Resource Allocation in Distributed Service Networks

Nitish Kumar Panigrahy
University of Massachusetts Amherst

RESOURCE ALLOCATION IN DISTRIBUTED SERVICE NETWORKS

A Dissertation Presented

by

NITISH K. PANIGRAHY

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

September 2021

College of Information and Computer Sciences

RESOURCE ALLOCATION IN DISTRIBUTED SERVICE NETWORKS

A Dissertation Presented

by

NITISH K. PANIGRAHY

Approved as to style and content by:

Don Towsley, Chair

Ramesh K. Sitaraman, Member

Prashant Shenoy, Member

Prithwish Basu, Member

James Allan, Chair
College of Information and Computer Sciences

DEDICATION

To my grand mother, Jeji.

ACKNOWLEDGMENTS

This thesis would not have been possible without the support of many people. Firstly, I would like to express sincere gratitude to my advisor, Don Towsley who has been an exceptional support and constant source of inspiration throughout the years of my PhD study. I can not thank him enough for continuously presenting me with so many wonderful opportunities. Thank you Don for believing and guiding me.

I would also like to thank Prithwish Basu (Raytheon BBN) for being a wonderful supervisor during my internship at BBN. It has been a great pleasure to collaborate with him and take advice on a life in industry. I am very grateful to my other thesis committee members: Ramesh Sitaraman (UMass) and Prashant Shenoy (UMass). Their useful feedback and suggestions certainly improved this thesis.

I am thankful to everyone with whom I have had a chance to work with. My early collaborators: Kris Hollot (UMass), Ananthram Swami (ARL), Kevin S. Chan (ARL), Kin K. Leung (Imperial) and Faheem Zafari (Imperial). All past and current members of the Computer Networks Research Group at UMass: Kun, James, Bo, Jian, Gayane, Amir, Stefan, Arman, Albert, Janice, Thiru, Matheus, Mohammad and Wenhan. My current collaborators: Philippe Nain (Inria) and Giovanni Neglia (Inria). Staff members at UMass: Laurie Connors, Leeanne Leclerc, and Eileen Hamel.

I am very grateful to my parents Pratibha and Bipin for their constant support and motivation which helped me believe I can successfully complete this thesis. I would also like to thank my grandmother Jeji. Jeji! you have been in my thoughts, and prayers, miss you. I am grateful to my brother Pritish, sister-in-law Sutapa, mother-in-law Sandhya, father-in-law Puspendra and brother-in-law Soubhagya for being such an awesome family. Thank you my nephew Neevu and niece Aadya for being so cute and adorable.

Last but not least thanks to my wife Prachi for being there. Through thick and thin. Thank you for being the wonderful life partner, friend and guide you are. Love you.

ABSTRACT

RESOURCE ALLOCATION IN DISTRIBUTED SERVICE NETWORKS

SEPTEMBER 2021

NITISH K. PANIGRAHY

B.Tech., NATIONAL INSTITUTE OF TECHNOLOGY, ROURKELA

M.Tech., INDIAN STATISTICAL INSTITUTE, KOLKATA

Ph.D., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Don Towsley

The past few years have witnessed significant growth in the use of a large number of smart devices, computational and storage resources. These devices and resources distributed over a physical space are collectively called a *distributed service network*. One of the new applications closely related to a distributed service network is the *internet of things* (IoT) system. Efficient resource allocation for such high performance IoT system remains one of the most critical problems. In this thesis, we model and optimize the allocation of resources in a distributed service network. This thesis contributes to two different types of service networks: caching, and spatial networks; and develops new techniques that optimize the overall performance of these services.

First, we propose a new method to compute an upper bound on hit probability for all non-anticipative caching policies in a distributed caching system. We order the contents according to the ratio of their Hazard Rate (HR) function values to their sizes and place in the cache the contents with the largest ratios till the cache capacity is exhausted. We find our bound to be tighter than the state-of-the-art upper bounds for a variety of content request arrival processes.

We then develop a utility based framework for content placement in a network of caches for efficient and fair allocation of caching resources through timer-based (TTL) caching policies. We develop provably optimal distributed algorithms that operate at each network cache to maximize the overall network utility. Our TTL-based optimization model provides theoretical answers to how long each content must be cached, and where it should be placed in the cache network.

Next, we develop and evaluate assignment policies that allocate resources to users with a goal to minimize the expected distance traveled by a user request (request distance), where both resources and users are located on a line. We consider unidirectional assignment policies that allocate resources only to users located to their left. We show that when user and resource locations are modeled by statistical point processes, the spatial system under unidirectional policies can be mapped into bulk service queueing systems, thus allowing the application of many queueing theory results that yield closed form expressions. We also consider bidirectional policies where there are no directional restrictions on resource allocation and develop an algorithm for computing the optimal assignment which is more efficient than known algorithms in the literature when there are more resources than users.

