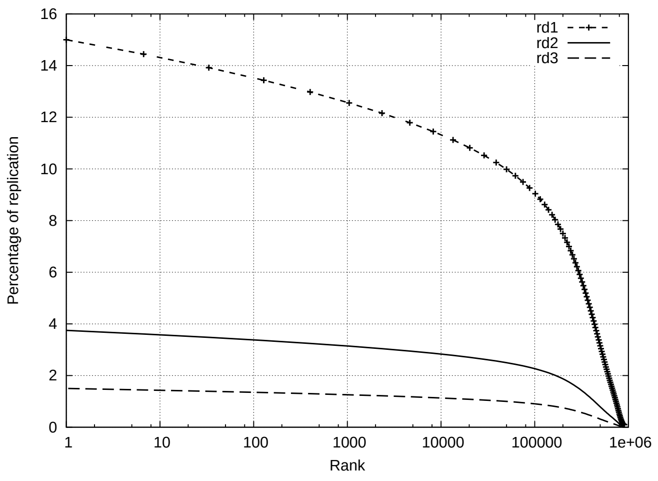


Figure 4.11: The three replica distributions used in the simulation.

For each simulation we run 100,000 search events. The number of generated search events for a particular video is proportional to its popularity (the number of views). The origin node of the search is chosen randomly with a uniform distribution.

4.5.3 *Small World Analysis*

A small world is a kind of graph defined by Watts [97] based on two metrics: clustering coefficient (γ) and characteristic path length (L). The *clustering coefficient* represents the probability that the neighbors of a vertex are connected. It is calculated as the average of the number of connections among the neighbors of all vertices of a graph. The *characteristic path length* is the mean of the path lengths between all pairs of vertices of a graph. A graph is *a small world* if, compared with a random graph with the same number of vertices and edges, it has roughly the same characteristic path length and a significantly larger (orders of magnitude) clustering coefficient. Formerly a graph is a small world if satisfies $L_0 \approx L_{random}$ and $\gamma_0 \gg \gamma_{random}$.

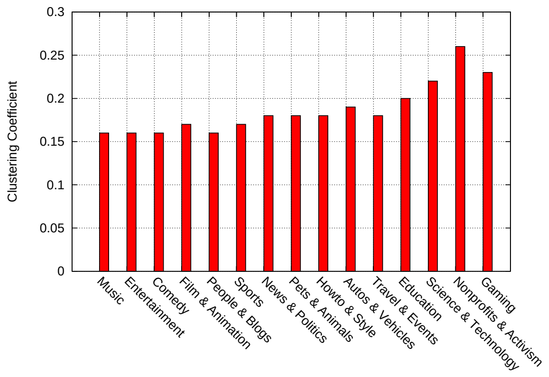
It is known that small world graphs present certain properties such a high degree connectivity, which in large networks permit to reduce the total number of hops between distant nodes. The analysis presented in this Section targets to highlight network characteristics related to the potential efficiency of content lookup both at local level (AP2P cluster) and global level (AP2P network). Locally, a small world with a low average path length inside a cluster will indicate that a flooding search is more likely to reach all nodes storing relevant content in an efficient way. Globally, the small world property suggests that cluster lookup is likely to be quickly directed to the target cluster.

	L	L_{random}	γ	γ_{random}
$AP2P_{logical}$	14.06	3.57	0.16	0.003
$AP2P_{physical}$	2.57	2.48	0.15	0.0262
Gnutella 0.6 [84]	4.17-4.23	3.75	0.018	0.00038
Gnutella 0.4 [50]	3.30-4.42	3.66	0.02	0.002
MovieActors [97]	3.65	2.99	0.79	0.00027
PowerGrid [97]	18.7	12.4	0.08	0.005
C. Elegans [97]	2.65	2.25	0.28	0.05

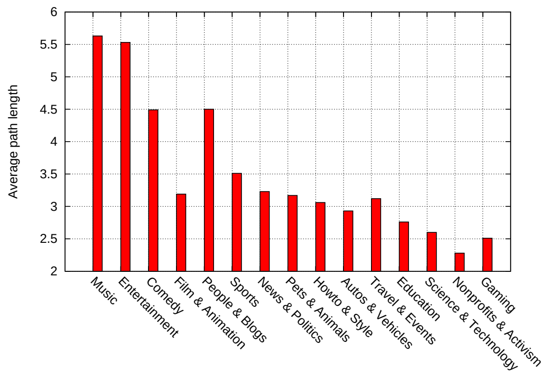
Table 4.2: Comparison of clustering coefficient and characteristic path length for various real networks.

Previous studies [50, 84, 97] demonstrate that Gnutella and many real networks (e.g., actors, power grid and C. Elegans) exhibit small-world properties. Table 4.2 shows γ and L computed in these studies along with the values for AP2P logical and physical networks corresponding to our YouTube dataset. We note that the characteristic path length for $AP2P_{logical}$ network is 14.06 which is roughly $\log(n)$, corresponding to the characteristic path length of a Chord-like ring. However, in AP2P each physical node has several logical presences in the network. The γ and L values for the $AP2P_{physical}$ show that AP2P satisfies the small world conditions. This fact shows that by using logical nodes, two physical nodes can be connected with less hops acting like shortcuts.

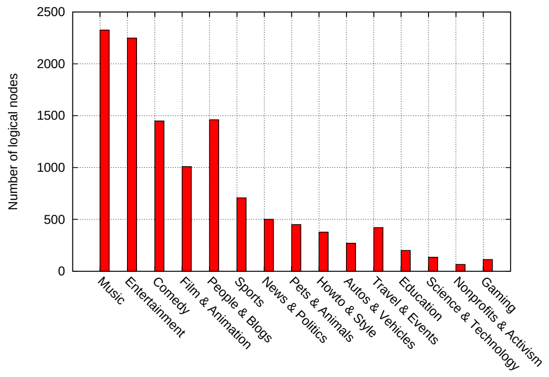
AP2P is structured as a collection of interconnected clusters. In order to investigate the local characteristics of each cluster, we computed the values for local characteristic path length of a cluster L_{local} and clustering coefficient of a cluster γ_{local}. This approach was inspired by the Watts' model [97] used for analyzing the local and global length scale of a small world graph.



(a) γ_{local}



(b) L_{local}



(c) Cluster size (number of logical nodes)

Figure 4.12: γ_{local}, L_{local} and cluster size for $AP2P_{logical}$

Figure 4.12 shows the values for γ_{local}, L_{local} and the number of logical nodes of each cluster. The clustering coefficients of all clusters are between 0.16 and 0.26. Nevertheless, the clustering coefficients of popular clusters are within a small range, i.e., 0.16 and 0.18 and do not appear to depend on the cluster size; for instance, the difference between sizes of clusters Music and Auto & Vehicles is one order of magnitude. L_{local} values appear to depend on the cluster size. Figure 4.13 shows that this dependence is sublogarithmic. These characteristics show that the content is highly clustered inside each AP2P cluster with a low average path among nodes. Consequently, the flooding inside each cluster is highly probable to reach relevant content with a low latency (number of hops).

4.5.4 *Cluster placement sensitivity analysis*

In this subSection we evaluate the benefits that can be achieved from a proper order of cluster placement on the AP2P ring. A good order translates into a high content locality

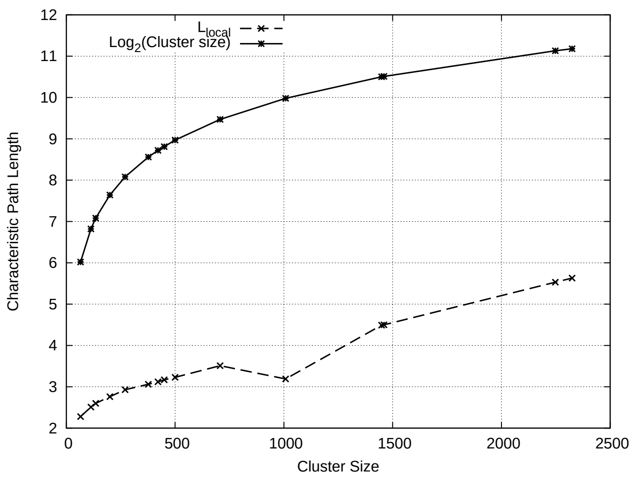
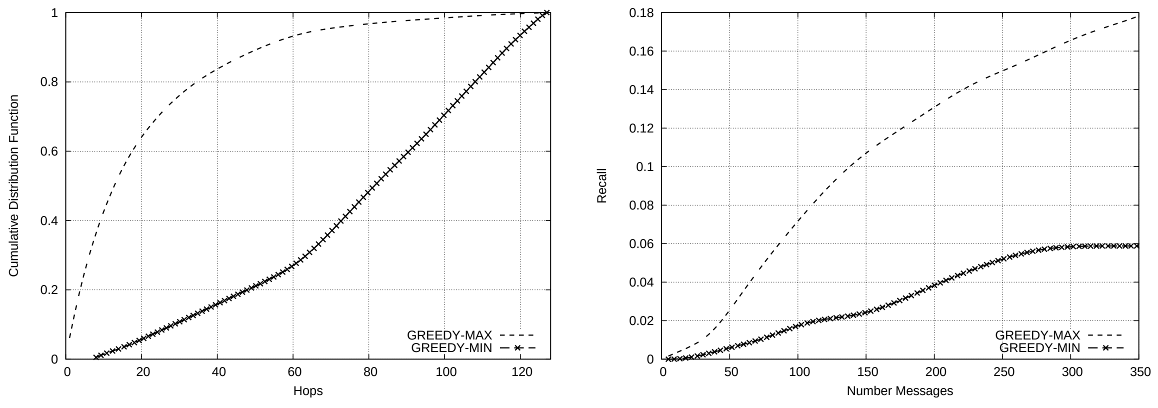


Figure 4.13: Variation of cluster-local characteristic path length (L_{local}) with the cluster size.

across clusters. For the evaluation we have computed two possible orders by applying the GREEDY-MAX and GREEDY-MIN algorithms on the same network.

Figure 4.14 compares search latency and recall for 4,000 nodes. We note that the order produced by GREEDY-MAX results in significant improvements in both latency and recall. In terms of latency, for GREEDY-MAX order more than 80% of the searches return the first replica in 40 hops. For the same number of hops GREEDY-MIN order finds the first replica only for less than 20% of the searches. The recall scales almost linearly with the number of messages for both orders. However, for GREEDY-MAX the slope is significantly greater. For instance, for 300 messages the recall is almost 17% for GREEDY-MAX and approximately 6% for GREEDY-MIN. GREEDY-MAX and GREEDY-MIN can be seen as an approximation of the upper bound and a lower bound on a spectrum of different possible cluster orders. The locality of content is improved by GREEDY-MAX, as affine clusters are placed near each other.



(a) Latency comparison.

(b) Recall comparison.

Figure 4.14: Comparisons of GREEDY-MIN and GREEDY-MAX algorithms in a 4,000 nodes network using replica distribution rd2.

4.5.5 *Effects of replication*

In this subSection we present an evaluation of the effect of replication on the search latency. We have used the three replication distributions introduced in Section 4.5.1 for a network composed of 4,000 physical nodes after applying GREEDY-MAX reordering algorithm. Figure 4.15 shows the latency results obtained from simulations.

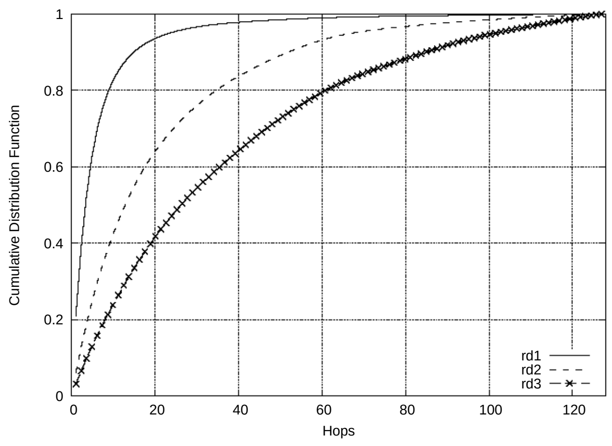


Figure 4.15: Latency comparison for a 4,000 nodes network using different replica distributions.

We note that a higher replication implies a lower latency. For instance, for rd1 more than 95% of the searches find the first replica in at most 20 hops. For comparison, in the same number of hops 65% and 40% of the searches return the first result for rd2 and rd3, respectively.

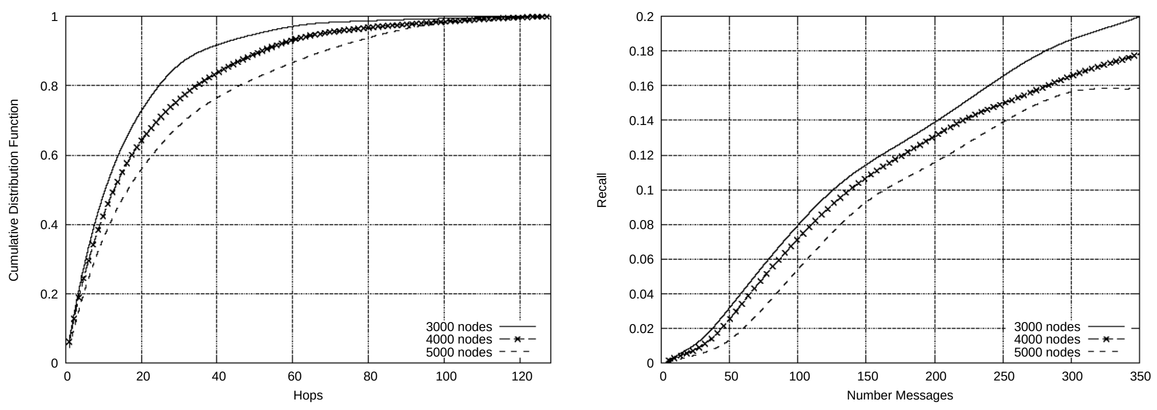
4.5.6 *Sensitivity to network size*

In this Section we evaluate the sensitivity of latency and recall to the AP2P network size. Figures 4.16a and 4.16b show the results when the network size increases from 3,000 to 5,000 physical nodes, while maintaining unchanged all the other parameters. The replication distribution used in this experiment was rd2. The sizes of AP2P network in terms of logical nodes were 8,897, 11,732 and 14,718, for 3,000, 4,000 and 5,000 physical nodes, respectively.

It can be noticed that the variation of both latency and recall are small, when increasing the number of nodes by 66% (from 3,000 to 5,000 physical nodes). For instance, in numerical terms, for 20 hops, 17% more searches return the first replica for 3,000 nodes than for 5,000 nodes.

The recall (Figure 4.16b) also decreases with the increase in the number of nodes. These results are intuitive, as the distance between replicas is expected to increase with the number of nodes and the FH is unmodified. Recall can be improved by increasing the FH as a function of the network size. AP2P recall results are good compared with other solutions. For example, the efficient search algorithm from [49] shows a recall value of 14% for a 5,000 nodes network, whereas AP2P obtains a recall value of 16%.

Table 4.3 shows a comparison of the search success rate for different network configurations. As expected, the success rate increases with the degree of replication. The success rate decreases slowly with the number of nodes showing good scalability with



(a) Latency comparison.   (b) Recall comparison.

Figure 4.16: Evaluation of scalability for different network sizes using replica distribution rd2.

		Number of nodes		
		3000	4000	5000
Replica distribution	rd1	0.9968	0.9884	0.9910
	rd2	0.9783	0.9553	0.9520
	rd3	0.9134	0.8624	0.8014

Table 4.3: Rate of successful searches for different number of nodes and different replica distributions.

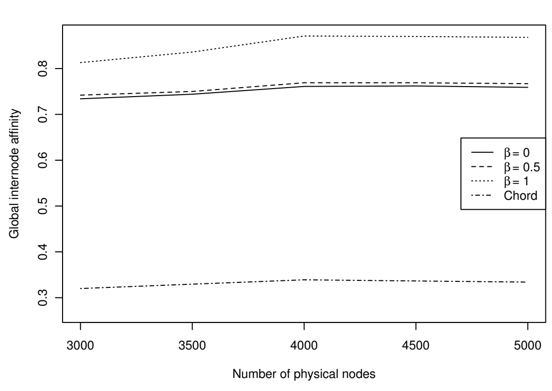
network size. For instance, increasing by 66% the number of physical nodes (from 3,000 to 5,000), the success rate is reduced by 0.5%, 2.7%, and 11.2% for the three replication distributions. It can also be noticed that the decreasing of the success rate is slower for higher degrees of replication. However, in our simulations the flooding horizon (FH) was kept constant at 64. The success rate directly depends on FH and, it can be increased by choosing a higher FH.

4.5.7 *Locality evaluation*

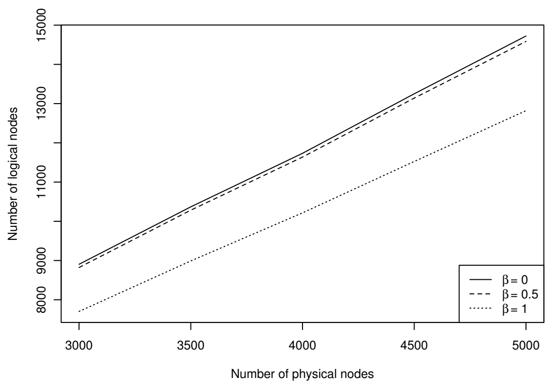
In order to estimate the global locality we compute the average inter-node affinity for all the connected nodes in AP2P and Chord. For each node n, we compute the vector v of size k, where v_k is the percentage of resources of node n classified in the category of cluster C_k. Given two nodes n_1 and n_2 with vectors u and v, we define inter-node affinity as $MAX(MIN(u_i, v_i)) \forall i \in \{0, 1, ..., c-1\}$. Intuitively, MIN operation computes the taste overlap per category for two users, while MAX computes the category with highest taste overlap. For instance, the inter-node locality is 1 for two nodes storing resources only in Music category, and 0 if one node stores exclusively resources classified in Music and another one exclusively resources classified in Movies.

In order to compare Chord and AP2P locality we build both networks using the same number of physical nodes and finger table size. Then, we insert the same documents in the network. Finally, we compute the average internode affinity for both Chord and AP2P networks.

Figure 4.17 plots (a) the average inter-node affinity and (b) logical node size for physical network size ranging from 3,000 to 5,000 nodes and for β from algorithm 1 ranging from 0 to 1. If $\beta = 0$ a node is added to secondary clusters if it has at least one resource in that cluster. This value is expected to reduce the affinity and to increase the logical network size. If $\beta = 1$ a node is added to secondary clusters if the number of its resources classified in that cluster is larger than the affinity between node's primary cluster and target cluster. This value is expected to increase global affinity and decrease logical network size (as nodes are added to secondary clusters selectively based on a high affinity). The value of α was fixed to 0, meaning that a node is considered to be added in secondary clusters based only on the relative number of resources classified in the respective category.



(a) Average inter-node affinity sensitivity to the physical network size and relative factor of logical node presence (β).



(b) Logical network size sensitivity to physical network size and relative factor of logical node presence (β).

Figure 4.17: Locality evaluation of AP2P network.

The average inter-node affinity for Chord represents less than half of the minimum value for AP2P and less than a third of the maximum value. As expected, the highest locality is obtained in all cases for $\beta = 1$, that is when secondary clusters are joined selectively. In this case the affinity in each cluster is increased, minimizing the probability of finding less affine resources. When decreasing β, the global locality decreases, the logical network size increases (Figure 4.17(b)), while the probability of finding resources with the lower degree of affinity faster increases.

4.6 SUMMARY

In this Chapter, we have explored a P2P file-sharing solution that is the base of the distributed collaborative caching infrastructure of our hierarchical architecture. This file solution, called AP2P (Affinity P2P), leverages users preferences and community knowledge in order to improve content locality and search latency. AP2P employs a cluster-based locality-aware self-organizing peer-to-peer network that leverages collaborative classifications in order to self-organize for a higher content locality. AP2P self organizes at two levels: node and cluster level. Each node may decide autonomously to join or leave clusters based on the own composition of content or interests. Clusters may change their positions in order to achieve a high inter-cluster improving search latency.

Based on analytical and experimental results, we make the following claims. AP2P shows small world characteristics, and, therefore, facilitates the localization of content. By joining the network at different points as logical nodes, physical nodes offer high content locality in different clusters, substantially improving the search performance. Additionally, the dynamic reorganization at cluster level provides an improvement in both search latency and recall. In particular, we find that cluster reorganization can improve latency 100% and multiply by three recall compared with inadequate cluster organizations. We also demonstrate empirically that replication of content according to popularity may be leveraged for achieving a lower latency with a relative small amount of replication. Finally, we show that the search latency, recall and rate of success scale smoothly with the number of nodes.

5 WORKLOAD MODELING

In the previous Chapter we presented a distributed collaborative caching infrastructure designed to mitigate the pressure in the elastic server infrastructure. When a requested content is not found inside this caching infrastructure, the request is forwarded to the elastic server infrastructure. This is highly probable to occur in the case of newly available and unpopular contents. Additionally, we can expect a portion of users to access the server infrastructure without joining the decentralized caching infrastructure. This decentralized infrastructure is not elastic in the sense that we cannot control its size. However, our elastic infrastructure must adapt its size to the incoming workload in order to save costs.

In order to determine the system size we need to understand the dynamics of the workload to be served. As is described in the related work of this thesis (Chapter 2) the workload of web applications is known to have periodic patterns. Additionally, several studies [7, 23, 37, 29, 28, 65, 91, 100, 56, 26] have revealed the workload of web systems to be highly variable depending on seasonal factors (time of the day, day of the week), trends, expected and unexpected events. While the periodicity of the workload makes easy to determine the amount of resources to employ, the variability makes necessary to overprovision resources to avoid the incoming workload to exceed the available resources. However, this incurs in a waste of resources during low workload periods [42, 11]. Anticipating the incoming workload may permit to adequate the system size increasing machine utilization and reducing costs. This problem can be done using several techniques. However, we particularly focus on the utilization of automatically defined autoregressive models.

This Chapter covers the second objective of this thesis by investigating the data access patterns of Internet applications and analyzing the limits of workload predictability. We explore the automatic generation of time series models in the prediction of web systems workloads. First, we overview the automatic generation of prediction models. Afterwards, we describe three workload datasets extracted from real systems and finally we provide an experimental evaluation based on these datasets.

5.1 FORECASTING OVERVIEW

The workload of a system can be modeled as a time series X_t with values $(x_{t-k}, x_{t-k-1}, \cdots, x_{t-2}, x_{t-1})$ where t is the current time. Further, the workload can be represented as the addition of four different time series:

$$X_t = T_t + C_t + S_t + R_t \tag{5.1}$$

where T_t, C_t, S_t and R_t are the trend, cycle, seasonality and random components of the time series. The trend T_t describes the long term movement of the time series. The cyclic component C_t describes cyclic behaviors with a constant level in the long term. Seasonality consists of patterns with fixed length influenced by seasonal factors (e.g., the month, the day of the week). Finally, the random component R_t is an irregular component to be described in terms of random noise. Additionally, the time series may have outliers. The outliers are observations of the time series that are unlikely to be a result of the observed process. These exceptional situations are particularly difficult to detect and are manually included inside the models.