Lastly, we design and evaluate resource proximity aware user-request allocation policies with a goal to reduce the cost associated with moving a request/job to/from its allocated resource or *implementation cost* and balance the number of requests allocated to a resource or the *load*. We consider a class of proximity aware Power of Two (POT) choice based assignment policies for allocating user-requests to resources, where both users and resources are located on a two-dimensional Euclidean plane. In this framework, we investigate the tradeoff between the implementation cost, and load balancing performance of different allocation policies. We propose two non-uniform resource sampling based POT policies that achieve the best of both implementation cost and load balancing performance. We then extend our analysis to the case where resources are interconnected as an arbitrary graph. Depending on topology, our proposed policies achieve a $8\% - 99\%$ decrease in implementation cost as compared to the state-of-the-art.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

The past few years have witnessed significant growth in the use of distributed network analytics involving agile code, data and computational resources. In many such networked systems, for example, Internet of Things (*IoT*) [10], a large number of smart devices, sensors, computational and storage resources are widely distributed in the physical world. These devices and resources are accessed by various end users/applications that are also distributed across the physical space. Delivering the content generated from the deployed smart devices to users efficiently and assigning users or applications to various resources are critical to sustained high performance operation of such a distributed system.

This thesis covers two main topics: the first concerns characterizing the fundamental limits and optimal content placement problem in a network of storage resources/caches, and the second concerns optimal resource allocation and load balancing problem in distributed spatial networks. The main focus of this thesis will be on a setting where resources or services are distributed across a physical space. We collectively call such a set of resources/services a *distributed service network*. We develop new analytical techniques and performance bounds for a caching network with the goal to efficiently and fairly deliver content to the end user. We also design and evaluate techniques to optimize the allocation of storage and computational resources to end users over a physical space with a goal to distribute these user requests as evenly as possible among resources.

1.1 Research Questions and Contributions

This thesis addresses three research questions motivated by two types of distributed service networks. The first two questions concern with a service network consisting of a network of caches while the third question deals with a network of spatially distributed services.

1.1.1 Can we provide an upper bound on the cache hit probability for non-anticipative caching policies?

This question is pertinent to a wide verity of distributed caching systems such as Content Deliver Networks (CDNs), Information and Content-centric Networks (ICNs/CCNs). In such caching systems, end-to-end application performance heavily depends on the fraction of contents transferred from the cache, also known as the *cache hit probability.* Many caching policies have been proposed and implemented to improve the hit probability. Thus the following natural question arises. *With limited statistical knowledge of the content arrival process and no look ahead option, can we provide an upper bound on the cache hit probability for any feasible non-anticipative caching policy?* To answer the above question, we propose a new method to compute an upper bound on hit probability for all non-anticipative caching policies, i.e., for policies that have no knowledge of future requests. Our key insight is to order the contents according to the ratio of their Hazard Rate (HR) function values to their sizes and place in the cache the contents with the largest ratios till the cache capacity is exhausted. Here, the HR function is the conditional density of the occurrence of a content request, given the realization of the request process [29]. Under fairly weak statistical assumptions, we prove that our proposed HR to size ratio based ordering model computes the maximum achievable hit probability and serves as an upper bound for all non-anticipative caching policies. We find our bound to be tighter than state-of-the-art upper bounds for a variety of content request arrival processes.

1.1.2 How do we build caching networks that provide differential services?

In modern caching systems, different content types have different quality of service requirements. Service differentiation among different content classes provide economic incentives along with technical gains. However, traditional caching policies are oblivious to these service requirements. This brings us to the following question. *How do we build caching networks that provide differential services?* To answer this question, we associate each content with a utility which is a function of the corresponding content hit rate. We model the caching network as a general graph $G = (V, E)$ where each vertex $v \in V$ represents a cache. We assume caches are *timer-based*, i.e., each cache $v \in V$ is a *Time-To-Live (TTL)* cache. Our main contributions can be summarized as follows.

- We formulate a cache utility maximization framework for a single cache under stationary request arrivals, where each content is associated with a utility and content is managed with a timer whose duration is set to maximize the aggregate utility for all contents. We propose computationally efficient online algorithms that adapt to different stationary requests using limited information.

- We further consider a more general cache network where each local edge cache can serve common contents, i.e., there are common contents among local edge caches. This introduces non-convex constraints, resulting in a non-convex utility maximization problem. We show that although the original problem is non-convex, the duality gap is zero. Based on this, we design a distributed iterative primal-dual algorithm for optimal content placement in the cache network.

- We apply the results obtained from our utility maximization framework to a wireless sensor network setting with the goal to jointly compress and cache requested content.

1.1.3 How do we allocate resources to users in a geographically dispersed network?

Consider another variation of a service network where both resources and end users are distributed across a physical space and the goal is to allocate user requests to resources on servers. Not surprisingly, the spatial distribution of resources and users in the network is an important factor in determining the overall performance of the service. A key measure of performance in such networks is *average request distance*, that is average distance between a user and its allocated resource/server (where distance is measured on the network). This directly translates to latency incurred by a user when accessing the service, which is arguably among the most important metrics in distributed service applications. Furthermore, in wireless networks, signal attenuation is strongly coupled to request distance, therefore developing allocation policies to minimize request distance can help reduce energy consumption, an important concern in battery-operated wireless networks.

Another important practical constraint in distributed service networks is the number of users assigned to a resource, or the *load*. One of design goals in such a system is to distribute users among distributed resources/servers as evenly as possible. While the optimal resource selection problem can be solved centrally, due to scalability concerns, it is often preferred to adopt randomized load balancing strategies to distribute these users among servers. This interpretation leads to formulating a randomized *load balancing* problem for the distributed systems with the goal to make the overall user-to-resource assignment as fair as possible. In a non-geographic setting, one of the widely used randomized load balancing policies is the *Power of Two* (POT) choices policy. In POT, each user uniformly at random selects two resources and gets allocated to the one with the least load. While POT can directly be used for user to resource assignment in a geographic setting, it is oblivious to the spatial distribution of resources and users.

Thus a fundamental research question that is pertinent to all geographically dispersed service network designs is : *How to assign users to suitable resources so as to minimize average request distance and the server load?* To answer this question, we consider three variations of the spatial network - (i) *one dimensional service network*: where users and resources are distributed over a one dimensional line. (ii) *two dimensional service network*: where users and resources are distributed over a two dimensional euclidean plane. (iii) *Generic Graph Network*: where resources are vertices of an arbitrary graph and user requests arrive on any of one of the vertices. Our contributions are summarized as follows.

- We develop and analyze simple allocation policies in a one dimensional service network with the goal of characterizing and minimizing average request distance.

    1. We show that when user and resource locations are modeled by statistical point processes, the spatial system under unidirectional policies (policies that allocate resources only to users located to their left) can be mapped into classical bulk service queueing systems. As we consider a case where different resources can satisfy different numbers of users, we also generate new results for bulk service queues.

    2. We propose a novel dynamic programming based algorithm for optimal bi-directional resource allocation that improves state-of-the-art time complexity by $O(n)$ with n being the number of resources.

- We study and develop spatial load balancing policies when resources are placed on a two dimensional Euclidean plane.

    1. We propose a spatially motivated POT policy: *spatial POT* (sPOT) in which each user is allocated to the least loaded resource among its two geographically nearest resources. When both servers and users are placed uniformly at random in the Euclidean plane, we map sPOT to a classical balls

and bins allocation policy with bins corresponding to the Voronoi regions associated with the second order Voronoi diagram of the set of servers. We show that sPOT performs better than classical POT in terms of average request distance. However, a lower bound analysis on the asymptotic expected maximum load for sPOT suggests that POT load balancing benefits are not achieved by sPOT.

    2. We propose two non-uniform resource sampling based POT policies to improve load and request distance behavior.

        (a) A candidate set based policy that samples k nearest servers from a user and applies POT on the candidate set.

        (b) A non-uniform distance decaying sampling based POT in which each user samples two servers with probability inversely proportional to square of the distance to the servers.

- We design proximity aware randomized load balancing policies when resources are placed on the vertices of an arbitrary graph.

    1. Our simulations demonstrate a total variation distance, a metric to determine closeness of two probability distributions, as low as $0.002 - 0.005$ between load distributions of classical POT and proposed proximity based policies across a wide range of network topologies.

    2. We achieve a significant reduction in implementation cost on the order of $20\% - 99\%$ for our proposed proximity based policies as compared to classical POT.

1.2 Thesis Outline

The remainder of the thesis is outlined as follows.

- Chapter 2 of this thesis details our proposed model to compute the maximum achievable cache hit probability for all non-anticipative caching policies. This model has been published in [77], [78].

- In Chapter 3, we formulate a utility-driven caching framework for general cache networks, which has also been developed in several papers [74], [76].

- Chapter 4 discusses an application of utility-driven caching framework in a wireless sensor network that was published in [75].

- Chapter 5 proposes resource allocation in a one dimensional distributed service network and has been published in [68], [72].