Modeling X_t can be described as the addition of its components. These models are generated based on the observations of past X_t values. In this thesis, we only focus on approaches that can automatically identify and generate these models for a set of past observations $(x_{t-k}, ..., x_{t-1})$ with length k. These observations are enough to define a model $M(X_t|x_{t-k}, ..., x_{t-1})$ for X_t. In this way, we can employ a model generator whose input is a set of observations and the output is the generated model (Figure 5.1). This process abstracts the user from the details of the model generation.

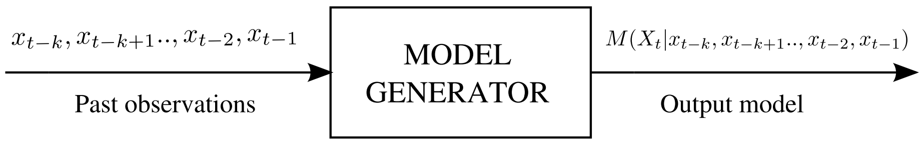

$x_{t-k}, x_{t-k+1}.., x_{t-2}, x_{t-1}$
Past observations
MODEL GENERATOR
$M(X_t|x_{t-k}, x_{t-k+1}.., x_{t-2}, x_{t-1})$
Output model

Figure 5.1: Generation of time series models.

Finally, a model can generate a set of predictions $(\tilde{x}_{t+1}, \tilde{x}_{t+2}, ..., \tilde{x}_{t+h})$ for a prediction window h using a set of observations (Figure 5.2).

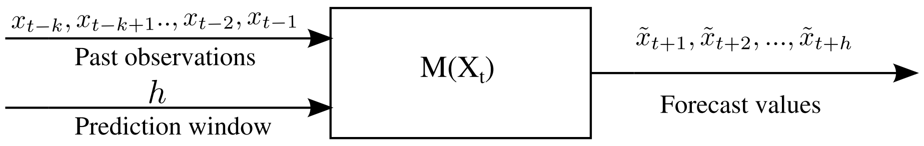

$x_{t-k}, x_{t-k+1}.., x_{t-2}, x_{t-1}$
Past observations
h
Prediction window
M(X$_t$)
$\tilde{x}_{t+1}, \tilde{x}_{t+2}, ..., \tilde{x}_{t+h}$
Forecast values

Figure 5.2: Model predictions based on past observations.

In this thesis, we focus on the utilization of autoregressive models. These models predict future values of the time series by weighting past observations. The general formulation of these models is:

$$\sum_{z=i_1}^{i_m} \alpha_{t-z} x_{t-z} + \omega_{t-z} \tag{5.2}$$

where z is the time lag in the set $(i_1, ..., i_m)$, α is the parameter that weights the observation at time $t - z$, and ω is the observation modifier. The weights, lags, and modifiers of the formula above are determined during the model definition. The model definition follows three stages according to the Box-Jenkins methodology [36]: (1) identification and selection, (2) estimation, and (3) diagnostics. The first stage determines the configuration of the model. The second, adjusts the weights of the model. Finally, the diagnostics stage identifies the errors generated during the model definition.

IDENTIFICATION AND SELECTION. Autoregressive methods are designed to work with stationary series. This means that it must exist temporal correlation between lagged observations. To identify stationarity it may be necessary to employ additional analysis such as detrending or differencing the time series. When large correlations are found at defined lags, this indicates observations that particularly capture relevant information. This permits to define the lags $(i_1, ..., i_m)$ to be used in the equation 5.2.

PARAMETER ESTIMATION. Once the observations to be used are chosen, the parameter estimation sets α values from equation 5.2. These values are calculated by selecting the best values that approximate the model equation to a given set of past observations. These values are typically calculated using an estimation method such as OLS (Ordinary Least Squares) or ML (Maximum Likelihood).

DIAGNOSTICS. The diagnostics stage analyzes the errors obtained by the model. In particular, this stage analyzes the residuals and identifies temporal correlation between them. In the case that the residuals are detected to have temporal patterns, we return to stage one and repeat the process until a better solution is found.

The general approach is to generate simultaneously several models that pass the diagnostics stage and compare them using a comparison criterion. Models comparison criteria such as AIC or BIC select the model with the lowest error. Additionally, in the case of BIC the number of observations is also taken into account.

These three stages permit to automatically define a model given a set of past observations. Depending on the idiosyncrasy of each model, it may exist differences in the implementation on the stages. However, this is the most common methodology to follow in order to identify and define an autoregressive model.

5.1.1 *Autoregressive forecasting models*

Although the prediction methodology used in this thesis is independent of the employed time series model, we focus on three family models: ARIMA, AR_z and Holt-Winters. We choose these family models for their utilization in heterogeneous problem domains, their inclusion in several tools and the implementation of automatic model generation methods. Below we briefly describe the main features of these models.

ARIMA. ARIMA models define a wide family of linear autoregressive functions denoted by $ARIMA(p, d, q)$, where p, d, and q are called orders. The parameter d identifies the order of differences applied to eliminate trends, i.e.,: $d = 1$ removes linear trends, $d = 2$ quadratic trends, etc.

ARIMA models are a combination of AR (AutoRegressive) and MA (Moving Average) models with orders p and q respectively. In this way an $ARIMA$ model can be an AR model with order p denoted by $ARIMA(p, 0, 0)$. Similarly, MA models are denoted by $ARIMA(0, 0, q)$. The identification of the model parameters follows the Box-Jenkins methodology through the observation of ACF and PACF values. These criteria are summarized in the table below:

	$ARIMA(p, 0, 0)$	$ARIMA(0, 0, q)$	$ARIMA(p, 0, q)$
ACF	Tails off	Cuts off after lag q	Tails off
PACF	Cuts off after lag p	Tails off	Tails off

Table 5.1: AR and MA models selections based on ACF and PACF according to Box-Jenkins methodology [82].

ARIMA models offer prediction capabilities in non-stationary scenarios using detrending indicated by the order d. Several extensions of ARIMA models exist, being particularly useful the seasonal ARIMA models. These models improve the forecast of time series adding seasonal patterns. These models are denoted by $ARIMA(p, d, q) \times (p, d, q)_s$ where the second component is an additional ARIMA model that employs the observations with lag s.

AR_z. $ARIMA$ models are defined using consecutive observations ($(i_t, i_{t+1}, \cdots, i_{t+m})$ according to Equation 5.2). For some time series it is usual to find an order p such as 100 or greater. This means that an $ARIMA$ model has to use a linear equation employing at least the 100 previous observations. This incurs in an increment of the time needed to define the model, adds noise to the predictions, and reduces the score obtained in AIC or BIC criteria. In order to mitigate this problem, McLeod and Zhang [63] propose a subfamily of $ARIMA$ models (actually AR models) called AR_z which uses non consecutive observations. While the observations of an $ARIMA$ model are $x_{t-1}, x_{t-2}, \ldots, x_{t-p}$, AR_z uses $(i_1, \ldots, i_m)$ observations not being mandatory these values to be consecutive.

McLeod and Zhang propose a novel autocorrelation function [63] to identify the weights to employ in the AR_z models. They claim this autocorrelation to bring fast automatic identification and estimation for large and complex time series. Additionally, they describe a complete set of tools for automatic model identification following the Box-Jenkins methodology.

HOLT-WINTERS. Holt-Winters models are an extended version of the exponential smoothing equation [82] that adds trending and seasonal components. Holt-Winter models assume predictions to fluctuate around a reasonably stable mean. For this reason, Holt-Winters models have been proposed as a solution for short-range predictions [15]. These models do not determine what lags to use. The parameters are estimated by recursively setting the equation weights to the given set of observations. This reduces the elapsed time for the generation of models.

The three autoregressive family models above permit to model a wide spectrum of time series. $ARIMA$ itself permits to model seasonality, trending and periodicity. AR_z is indicated for models defined using a large number of observations. Finally, Holt-Winters is a versatile and simple predictive model with a simplified identification and selection stage.

5.2 DATASETS

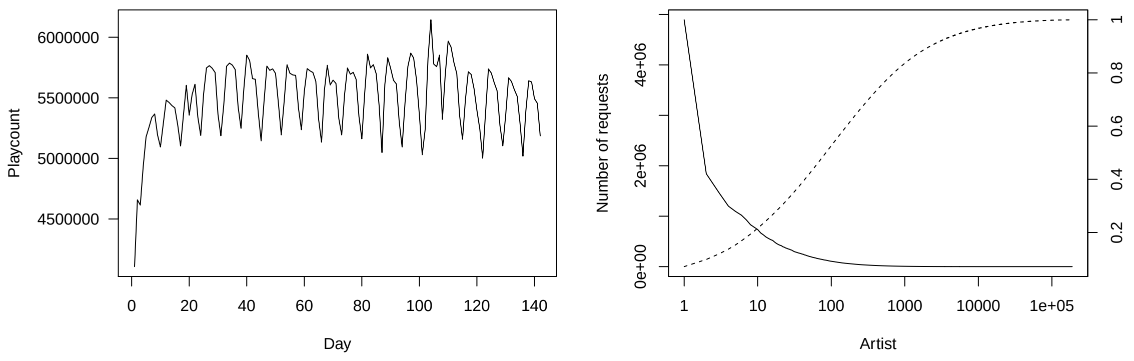
In this Chapter we evaluate different facets of workload prediction. Our evaluations have been done with real datasets taken from real systems. Prior to presenting our evaluation results, we present the datasets employed in our evaluations. In particular, we use traces extracted from the 1998 World Cup [7], Last.FM [58] and Wikipedia [91].

5.2.1 *Last.FM*

Last.FM is one of the largest music portals with social networking features. Last.FM community has currently more than 30 million users from more than 200 countries [57]. Labels and authors can freely share music on the portal. Users can listen to radio stations or to previews in either full-length or as 30 second samples. A radio station is created by

Last.FM based on different criteria: user library, recommendation, loved tracks, similar artists, tracks sharing the same tag, etc. Track order play in radio cannot be controlled by the users. Previews are generally free, while radio is a subscriber feature in most countries. Users have the possibility of connecting each other through friendship relationships. Last.FM records play counts on a daily base in order to generate charts for artists and tracks.

We crawled Last.FM through its public API by using a distributed crawler deployed in a cluster over 20 machines. We traversed the friendship graph in a breadth-first search manner and extracted the profiles of a set of 250,000 users including the listened artists, daily for the period between January 1^{st} to May 22^{nd}, 2009 (142 days). The extracted social graph has an average degree of 14.9, a diameter of 8, an average path length of 4.37, and an average clustering coefficient of 0.17 [97]. The total number of artists users listened to was 2,390,970, amounting to a total play count of 780,579,318.



(a) Last.FM daily workload.

(b) Last.FM popularity distribution.

Figure 5.3: LastFM dataset characterization summary.

Figure 5.3a shows the load evolution for the 142 days of our data trace. The first days coincide with the beginning of the year (starting January 1^{st}), when the system load was apparently lower. The long-term evolution shows a clear periodic pattern with slightly increasing and decreasing trends. The ACF analysis confirms a weekly seasonal pattern with decreasing activity during week-ends and maximum traffic in the middle of the week. Around the day 110 we observe an unexpected variation in the workload. We believe this was due to the fact that Last.FM become a pay per use service in that date.

Figure 5.3b shows the popularity distribution of the artists for the whole trace duration. The distribution has a long tail, with a small number of very popular artists and the major part of the artists in the long tail. This type of distribution is commonly found on social networks referred to different characteristics, as shown in previous works [24, 23, 55].

A limitation of the trace is the fact that we can reconstruct only the aggregate daily load of the system. Therefore, the trace does not contain load variation at minute and hour granularities. In order to synthetically extend our trace at minute granularity we leverage patterns of workload observed for other traces. In particular Gill et al. [37] plotted the hourly patterns of video accesses of a large community of YouTube users as shown in Figure 5.4. We synthetically model the hourly LastFM accesses based on the probability distribution of YouTube accesses. While this distribution is particular for YouTube (video sharing), we understand that the shape is representative for several workloads. Inside each hour, we consider a Poisson distribution of user arrivals at minute granularity.

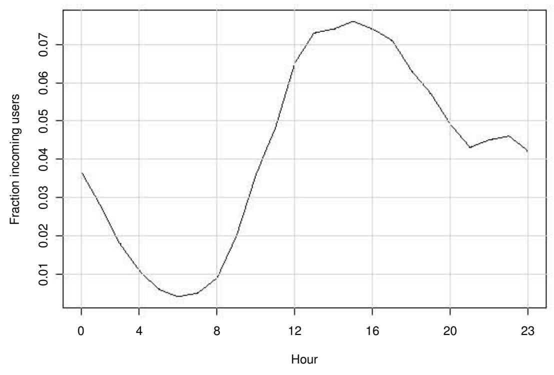
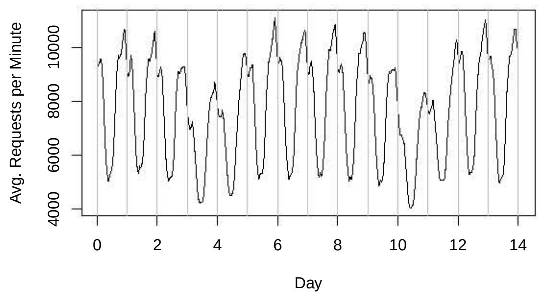


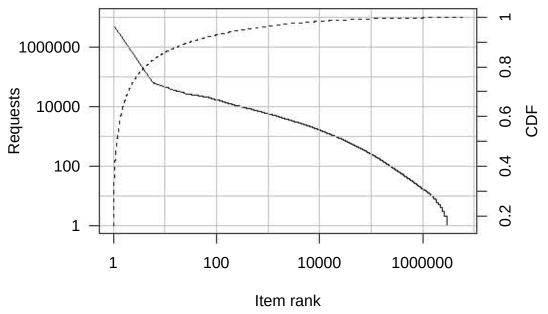
Figure 5.4: Probability of user arrivals in Last.FM dataset.

5.2.2 *Wikipedia*

The wikipedia dataset contains 10% of all the requests directed to Wikipedia proxy caches from September 19th 2007 to January 2nd, 2008 accounting for 20.6 billion requests. Each request in the trace contains a unique identifier, a time stamp and the URL of the request. For evaluation purposes, we only use requests to articles of the English Wikipedia during the first two weeks of trace. After filtering and cleaning the original trace, these two weeks account for 145 million requests and about 6 million available items.



(a) Wikipedia dataset daily workload.



(b) Wikipedia dataset popularity distribution.

Figure 5.5: Wikipedia dataset characterization summary

Figure 5.5a shows the number of requests per minute for the first two weeks. The workload is clearly periodic and similar to the one shown in [91]. There is a weekly pattern according to which the workload diminishes on weekends. During a single day, the number of requests doubles and decreases again. This daily pattern is also found in other systems [38, 37], confirming the existence of a low activity period, followed by a large peak of load and then a workload decrement. The popularity distribution of the

Name	t_0	t_1	t_2	t_3	M	V	H
w_1	100	160	1360	1439	1.05	0.01	64
w_2	900	1005	1065	1095	4.97	0.06	159
w_3	400	455	1255	1400	2.43	0.01	50

Table 5.2: Characterization of the Wikipedia synthetically generated spikes according to the methodology defined in [17].

items shown in Figure 5.5b follows a long tail distribution, where the first 4 items account for 80% of the total requests.

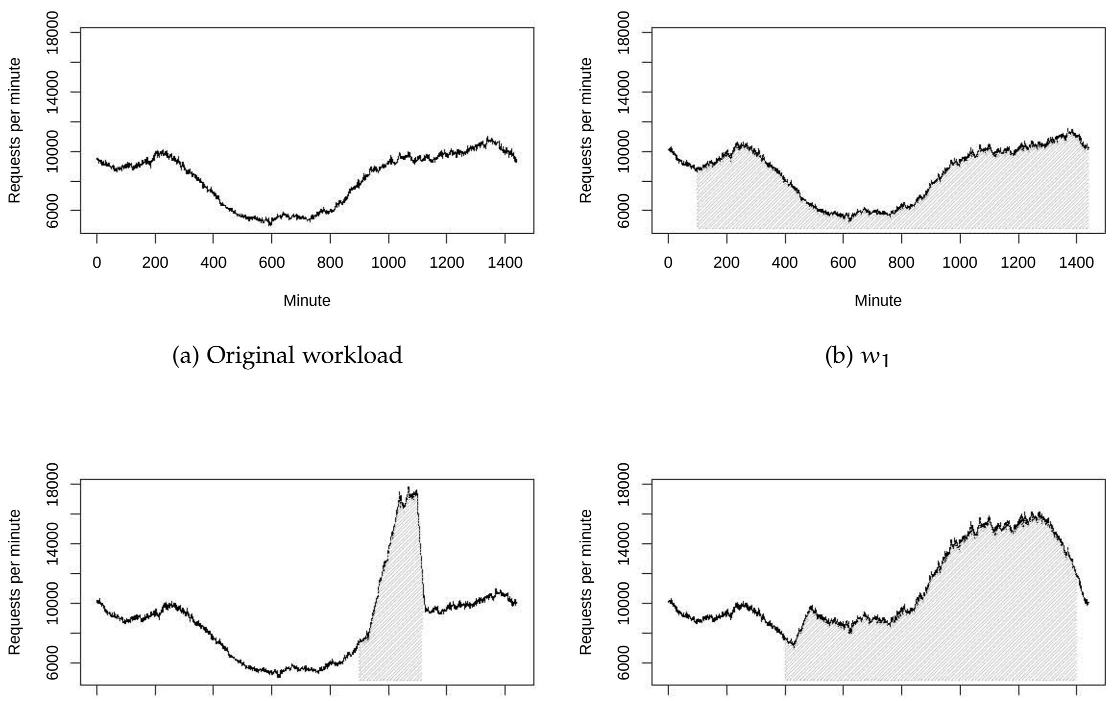
5.2.3 *Synthetic generation of workload peaks*

The Wikipedia dataset contains a large number of objects during a large period of time. The popularity of the items in the dataset is known to vary [91]. However, the total workload does not present any unexpected variation. For this reason, we synthetically modify this dataset to include three workload spikes using the methodology provided by Bodík et al. in [17]. From that work we select three significant spikes based on real traces. These are the requests served by Wikipedia after Michael Jackson's death (w_1), server demand variations during the World Cup 1998 [7] (w_2), and, finally a peak demand to Ebates.com servers [15] (w_3).

According to this methodology, a spike is identified by a tuple (t_0, t_1, t_2, t_3, M, V, H). The values t_0, t_1, t_2, and t_3 indicate the times when the spike starts, when it reaches its peak, the end of the peak period, and the end of the spike, respectively. M is the magnitude of the peak compared with the baseline workload. V is the variability of the popularity of the hotspot items during the spike compared to the baseline workload. This value has to satisfy $0 < V < \frac{H-1}{H^2}$ as defined in the methodology. In the three scenarios we select the largest two decimals value that satisfies the mentioned condition, this results in an exponential distribution. Finally, H is the number of hotspot items that are responsible for the increment of traffic. Table 5.2 indicates the values of these parameters used in w_1, w_2, and w_3 and Figure 5.6 shows the resulting global workloads. The shadowed area indicates the duration of the spike. For w_1 there is no significant change in the magnitude of the spike, only the popularity distribution of items varies, w_2 is characterized by a fast growth and fast decay, and finally, w_3 is characterized by a steady growth followed by a moderately fast decay.

5.2.4 *1998 World Cup*

This dataset [7] contains the requests between April 30th, and July 26th, 1998 from the infrastructure serving the 1998 World Cup videos. The workload shows the number of requests to web pages containing the videos of the event. From the original dataset we focus on the set of five consecutive days with the largest demand. This set starts from June 14th. Figure 5.7a shows the workload per hour during the five days. We observe a periodic component that repeats daily. The workload during the first two days is lower than the observed during the remaining three days. However, we still observe a similar pattern with two peaks in the morning and in the evening.



(a) Original workload

(b) w_1

(c) w_2

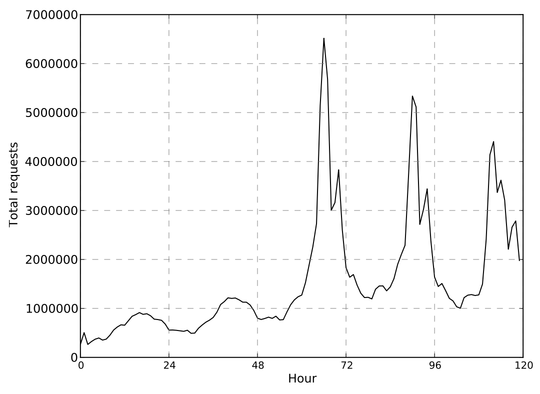
(d) w_3

Figure 5.6: Wikipedia original workload compared to the synthetically generated scenarios.

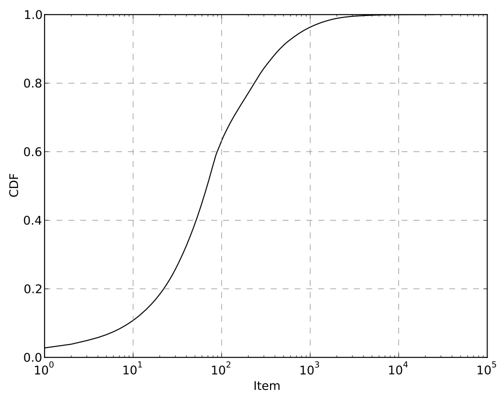
The original dataset shows the demand for 89,997 different available items. The trace excerpt we use only shows the demand for 24,933 items which accounts for 27% of the content in the original dataset. Figure 5.7b shows the popularity distribution of the items for our dataset. We estimate that 233 items (0.9% of the total) account for 80% of the total demand during the studied five days.

5.3 WORKLOAD FORECASTING

Following the time series automatic modeling approach depicted in Figures 5.1 and 5.2, three parameters have to be defined prior to get workload predictions: time granularity, prediction horizon (h), and number of observations (k). The time granularity and the prediction horizon depends on the application domain. For example, an application interested on calculating workload per hour would employ the observations per hour. The prediction horizon depends on the employed granularity and the utilization of short or long-term predictions. Finally, the number of observations to employ typically matches the number of observations needed to cover a workload pattern. In this manner, we can simplify the model identification stage by introducing time correlated observations. As a rule of thumb, the more information employed in the model generation the better. Nevertheless, many observations or a wrong selection of them might bring inaccurate models.



(a) 1998 World Cup workload per hour.



(b) 1998 World Cup workload per hour.

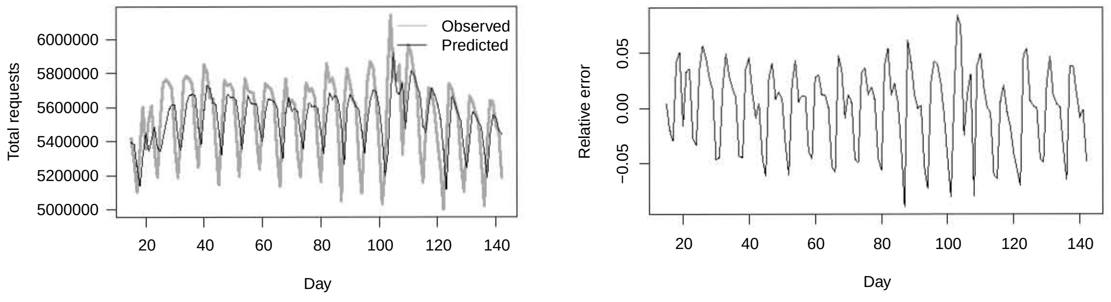
Figure 5.7: 1998 World Cup dataset popularity distribution.