- Chapter 6 discusses resource allocation in a two dimensional distributed service network, which has also been published in [73].

- Chapter 7 develops proximity aware randomized load balancing policies for the case when resources are placed on the vertices of an arbitrary graph and has been developed in [79].

- Chapter 8 concludes this thesis and provides a list of ongoing and future work.

CHAPTER 2

A NEW UPPER BOUND ON CACHE HIT PROBABILITY FOR NON-ANTICIPATIVE CACHING POLICIES

2.1 Background

Caches are pervasive in computing systems, and their importance is reflected in many networks and distributed environments including *content delivery networks* (CDNs). In such networks, end user quality of experience primarily depends on whether the requested object[1] is cached near the user. Thus the cache *hit probability*, i.e., the percentage of requests satisfied by the cache, plays an important role in determining end-to-end application performance. In general, the number of objects available in a system is much larger than the cache capacity. Hence, design of caching algorithms typically focuses on maximizing the overall cache hit probability. Also, maximizing the cache hit probability corresponds to minimizing the expected retrieval time, the load on the server and on the network when object sizes are equal.

One possible way to improve cache hit probability is by increasing cache capacity. However, increasing cache capacity only logarithmically improves cache hit probability[22, 8]. Thus improving caching policies seems to be more effective for maximizing the overall cache hit probability. In practice, most caches employ *least-recently used* (LRU) or its variants often coupled with call admission or prefetching [19]. Apart from LRU, other well known eviction policies include LFU, FIFO, RANDOM. There has been plethora of work [52, 65, 90, 9, 16, 25, 51] on improving cache hit probabilities in the literature. In order to gauge the potential effectiveness of these

[1]We use object and content interchangeably.

eviction policies, an upper bound on maximum achievable cache hit probability for a given cache capacity has been widely adopted [9].

2.1.1 Offline upper bound

For equal size objects, Bélády's algorithm or MIN [6] has been widely used as an upper bound for cache hit probability among all feasible on demand and online caching policies, togetherly known as *non-anticipative* policies. However, Bélády's algorithm is an offline algorithm, i.e., it assumes exact knowledge of future requests. Offline upper bounds on object hit probability have been proposed for variable (different) size object [19]. Often system designers do not have access to the exact request trace, but can estimate the statistical properties of the object request process such as the *inter-request time* (irt) distribution. Also, caching studies typically include model driven simulations. Thus the following natural question arises. *With limited knowledge of the object arrival process and no look ahead option, can we provide an upper bound on the cache hit probability for any feasible non-anticipative caching policy?*

2.1.2 Our Approach: Hazard Rate based upper bound

When object requests follow the *Independent Reference Model* (IRM), i.e., when objects are referenced independently with fixed probabilities, Least-Frequently Used (LFU) caching policy is asymptotically optimal in terms of object hit probability. However, general request processes are more complex and correlated.

In this chapter, we assume a larger class of statistical models for object reference streams. We also assume that the *hazard rate* (HR) function (or conditional intensity) associated with this point process is well defined and can be computed at all points of time t. Here, the HR function is the conditional density of the occurrence of an object request at time t, given the realization of the request process over the interval $[0, t)$ [29].

Note that, the hazard rate function may have different interpretations depending on the context it is used. For example, in survival analysis, the hazard rate corresponds to the propensity or the likelihood of an item to die or fail conditioned on the fact that it has survived until a certain age. When the failure process of an item is described by a Poisson process, due to its memoryless property, the hazard rate is equivalent to the request rate for that item. In caching terminology, we can treat failure/death of an item as an object being requested.

We now propose the HR based upper bound as follows. When objects have equal size, at any time t we determine the HR values of each object and place in the cache the B objects which have the largest HR values. When objects have different sizes, we sort the objects according to the ratio of their HR values at time t to their sizes in decreasing order. Note that, an ideal LFU policy keeps track of number of times an object is referenced and order them accordingly in the cache. Similarly, in our upper bound, we keep an ordered list but on the basis of ratio of HR values to object sizes. We then place in the cache the objects with the largest ratios till the cache capacity is exhausted.

Our contributions are summarized below:

1. We present a new upper bound for cache hit probability among all non-anticipative caching policies that allow prefetching:

    - When objects have equal sizes, a simple HR based ordering for the objects provides an upper bound on cache hit probability.

    - For variable size objects, we order the objects with respect to the ratio of their HR function values to their objects sizes and provide upper bounds on the byte and object hit probabilities.

2. We derive closed form expressions for the upper bound under some specific object request arrival processes.

3. We evaluate and compare the HR based upper bound with different cache replacement policies for both synthetic and real world traces.

2.2 Equal Size Objects

We consider a cache of capacity B serving requests for n distinct equal size objects. Without loss of generality we assume that all objects have size one. Later in Section 2.3, we also consider objects with different sizes. Let $\mathcal{D} = \{1, \ldots, n\}$ be the set of objects, with $n > B$.

2.2.1 Number of Hits for General object Arrival Processes

Let $\{0 < T_{i,1} < T_{i,2} < \cdots\}$ denote the successive time epochs when object i is requested. Assume $\{T_{i,k}\}_k$ is a regular point process, that is it possesses an intensity function [29, Definition 7.1.I., p. 213]. Define $X_{i,k} = T_{i,k} - T_{i,k-1}$ for $k \geq 2$ and $X_{i,1} = T_{i,1}$. For $t > 0$, define $\mathcal{H}_{i,t} = \{T_{i,k}, k \geq 1 : T_{i,k} < t\}$ the history of the point process $\{T_{i,k}\}_k$ in $[0, t)$.

Let $\{0 < T_1 < T_2 < \cdots\}$ be the point process resulting from the superposition of the point processes $\{T_{i,k}\}_k$, $i = 1, \ldots, n$. Call $R_k \in \{1, \ldots, n\}$ the object requested at time T_k. Define $\mathcal{H}_t = \{(T_k, R_k), k \geq 1 : T_k < t\}$, the history of point processes $\{T_{1,k}\}_k, \ldots, \{T_{n,k}\}_k$ in $[0, t)$. Notice that $\mathcal{H}_{i,t}$ is right-continuous for all i and so is $\mathcal{H}_t$. In particular, $T_k \notin \mathcal{H}_{T_k}$ for all k.

Define $k_i(t) = \max\{k \geq 1 : T_{i,k-1} < t\}$, so that exactly $k_i(t) - 1$ requests for object i have been made in $[0, t)$.

Assume that the request object processes $\{T_{1,k}\}_k, \ldots, \{T_{n,k}\}_k$ are conditionally independent $\forall t > 0$, in the sense that

$$\mathbb{P}(T_{1,k_1(t)} \geq t_1, \cdots, T_{n,k_n(t)} \geq t_n \,|\, \mathcal{H}_t) = \prod_{i=1}^{n} \mathbb{P}(T_{i,k_i(t)} \geq t_i \,|\, \mathcal{H}_{i,t}), \tag{2.1}$$

for all $t_1 \geq t, \ldots, t_n \geq t$.

Given $T_{i,k} = t_{i,k}$ for $k \geq 1$, the hazard rate function of $\{T_{i,k}\}_k$ at time t is defined by the piecewise function [29, Definition 7.2.II, p. 231]

$$\lambda_i^*(t) = \begin{cases} \frac{\frac{d}{dt}P(X_{i,1}<t)}{P(X_{i,1}>t)} & \text{for } 0 < t \leq t_{i,1}, \\ \frac{\frac{d}{dt}P(X_{i,k}<t-t_{i,k-1}\,|\,T_{i,j}=t_{i,j},j\leq k-1)}{P(X_{i,k}>t-t_{i,k-1}\,|\,T_{i,j}=t_{i,j},j\leq k-1)} & \text{for } t_{i,k-1} < t \leq t_{i,k},\ k \geq 2. \end{cases} \tag{2.2}$$

In (2.2) the existence of $\frac{d}{dt}P(X_{i,1} < t)$ and $\frac{d}{dt}P(X_{i,k} < t - t_{i,k-1}\,|\,T_{i,j} = t_{i,j}, j \leq k-1)$ for $k \geq 2$, follows from the assumption that $\{T_{i,k}\}_k$ is a regular point process [29, Definition 7.1.I., p. 213].

We assume that the cache is empty at time $t = 0$ to avoid unnecessary notational complexity but all results in the chapter hold without this assumption as long as the probability distribution of the state of the cache is known at time $t = 0$. A caching policy π determines at any time t which B objects among the n available objects are cached. Formally, π is a measurable deterministic mapping from $\mathbb{R} \times (\mathbb{R} \times \{1, \ldots, n\})^\infty \to S_B(n)$, where $S_B(n)$ is the set of subsets of $\{1, \ldots, n\}$ that contain B elements. In this setting, $\pi(t, \mathcal{H}_t)$ gives the B objects that are cached at time t based on the knowledge of the overall request process up to t. Let Π be the collection of all such policies. Note that policies in Π are *non-anticipative*, in that they do not know when future requests will occur.

We will only consider deterministic policies although the setting can easily be extended to random policies (in this case $\pi : \mathbb{R} \times (\mathbb{R} \times \{1, \ldots, n\})^\infty \to \mathcal{Q}(S_B(n))$, where $\mathcal{Q}(S_B(n))$ is the set of probability distributions on $S_B(n)$).