In our approach to the problem of the workload prediction, we consider that the people interested in the utilization of these models are not experts in the field of time series forecasting. This assumption is directly related with the utilization of automatic model generation, where a model is expected to provide proper predictions with a minimal human intervention. Other approaches [26, 56], assume the utilization of a model defined by an expert. By using the Last.FM and 1998 World Cup datasets we illustrate how even when a workload seems easily predictable or a model seems to be working properly the utilization of manually defined models can lead to inaccurate results.

First, we show an example of how to manually define a prediction model for the Last.FM dataset. All the models and predictions are generated employing the R statistical suite [71] using the *Forecast* [99] and *FitAR* [63] packages. These packages offer a suite of methods that permit to automatically generate ARIMA, Holt-Winters and AR_z models. In a first attempt to come out with an accurate model we just apply an ARIMA(1,0,0) model with $k = 14$ for parameter estimation. The granularity of the time series is daily, therefore with $h = 1$ we predict the total workload for the next day. Figure 5.8a shows in black the predictions and in grey the observations. The model obtains a reasonable accuracy with a maximum $\pm 8\%$ of relative error. However, the periodic signal of the error in Figure 5.8b indicates the existence of temporal correlation of the errors. This, means that the model is not working properly.

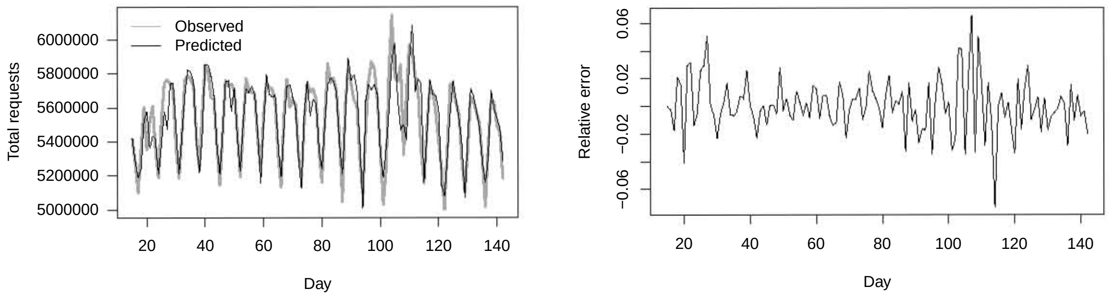
A temporal correlation in the error indicates that the model is not the correct one. In this case, there are two options: the manipulation of the time series to remove the weekly factor or to attach a seasonal factor to the model employed above. In particular, we extend the ARIMA model above to include the weekly seasonality. This model is an $ARIMA(1,0,0) \times (1,0,0)_7$ that results from adding to the previous $ARIMA(1,0,0)$ the same model with the observations of the previous week.

Figure 5.9 shows the predictions and errors after using the seasonal ARIMA model. The accuracy has been increased reducing the maximum relative error to $\pm 6\%$ instead of $\pm 8\%$ and eliminating the temporal correlation between the errors. This manual process of analysis and correction can be difficultly carried out by a person without a background in time series forecasting. Moreover, the first analysis we have carried out, although correct would have generated significant errors. This approximation might not be a challenge in



(a) Prediction obtained. (b) Relative error.

Figure 5.8: Prediction and error obtained using ARIMA(1,0,0) models in Last.FM dataset for $k = 14$.



(a) Prediction obtained. (b) Relative error.

Figure 5.9: Prediction and error obtained using ARIMA(1,0,0)x(1,0,0)7 models in Last.FM dataset for $k = 14$.

scenarios with regular and stable workload patterns like in the Last.FM dataset. In the case of scenarios with spontaneous peak loads like the 1998 World Cup, the utilization of fixed models not suitable.

Second, we consider the World Cup dataset using 5 minutes granularity observations to demonstrate that fixed prediction models are not suitable. In order to identify the orders of the ARIMA model, we follow the Table 5.1. The analysis of ACF and PACF indicates $ARIMA(1,0,1)$ to be an appropriate model. The workload is expected to be periodic with a daily pattern. For that reason we use $k = 288$ (12 observations per hour $\times$ 24 hours) to employ a whole day of observations in the model definition. The model works properly with an acceptable error until we try to define the model at time 661. At this point, the parameter estimation method does not converge. This lack of convergence, indicates that the weights of the model $ARIMA(1,0,1)$ cannot be calculated (Figure 5.10). The model is no longer valid.

This situation is known as the model change point [10]. Suitable models become inaccurate due to the dynamics of time series. In this case, we can easily note that $ARIMA(1,0,1)$ stops being accurate, as the parameter estimation does not converge.

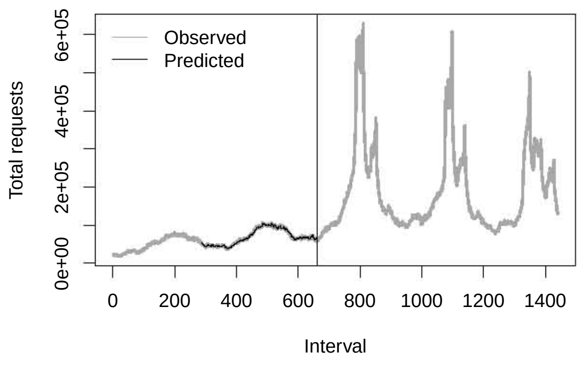
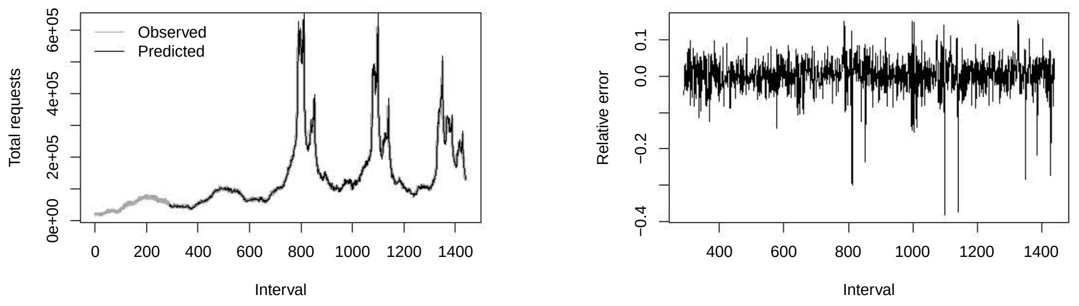


Figure 5.10: Predictions obtained using ARIMA(1,0,1) with $k = 288$ in the 1998 World Cup dataset for intervals of five minutes. The vertical line indicates the time where the model weights cannot be calculated. This indicates that the model is no longer valid.

Other prediction methods can be used to obtain working models. We employ the predictions generated using Holt-Winters without seasonal components and $AR_z(1)$ as shown in Figures 5.11 and 5.12. Both kinds of models come up with accurate predictions with ±1% average relative error, which is a reasonable accuracy.

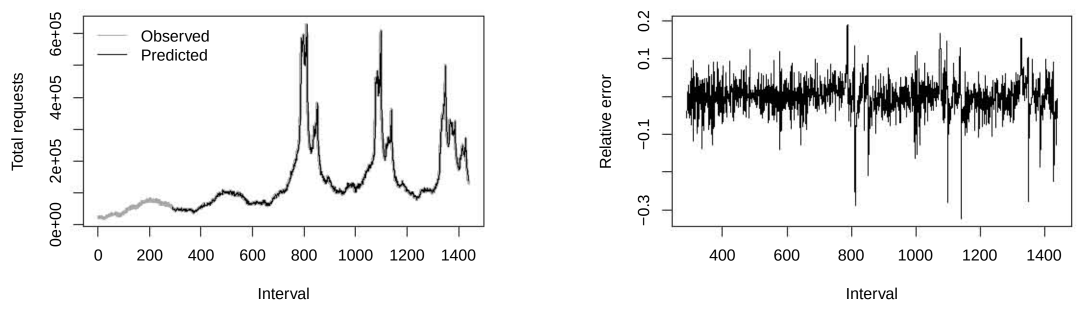


(a) Prediction obtained. (b) Relative error.

Figure 5.11: Prediction and error obtained using Holt-Winters models with $k = 288$ in the 1998 World Cup dataset for intervals of five minutes.

5.3.1 *Multimodel prediction approach*

We have demonstrated in the previous Section the complexity of manually determining the most appropriate prediction model given a set of past observations. Additionally, we demonstrate that the automatic generation of prediction methods can be used to address variations of the expected workload. Based on automatic model generation tools for each model family, we propose the simultaneous generation of multiple prediction models for the same set of past observations. Afterwards, a model selection criterion selects a candidate to generate the predictions as depicted on Figure 5.13.



(a) Prediction obtained.

(b) Relative error.

Figure 5.12: Prediction and error obtained using $AR_z(1)$ models with $k = 288$ in the 1998 World Cup dataset for intervals of five minutes.

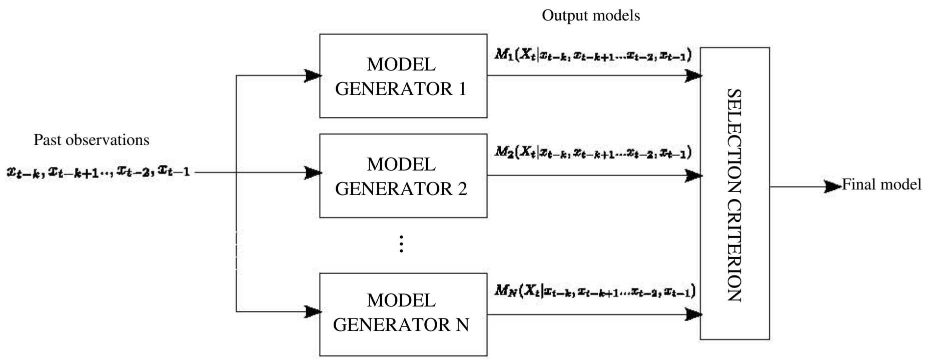

Output models
Past observations
$x_{t-k}, x_{t-k+1} .., x_{t-2}, x_{t-1}$
MODEL GENERATOR 1
MODEL GENERATOR 2
MODEL GENERATOR N
$M_1(X_t | x_{t-k}, x_{t-k+1} ... x_{t-2}, x_{t-1})$
$M_2(X_t | x_{t-k}, x_{t-k+1} ... x_{t-2}, x_{t-1})$
$M_N(X_t | x_{t-k}, x_{t-k+1} ... x_{t-2}, x_{t-1})$
SELECTION CRITERION
Final model

Figure 5.13: Generation and selection of multiple models.

Several model selection criteria can be employed in our multimodel approach. We particularly propose two methods based on the analysis of fitting errors (minFE) and prediction errors (minPE). The minFE criterion selects the model that obtained the minimum fitting error during its generation. On the other hand, minPE selects the model that obtained the smallest prediction error in the last predictions.

5.4 EXPERIMENTAL EVALUATION

For a deeper comprehension of the benefits and limitations of the multiple model approach, the following subSections present an evaluation of modeling overhead, prediction horizon sensitivity, accuracy depending on the number of observations, accuracy under unexpected events, and accuracy combining several models. In order to evaluate the error we employ the metrics shown in Table 5.3. The mean average error (MAE), the root mean square error (RMSE), and the mean absolute percentage error (MAPE). The MAE calculates the average absolute magnitude of the errors, while the RMSE measures the quadratic average magnitude of the error. The RMSE is particularly affected by outliers, while MAE weights all the residuals equally [56]. Situations where RMSE>>MAE are highly undesirable. MAE and RMSE are both scale-dependent measures, for relative comparisons we employ the MAPE as it measures the accuracy in percentage.

Error	Definition
MAE	$\frac{1}{n}\sum_{i=1}^{n}\lvert X_i - \tilde{X}_i \rvert$
RMSE	$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(X_i - \tilde{X}_i)^2}$
MAPE	$\frac{100}{n}\sum_{i=1}^{n}\left\lvert \frac{X_i - \tilde{X}_i}{X_i} \right\rvert$

Table 5.3: Definition of employed error metrics.

Package	Version	Provided methods
Forecast [99]	3.24	ARIMA and Holt-Winter models
FitAR [63]	1.92	AR_z models
R [71]	2.15	General statistical functions

Table 5.4: R packages versions used in the experiments run in an Intel Xeon with four cores at 2 GHz and 4 GBytes of memory.

5.4.1 *Modeling overhead*

In order to have a better comprehension of the modeling overhead, we automatically generate the predictions for our datasets using ARIMA, Holt-Winters and AR_z models for different k observations. Figure 5.14 shows the time elapsed to generate the models (identification and parameter estimation stages) for the Last.FM dataset. We use a granularity of days, computing for each k value one different model each time we require a new prediction. The showed values are the mean of 5 executions. AR_z models are discarded from this comparison as these models can only be computed when the model order p is larger than the number of observations, requirement not fulfilled in this dataset. We highlight two results. First, it exists a significant difference in the time elapsed between ARIMA and Holt-Winters. This is due to the exhaustive search of temporal correlations done by ARIMA. This fact permits Holt-Winters to spend less time in the identification stage. Second, a larger number of observations incur in a larger overhead.

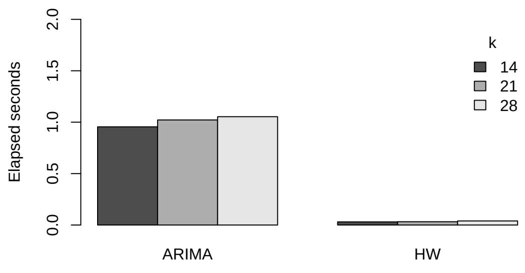
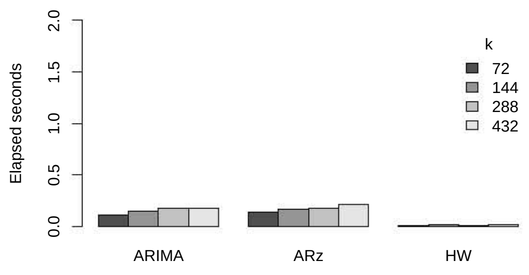
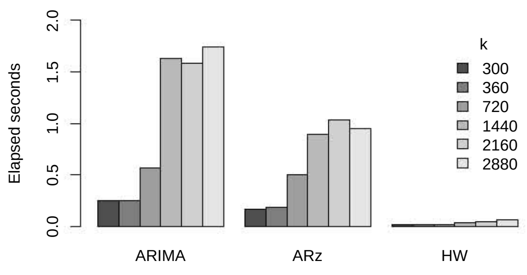


Figure 5.14: Model generation overhead for the Last.FM dataset.

We repeat the same experiment for the 1998 World Cup dataset using intervals of 5 and 1 minutes as shown in Figures 5.15a and 5.15b respectively. We know that the workload has a daily pattern, thus we employ a set of observations that captures this daily pattern. In particular for the 5 minutes intervals we use 6, 12, 24, 36 and 48 hours that correspond



(a) Using intervals of 5 minutes.



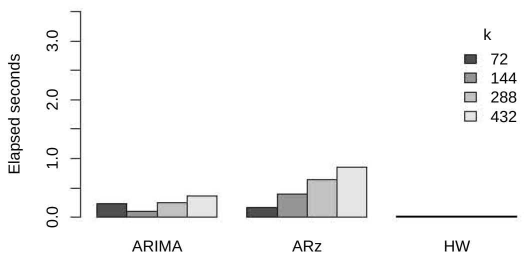
(b) Using intervals of 1 minute.

Figure 5.15: Model generation overhead for the 1998 World Cup dataset.

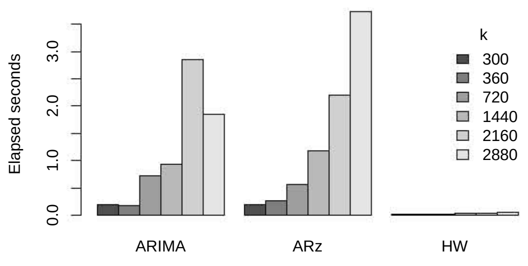
with k values 72, 144, 288, 432 and 576 observations respectively. A similar approach is followed in the case of one minute observations.

We observe similar elapsed times for $k \leqslant 360$ using 1 minute granularity and $k \leqslant 432$ in 5 minutes granularity. However, we note a dramatic increment when $k > 720$ for a granularity of 1 minute when using ARIMA and AR_z. For $k > 720$ (half day) ARIMA requires more than 1.5 seconds, whereas AR_z only needs around 1 second. The time reduction in AR_z for $k > 720$ is due to the utilization of the McLeod and Zhang functions that improve the convergence time. This results in 33% lower overhead when using AR_z for $k > 720$.

In the case of the Wikipedia dataset using intervals of five minutes (Figure 5.16a) the elapsed time is always smaller than one second. When we employ one minute intervals (Figure 5.15b), we find that even though we use the same k values as in the World Cup, the required time is larger. Using $k < 1440$ the elapsed time is lower than 1.5 second for all the methods. In particular for ARIMA the elapsed time is even smaller than in the World Cup dataset. We observe a significant increment of the elapsed time for $k > 1440$, particularly for AR_z.



(a) Using intervals of 5 minutes.



(b) Using intervals of 1 minute.

Figure 5.16: Model definition overhead in the Wikipedia dataset.

These experiments indicate the different overheads of automatic definition of ARIMA, AR_z and Holt-Winters models. We have observed how the overhead depends on the kind of models, the granularity of the observations and the selected k value. However, we can make general observations. Holt-Winters has the fastest model generation independently of the time series and k value. For $k > 1440$ the elapsed time increases dramatically particularly for AR_z. However, using $k \leqslant 1440$ requires around one second for both ARIMA and AR_z.

The modeling overhead showed in our experiments indicate that the utilization of these automatic model generation methods may be feasible under certain restrictions. In particular, the most important restriction is the minimum elapsed time between consecutive predictions. The implementations we have used could be improved by employing standard programming techniques such as multithreading or reducing the utilization of R functions that are slower than native methods. Nevertheless, in the worst case 3 seconds are enough to generate a prediction model. This overhead makes feasible the utilization of these methods in scenarios where the prediction have to be done in intervals larger than 3 seconds. Even with this restriction, these prediction models could have been used in solutions such as [17, 27, 60, 79] where the predictions use a granularity of minutes.

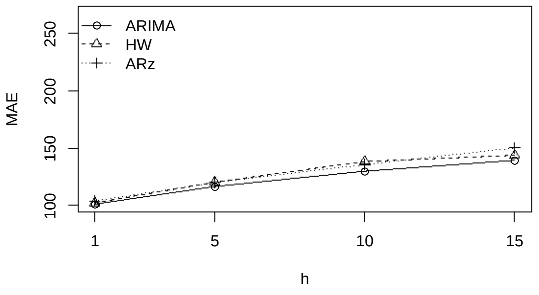
5.4.2 *Accuracy analysis varying the prediction horizon*

Intuitively, the larger the prediction horizon h is, the larger is the prediction uncertainty. We evaluate the prediction accuracy for various prediction horizons. Figures 5.17 and 5.18 show the error analysis for the Wikipedia and World Cup datasets for a fixed $k = 1440$ when varying h. In the Wikipedia dataset we observe that RMSE$\approx$MAE independently of h showing a stable error distribution. As expected, the MAPE increases with h for all the methods. The values obtained for AR_z and Holt-Winters are almost the same. ARIMA provides the most accurate predictions with an error lower than 2%. This accuracy and high stability is a result of the steady periodicity of the signal and the lack of unexpected variations.

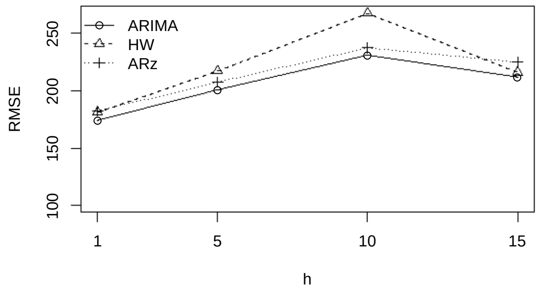
For the World Cup dataset (Figure 5.18) MAE and RMSE are significantly different. For $h = 1$ the RMSE almost doubles the obtained MAE. This difference gets even larger as h increases. The methods get similar errors with the exception of AR_z that slightly improves the RMSE for $h \geqslant 10$. We consider the obtained MAPE under 6% to be acceptable for all the prediction horizons. The World Cup dataset presents large variations between the first two days and the remaining dataset. This change in the workload makes challenging to identify suitable models.

5.4.3 *Accuracy analysis varying the number of observations*

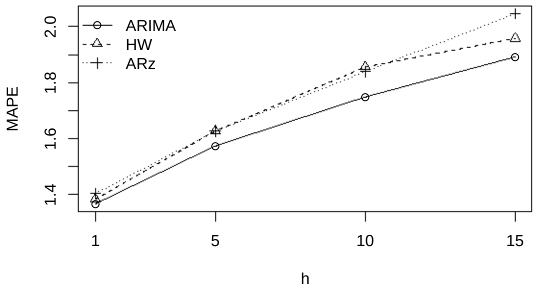
Intuitively, using large k values must bring the model generators a better vision of the workload than using small values. Generally, the number of observations to employ in the model generation should capture at least one period. In our particular case capturing one or two days of observations should be enough as the signal is periodic with a daily pattern. However, it exists a trade-off among the number of employed observations, the model accuracy and the definition overhead. For example, in the case of very long periods using all the observations in the model definition may result in a large overhead. In order to evaluate how the selection of k may affect the generated models, we calculate



(a) MAE



(b) RMSE



(c) MAPE

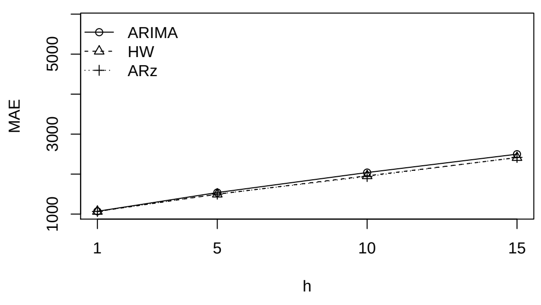
Figure 5.17: Wikipedia dataset error analysis varying h for $k = 1440$.

MAE, RMSE and MAPE measures for the Wikipedia and World Cup datasets. In this case we fix the prediction horizon to $h = 5$ and vary the k value.

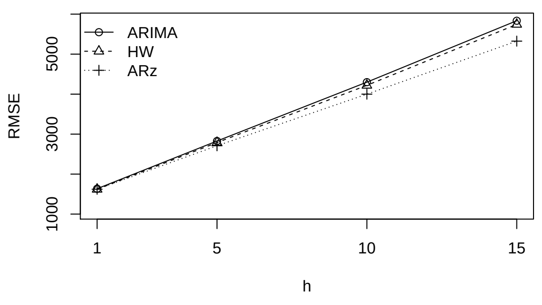
Figures 5.19 and 5.20 show the obtained results for the Wikipedia and World Cup datasets respectively. In the Wikipedia case we do not find any relevant variation in terms of MAE or RMSE. However, the MAPE is reduced using larger k values. In particular, we can get the best results using ARIMA with $k = 1440$ and $k = 2880$, which correspond to 1 and 2 days of observations respectively.