We introduce the hazard rate based *rule* for equal-size objects, abbreviated to HR-E. At any time t and given $\mathcal{H}_t$, HR-E (i) determines the hazard rate function of each object and (ii) places in the cache the B documents which have the largest hazard rates, i.e., if $\lambda_{i_1}^*(t) \geq \cdots \geq \lambda_{i_n}^*(t)$ then objects $i_1, \ldots, i_B$ are cached at time t (ties between equal rates are broken randomly). We call it a rule, not a policy and will use it as a way to upper-bound various performance metrics (see next)—which is the goal of this chapter—regardless of whether it can be implemented.

Let $B_k^\pi \in S_B(n)$ be the state of the cache just before time T_k under π, and define

$$H_k^\pi = \mathbb{1}(R_k \in B_k^\pi), \tag{2.3}$$

i.e., $H_k^\pi = 1$ if the k-th requested object is in the cache under rule π, and $H_k^\pi = 0$ otherwise. Denote by

$$N_K^\pi = \sum_{k=1}^{K} H_k^\pi, \tag{2.4}$$

the number of hits during the first K requests for an object.

The following theorem holds,

Theorem 1 (Expected number of hits)**.**

$$\mathbb{E}\left[N_K^{HR-E}\right] \geq \mathbb{E}\left[N_K^\pi\right], \quad \forall \pi \in \Pi,\ \forall K \geq 1. \tag{2.5}$$

Proof. Fix $\pi \in \Pi$. Given that a request for an object is made at time t and given that the history $\mathcal{H}_t$ is known, this request is for object i with probability

$$p_i(t) = \frac{\lambda_i^*(t)}{\sum_{j=1}^n \lambda_j^*(t)}. \tag{2.6}$$

Proof of (2.6) is given in Appendix A.1. This result relies on the conditional independence of point processes $\{T_{1,k}\}_k, \ldots, \{T_{1,k}\}_k$, expressed in (2.1). By definition of HR-E,

$$\sum_{i \in B_k^{HR-E}} \lambda_i^*(T_k) \geq \sum_{i \in B_k^\pi} \lambda_i^*(T_k), \quad \forall k \geq 1. \tag{2.7}$$

Therefore, for $k \geq 1$,

$$\begin{aligned}
\mathbb{E}\left[H_k^{HR-E} \mid \mathcal{H}_{T_k}, T_k\right] &= \mathbb{P}\left(R_k \in B_k^{HR-E} \mid \mathcal{H}_{T_k}, T_k\right) = \sum_{i=1}^{n} \mathbb{P}(R_k \in B_k^{HR-E} \mid R_k = i) p_i(T_k) \\
&= \sum_{i=1}^{n} \mathbb{1}(i \in B_k^{HR-E}) \frac{\lambda_i^*(T_k)}{\sum_{j=1}^{n} \lambda_j^*(T_k)} \quad \text{from (2.6)}, \\
&= \sum_{i \in B_k^{HR-E}} \frac{\lambda_i^*(T_k)}{\sum_{j=1}^{n} \lambda_j^*(T_k)} \geq \sum_{i \in B_k^{\pi}} \frac{\lambda_i^*(T_k)}{\sum_{j=1}^{n} \lambda_j^*(T_k)} \quad \text{from (2.7)}, \\
&= \mathbb{E}\left[H_k^{\pi} \mid \mathcal{H}_{T_k}, T_k\right]. \qquad (2.8)
\end{aligned}$$

Taking expectation on both sides of (2.8) to remove the conditioning yields $\mathbb{E}\left[H_k^{HR-E}\right] \leq \mathbb{E}\left[H_k^{\pi}\right]$. Summing both sides of the latter inequality for $k = 1, \ldots, K$ gives (2.5) from the definition of N_K^{π}. □

It is worth noting that Theorem 1 holds for any non-stationary request object processes. We now study a more specific request arrival process and derive an upper bound on the object hit probability.

2.2.2 Upper Bound on Stationary Hit Probability

We still assume that $\{T_{1,k}\}_k, \ldots, \{T_{n,k}\}_k$ are regular processes and that the conditional independence assumption (2.1) holds. We define the stationary hit probability of any policy $\pi \in \Pi$ as

$$h^{\pi} = \lim_{K \to \infty} \frac{1}{K} \sum_{k=1}^{K} H_k^{\pi} \quad \text{a.s.}, \qquad (2.9)$$

whenever this limit exists, where H_k^{π} is defined in (2.3).

We are now in position to state and prove the main result of the chapter.

Theorem 2 (Stationary hit probability).