In the World Cup dataset, as in the previous case, we observe that k does not affect the MAE and RMSE values. Nevertheless, the MAPE is particularly affected. In this dataset a larger k does not decrease the MAPE in the case of ARIMA and AR_z as it occurs for the Wikipedia dataset. We notice that the MAPE obtained by ARIMA increases with larger k values. We guess this might occur due to the difficulties of ARIMA to adapt to workload variations.

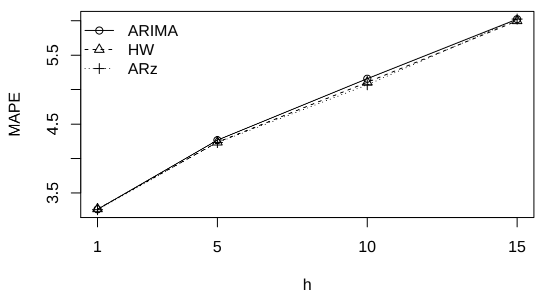
These experiments indicate that the selection of k does not particularly affect the stability of the predictions, but does affect the MAPE. It is important to select k values that correspond with the period of the signal. However, in the World Cup case we observe how we can drastically reduce the obtained error using a number of observations that does not correspond to any period. This situation may occur but it is difficult to identify. For this reason, the utilization of values of k that cover at least one period seems to be a reasonable approach.



(a) MAE



(b) RMSE



(c) MAPE

Figure 5.18: 1998 World Cup dataset error analysis varying h for $k = 1440$.

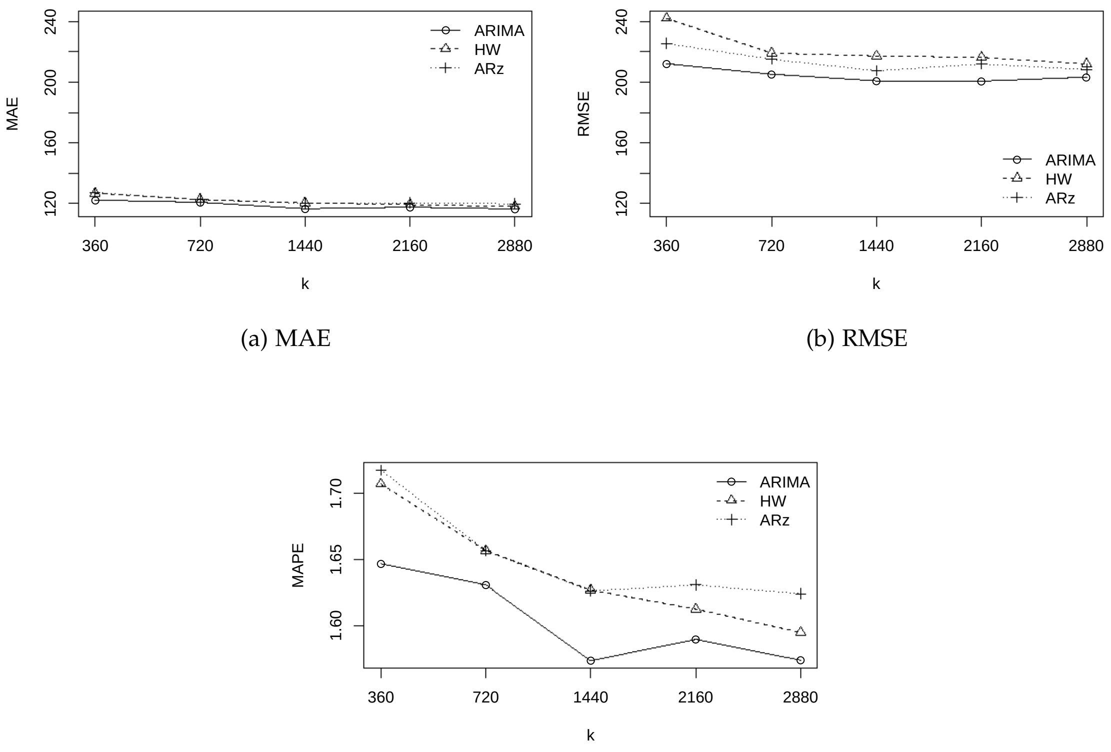
5.4.4 *Multimodel prediction approach*

In order to analyze the accuracy of *minPE* and *minFE* as a model selection criteria, we have computed the predictions for the Wikipedia and World Cup datasets for $k = 1440$ and $h = 5$. The results are shown in Tables 5.5 and 5.6 for the Wikipedia and World Cup respectively. For comparison purposes we have computed the minimum error that can be obtained by combining the predictions of the three methods that will generate the lower error. This approach is not realistic as we select the combination of predictions knowing the values to be predicted. However, it defines an accuracy baseline.

	MAPE	MAE	RMSE
Minimum	1.28	94	185
ARIMA	1.57	116	200
HW	1.62	120	217
AR_z	1.62	120	207
minPE	1.62	119	205
minFE	1.57	116	203

Table 5.5: Wikipedia dataset error comparison using $k = 1440$ and $h = 5$.

In the results for the Wikipedia dataset, *ARIMA* obtains the best results in terms of MAPE, MAE and RMSE. The *minFE* and *ARIMA* get similar results. This indicates that *minFE* has been selecting *ARIMA* as the candidate model most of the time which actually seems to be the most appropriate solution. However, *minPE* gets similar values to Holt-



(a) MAE (b) RMSE

(c) MAPE

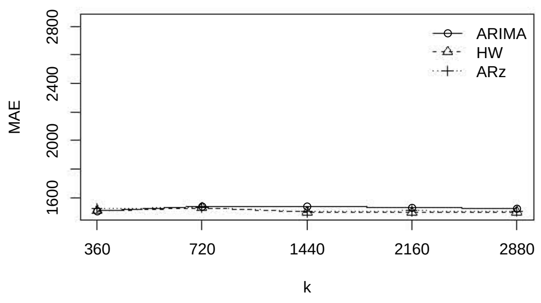
Figure 5.19: Wikipedia dataset error analysis varying k with $h = 5$.

Winters and AR_z, although it slightly decreases the RMSE. In the World Cup dataset AR_z gets the best results closely followed by Holt-Winters. In this case, *minFE* approximates the results obtained by Holt-Winters reducing the RMSE.

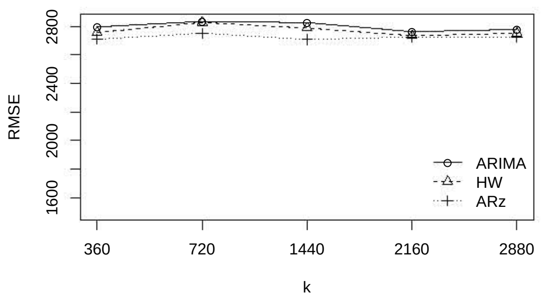
	MAPE	MAE	RMSE
Minimum	3.47	1228	2419
ARIMA	4.26	1538	2827
HW	4.23	1496	2788
AR_z	4.22	1499	2709
minPE	4.27	1510	2813
minFE	4.23	1510	2717

Table 5.6: World Cup dataset error comparison using $k = 1440$ and $h = 5$.

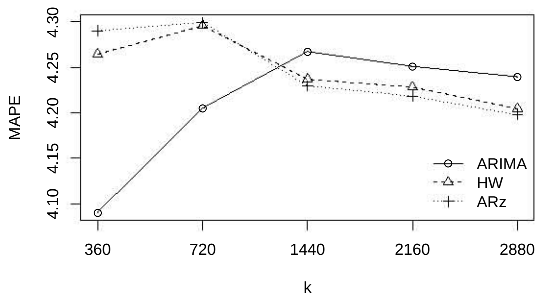
The previous results indicate that *minFE* is the most suitable solution in regular workload scenarios. However, we find that *minPE* works better during unexpected workload spikes. Figure 5.21 shows the MAPE obtained for the unexpected Wikipedia workload spikes using $k = 1440$ and $h = 5$



(a) MAE



(b) RMSE



(c) MAPE

Figure 5.20: 1998 World Cup dataset error analysis varying k with $h = 5$.

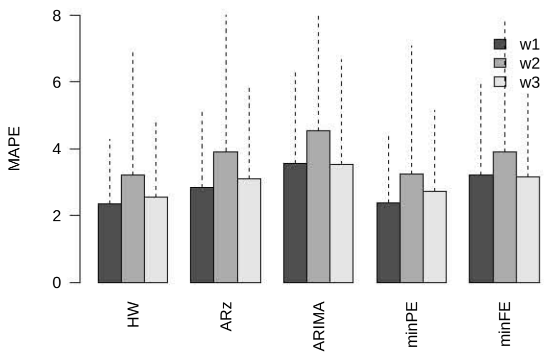


Figure 5.21: MAPE obtained for the Wikipedia with unexpected events dataset.

These results indicate that *minFE* is a suitable model selection criterion that can help to identify the most proper prediction method to use in different scenarios. This solution is appropriate to scenarios where we cannot priorly identify the most appropriate model.

5.5 SUMMARY

In this Chapter, we have covered the second objective of this thesis by investigating the data access patterns of Internet applications and analyzing the limits of workload predictability. First, we have studied the problem of workload forecasting. Second, we have described three workloads extracted from real systems. Finally, we provide a detailed evaluation of aspects such as model definition overhead, and prediction accuracy using different configurations.

We demonstrate that the utilization of fixed prediction models is not suitable for web-based applications. Moreover, we propose the utilization of automatic methods for the definition of prediction models. This permits to define models with a higher frequency while reducing the human intervention. Additionally, the automatic generation makes possible to combine several models over time selecting the most accurate one. In particular, we propose two methods called *minPE* and *minFE* that automatically select what model to use based on the past error and the fitting error respectively.

We present a detailed evaluation of four relevant aspects of prediction models: modeling overhead, accuracy when varying the prediction horizon, accuracy when varying the number of observations and accuracy under unexpected events. The overhead of automatic modeling depends on the number of employed observations and the facility to find periodic components among the given observations. We claim that our approach can be employed in scenarios with a prediction granularity of minutes. Our evaluation indicates that models accuracy significantly varies depending on the scenario and particularly if unexpected variations occur. In these cases, we demonstrate that the combination of multiple models through models selection criteria *minPE* and *minFE* can improve the accuracy.

6 ELASTIC SERVER INFRASTRUCTURE

The second component of the hierarchical architecture for content distribution proposed in Chapter 3 is an elastic server infrastructure based on a cloud solution (Figure 6.1). The cloud enables elastic horizontal scalability of server infrastructures, which allows to dynamically allocate resources and to pay only for their utilization. In Chapter 4, we described a distributed collaborative caching infrastructure that can be employed to mitigate the pressure in this elastic server infrastructure. By reducing the total workload served by the infrastructure we can reduce the amount of employed resources, and therefore save costs. However, efficiently exploiting the dynamic resource allocation mechanisms offered by clouds strongly depends on understanding and controlling the dynamics of workloads and on reducing the data traffic inside the data center. Increasing scale and demand variations pose huge challenges on developing, deploying, and evaluating control mechanisms and policies for efficient resource allocation.

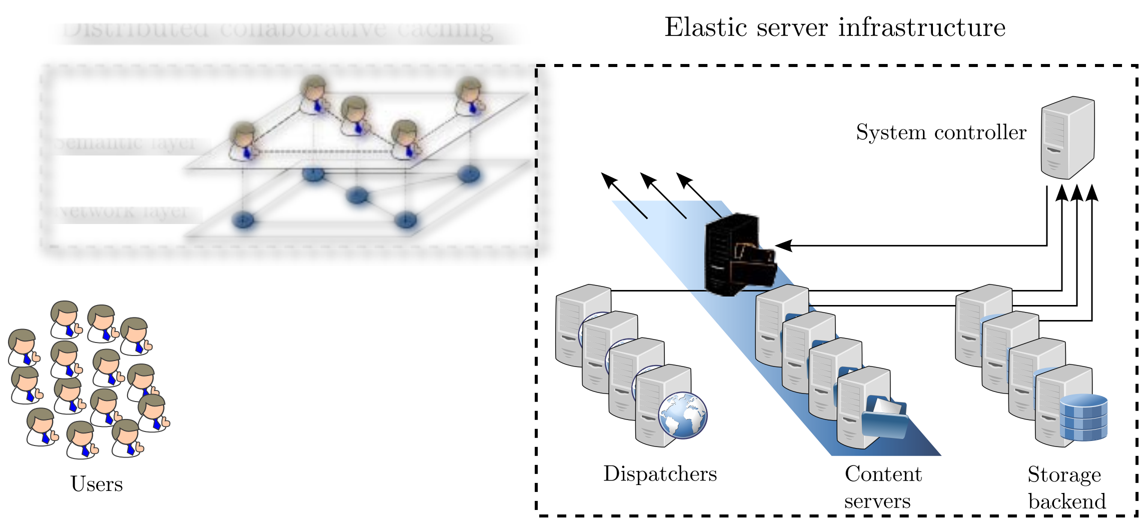


Figure 6.1: Detail of the elastic server infrastructure in the proposed architecture for content distribution.

Controlling data layout is critical for making feasible the management of large amounts of data, as the reorganization of large data sets is severely limited by the I/O infrastructure capabilities and must not negatively impact quality of service requirements [95]. However, controlling data layout has become an increasingly difficult task as the workloads of Internet applications show high variability due to factors such as periodic variations (seasonality), trends, expected and unexpected events. Traditionally, these variations have been addressed by overprovisioning the infrastructure, but this approach has been demonstrated to be costly and economically risky, as the peek volume is short-lived and the server utilization in normal traffic periods is between 10% and 50% [11]. In Chapter 5 we study and analyze the limits of automatic prediction methods confirming their suitability for predicting workloads. The utilization of proactive methods based on workload prediction models can improve the utilization of resources by anticipating the system size before needed.

This Chapter, covers the third objective of this thesis through the study of methods for prediction and control theory to enhance data distribution on elastic server infrastructures. We propose different methods to be employed in the system controller presented in Section 3.4. First, we introduce our solution. Second, we present the notation employed. Third, we describe the components of our proposed system controller: the adaptive prediction modeling, the adaptive data distribution and the system sizing. Finally, we experimentally evaluate our proposal using the datasets describe in Section 5.2.

6.1 OVERVIEW

The elastic server infrastructure is a three-layered server architecture as the one described in Chapter 2.1. It is composed of three layers: storage backend, content servers and dispatchers. The available contents are stored in the storage backend which acts as a long-term repository. Content servers answer the incoming users requests and acts as a short term storage layer or cache. These servers reduce the number of accesses to the storage backend. Finally, the dispatchers employ logical structures that permit to redirect user requests to the content servers. These structures permit to balance the traffic among servers avoiding bottlenecks and might employ additional intelligence to improve aspects such as content locality. All these elements are configured by the system controller.

We assume our architecture to employ existing methods provided by a cloud platform. These methods have to permit the addition and removal of resources from the elastic infrastructure. Our system controller uses these methods in order to set the system size. Internally, the system controller estimates the future state of the system and adds or removes resources accordingly. Additionally, it organizes the dispatchers to redirect the incoming users requests to the available content servers.

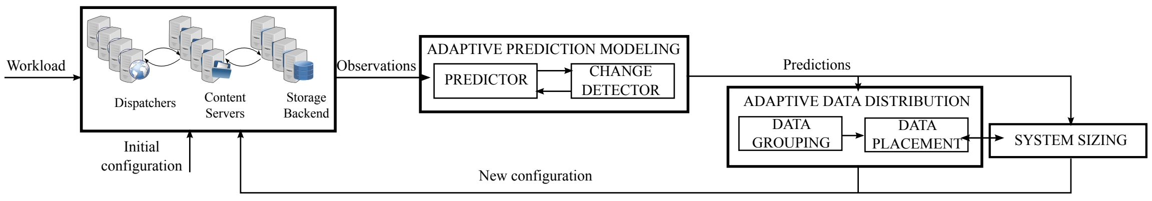


Figure 6.2: Components of the system controller

The system controller is composed of the elements showed in Figure 6.2:

- The *adaptive prediction modeling* component has two main tasks: a) to define, monitor, modify, and adapt time series workload models; b) to inform the adaptive data distribution component about model failures that could cause reactive resource provisioning actions. This component employs different workload observations to feed two interacting modules: a predictor module and a change detector module. The *predictor module* manages an extensible library of generic time-series models, which are employed for generating forecasts at global level and group level. The *change detector module* constantly monitors the prediction accuracy and takes decisions on model change based on user-defined criteria. The adaptive prediction component provides a series of statistics that can be used for estimating model accuracy or for defining criteria for model selection or model change. The predictions generated by this component are forwarded to the adaptive data distribution component.

- The *adaptive data distribution* component allows to employ elastic data distribution strategies for content servers based on two modules: data grouping and data placement. The *data grouping module* dynamically clusters storage objects into logical groups, which are the unit of placement and replication on the content servers. The *data placement module* allows to implement adaptive data placement strategies based on the information provided by the adaptive prediction component.
- The *system sizing* module dynamically scales the system by turning servers on and off based on the information provided by the adaptive prediction module.

Our system controller offers a modular design that permits to explore a vast space of possible solutions. We focus on exploring two issues: system sizing and data placement. We study the utilization of autoregressive prediction models to identify the amount of resources to use in the near future. These modifications in the system size implies modifications in the data placement. We investigate how to increase content locality independently of system size variations.

6.2 Concepts and notation

The system controller is composed of a set of elements whose interfaces receive input parameters and return a set of results based on a given configuration. For simplifying the reading, we present the notation employed in this Chapter using three tables. Observed and predicted variables are defined in Table 6.1. Table 6.2 shows the model and evaluation criteria used. Table 6.3 contains the notation for the observed and predicted values that are generated by the components of the system controller.

Variable description	**Observed value**	**Predicted value**
Global workload time at time t, measured in number of requests	$l(t)$	$\tilde{l}(t)$
Workload of server s at time t, measured in number of requests	$l_s(t)$	$\tilde{l}_s(t)$
Workload of group g_j, measured in number of requests	$l_{g_j}(t)$	$\tilde{l}_{g_j}(t)$
Number of servers at time t	$n(t)$	$\tilde{n}(t)$
Probability of dispatching a request for an object in group g_j to server s		$p_{g_j}(s)$

Table 6.1: Concepts and notation for observed and predicted variables.

Variable	Description
$\epsilon(t)$	Absolute prediction error at time t
$\eta(t)$	Relative prediction error at time t
RMSE	Root Mean Square Error
MAE	Mean Absolute Error
MAPE	Mean Absolute Prediction Error
minFE	Minimum Fitting Error selection criterion
minPE	Minimum Prediction Error selection criterion
GlobalCD	Global Change Detection is a method for evaluating the model of the global workload
LocalCD	Local Change Detection is a method for evaluating the models of the local workloads

Table 6.2: Concepts and notation for evaluation criteria.

Parameter	Description
$o_i(t)$	Number of requests for storage object o_i at time t
Prediction models	Time series models to be used
h	Prediction horizon
Model selection criteria	Criteria for selecting a model from several candidates
Change detection	Method of deciding model redefinition
η_G	Global overprovisioning threshold
ϵ_L	Local overprovisioning threshold
G	Optional set of categories $g_j, j = 1, ..., \|G\|$ classifying objects stored in the system
$\{(o_i, g_j)\}$	Optional assignment of the storage object o_i into the group/category g_j
Data grouping	Method used for joining the data items into placement groups
Data placement	Method used for distributing requests to servers
c_s	Maximum load of content server s measured in concurrent number of requests
d_s	Data cache capacity of server s in megabytes
thr_u	Server utilization target in %

Table 6.3: Concepts and notation for input parameters.

6.3 ADAPTIVE PREDICTION

The main goal of the prediction modeling component is to generate a set of predictions for a given set of historic workload observations. In Section 5.3 we demonstrated how the utilization of statically defined models is not appropriate in dynamic environments such as user-oriented Internet applications. In order to solve this problem, we propose the utilization of adaptive prediction techniques using model adaptability and model change detection.

We propose to continuously redefine the prediction model to be used for the next prediction horizon. This solves the mentioned problem of static models and permits to adequate the predictions to the dynamics of workload. This continuous redefinition of models may result in a delay due to the elapsed time to define the models. However, our analysis of the elapsed time in model generations (see Section 5.4.1) indicates that is feasible to continuously redefine models when using a time granularity of minutes. The change detection must identify when a model is no longer suitable. This may occur at least in two scenarios: when the generated model is not accurate enough or when the predictions may become problematic for the system (i.e., possible underprovisioning, low data locality, etc).

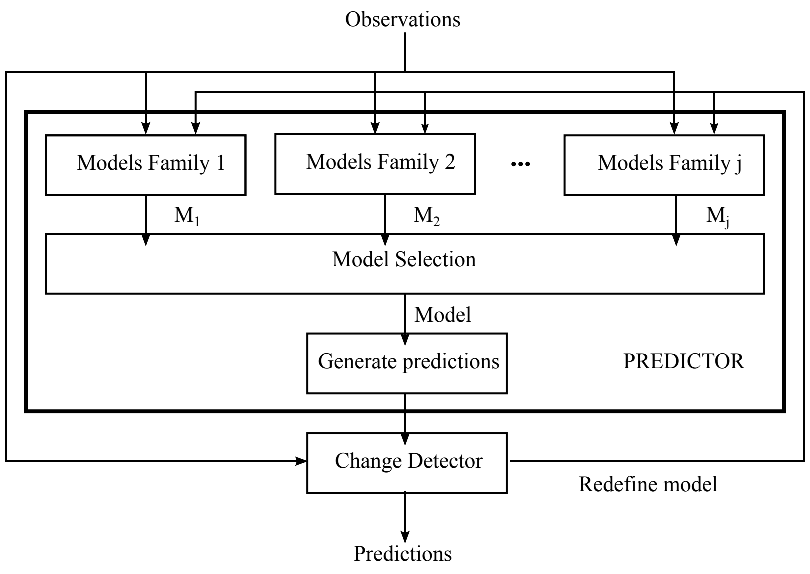


Figure 6.3: Internal structure of the adaptive prediction component.

Model adaptability and change detection are implemented in two separated interacting modules as shown in Figure 6.3: a predictor module and a change detector module. The predictor module automatically defines prediction models, selects among several candidates and generates predictions from these models. The change detector module estimates the model accuracy and relaunches the redefinition of models.

We build hierarchical prediction models at two levels corresponding to models for the global and local workloads. A global model predicts the total workload of the system, which helps to dynamically determine the system size. At a second level, a local model predicts the expected number of requests for a group of items. This second level of prediction permits to identify workload variations at a lower granularity and to control the dynamic assignment of groups to content servers. The decomposition of the global workload into local workloads is managed by the data grouping module, which is part of the adaptive data distribution component and is discussed later.

6.3.1 *Prediction module*

The prediction module manages models for predicting the system workload based on observing the system status. In particular, we employ the content servers history access. The prediction module receives as input a series of past observations $(l_{t-k}, ..., l_{t-2}, l_{t-1})$ and generates a time window of h predictions $(\tilde{l}_t, ..., \tilde{l}_{t+h-1})$ and the fitting error.

Following the multimodel approach described in Section 5.3.1, the prediction module is able to simultaneously generate various prediction models. We differentiate two stages: model generation and model selection.

MODEL GENERATION. The model generation follows the steps described in Section 5.1: model formulation and model estimation (fitting) [82]. The model formulation identifies the form of the internal equations used by the models, while the fitting stage estimates the model parameters by methods such as recursive least squares or maximum likelihood.

We employ the three family models described in Chapter 5: ARIMA [82], Holt-Winters [44], and AR_z [63]. For each of them the prediction module relies on automatic model formulation and estimation tools available for the R statistical software environment. Nevertheless, the available prediction modules can be extended, as the prediction module interface is generic: any uni-dimensional time series family model can be incorporated into the framework and evaluated together with other system components.

MODEL SELECTION. The predictor module allows to employ several concurrent prediction models and to select the best one based on various criteria, as shown in Figure 6.3. First, a candidate model is chosen based on a family model specific selector. For instance, for autoregressive models we use the AIC and BIC [82] criteria to select the model with the best combination of low fitting error and small number of parameters. Further, a model can be chosen from the selected candidate models based on two criteria. We employ the *minFE* and *minPE* criteria described in Section 5.3.1.

It has been noticed that during expected workloads the amount of requests for some items may significantly vary [17]. For this reason, we employ two kind of models: global and locals. The global model generates predictions for the total number of requests to be expected at time t $(\tilde{l}(t))$. Local models generate predictions for the group g_j at time t $(\tilde{l}_{g_j}(t))$. The number of local models to generate is the number of groups in the system. This depends on the data grouping module, which performs the dynamic group management.

6.3.2 *Change detector module*

As the workload patterns change, the prediction models can become inaccurate and have to be either refitted or redefined. The change detection module monitors the model accuracy and uses change detection criteria for discovering when a model prediction becomes inaccurate and redefines the model when necessary. Detecting when to redefine a model is a difficult task as it implies to distinguish between permanent changes in the workload and temporal fluctuations [56]. Our proposed adaptive prediction allows to define custom criteria for change detection. Below, we illustrate this process by presenting two change detection criteria we have already implemented for global and local models.

A model redefinition method periodically checks the accuracy of the predictions and decides based on change detection criteria when the models have to be redefined. For each model, the change detection module stores the past absolute error distribution $(\epsilon(t-k), ..., \epsilon(t-1))$ and the past absolute relative error distribution $(\eta(t-k), ..., \eta(t-1))$ from the previous predictions of a given model, where t is the current unit of time and k is the size of the interval for which the errors are recorded. Based on these distributions we propose here two model redefinition criteria. The first criterion is based on the detection of outliers from the relative error distribution. The second criterion detects errors beyond an absolute threshold. These thresholds are denoted η_G for the global model and ϵ_L for local models.

The first criterion GlobalCD (Global Change Detector) is described in Algorithm 5, where E and σ are the mean and the standard deviation of the error distributions in the time interval $[t-k, t-1]$. First, the algorithm checks if the last absolute relative error is an outlier of the error distribution. We use a standard outlier detection criteria, based on which an observation is an outlier if the absolute prediction error lies more than two standard deviations apart from the mean of the past prediction errors. If no outlier is detected, it checks if the last absolute error was larger than η_G, the ratio of the total planned free capacity in the system. For example, for $\eta_G = 0.5$, the global model is redefined if the last absolute error is larger than 50% of the overprovisioned capacity. Intuitively, η_G controls the stability of the system: decreasing its value increases the probability of model redefinition.

Algorithm 5 GlobalCD checks and redefines the global workload model

```
1: //Check if $\tilde{l}(t)$ is an outlier.
2: **if** $\eta(t) < E[\eta(t-k), ..., \eta(t-1)] + 2\sigma[\eta(t-k), ..., \eta(t-1)]$ **then**
3:     // Check the absolute error
4:     **if** $\epsilon(t) < \eta_G \cdot \tilde{n}(t) \cdot (1 - thr_u) \cdot c_s$ **then**
5:         return
6:     **end if**
7: **end if**
8: //Redefine the model if any criteria fails.
9: redefine(global workload model)
```

Similarly, Algorithm 6 describes the LocalCD (Local Change Detector) criterion. For each local model we apply the same outlier detection employed for the global model. The second criterion compares if the mean of the relative errors distribution is larger than ϵ_L. The threshold ϵ_L is the number of requests directed to a group an absolute error must not exceed. Intuitively, this threshold is necessary, as the relative error for large groups may correspond to a significant absolute error (in number of requests). Therefore, model change detection for large groups needs to become more sensitive to the absolute error in order to avoid system overload.

Algorithm 6 LocalCD checks and redefines workload models of groups

```
1:  for all j do
2:      //Check if $\tilde{l}_{g_j}(t)$ is an outlier.
3:      if $\eta(t) < E[\eta(t-k),...,\eta(t-1)] + 2\sigma[\eta(t-k),...,\eta(t-1)]$ then
4:          //Check the absolute error.
5:          if $E[\epsilon(t-k),...,\epsilon(t-1)] < \epsilon_L$ then
6:              return
7:          end if
8:      end if
9:      //Redefine the model if any criteria fails.
10:     redefine(workload model of group $g_j$)
11: end for
```

Both criteria GlobalCD and LocalCD are designed to asynchronously relaunch the model definition of the affected models. This action may not necessarily be straightforwardly translated into a new system configuration. We currently assume that the redefinitions of the global model have to be immediately translated into a new system size.

6.4 SYSTEM SIZING MODULE

The system sizing module elastically scales the system by turning servers up and down. We employ a method based on the time series of the number of requests entering the system. First, at time t, the global workload model generates h predictions. In order to reduce oscillations in the system size evaluation, we take the maximum predicted workload $\tilde{l}_{max}(t, t+h-1) = max(\tilde{l}(t), ..., \tilde{l}(t+h-1))$ as the reference value for system sizing for the next time window h. If we assume that all content servers have the same maximum capacity of c requests per unit of time ($c = c_s$ for all s) and that the desired utilization ratio is thr_u, the number of servers for the next prediction window h is given by:

$$\tilde{n}(t) = ... = \tilde{n}(t+h-1) = \left\lceil \frac{\tilde{l}_{max}(t, t+h-1)}{thr_u \cdot c} \right\rceil \tag{6.1}$$

Figure 6.4 shows the different variables involved in the system sizing process. Using $\tilde{l}_{max}(t)$ we provision enough resources for the maximum predicted workload in the next prediction horizon. When thr_u is close to 100%, the system resizes to employ the minimal number of resources to serve the predicted demand. This may incur in the overload of servers due to mispredictions or unexpected events. This effect can be mitigated by reducing the value of thr_u. Setting the thr_u value is equivalent to the utilization of a security margin or overprovisioning [16]. The most appropriate thr_u value depends on the application domain. For instance, certain applications may require an optimization of the number of employed resources rather than having unattended requests.

6.5 ADAPTIVE DATA DISTRIBUTION

The adaptive data distribution component is in charge of dynamically adapting the data distribution based on discrete information provided from the adaptive prediction compo-

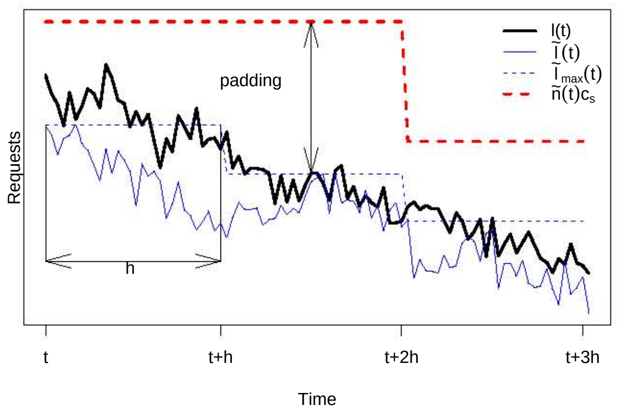


Figure 6.4: System sizing for a prediction horizon h.

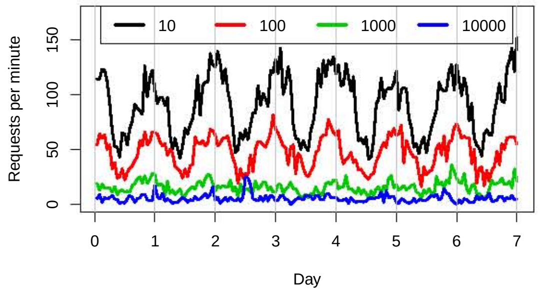
nent. This component has two main tasks: to identify how to group items and to define data placement policies. These tasks are carried out by the data grouping and data placement modules.

6.5.1 *Data grouping module*

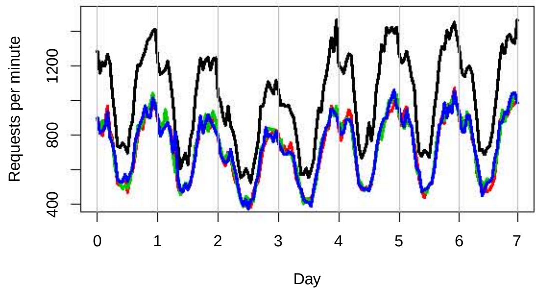
The data grouping module maps storage objects o_i (data items) to placement groups g_j. A placement group is a unit of data placement, i.e., it is not further divided into smaller logical units when stored on a server or on a disk (although it can be physically divided on several blocks when physically stored on a disk). We associate local prediction models with placement groups. In order to make this approach efficient, groups have to be large enough to reduce the modeling overhead and small enough to avoid that popularity variations of individual items remain undetected.

Grouping does not only bring a logical structure to the data provided by the system, it also permits to simplify the generation of prediction models by leveraging the periodicity of the groups. Figure 6.5 shows an example of how data grouping improves the capture of periodicity in the Wikipedia dataset. The workload before grouping corresponds to four items with four different popularity ranks. The workload time series has a clear periodic pattern for the two most popular items. However, the other items have workloads which periodicity is difficult to find. This would make extremely difficult the identification of appropriate prediction models for these items. This indicates that using one model per item apart from being unfeasible due to the number of models to generate, is difficult due to the workloads of unpopular items. In this example, we show how by grouping the items of the dataset into four equally-sized random groups the obtained workloads with clearly defined periodic patterns making easier the generation of prediction models. Each group may show a different pattern that should be identified by a different prediction model. Nevertheless, this dramatically reduces the number of models to manage while permitting to have an approximation of the future behavior for the data managed by the system.

The data grouping module currently supports two data grouping policies: random and affinity-based. The random grouping assigns storage objects to placement groups randomly by using a uniform distribution. Our affinity-based grouping is based on an affinity metric capturing the relationships between two categories. This metric is an adap-



(a) Workload for items in ranking 10th, 100th, 1000th and 10000th popularity.



(b) Workload after grouping the items in 5 equally sized groups by random selection.

Figure 6.5: Example of workload before and after random grouping in the Wikipedia dataset.

tation of the metric previously described in Section 4.2.1. The affinity a_{ij} represents the probability of a user accessing a storage object from categories K_i to access an object from category K_j. The affinity matrix A is a square matrix of size $|K| \times |K|$, where $|K|$ is the total number of categories in the system.

The a_{ij} value at time t is calculated using a version of the Jaccard coefficient. In particular, we calculate the probability of a user demanding objects from category K_i to demand objects from category K_j as:

$$a_{ij} = \frac{|U_i^t \cap U_j^t|}{|U_i^t \cup U_j^t|} \tag{6.2}$$

where U_i^t contains the set of users consuming content from K_i at time t.

Our affinity metric is calculated for the object category, not for the object itself. While it is possible for an object to have its own category, we are interested in grouping objects by category, whenever a category is probable to reflect potential spatial and temporal locality patterns. The affinity metrics are calculated based on a configurable time window. The time to recalculate the affinity metrics is intended to be large, as user tastes may not vary in the short term.

The Algorithm 7 describes how our affinity-based data grouping works. First, we select the most popular category in the system as seed for the algorithm. With this seed, we select the K_{min} categories that maximize the affinity. If the load calculated for the categories contained in the group does not reach the threshold thr_G, we keep on adding categories until we reach it. The process is repeated iteratively until all the categories are assigned to a group.

For selecting groups of categories maximizing the sum of their mutual affinities, we use a greedy algorithm that selects the pair of categories (i, j) with affinity a_{ij} and adds it to an empty result set. Subsequently, it iteratively selects from the remainder categories, the one that maximizes the affinity of the result set. This algorithm has a complexity of $O(g|K|^2)$, where g is the size of the group. The grouping algorithm is intended to be applied only for the most popular categories. This approach drastically reduces the number of categories considered for grouping. Intuitively, grouping the categories from the long tail does not bring substantial benefit to locality, as they are sporadically accessed.

Algorithm 7 Affinity-based data grouping

```
1: $i = 0$
2: $pending = K$
3: while $pending \neq \{\emptyset\}$ do
4:     $i \leftarrow i + 1$
5:     Select a subset $T$ with $K_{min}$ categories from $pending$ such that the sum of their
       mutual affinities is maximized
6:     Assign subset $T$ to group $g_i$
7:     Remove $T$ from $pending$
8:     Calculate $g_i$ workload $\tilde{l}_{g_i}$
9:     while $\tilde{l}_{g_i} < thr_G$ do
10:        Select a category $k_j$ that maximizes the mutual affinities with the categories
           assigned to $g_i$
11:        Assign $k_j$ to $g_i$
12:        Remove $k_j$ from $pending$
13:        Update $\tilde{l}_{g_i}$
14:    end while
15: end while
```

6.5.2 *Data placement module*

The data placement module is in charge of managing the dynamic distribution of the groups over content servers. This distribution is controlled through two data structures: the items catalog and the dispatching table. The items catalog is a dictionary provided by the data grouping module that identifies what group an item belongs to. The dispatching table maps placement groups onto content servers and is used for redirecting user requests to these content servers. The table consists of three columns: placement groups, servers, and dispatching probabilities. Each group is associated with a set of servers with a certain dispatching probability. Each server is associated a probability interval, which controls the amount of requests redirected to each server at placement group granularity. To determine where to redirect a request, the dispatcher generates a uniform random number between 0 and 1 and identifies the server in charge of the group based on the probability distribution interval which contains that number.

Figure 6.6 shows an illustrative use case where a user requests the object o_{100} to one available dispatcher (1). The dispatcher checks the item catalog and gets the group the item belongs to, in this case g_1 (2). Once the group is identified, assume that the dispatcher generates the random number 0.48. After checking the dispatching probabilities for g_1, it selects the server s_2 as 0.48 falls in the second dispatching probability interval (3). Finally, the request is processed by server s_2 (4).

In our design, storage objects are transferred from the storage backend to the content servers when these does not find the requested content. This means that an intelligent distribution of requests among servers can reduce the traffic in the storage backend. In this respect, the data placement module allows for custom implementations of the dispatching table construction. Nevertheless, the final construction has to guarantee a fair balance of the requests among the content servers while not exceeding the desired utilization threshold thr_u.

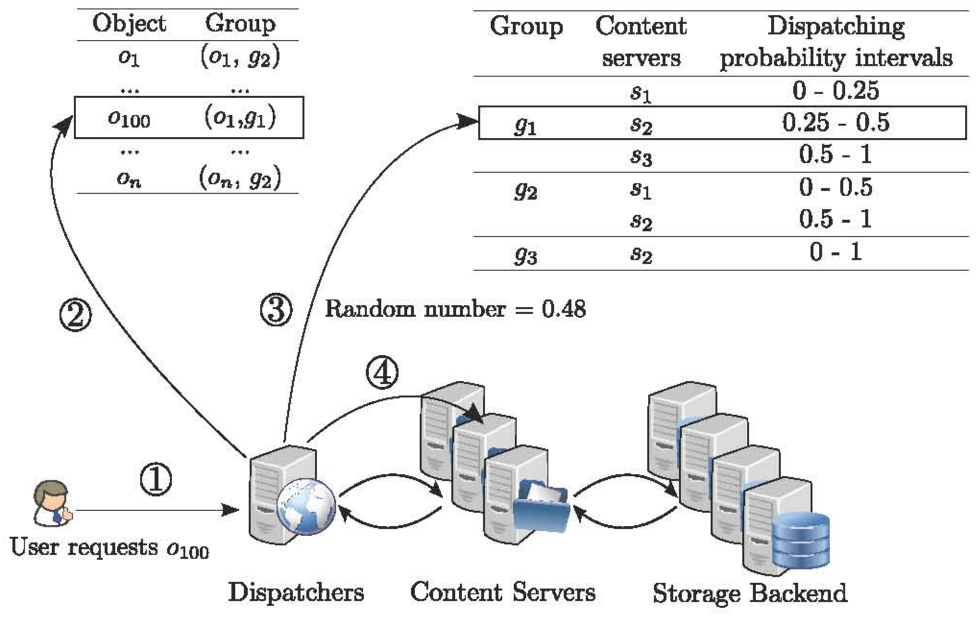


Figure 6.6: Data placement module use case example showing the utilization of the items catalog and the dispatching table.

We address the table construction as a bin-packing problem [93] where the group g_j with size $\tilde{l}_{g_j}(t)$ have to be assigned to content servers $s_1, s_2, ..., s_n$, which are bins with capacities $thr_u \cdot c_s$. As the bin-packing problem is NP–Complete, we employ the first-fit greedy approximation. Algorithm 8 shows how to construct the dispatching table through this method assuming that the capacities of all servers are the same ($c_s = c$ for all s) and that the target utilization threshold is $thr_u \cdot c_s$. This approach iteratively assigns the group with the largest predicted workload to the server with the largest free capacity until all the groups have been assigned.

Algorithm 8 Dispatching table update.

```
1: for all $j$ do
2:     //Total group load prediction to assign
3:     $toAllocate = \tilde{l}_{g_j}$
4:     //While there is some group load to assign
5:     while $toAllocate > 0$ do
6:         //Select a server with available load
7:         $s = selectNextServer()$
8:         //Decide how much load to assign to this server
9:         $assignedLoad = assign(toAllocate, s)$
10:        //Update the dispatching table
11:        $updateDispatchingProbabilities(s, assignedLoad)$
12:        //Calculate the remaining load to assign
13:        $toAllocate = toAllocate - assignedLoad$
14:    end while
15: end for
```

The $selectNextServer$, $assign$, and $updateDispatchingProbabilities$ methods allow for implementing multiple policies in the dispatching table. The simplest approach is to uniformly distribute the groups among the servers (RRD policy) with the same probability for every group in every server. This solution does not take advantage of local model predictions. To take advantage of local models, we propose *OLARD* (Oblivious Locality-Aware Request Distribution). *OLARD* calculates the dispatching probability interval as the ratio of the local predictions $\tilde{l}_{g_j}(t)$ assigned to a server. *OLARD* defines a minimum number of groups that have to be stored in one server to avoid situations where a server only returns objects belonging to a single group. This solution adaptively modifies the amount of traffic redirected to the servers depending on the given predictions. The idea behind *OLARD* is to distribute the workload of groups becoming popular to additional servers while trying to maintain content locality even under system size variations.

OLARD is oblivious with respect to previous assignments of placement groups to content servers. This is highly undesirable as it does not exploit data locality on the servers. This fact adds more complexity to the problem as content locality has to be preserved among servers through the constructions of the dispatching table. To mitigate this problem we propose the *ABLARD* (Affinity-Based Locality-Aware Request Distribution) approach that takes into account the groups that were previously assigned to a server. In order to do this, the *ABLARD* policy uses a content summary table. This table contains the ratio of dispatched requests per group and server. When assigning a group to a server, the summary table is used to greedily select the server with the largest previous assignment share for that group. In this way, we increase the chance to allocate content in a server that previously stored it.

6.5.3 *Evaluation metrics*

This Section presents the metrics used in our experimental evaluation, which are summarized in Table 6.4. We discuss three categories of metrics: system sizing metrics, server utilization metrics, and content locality metrics.

SYSTEM SIZING METRICS. These metrics estimate the efficiency of server provisioning based on the optimal number of servers that could have served that workload. Their main goal is to evaluate how far the provided resources are from the optimum. For a given server capacity c_s, the optimal number of provisioned servers can be calculated by:

$$n_{opt}(t) = \left\lceil \frac{l(t)}{c_s} \right\rceil \tag{6.3}$$

where $l(t)$ is the total number of requests received by the system at time t. Using the optimal server provisioning value we define the relative provisioning error $\eta_p(t)$ as:

$$\eta_p(t) = \frac{\tilde{n}(t) - n_{opt}(t)}{n_{opt}(t)} \tag{6.4}$$

where $\tilde{n}(t)$ is the number of provisioned servers at time t. Based on the calculated values for the relative provisioning errors in a time interval between t_1 and t_2, we calculate the overprovisioning rate $r_o(t_1, t_2)$ as the mean of all positive relative

System sizing metrics	
$n(t)$	Number of servers provisioned at time t
$n_{opt}(t)$	Optimal number of servers at time t for a server load target thr_u
$\overline{n}(t_1, t_2)$	Average number of servers provisioned between t_1 and t_2
$\eta_p(t)$	Relative provisioning error at time t
$r_o(t_1, t_2)$	Overprovisioning rate of all servers between t_1 and t_2
$r_u(t_1, t_2)$	Underprovisioning rate of all servers between t_1 and t_2
Server utilization metrics	
$u_s(t)$	Utilization of server s at time t (in %)
$\overline{u}(t_1, t_2)$	Average server utilization between t_1 and t_2 (in %)
$\sigma_u(t_1, t_2)$	Standard deviation of server utilization between t_1 and t_2
Locality metrics	
$hit_s(t_1, t_2)$	Hit rate of server s between t_1 and t_2
$\overline{hit}(t_1, t_2)$	Average hit rate of all servers between t_1 and t_2
$\overline{hit}(t)$	Average hit rate of all servers at time t

Table 6.4: Evaluation metrics.

provisioning errors and the under-provisioning rate $r_u(t_1, t_2)$ as the negated mean of all negative relative provisioning errors (i.e. both r_o and r_u have positive values).

$$r_o(t_1, t_2) = \frac{1}{|T|} \sum_{t \in T} \eta_p(t) \text{ with } T = \{t_i, ..., t_j\}, \text{where } \eta_p(i) > 0 \quad \forall i \in T \tag{6.5}$$

$$r_u(t_1, t_2) = -\frac{1}{|T|} \sum_{t \in T} \eta_p(t) \text{ with } T = \{t_i, ..., t_j\}, \text{where } \eta_p(i) < 0 \quad \forall i \in T \tag{6.6}$$

As far as we employ a desired threshold utilization $thr_u > 0$ there will be overprovisioning ($r_o \neq 0$). Nevertheless, to obtain $r_u > 0$ is not desirable as it may indicate a decrement in the quality of service.

SERVER UTILIZATION METRICS. We calculate the utilization of a server s at time t, denoted $u_s(t)$, as the ratio between the number of served requests l_s and the server capacity c_s. Based on these values we calculate aggregation metrics such as the average server utilization between t_1 and t_2 denoted $\overline{u}(t_1, t_2)$. This metric allows for identifying how close the server utilization is to the desired threshold thr_u. The servers load balance for that interval is estimated as the standard deviation of server utilization and denoted by $\sigma_u(t_1, t_2)$. The lower σ_u, the better load balance obtained in the system.

LOCALITY METRICS. Each server maintains an object cache with a configurable size d_s. To measure the content locality we evaluate the hit rate in each server. This hit rate for a period between t_1 and t_2 is denoted as $hit_s(t_1, t_2)$ and measures the rate of requested objects found in cache memory for that period. We denote the average hit rate of all servers at time t as $\overline{hit}(t)$. For a period of time we note the average server utilization as $\overline{hit}(t_1, t_2)$.