For any $\pi \in \Pi$, assume that the limit in (2.9) exists and that h^{π} is a constant. Then,

$$h^{HR-E} \geq \max_{\pi \in \Pi} h^{\pi}.$$

Proof. Let $\pi \in \Pi$. Taking the expectation on both sides of (2.9), using the fact that h^π is a constant and then invoking Lebesgue's dominated convergence theorem gives

$$h^\pi = \mathbb{E}[h^\pi] = \mathbb{E}\left[\lim_{K\to\infty} \frac{1}{K}\sum_{k=1}^{K} H_k^\pi\right] = \lim_{K\to\infty} \frac{1}{K}\sum_{k=1}^{K} \mathbb{E}[H_k^\pi] = \lim_{K\to\infty} \frac{\mathbb{E}[N_K^\pi]}{K},$$

by using (2.4). The proof is concluded by using Theorem 1. $\square$

Let us now discuss the existence of the limit in (2.9). When the inter-request time sequences $\{X_{1,k}\}_k, \ldots, \{X_{n,k}\}_k$ are stationary, ergodic and mutually independent, the sequence $\{(X_k, R_k)\}_k$ is stationary[2] and ergodic (see e.g., [13, pp. 33-34]). The latter result coupled with the fact that, for any $\pi \in \Pi$, there exists a measurable mapping $\varphi^\pi : (\mathbb{R}\times\{1,\ldots,n\})^\infty \to \{0,1\}$ such that $H_k^\pi = \varphi^\pi((T_j, R_j), j \le k-1)$ (H_k^π is defined in (2.3)), shows that the sequence $\{H_k^\pi, k \in \mathbb{Z}\}$ is stationary and ergodic (e.g., see [82, Thm p. 62]). The ergodic theorem then ensures the existence of the limit in (2.9) and that (see e.g., [57, Thm 1])

$$h^\pi = \mathbb{P}(H_k^\pi), \quad \forall \pi \in \Pi.$$

Other instances when the limit in (2.9) exists and is a constant are discussed in Section 2.4.

2.3 Variable Size Objects

We now assume object i has size $s_i \in \mathbb{R}^+$ for all $i \in \mathcal{D}$ and the capacity of the cache is B bytes.

[2]Let X_i be a random variable with the same distribution as that of the stationary inter-request time for object i. The stationary version of the inter-request times for object i is the sequence $\{X_{i,k}\}_k$ where $P(X_{1,k} < x) = 1/\mathbb{E}[X_i]\int_0^x \mathbb{P}(X_i > u)du$ and $X_{i,k} =_{\text{st}} X_i$ for $k \ge 2$.

2.3.1 Number of byte hits and fractional knapsack problem

The setting and assumptions are that of Section 2.2.1 but *Fractional Caching* (FC)[3] is now allowed, i.e., we allow to store a fraction of an object in the cache instead of the whole ones. We refer interested reader to [63], [80] for more details. We denote by Π_{FC} the set of all FC policies.

For $\pi \in \Pi_{FC}$, let V_k^π denote the number of bytes served from the cache at the kth request for an object. Let $x_{i,k}$ denote the fraction of object i in the cache at the time of the k-th request. Then $V_k^\pi = s_i x_{i,k}^\pi$ if the request is for object i. Let $W_K^\pi = \sum_{k=1}^K V_k^\pi$ denote the total number of bytes served from the cache during the first K requests for an object.

Given a request for an object is made at time t and that the history $\mathcal{H}_t$ is known, we have already observed (see (2.6)) that this request is for object i with the probability $\lambda_i^*(t)/\sum_{j=1}^n \lambda_j^*(t)$. Therefore,

$$\mathbb{E}[V_k^\pi \,|\, \mathcal{H}_{T_k}] = \sum_{i=1}^n \mathbb{E}[V_k^\pi \,|\, \mathcal{H}_{T_k}, \text{object } i \text{ is requested}] \times \frac{\lambda_i^*(T_k)}{\sum_{j=1}^n \lambda_j^*(T_k)} = \frac{\sum_{i=1}^n s_i x_{i,k}^\pi \lambda_i^*(T_k)}{\sum_{j=1}^n \lambda_j^*(T_k)}. \tag{2.10}$$

Our goal is to find $\pi \in \Pi_{FC}$ that maximizes $\mathbb{E}[V_k^\pi \,|\, \mathcal{H}_{T_k}]$ subject to the capacity constraint on the size of the cache. This can be done by solving the optimization problem,

$$\begin{aligned} \max \quad & \sum_{i=1}^n s_i x_i \lambda_i^*(t) \\ \text{subject to} \quad & \sum_{i=1}^n s_i x_i \leq B \\ & 0 \leq x_i \leq 1, \quad i = 1, \cdots, n, \end{aligned}$$