6.6 EXPERIMENTAL EVALUATION

This Section demonstrates how our system controller can be used for browsing the parameter space of the data distribution in an elastic server infrastructure. Our evaluation employs the datasets described in Section 5.2 based on the traces from Last.FM, Wikipedia and the 1998 World Cup described in Section 5.2. Table 6.5 overviews the set of input parameters employed by the experiments discussed here.

For brevity reasons we use the following naming conventions. Adding the suffix "+A" indicates that a method uses model adaptability based on change detections. For instance HW+A denotes the utilization of Holt-Winters with the change adaptivity enabled. We evaluate a three-tier platform as the one showed in Figure 6.1 using an event-discrete simulator. For simplifying the analysis we make the following assumptions. The system has a variable number of dispatchers, which simply scale with the number of client requests. The client requests coming from the outside world are uniformly distributed over all dispatchers. The system has a variable number of content servers, whose elasticity is controlled by our system. Finally, the storage system has a fixed size and a uniform service time.

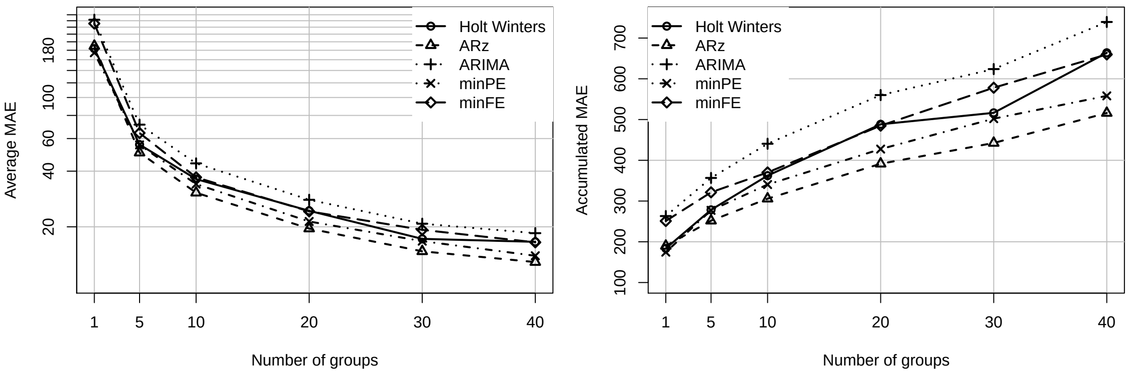
This Section is organized into five parts, which illustrate how our proposed system controller can be used for evaluating adaptive data placement strategies: evaluation of prediction accuracy for global and local models (Section 6.6.1), change detection stability (Section 6.6.2), system sizing (Section 6.6.3), server utilization and load balance (Section 6.6.4), data locality (Section 6.6.5), over- and underprovisioning (Section 6.6.6).

Parameter	**Value**
Prediction models	Holt Winters (HW), ARIMA, and AR_z
Model selection criteria	minPE and minFE
Change detection methods	GlobalCD and LocalCD
h	30
η_G	0.5
ϵ_L	200
System sizing	Predictive, Predictive+Adaptive
Data grouping	Random and Affinity-based
Data placement	*RRD*, *OBLARD*, and *ABLARD*
d_s	512, 1024 and 2048 MBytes
thr_u	75%, 80%, 85%, 90%, 95%
c_s	2000

Table 6.5: Evaluation parameters.

6.6.1 *Local models accuracy*

In this Section, we compare the prediction accuracy for local models based on the number of groups generated using the Wikipedia dataset. For comparison we consider both single family models (HW, ARIMA, and AR_z) and combination of family models (minPE and minFE).



(a) Average MAE of prediction per group. (b) Accumulated MAE prediction per group .

Figure 6.7: Average and accumulated prediction error for local models, when using 1, 5, 10, 20, and 40 placement groups in the Wikipedia dataset. Using 1 group means that only the global model is used.

We measure the prediction errors for equally-sized random groups. We split the available items into 5, 10, 20, 30 and 40 groups. Figure 6.7a shows the average of the mean absolute error (MAE) per group when predicting the workload at the granularity of a group($\tilde{l}_{g_j}(t)$). It can be noticed that the error decreases as the number of groups increments, which indicates that the utilization of local models increases the accuracy of local workload predictions. However, even though the average error per group decreases, the accumulated MAE of all groups increases, as it can be observed in Figure 6.7b. This plot demonstrates that using one model for predicting the global workload results in better accuracy than the aggregation of the predictions of the local models.

6.6.2 *Model change detection*

The utilization of the change detector module of the adaptive prediction modeling component addresses situations of unexpected workload variations that cannot be detected by the working models. This adaptivity has to reduce the obtained prediction error, and should also permit to dynamically modify the prediction window based on the obtained error.

Figure 6.8 shows the MAPE of global workload models for w_1, w_2, and w_3 from the Wikipedia dataset with unexpected workload variations. The average error is under 5% in all scenarios (under 8% if we add the standard deviation). We observe that the largest error and standard deviation is obtained for w_2 due to the sudden volume spike. The utilization of adaptability based on change detection significantly reduces both the average error and the standard deviation. In particular HW+A, AR_z+A and minPE+A provide the best results.

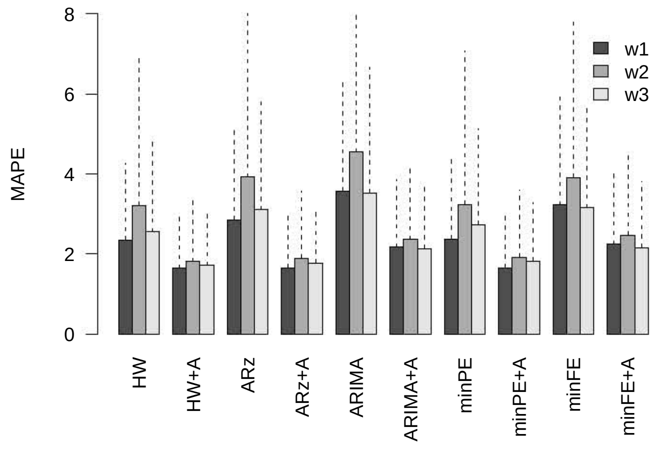


Figure 6.8: MAPE of the global workload models for w_1, w_2, and w_3 in the Wikipedia dataset.

Figure 6.9 shows the average time to force a redefinition of the prediction models for w_1, w_2, and w_3 workload spikes of the Wikipedia dataset using the prediction horizon $h = 30$. We observe that all the methods recalculate the model each 10 minutes on average. In particular, minPE+A takes on average 11 minutes for w_2. This corresponds to the fact that minPE+A obtains the best predictions in that scenario (Figure 6.8), with a lower error the redefinition of models is delayed. The fact that all the methods redefine their models every 10 minutes reveals that $h = 30$ is too large for this scenario under the given configuration.

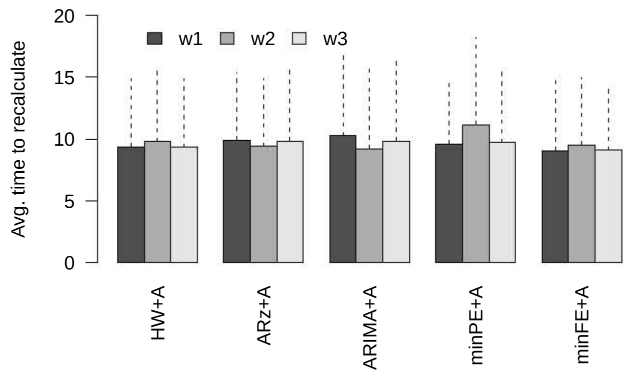
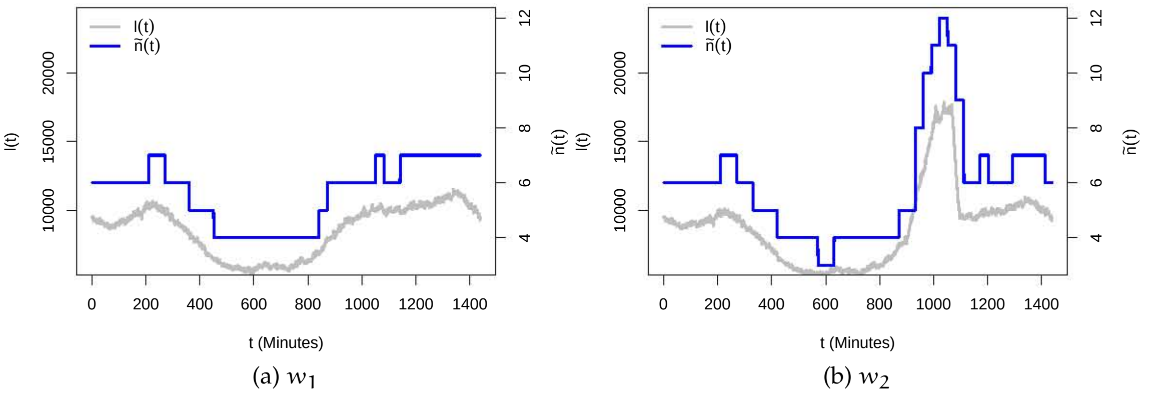


Figure 6.9: Average time used by proactive prediction methods to recalculate their models using $h = 30$.

6.6.3 *System sizing*

In this Section, we explore the evolution of system sizes. Figure 6.10 shows the global load $l(t)$ and the total number of provisioned servers $\tilde{n}(t)$ for the three Wikipedia workloads with unexpected events (w_1, w_2, and w_3) using $thr_u = 80\%$, random grouping, *OLARD* based on Holt Winters, and $d_s = 2048$ MB. We observe how the number of provisioned machines adequately evolves with $l(t)$ at a certain distance due to the overprovisioning. Most of the size variations only need to add or remove one machine. This changes with the peak in w_2. The workload increases so fast that the predictions recommend to add three machines at a time.



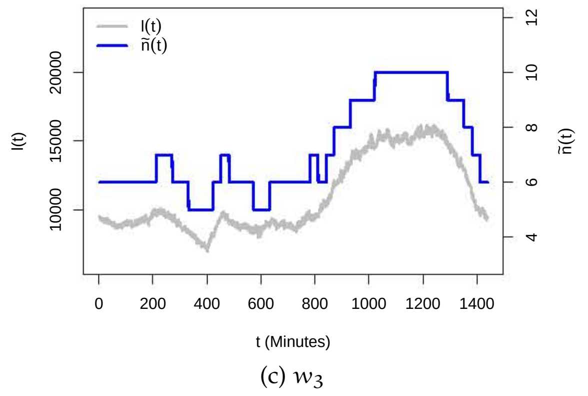


Figure 6.10: Example of system size variation for Wikipedia workloads, using $\text{thr}_u = 80\%$, random grouping, *OLARD* based on Holt-Winters, and $d_s = 2048$ MB.

Obviously different prediction models generate different predictions over time, therefore different amount of resources may be provisioned. Figure 6.11 shows the saved machine time of various server provisioning approaches, when compared with the traditional approach of statically provisioning a percentage over the maximum workload for w_1, w_2, and w_3 workloads of the Wikipedia dataset. In this experiment, we statically provision 10% over the maximum workload i.e. $\text{thr}_u = 90\%$. We employ all the prediction methods currently available in our solution with change detection disabled and enabled (ARIMA, ARIMA+A, AR_z, AR_z+A, HW, HW+A, minPE, minPE+A, minFE, and minFE+A), $\text{thr}_u = 90\%$, random grouping, *OLARD*, and $d_s = 2048$ MB. We calculate for both approaches the statically and the elastically the total amount of time machines that have been provisioned. Then, we calculate the saved machine time by comparing both values. The results indicate that the saved machine time is around 30% for w_1 and w_3, and 50% for w_2. The savings are larger for w_2 as this workload has a higher peak than the other workloads. This lower machine time makes our solution suitable for energy-aware solutions that try to reduce energy consumption by turning off machines.

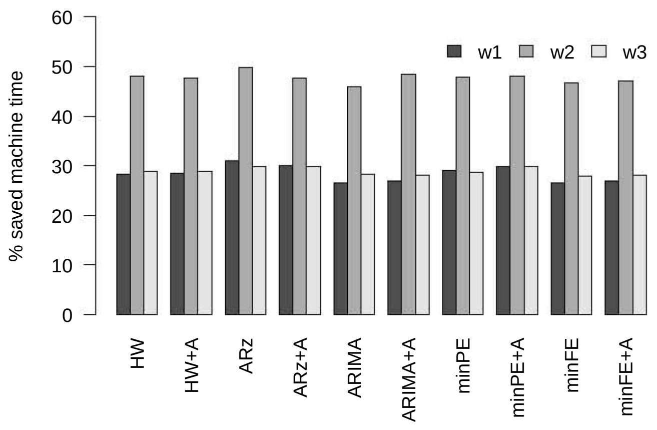


Figure 6.11: Saved machine time compared with a static approach provisioning the maximum number of servers needed to serve the given workload. The example uses $thr_u = 90\%$, random grouping, *OLARD*, and $d_s = 2048$ MB.

6.6.4 *Server utilization and load balance*

One of the most important objectives of any data placement technique is to maximize the server utilization, while perfectly balancing the load. We estimate for a time interval (t_1, t_2) the average server utilization $\overline{u}(t_1, t_2)$ and the load balance $\sigma_u(t_1, t_2)$. In the optimal case the servers are 100% loaded and, thus, perfectly load balanced. However, this is impractical for two reasons. First, unexpected peaks will be delayed until additional resources will be available. Second, as the load of a server gets closer to 100%, the variance of the response time is known to increase [81]. Therefore, a small over provisioning of $1 - thr_u$ has the role of mitigating both problems.

Figure 6.12 shows how our solution allows to estimate $\overline{u}(t_1, t_2)$ and $\sigma_u(t_1, t_2)$ using *OLARD*, random grouping, $thr_u = 90\%$, three prediction models (HW, AR_z, ARIMA, minPE, and minFE) with and without change detection enabled. How close the server utilization is to the desired utilization threshold thr_u depends on the accuracy of the prediction methods, global and local. The global model defines the system size, this may explain why using ARIMA the average utilization is closer to 80% instead of 90%. However, generalizing we can claim that the average server utilization does not exceed the threshold in any case. We observe that the utilization of change detection improves the average server utilization in all cases. The load balance is stable: σ_u is approximately 2% for w_1 and w_3, and 3% for w_2.

6.6.5 *Locality-aware requests distribution analysis*

An important indicator of the efficiency of a data placement policy is data locality. Elastic infrastructures have the additional problem of system variations when trying to conserve content locality. When turning off a machine the requests have to be distributed to the working servers. Every request to an object not stored in cache results in a request to the storage backend. A large data locality is translated into less requests to the storage backend. We analyze the data locality through time by calculating the average hit rate at time (t $\overline{hit}(t)$) for w_1, w_2 and w_3 Wikipedia scenarios using $thr_u = 80\%$, $h = 5$

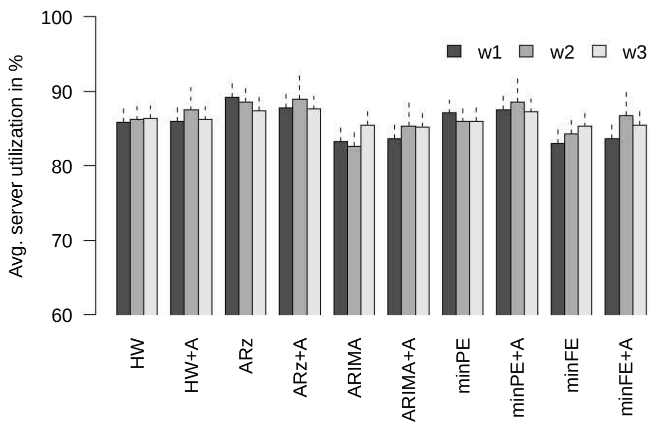
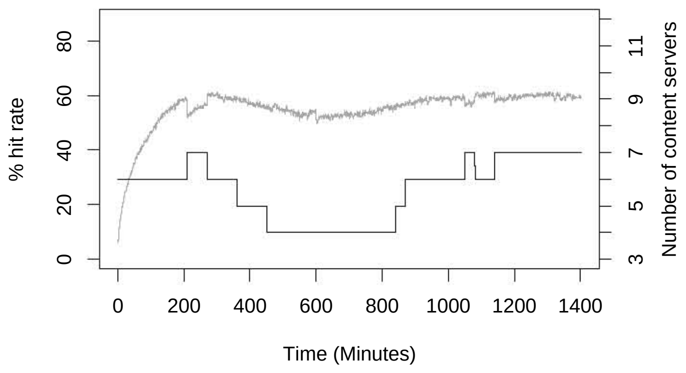


Figure 6.12: Average server utilization $\overline{u}(t_1, t_2)$ (bars) and load balance $\sigma_u(t_1, t_2)$ (dashed lines) for w_1, w_2 and w_3, $thr_u = 90$, and *OLARD* data placement.

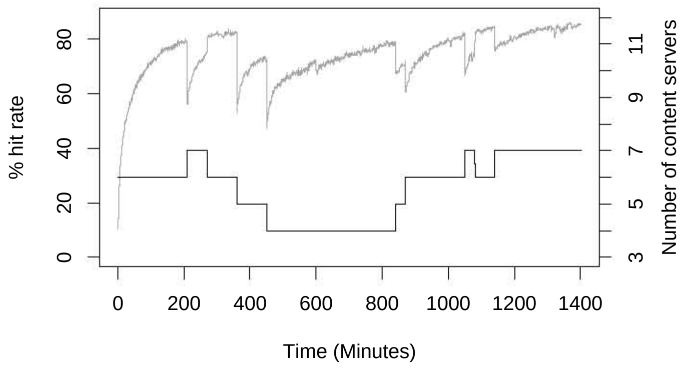
and Holt-Winters for the local and global models. Figures 6.13, 6.14 and 6.15 show the obtained results with the grey line indicating the hit rate and the black line the number of machines.

We observe that there is no difference among the three policies in the number of employed machines as they employ the same global prediction method. However, the hit rate is affected by the selected policy. Server caches are empty when starting the experiment. The hit rate increases for all the policies until the first size variation occurs at minute 200. From this point on, we observe clear differences depending on the employed policy. *RRD* shows a low hit rate variability in all scenarios, except during the unexpected workload peak in w_2. On contrast, *OLARD* increases the hit rate in all the scenarios when compared with *RRD*. We observe that certain system size variations produce short-term drops that do not occur in *RRD*. This occurs due to the redirection of requests from groups to other servers that now have to serve objects that might not have stored before. This is due to the fact that *OLARD* does not take into account past cache contents when updating the dispatching table. As explained in Section 6.5.2 *ABLARD* is intended to mitigate this problem by taking into account the past cache contents when updating the dispatching table. *ABLARD* obtains a hit rate similar to the one obtained in *OLARD* when no variations are present. However, *ABLARD* achieves a higher hit rate stability during system size variations. In particular, we observe that, when removing machines, *ABLARD* gets up to 10% more hit rate than *OLARD*.

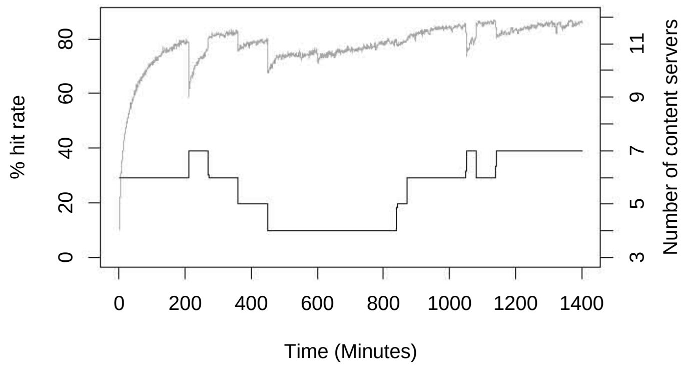
Apart from analyzing the system data locality over time, we summarize the obtained results by calculating the average hit rate for a period of time over all the servers as $\overline{hit}(t_1, t_2)$. Figure 6.16 shows a comparison of average hit rates among the mentioned three data placement policies employing Holt Winters as prediction model. We evaluate the average hit ratio for the Wikipedia workloads using $thr_u = 90\%$, random grouping and $d_s = 2048$. The results show that our proposed policies get a significantly higher average hit rate when compared with *RRD* for the three workloads. *OLARD* gets 35% more hit rate than *RRD* for w_1. For w_2 and w_3 our solutions get 20% higher hit rate than *RRD*. Finally, there is not a significant improvement on average between *OLARD* and *ABLARD*.



(a) *RRD*

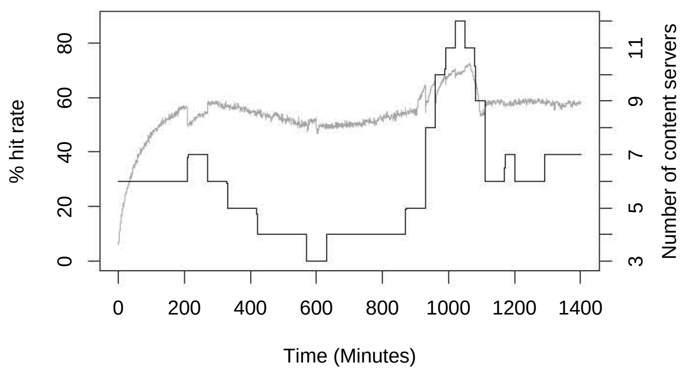


(b) *OLARD*

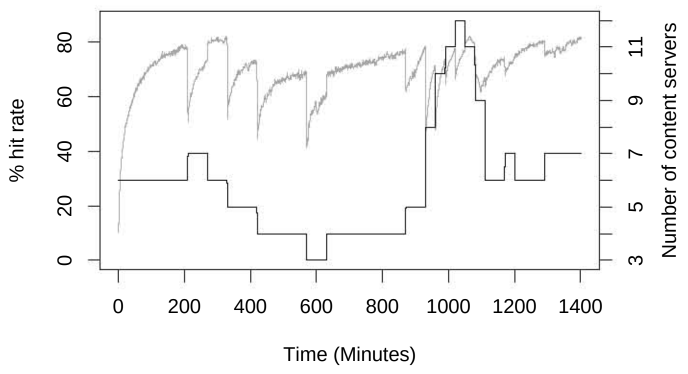


(c) *ABLARD*

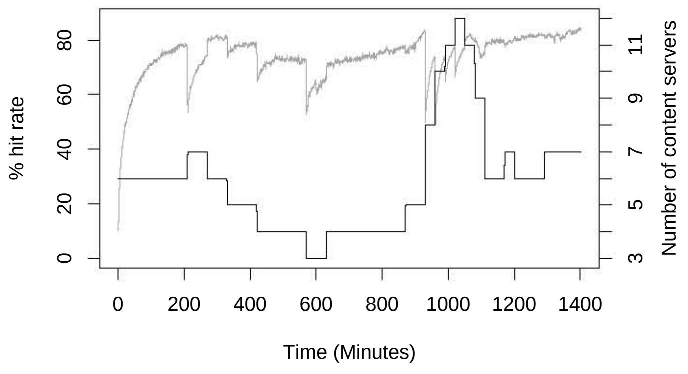
Figure 6.13: Hit rate and number of servers for w_1 from the Wikipedia dataset for $\mathrm{thr}_u = 80\%$, $h = 5$, $d_s = 2048$ MB, and Holt-Winters as prediction method.



(a) *RRD*

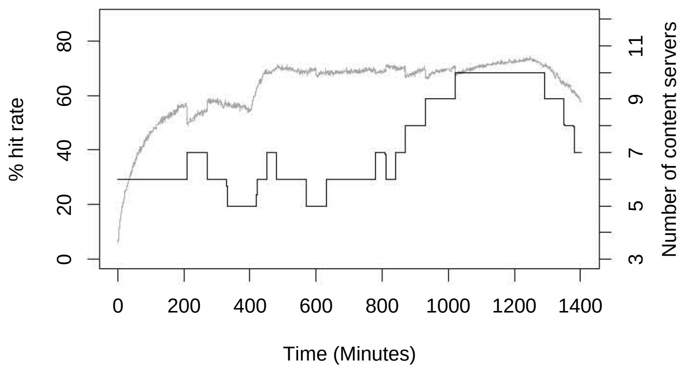


(b) *OLARD*

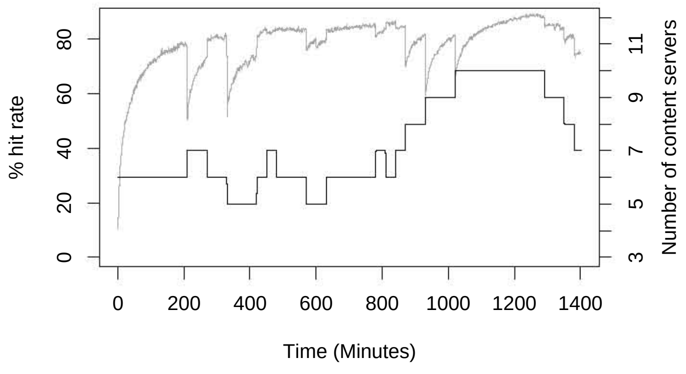


(c) *ABLARD*

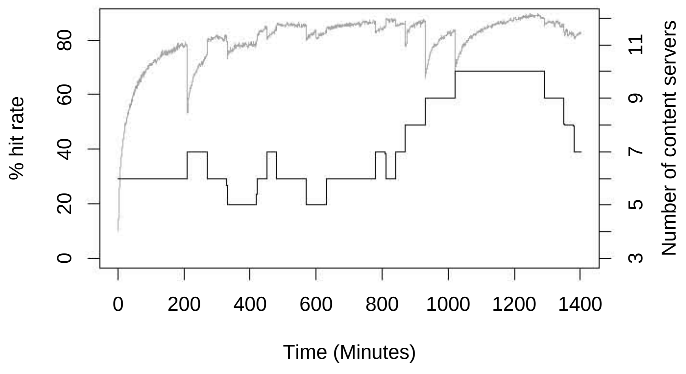
Figure 6.14: Hit rate and number of servers for w_2 from the Wikipedia dataset for $\mathrm{thr}_u = 80\%$, $h = 5$, $d_s = 2048$ MB, and Holt-Winters as prediction method.



(a) *RRD*



(b) *OLARD*



(c) *ABLARD*

Figure 6.15: Hit rate and number of servers for w_3 from the Wikipedia dataset for $\mathrm{thr}_u = 80\%$, $h = 5$, $d_s = 2048$ MB, and Holt-Winters as prediction method.

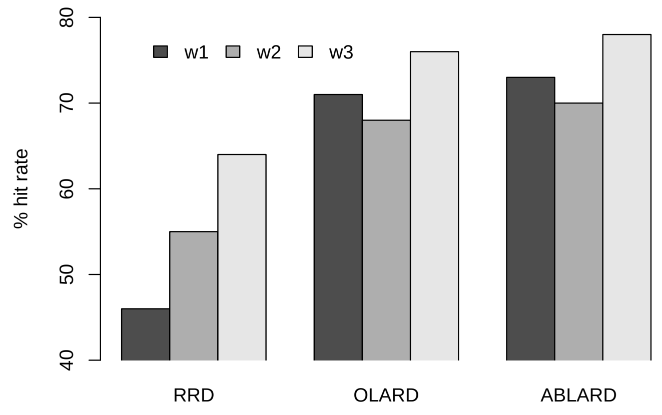


Figure 6.16: Hit rate for w_1, w_2 and w_3 from the Wikipedia dataset using $\mathrm{thr}_u = 90\%$, random grouping, Holt Winters, and $d_s = 2048$ MB with *RRD*, *OLARD* and *ABLARD*.

In a second analysis, we study the content locality resulting when varying the cache size d_s. We use the same parameters as in the previous experiment, except that we only focus on w_2 workload using three different cache sizes $d_s = 512$ MB, 1024 MB, and 2048 MB. As expected, the hit rate increases with the cache capacity for all methods. However, *OLARD* and *ABLARD* get 15% improvement over *RRD* for the three cache sizes. This demonstrates that independently of the employed cache size our solutions outperform *RRD*.

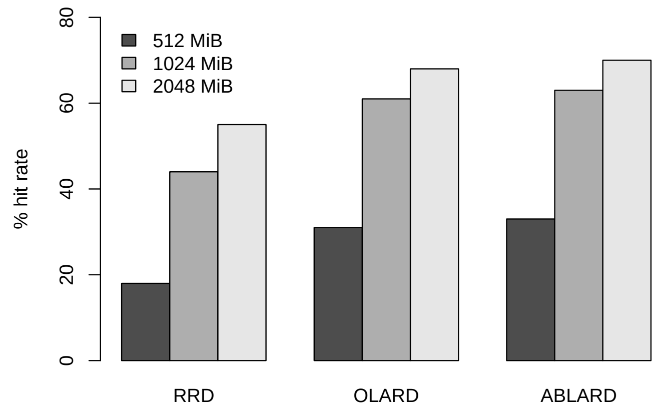


Figure 6.17: Hit rate for w_2 from the Wikipedia dataset, using $\mathrm{thr}_u = 90\%$, random grouping, and various server cache sizes $d_s = 512$ MB, 1024 MB, and 2048 MB for *RRD*, *OLARD* and *ABLARD*.

In Section 6.5.1 we describe an affinity-based data grouping method. We analyze the evolution of the contents assigned to the servers calculating the average Jaccard coefficient for the groups assigned to a server through time. The closer the metric is to 1, the better is the temporal content stability. Figure 6.18 shows for each of the 30 servers used in the simulation the mean of the values for this metric over the whole trace duration and the associated standard deviations for an execution using the Last.FM dataset. The first 18 servers show a value larger than 0.8, demonstrating that the popular and affine items have a high temporal stability. The value diminishes for servers 19, 20 and 21. These servers are likely to store contents that are less popular being more likely to migrate content between servers. Finally, the less popular events are mapped on the last servers which explains the high stability.

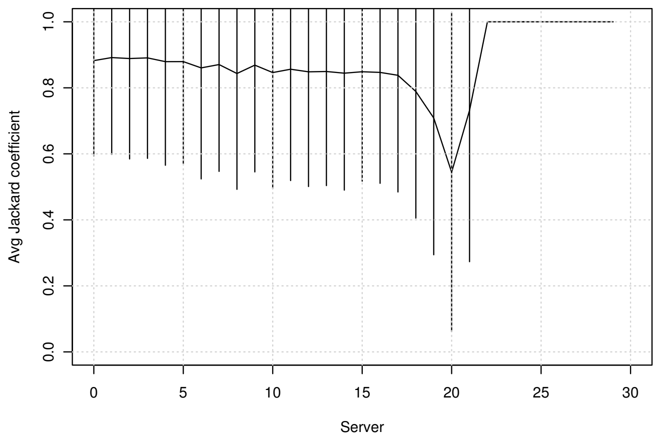


Figure 6.18: Dynamics of content across servers for a simulation using the Last.FM dataset measured using the Jaccard coefficient.

6.6.6 *Over- and underprovisioning evaluation*

CONDESA permits to define a desired utilization threshold $\mathtt{thr}_u$ that configures the system to work with a pre-defined ratio of overprovisioning. However, even when the average server utilization is the indicated in the configuration, it exists the possibility of having periods of underprovisioning specially during unexpected workload peaks. This problem can be solved by overprovisioning a large portion of resources, but this is not desirable. It exists a trade-off between increasing $\mathtt{thr}_u$ and avoid underprovisioning that depends on the accuracy of the employed prediction method, the configured $\mathtt{thr}_u$ and the workload variations.

We explore the over- and underprovisioning for all the proposed prediction methods and various $\mathtt{thr}_u$ values in the w_2 Wikipedia scenario. Figure 6.19 shows the overprovisioning and underprovisioning rates for executions using random grouping and *OLARD*. For each $\mathtt{thr}_u$ value, there are ten points corresponding to the employed prediction method: ARIMA, ARIMA+A, AR_z, AR_z+A, HW, HW+A, minPE, minPE+A, minFE, and minFE+A. The x-axis of each point on the plot represents the underprovisioning rate r_u and the y-axis the overprovisioning rate r_o, respectively.

We observe in Figure 6.19 three main clusters of points. First, the upper left cluster with r_o larger than 0.15 and r_u lower than 0.01 contains mostly points corresponding to target utilization rates $\mathtt{thr}_u$ of 75% and 80%. Second, the lower left cluster is the closest to the ideal under- and overprovisioning values and contains r_o values between 0.0 and 0.15 and r_u values between 0.0 and 0.01 corresponding mostly to target utilization rates $\mathtt{thr}_u$ with values between 85% and 95%. Third, the lower right cluster contains points representing low r_o values and r_u values larger than 0.01, corresponding to target utilization rates $\mathtt{thr}_u$ with values larger than 95%. We notice that, as expected, as the target system utilization gets closer to the 100% value, the underprovisioning rate increases, which has to be avoided. Independently of the underprovisioning that can be assumed, the suitable combinations of prediction method and thr_u are in the lower left cluster.

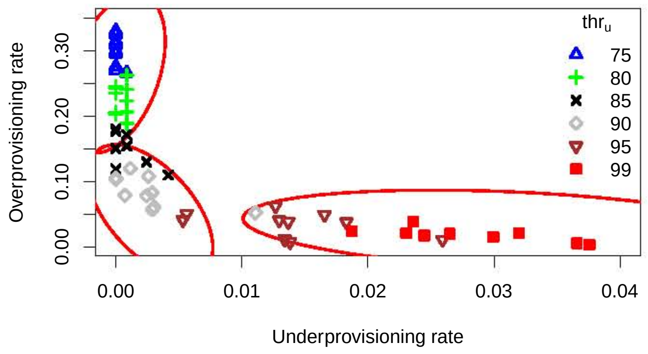


Figure 6.19: Overprovisioning rate r_o and underprovisioning rate r_u for workload w_2 from the Wikipedia dataset using random grouping, *OLARD*, various values of thr_u (75%, 80%, 85%, 90%, 95%, and 99%), five prediction methods (ARIMA, AR_z, Holt Winters, minPE, and minFE), and enabled/disabled change detection for the prediction models.

For $\text{thr}_u > 90\%$ we observe that becomes very difficult to avoid underutilization independently of the employed method. When using $\text{thr}_u \leqslant 90$ there are methods that avoid underprovisioning thus, we can consider 0.9 as the upper bound for the employed prediction methods here. To define what method is the most appropriate, we select the method that produces the lowest r_u for each combination of scenario and thr_u. We show the candidates in Table 6.6. We observe that for w_2 the best candidates employ change detection. In contrast, in scenarios w_1 and w_3 where the variation of the global workload is slower, disabling the change detector may get suitable results.

thr_u	w_1	w_2	w_3
75	AR_z+A	AR_z+A	AR_z+A
80	AR_z	AR_z+A	AR_z
85	HW+A	AR_z+A	AR_z
90	ARIMA	minFE+A	HW+A
95	minFE	minFE+A	ARIMA+A

Table 6.6: Best prediction models for different scenarios and thr_u for w_2 Wikipedia scenario.

6.7 SUMMARY

This Chapter covers the third objective of this thesis by exploring the design space of control methods for elastic server infrastructures. This infrastructure is one of the components of our proposed hierarchical architecture for content distribution. The infrastructure is supervised and configured by a system controller. This controller is designed to configure two aspects of the infrastructure: the system size and the data distribution. We employ automatically defined prediction models that permit to proactively identify the incoming workload. Moreover we add adaptivity to changes by detecting inaccurate models and reacting under unexpected events. Using the system controller we study the interplay between five data related strategies: workload prediction, adaptive control of data distribution and server provisioning, adaptive data grouping, adaptive data placement, and adaptive system sizing.

We have demonstrated how our system controller provides a powerful framework for browsing the design space of adaptive data distribution policies. We carry out an experimental evaluation using real traces extracted from Wikipedia and a collection of synthetically modified workloads with unexpected events. First, we analyze the accuracy of local models and demonstrate their utility to identify local workload variations. Second, we evaluate the impact of model change detection on prediction accuracy and how adaptive methods can be used for choosing and adequate prediction horizon. In particular, we observe an improvement in the stability of the predictions and a reduction in the obtained error. Third, we demonstrate how the adaptive prediction can be used for sizing a server system with up to 30% of time machine saving when compared with static approaches. Fourth, we have shown how prediction models can be employed in the design of locality-aware content placement policies. We demonstrate that independently of system size variations our locality-aware techniques can reduce up to 20% the traffic in the storage backend. Finally, we have shown how prediction models, change detection strategies and data placement policies can be combined and compared based on sever utilization, load balance, over- and underprovisioning, and data locality.

7 CONCLUSIONS

In this thesis, we have **proposed and studied a hierarchical content distribution architecture for large scale on-line communities that combines decentralized systems and cloud-based infrastructures**. We have fulfilled the three objectives presented in Section 1.2:

O1 **Explore a decentralized content distribution solution that leverages users preferences and community knowledge, in order to reduce pressure on the back-end infrastructures by improving content locality**. In Chapter 4, we present a cluster-based locality-aware collaborative system that employs user knowledge to improve data locality. We employ a scalable solution based on a structured network topology that organizes users into clusters of interest. We carry out an experimental evaluation with real traces demonstrating that our solution increases content locality while improving search latency and recall. Furthermore, we demonstrate how community knowledge can be employed to identify the affinity between clusters of interest and how to use this knowledge.

O2 **Study the data access patterns of Internet applications, analyze the limits of workload predictability, and investigate solutions to adapt to workload variability**. In Chapter 5, we explore the problem of forecasting web application workloads. We propose to employ automatically defined autoregressive prediction models to abstract away the user from the model definition process. Additionally, we propose the utilization of a multi-model approach that relieves the user from choosing which family of models is the best. Our experimental evaluation with three different real workloads and standard tools indicate that the overhead of automatically defining the prediction models is acceptable in the scope of this thesis. Moreover, we propose two model selection criteria that permit to combine multiple models demonstrating how this combination of models permit to improve accuracy.

O3 **Propose and study methods of prediction and control theory to enhance data distribution on elastic server infrastructures**. In Chapter 6, we explore the interplay of five data related strategies: workload prediction, adaptive control of data distribution and server provisioning, adaptive data grouping, adaptive data placement, and adaptive system sizing. We propose the design of a modular system controller in charge of supervising and configuring elastic infrastructures. We demonstrate that automatically generated prediction models can be employed to determine the system size. Additionally, we describe adaptive methods to detect unexpected demand variations that permit to adapt the system accordingly. Moreover, we demonstrate that the utilization of prediction methods can be employed in the design of locality-aware data distribution techniques that can reduce the traffic in the storage backend while maintaining load balance.

The design and evaluation of content distribution solutions is a complex task. The design space is vast and impossible to cover completely in a thesis. In this thesis, we have explored ideas to design low cost and highly scalable solutions by combining existing technologies such as P2P and cloud computing. We have made a significant effort to design methods to improve content locality in both distributed and server infrastructures. In the case of distributed infrastructures a high locality permits to improve search latency and recall. In server infrastructures the locality reduces the traffic in the storage backend. Additionally, we have explored the limits of workload prediction using different real workloads. We have extended the classic vision of workload forecasting in existing works [22, 38, 79, 17] by adding reactive components to prediction models in order to adapt to unexpected workloads. The combination of reactive and proactive models permit to have a future vision of the system while reacting in case of unexpected workloads or inaccurate models.

The utilization of proactive and reactive methods to determine the system size can be extended to other fields not directly related with content distribution. We have proposed a solution to design locality-aware content distribution policies based on this proactive and reactive duality. In this sense, we have proposed a set of data structures and updating mechanisms that deliver the content among a set of servers with limited resources taking into account the future demand. Our current approach is based on the idea of allocating contents taking into account the server capacity. However, other approaches taking into account response time, SLAs, energy consumption, cost, etc., can be easily designed based on our solution.

The evaluation of content distribution architectures is complex and has many aspects to be covered. We have evaluated an important set of them including content locality, resources utilization, load balance or over- and underprovisioning. However, some of them such as average response time, SLA violations, energy consumption, cost, etc., remain outside the scope of this thesis. We strongly believe that our methods and solutions can improve the results obtained in many of the mentioned aspects. However, evaluating these aspects requires complete implementations and remains for future work.

Finally, we find extremely difficult to make fair comparisons of our approach with other existing solutions. As mentioned in Chapter 2 it does not exist a common evaluation dataset that permit to compare values such as models accuracy or content locality among others. Additionally, existing works do not make their implementations available, or simply preparing a comparable evaluation scenario becomes time-consuming, unfeasible or even impossible due to the lack of information about how the evaluation was done. We believe that the definition of a common evaluation dataset or comparison framework remains crucial for the simplification of existing solutions.

7.1 CONTRIBUTIONS

This thesis makes the following contributions:

C1 **We design a cluster-based locality-aware self-organizing P2P system leveraging collaborative classification**. First, the system exploits Web 2.0 collaborative content classification in order to automatically improve content locality. Second, we propose a novel affinity-based metric for estimating the distance between clusters of interests. Third, we use a logarithmic-time parallel flooding algorithm that aims to achieve high data recall, high tolerance to node failure, and no redundant communication. Fourth, we propose a greedy cluster placement algorithm, which reorganizes clusters based on affinity in order to increase content locality.

C2 **We propose a novel multi-model technique for improving hierarchical workload prediction accuracy at aggregate and group levels**. The utilization of a hierarchy of models allows to detect workload changes at a finer level. Additionally, employing simultaneously multiple automatically generated prediction models permits to determine the appropriate prediction models to be used in each situation increasing the prediction accuracy.

C3 **We propose novel locality-aware data management policies based on predictive models for elastic server infrastructures and demonstrate that they improve content locality**. We demonstrate that employing prediction models in data management permits to design efficient locality-aware content distribution policies. Additionally, we propose techniques to combine several predictions models and demonstrate that they improve predictions accuracy and permit to abstract the user away from the task of selecting the most adequate models.

C4 **We propose novel techniques for adaptive control of elastic server infrastructures**. These techniques provide adaptability to changes by automatically detecting inaccurate models and reactive model redefinition. We demonstrate how the utilization of these techniques permits to estimate change point detections increasing accuracy. Moreover, we introduce novel metrics for evaluating efficiency of dynamic content distribution and server allocation policies. Using these metrics we provide a detailed evaluation of several aspects such as content locality, resource utilization and overprovisioning. In our evaluation we use traces extracted from real systems synthetically extended to contain unexpected workload peaks.

7.2 FUTURE WORK

After the completion of this thesis we identify various future research lines:

- **Elastic distributed file systems**. In this thesis, we have proposed an elastic server architecture for content distribution. However, this architecture can be extended to other application scenarios such as elastic distributed file systems. Our methods for determining the system size, data groups, and data placement can be easily adapted to this scenario. However, the utilization of our prediction mechanisms to forecast the number of I/O operations, how they affect the system performance and how to take advantage of proactive are research challenges to be addressed.
- **Framework for the design and comparison of system controllers**. In this thesis, we have proposed the design of a system controller for elastic server infrastructures. We have used this controller to evaluate several methods for adaptively determining the system size and configuring the content distribution while improving content locality. Based on our experience, we have already started to extend this framework as a research tool for designing, implementing and evaluating methods for elastic data distribution and management. Our idea is to offer this framework as a tool to combine existing methods for determining system size or configuring data distribution. We strongly believe that the design and share of a generic framework

might significantly facilitate the comparison of existing solutions, making easier to decide which is the best one.

- **Integration with real infrastructures**. Based on our experimental evaluations, we have acquired a detailed background about the problems of elastic server infrastructures. Additionally, we have proposed solutions for several aspects such as determining the system size or increasing content locality. We plan to integrate our solutions with real infrastructures such as Amazon EC2 [2] or OpenStack [66]. Our idea is to provide an on-line tool for elastic content server infrastructures. Moreover, this integration will permit us to further adapt our solutions to the particularities of each infrastructure and to deal with aspects such as cost or SLAs.

- **Energy-aware infrastructures**. Our current elastic server infrastructure approach permits to save energy by shutting down machines when they are not needed. The ideas proposed in this thesis can be extended to develop energy-aware solutions in order to minimize the energy consumption. Our ideas can be particularly relevant for owners of data centers where intelligent methods to adequate resources are directly translated into a reduction of energy costs. It is also possible to extend our solutions to design methods that explore the amount of energy needed in order to translate content among servers. This results into an optimization problem that tries to find the best data movements in order to save energy, while satisfying the incoming demand.

7.3 THESIS RESULTS

The main contributions of this thesis have been published in international conferences and journals. We enumerate the publications classified into journals, conferences and workshops. Additionally, we indicate the internships, grants and projects related to the results of this thesis.

- **Journals**
  1. J. M. Tirado, D. Higuero, J. Blas, F. Isaila, and J. Carretero, "CONDESA: A Framework for Controlling Data Distribution on Elastic Server Architectures," *Transactions on Parallel and Distributed Systems*, Under Review
  2. J. M. Tirado, D. Higuero, F. Isaila, J. Carretero, and A. Iamnitchi, "Affinity P2P: A self-organizing content-based locality-aware collaborative peer-to-peer network," *Computer Networks*, vol. 54, pp. 2056–2070, August 2010.
  3. D. Higuero, J. M. Tirado, J. Carretero, F. Felix, and A. Fuente "HIDDRA: A Highly Independent Data Distribution and Retrieval Architecture for Space Observation Missions," *Journal of Astrophysics & Space Science*, vol. 321, issue 3, pp. 169–175, 2009.

- **Conferences**
  1. J. M. Tirado, D. Higuero, F. Isaila, and J. Carretero, "Reconciling dynamic system sizing and content locality through hierarchical workload forecasting." in *18th IEEE International Conference on Parallel and Distributed Systems (ICPADS)*, 2012.

2. D. Higuero, J. M. Tirado, F. Isaila, and J. Carretero, "Enhancing file transfer scheduling and server utilization in data distribution infrastructure." in *IEEE 20th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2012.
3. J. Carretero, F. Isaila, A-M. Kermarrec, F. Taiani, J. M. Tirado, "Geology: Modular Georecommendation in Gossip-Based Social Networks" in *IEEE 32nd International Conference on Distributed Computing Systems (ICDCS)*, 2012.
4. J. M. Tirado, D. Higuero, F. Isaila, and J. Carretero, "Multi-model prediction for enhancing content locality in elastic server infrastructures," in *IEEE International Conference on High Performance Computing*, 2011.
5. J. M. Tirado, D. Higuero, F. Isaila, and J. Carretero, "Predictive data grouping and placement for cloud-based elastic server infrastructures," in *2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CC-Grid)*, 2011, pp. 285 –294.

- **Workshops**
  1. J. M. Tirado, D. Higuero, F. Isaila, and J. Carretero. "Analyzing the Impact of Events in an Online Music Community," in *Workshop on Social Network Systems*, held at Eurosys, 2011.

- **Research internships**

  INRIA Rennes Bretagne under the supervision of Anne-Marie Kermarrec, from September to December, 2011.

- **Grants**
  - Programa para la Formación del Profesorado Universitario (FPU) 2007, PhD fully granted (4 years), Ministerio de Educación y Ciencia.
  - Programa propio de ayudas para estancias, 1500 €, 2011, Universidad Carlos III.
  - IEEE ICDCS attendance grant (full registration), 650$, IEEE, 2012
  - IEEE ISPA student grant (full registration), 215 €, IEEE, 2012

- **Projects**
  - FPU program AP2007-03530
  - United States National Science Foundation grant CNS-0831785
  - Spanish Ministry of Education, TIN2007-63092
  - Spanish Ministry of Education, TIN2010-16497
  - ERC Starting Grant GOSSPLE number 204742

BIBLIOGRAPHY

[1] Karl Aberer, Luc Onana Alima, Ali Ghodsi, Sarunas Girdzijauskas, Seif Haridi, and Manfred Hauswirth. The essence of P2P: A reference architecture for overlay networks. In *Fifth IEEE International Conference on Peer-to-Peer Computing (P2P 2005), 31 August - 2 September 2005, Konstanz, Germany*, pages 11–20. IEEE Computer Society, 2005.

[2] Amazon. Amazon elastic compute cloud, 2012. URL http://aws.amazon.com/ec2.

[3] Amazon. Amazom auto-scaling. http://aws.amazon.com/autoscaling/, 2012.

[4] Stephanos Androutsellis-Theotokis and Diomidis Spinellis. A survey of peer-to-peer content distribution technologies. *ACM Comput. Surv.*, 36(4):335–371, December 2004.

[5] Zahid Anwar, William Yurcik, Vivek Pandey, Asim Shankar, Indranil Gupta, and Roy H. Campbell. Leveraging social-network infrastructure to improve peer-to-peer overlay performance: Results from orkut. *CoRR*, abs/cs/0509095, 2005.

[6] Danilo Ardagna, Sara Casolari, Michele Colajanni, and Barbara Panicucci. Dual time-scale distributed capacity allocation and load redirect algorithms for cloud systems. *Journal of Parallel and Distributed Computing*, 72(6):796 – 808, 2012.

[7] M. Arlitt and T. Jin. A workload characterization study of the 1998 world cup web site. *Network, IEEE*, 14(3):30–37, 2000.

[8] Martin F. Arlitt and Carey L. Williamson. Internet web servers: workload characterization and performance implications. *IEEE/ACM Trans. Netw.*, 5(5):631–645, October 1997.

[9] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. A view of cloud computing. *Commun. ACM*, 53:50–58, April 2010. ISSN 0001-0782.

[10] J. Bai. Estimation of a change point in multiple regression models. *The Review of Economics and Statistics*, pages 551–563, 1997.

[11] L.A. Barroso and U. Holzle. The case for energy-proportional computing. *Computer*, 40(12):33 –37, dec. 2007. ISSN 0018-9162.

[12] Fabrício Benevenuto, Tiago Rodrigues, Meeyoung Cha, and Virgílio Almeida. Characterizing user behavior in online social networks. In *IMC '09: Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, pages 49–62, New York, NY, USA, 2009. ACM.

[13] Marin Bertier, Davide Frey, Rachid Guerraoui, Anne-Marie Kermarrec, and Vincent Leroy. The gossple anonymous social network. In *Proceedings of the ACM/IFIP/USENIX 11th International Conference on Middleware*, Middleware '10, pages 191–211, Berlin, Heidelberg, 2010. Springer-Verlag. ISBN 978-3-642-16954-0.

[14] BitTorrent. Bittorrent protocol, 2012. URL http://www.bittorrent.org.

[15] P. Bodik, G. Friedman, L. Biewald, H. Levine, G. Candea, K. Patel, G. Tolle, J. Hui, A. Fox, M.I. Jordan, et al. Combining visualization and statistical analysis to improve operator confidence and efficiency for failure detection and localization. In *Autonomic Computing, 2005. ICAC 2005. Proceedings. Second International Conference on*, pages 89–100. IEEE, 2005.

[16] Peter Bodik, Rean Griffith, Charles Sutton, Armando Fox, Michael I. Jordan, and David A. Patterson. Statistical machine learning makes automatic control practical for internet datacenters. In *In Workshop on Hot Topics in Cloud Computing (HotCloud '09)*, 2009.

[17] Peter Bodik, Armando Fox, Michael Franklin, Michael Jordan, and David Patterson. Characterizing, modeling, and generating workload spikes for stateful services. In *In Symposium on Cloud Computing (SOCC)*, 2010.

[18] R. Bolla, R. Gaeta, A. Magnetto, M. Sciuto, and M. Sereno. A measurement study supporting p2p file-sharing community models. *Computer Networks*, 53(4):485 – 500, 2009. ISSN 1389-1286. Content Distribution Infrastructures for Community Networks.

[19] Valeria Cardellini, Emiliano Casalicchio, Michele Colajanni, and Philip S. Yu. The state of the art in locally distributed web-server systems. *ACM Comput. Surv.*, 34 (2):263–311, 2002.

[20] Bengt Carlsson and Rune Gustavsson. The Rise and Fall of Napster - An Evolutionary Approach. In *Proceedings of the 6th International Computer Science Conference on Active Media Technology*, AMT '01, pages 347–354, 2001.

[21] Kevin M. Carter, Richard P. Lippmann, and Stephen W. Boyer. Temporally oblivious anomaly detection on large networks using functional peers. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, IMC '10, pages 465–471, 2010. ISBN 978-1-4503-0483-2.

[22] Sara Casolari and Michele Colajanni. Short-term prediction models for server management in internet-based contexts. *Decision Support Systems*, 48(1):212 – 223, 2009. ISSN 0167-9236.

[23] M. Cha, H. Kwak, P. Rodriguez, Y.Y. Ahn, and S. Moon. I tube, you tube, everybody tubes: analyzing the world's largest user generated content video system. In *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, page 14. ACM, 2007.

[24] Meeyoung Cha, Alan Mislove, and Krishna P. Gummadi. A measurement-driven analysis of information propagation in the flickr social network. In *Proceedings of*

the 18th international conference on World wide web, WWW '09, pages 721–730, New York, NY, USA, 2009. ACM.

[25] Meeyoung Cha, Hamed Haddadi, Fabricio Benevenuto, and Krishna P. Gummadi. Measuring User Influence in Twitter: The Million Follower Fallacy. In *In Proceedings of the 4th International AAAI Conference on Weblogs and Social Media (ICWSM)*, 2010.

[26] Abhishek Chandra, Weibo Gong, and Prashant Shenoy. Dynamic resource allocation for shared data centers using online measurements. In *Proceedings of the 2003 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, SIGMETRICS '03, pages 300–301, New York, NY, USA, 2003. ACM. ISBN 1-58113-664-1.

[27] Jeffrey S. Chase, Darrell C. Anderson, Prachi N. Thakar, Amin M. Vahdat, and Ronald P. Doyle. Managing energy and server resources in hosting centers. *SIGOPS Oper. Syst. Rev.*, 35(5):103–116, 2001. ISSN 0163-5980.

[28] Xu Cheng and Jiangchuan Liu. NetTube: Exploring social networks for peer-to-peer short video sharing. In *Proceedings of INFOCOM 2009*, 2009.

[29] Xu Cheng, C. Dale, and Jiangchuan Liu. Statistics and Social Network of YouTube Videos. In *Quality of Service, 2008. IWQoS 2008. 16th International Workshop on*, pages 229 –238, 2008.

[30] Arturo Crespo and Hector Garcia-Molina. Semantic overlay networks for P2P systems. In *Agents and P2P Computing*, volume 3601, pages 1–13. Springer, 2005. ISBN 3-540-29755-3.

[31] Brian Eriksson, Paul Barford, Rhys Bowden, Nick Duffield, Joel Sommers, and Matthew Roughan. Basisdetect: a model-based network event detection framework. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, IMC '10, pages 451–464, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0483-2.

[32] Facebook, 2012. URL `http://www.facebook.com`.

[33] A. Forestiero, C. Mastroianni, and M. Meo. Self-Chord: a Bio-Inspired Algorithm for Structured P2P Systems. In *Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid-Volume 00*, pages 44–51. IEEE Computer Society, 2009.

[34] I. Foster, Yong Zhao, I. Raicu, and S. Lu. Cloud computing and grid computing 360-degree compared. In *Grid Computing Environments Workshop, 2008. GCE '08*, pages 1 –10, nov. 2008.

[35] Wikipedia Foundation. Wikipedia page counters, 2012. URL `http://stats.grok.se/en/200906/michael%20jackson`.

[36] Gregory Reinsel George Box, Gwilym M. Jenkins. *Time Series Analysis and Control*. Wiley, 1970.

[37] Phillipa Gill, Martin Arlitt, Zongpeng Li, and Anirban Mahanti. Youtube traffic characterization: a view from the edge. In *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, IMC '07, pages 15–28. ACM, 2007.

[38] Daniel Gmach, Jerry Rolia, Ludmila Cherkasova, and Alfons Kemper. Workload analysis and demand prediction of enterprise data center applications. In *Proceedings of the 2007 IEEE 10th International Symposium on Workload Characterization*, pages 171–180, 2007.

[39] Z. Gong, X. Gu, and J. Wilkes. Press: Predictive elastic resource scaling for cloud systems. In *Network and Service Management (CNSM), 2010 International Conference on*, pages 9–16. IEEE, 2010.

[40] Jan G. De Gooijer and Rob J. Hyndman. 25 years of time series forecasting. *International Journal of Forecasting*, 22(3):443–473, 2006.

[41] Vijay Gopalakrishnan, Rittwik Jana, K. K. Ramakrishnan, Deborah F. Swayne, and Vinay A. Vaishampayan. Understanding couch potatoes: measurement and modeling of interactive usage of iptv at large scale. In *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*, IMC '11, pages 225–242. ACM, 2011. ISBN 978-1-4503-1013-0.

[42] Albert Greenberg, James Hamilton, David A. Maltz, and Parveen Patel. The cost of a cloud: research problems in data center networks. *SIGCOMM Comput. Commun. Rev.*, 39(1):68–73, 2008.

[43] Xiaojun Hei, Chao Liang, Jian Liang, Yong Liu, and K.W. Ross. A Measurement Study of a Large-Scale P2P IPTV System. *Multimedia, IEEE Transactions on*, 9(8): 1672–1687, 2007.

[44] Rob J. Hyndman and Yeasmin Khandakar. Automatic time series forecasting: The forecast package for r. *Journal of Statistical Software*, 27(3):1–22, 7 2008. ISSN 1548-7660.

[45] A. Iamnitchi and I. Foster. Interest-aware information dissemination in small-world communities. In *HPDC '05: Proceedings of the High Performance Distributed Computing, 2005. HPDC-14. Proceedings. 14th IEEE International Symposium*, pages 167–175, Washington, DC, USA, 2005. IEEE Computer Society. ISBN 0-7803-9037-7.

[46] A. Iamnitchi, S. Doraimani, and G. Garzoglio. Filecules in high-energy physics: Characteristics and impact on resource management. *15th IEEE International Symposium on High Performance Distributed Computing (HPDC)*, pages 69–79, 2006.

[47] Adriana Iamnitchi, Matei Ripeanu, and Ian Foster. Small-world file-sharing communities. In *The 23rd Conference of the IEEE Communications Society (InfoCom 2004)*, volume 2, pages 952–963, Hong Kong, 2004.

[48] Sibren Isaacman, Stratis Ioannidis, Augustin Chaintreau, and Margaret Martonosi. Distributed rating prediction in user generated content streams. In *Proceedings of the fifth ACM conference on Recommender systems*, RecSys '11, pages 69–76, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0683-6.

[49] Hai Jin, Xiaomin Ning, and Hanhua Chen. Efficient search for peer-to-peer information retrieval using semantic small world. In *WWW '06: Proceedings of the 15th international conference on World Wide Web*, pages 1003–1004, New York, NY, USA, 2006. ACM.

[50] M. Jovanović, F. Annexstein, and K. Berman. Modeling peer-to-peer network topologies through small-world models and power laws. In *IX Telecommunications Forum, TELFOR*, 2001.

[51] Christos Karamanolis, Magnus Karlsson, and Xiaoyun Zhu. Designing controllable computer systems. In *Proceedings of the 10th conference on Hot Topics in Operating Systems - Volume 10*, HOTOS'05, 2005.

[52] B. Wittenmark. K.J. Astrom. *Adaptive control.* Addison-Wesley, 1995.

[53] Andrew Krioukov, Prashanth Mohan, Sara Alspaugh, Laura Keys, David Culler, and Randy Katz. Napsac: Design and implementation of a power-proportional web cluster. In *n Proceedings of the First ACM SIGCOMM Workshop on Green Networking I*, New Delhi, India, August 2010.

[54] Dara Kusic, Jeffrey O. Kephart, James E. Hanson, Nagarajan Kandasamy, and Guofei Jiang. Power and performance management of virtualized computing environments via lookahead control. *Cluster Computing*, 12(1):1–15, 2009.

[55] Haewoon Kwak, Changhyun Lee, Hosung Park, and Sue Moon. What is twitter, a social network or a news media? In *Proceedings of the 19th international conference on World wide web*, WWW '10, pages 591–600, 2010.

[56] Mario Lassnig, Thomas Fahringer, Vincent Garonne, Angelos Molfetas, and Miguel Branco. Identification, modelling and prediction of non-periodic bursts in workloads. In *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, CCGRID '10, pages 485–494, Washington, DC, USA, 2010. IEEE Computer Society. ISBN 978-0-7695-4039-9.

[57] Last.FM. Site statistics, March 2009. URL `http://blog.last.fm/2009/03/24/lastfm-radio-announcement`.

[58] LastFM. Lastfm web page, 2012. URL `http://www.last.fm`.

[59] Jiangchuan Liu, S.G. Rao, Bo Li, and Hui Zhang. Opportunities and Challenges of Peer-to-Peer Internet Video Broadcast. *Proceedings of the IEEE*, 96(1):11–24, 2008.

[60] Xue Liu, Jin Heo, Lui Sha, and Xiaoyun Zhu. Adaptive control of multi-tiered web applications using queueing predictor. In *Network Operations and Management Symposium, 2006. NOMS 2006. 10th IEEE/IFIP*, pages 106 –114, april 2006.

[61] M. Maurer, I. Brandic, V.C. Emeakaroha, and S. Dustdar. Towards knowledge management in self-adaptable clouds. In *Services (SERVICES-1), 2010 6th World Congress on*, pages 527 –534, july 2010.

[62] P. Maymounkov and D. Mazieres. Kademlia: A peer-to-peer information system based on the xor metric, 2002.

[63] A. I. McLeod and Y. Zhang. Improved Subset Autoregression With R Package. *Journal of Statistical Software*, 28(2):1–28, 10 2008.

[64] E. Meshkova, J. Riihijärvi, M. Petrova, and P. Mähönen. A survey on resource discovery mechanisms, peer-to-peer and service discovery frameworks. *Computer Networks*, 52(11):2097 – 2128, 2008. ISSN 1389-1286.

[65] Alan Mislove, Massimiliano Marcon, Krishna P. Gummadi, Peter Druschel, and Bobby Bhattacharjee. Measurement and analysis of online social networks. In *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, IMC '07, pages 29–42, 2007.

[66] OpenStack. Openstack open source cloud computing software, 2012. URL http://www.openstack.org/.

[67] Nish Parikh and Neel Sundaresan. Scalable and near real-time burst detection from ecommerce queries. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '08, pages 972–980, 2008.

[68] Josiane Xavier Parreira, Sebastian Michel, and Gerhard Weikum. P2PDating: Real life inspired semantic overlay networks for web search. *Information Processing & Management*, 43(3):643–664, 2007.

[69] J. A. Pouwelse, P. Garbacki, J. Wang, A. Bakker, J. Yang, A. Iosup, D. H. J. Epema, M. Reinders, M. R. van Steen, and H. J. Sips. Tribler: a social-based peer-to-peer system: Research articles. *Concurrency and Compututation: Practice and Experience*, 20 (2):127–138, 2008. ISSN 1532-0626.

[70] Anna Puig-Centelles, Oscar Ripolles, and Miguel Chover. Reviewing P2P network community detection. In *IASTED European Conference on Proceedings of the IASTED European Conference: internet and multimedia systems and applications*, IMSA'07, pages 122–127, 2007.

[71] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2012. URL http://www.R-project.org/. ISBN 3-900051-07-0.

[72] P. Raftopoulou and E.G.M. Petrakis. iCluster: A Self-organizing Overlay Network for P2P Information Retrieval. In *Advances in information retrieval: 30th European Conference on IR Research, ECIR 2008*, page 65, 2008.

[73] Jia Rao, Xiangping Bu, Cheng-Zhong Xu, and Kun Wang. A distributed self-learning approach for elastic provisioning of virtualized cloud resources. In *Modeling, Analysis Simulation of Computer and Telecommunication Systems (MASCOTS), 2011 IEEE 19th International Symposium on*, pages 45 –54, july 2011.

[74] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard M. Karp, and Scott Shenker. A scalable content-addressable network. In *SIGCOMM*, pages 161–172, 2001.

[75] Matei Ripeanu. Peer-to-peer architecture case study: Gnutella network. In *Proceedings of the 1st IEEE International Conference on Peer-to-Peer Computing*, 8 2001.

[76] Habib Rostami, Jafar Habibi, and Emad Livani. Semantic routing of search queries in P2P networks. *Journal of Parallel Distributed Computing*, 68(12):1590–1602, 2008. ISSN 0743-7315.

[77] Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. *Lecture Notes in Computer Science*, 2218:329–350, 2001. ISSN 0302-9743.

[78] Sandvine. Sandvine global internet phenomena spotlight. Technical report, Sandvine, 2011. URL http://www.sandvine.com/general/document.download.asp?docID=51&sourceID=0.

[79] Carlos Santana, Julius Leite, and Daniel Moss. Power management by load forecasting in web server clusters. *Cluster Computing*, 14:471–481, 2011.

[80] Fabian Schneider, Anja Feldmann, Balachander Krishnamurthy, and Walter Willinger. Understanding online social network usage from a network perspective. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, IMC '09, pages 35–48, 2009. ISBN 978-1-60558-771-4.

[81] Piyush Shivam, Varun Marupadi, Jeffrey S. Chase, Thileepan Subramaniam, and Shivnath Babu. Cutting corners: Workbench automation for server benchmarking. In *USENIX Annual Technical Conference*, pages 241–254, 2008.

[82] R.H. Shumway and D.S. Stoffer. *Time series analysis and its applications, 3rd edition*. Springer Verlag, 2010. ISBN 0387989501.

[83] Ion Stoica, Robert Morris, David R. Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM*, pages 149–160, 2001.

[84] D. Stutzbach, R. Rejaie, and S. Sen. Characterizing Unstructured Overlay Topologies in Modern P2P File-Sharing Systems. *Networking, IEEE/ACM Transactions on*, 16(2):267–280, April 2008.

[85] A. Colman T. Patikirikorala. Feedback controllers in the cloud. In *Proceedings of APSEC*, 2010.

[86] Christoph Tempich, Steffen Staab, and Adrian Wranik. Remindin': semantic query routing in peer-to-peer networks based on social metaphors. In *WWW '04: Proceedings of the 13th international conference on World Wide Web*, pages 640–649, New York, NY, USA, 2004. ACM Press. ISBN 158113844X.

[87] UMASS trace repository. YouTube traces, 2008. URL http://traces.cs.umass.edu/index.php/Network/Network.

[88] Trendistic. Top 20 trends in 2009, 2012. URL http://trendistic.com/_top-twenty-trending-topics-2009/.

[89] Twitter. Inauguration day on twitter, 2012. URL http://blog.twitter.com/2009/01/inauguration-day-on-twitter.html.

[90] Twitter. Twitter celebrates its sixth anniversary, 2012. URL http://blog.twitter.com/2012/03/twitter-turns-six.html.

[91] Guido Urdaneta, Guillaume Pierre, and Maarten van Steen. Wikipedia workload analysis for decentralized hosting. *Comput. Netw.*, 53:1830–1845, July 2009.

[92] Bhuvan Urgaonkar, Giovanni Pacifici, Prashant Shenoy, Mike Spreitzer, and Asser Tantawi. An analytical model for multi-tier internet services and its applications. In *Proceedings of the 2005 ACM SIGMETRICS international conference on Measurement*

and modeling of computer systems, SIGMETRICS '05, pages 291–302, New York, NY, USA, 2005. ACM. ISBN 1-59593-022-1.

[93] V.V. Vazirani. *Approximation algorithms*. Springer Verlag, 2001.

[94] T. Vercauteren, P. Aggarwal, Xiaodong Wang, and Ta-Hsin Li. Hierarchical forecasting of web server workload using sequential monte carlo training. *Signal Processing, IEEE Transactions on*, 55(4):1286 –1297, 2007.

[95] Akshat Verma, Ricardo Koller, Luis Useche, and Raju Rangaswami. Srcmap: energy proportional storage using dynamic consolidation. In *Proceedings of the 8th USENIX conference on File and storage technologies*, FAST'10, pages 20–20, 2010.

[96] S. Voulgaris, M. van Steen, and K. Iwanicki. Proactive gossip-based management of semantic overlay networks. *Concurrency and Computation*, 19(17):2299–2311, 2007.

[97] Duncan J. Watts. *Small Worlds : The Dynamics of Networks between Order and Randomness (Princeton Studies in Complexity)*. Princeton University Press, November 2003. ISBN 0691117047.

[98] Adepele Williams, Martin Arlitt, Carey Williamson, and Ken Barker. Web workload characterization: Ten years later. In *Web Content Delivery*, volume 2 of *Web Information Systems Engineering and Internet Technologies Book Series*, pages 3–21. Springer US, 2005. ISBN 978-0-387-24356-6.

[99] Rob J Hyndman with contributions from Slava Razbash and Drew Schmidt. *forecast: Forecasting functions for time series and linear models*, 2012. URL http://CRAN.R-project.org/package=forecast. R package version 3.24.

[100] Hao Yin, Xuening Liu, Feng Qiu, Ning Xia, Chuang Lin, Hui Zhang, Vyas Sekar, and Geyong Min. Inside the bird's nest: measurements of large-scale live VoD from the 2008 olympics. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, IMC '09, pages 442–455, 2009.

[101] YouTube, 2012. URL http://www.youtube.com.

[102] Hui Zhang, Guofei Jiang, Haifeng Chen, Kenji Yoshihira, and Akhilesh Saxena. Intelligent workload factoring for a hybrid cloud computing model. In *International Workshop on Cloud Services*, 2009.

[103] Jia Zhou, Yanhua Li, Vijay Kumar Adhikari, and Zhi-Li Zhang. Counting YouTube videos via random prefix sampling. In *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*, IMC '11, pages 371–380, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-1013-0.

[104] Xiaoyun Zhu, Mustafa Uysal, Zhikui Wang, Sharad Singhal, Arif Merchant, Pradeep Padala, and Kang Shin. What does control theory bring to systems research? *SIGOPS Oper. Syst. Rev.*, 43(1):62–69, January 2009.

INDEX