[3]Fractional caching has its applications in large video object delivery systems where objects are composed of chunks stored independently.

which is nothing but the Fractional Knapsack Problem (FKP) [44, Chapter 5.1]. It is well known that its solution depends on the respective values of the ratios $s_i \lambda_i^*(t)/s_i = \lambda_i^*(t)$ for $i = 1, \ldots, n$. More specifically, assume that

$$\lambda_{i_1}^*(t) \geq \lambda_{i_2}^*(t) \geq \cdots \geq \lambda_{i_n}^*(t),$$

Then, the solution of (2.11) is given by $x_{i_j} = 1$ for $1 \leq j \leq a := \max\{a : s_{i_1} + \cdots + s_{i_a} \leq B\}$, $x_{i_{a+1}} = (B - s_{i_1} - s_{i_2} - \cdots - s_{i_a})/s_{i_{a+1}}$, and $x_{i_j} = 0$ for $j = a+2, \ldots, n$.

Call HR-VB the rule which at any time t places *entirely* in the cache objects with the highest hazard rates until an object cannot fit; if object k is the first one that cannot entirely fit in the cache and objects $i_1, \ldots, i_j$ are already in the cache, then $B - \sum_{l=1}^{j} s_l$ bytes of object k are cached. All other objects are not cached. Then, by (2.10), for any policy $\pi \in \Pi_{FC}$,

$$\mathbb{E}[V_k^{HR-VB} \,|\, \mathcal{H}_{T_k}] \geq \frac{\sum_{i=1}^{n} s_i x_{i,k}^{\pi} \lambda_i^*(T_k)}{\sum_{j=1}^{n} \lambda_j^*(T_k)} = \mathbb{E}[V_k^{\pi} \,|\, \mathcal{H}_{T_k}].$$

Removing the conditioning on $\mathcal{H}_{T_k}$ yields

$$\mathbb{E}[V_k^{HR-VB}] \geq \mathbb{E}[V_k^{\pi}] \tag{2.12}$$

Summing both sides of this inequality for $k = 1, \ldots, K$ gives

$$\mathbb{E}[W_K^{HR-VB}] \geq \mathbb{E}[W_K^{\pi}].$$

2.3.2 Upper Bound on the Byte Hit Probability for Stationary and Ergodic Object Arrival Processes

Throughout this section π is fixed in Π_{FC}, the set of fractional caching policies. We assume that object request arrival processes are stationary, ergodic, and independent processes, so that the joint sequence of request arrival times and requested objects $\{(T_k, R_k)\}_k$ is stationary and ergodic (setting of Section 2.2.2). Recall that V_k^π is the number of bytes served by the cache at the kth request for an object. We denote by $\sigma_k = \sum_{i=1}^n s_i \mathbb{1}(R_k = i)$ the number of bytes requested from the cache at the kth request.

Define the stationary byte hit probability as

$$\widehat{h}^\pi = \lim_{K\to\infty} \frac{\sum_{k=1}^K V_k^\pi}{\sum_{k=1}^K \sigma_k}, \quad \text{a.s.},$$

whenever this limit exists. $\widehat{h}^\pi$ can be rewritten as

$$\widehat{h}^\pi = \lim_{K\to\infty} \frac{1}{K}\sum_{k=1}^K V_k^\pi \times \frac{1}{\lim_{K\to\infty} \frac{1}{K}\sum_{k=1}^K \sigma_k}, \quad \text{a.s.}$$

The sequence $\{R_k\}_k$ being stationary and ergodic the sequence $\{\sigma_k\}_k$ enjoys the same properties, and $\lim_{K\to\infty} \frac{1}{K}\sum_{k=1}^K \sigma_k = \mathbb{E}[\sigma]$ a.s., where σ a rv with the same distribution as σ_k. On the other hand, there exists a mapping $\psi^\pi : (\mathbb{R}\times\{1,\ldots,n\})^\infty \to [0,\infty)$ such that $V_k^\pi = \psi^\pi((T_j, R_j), j \le k)$, which shows that the sequence $\{V_k^\pi\}_k$ is stationary and ergodic [82, Thm p. 62]. Hence, $\lim_{K\to\infty} \frac{1}{K}\sum_{k=1}^K V_k^\pi = \mathbb{E}[V^\pi]$ a.s., with V^π a rv with the same distribution as V_k^π. Therefore,

$$\widehat{h}^{HR-VB} = \frac{\mathbb{E}[V^{HR-VB}]}{\mathbb{E}[\sigma]} \ge \frac{\mathbb{E}[V^\pi]}{\mathbb{E}[\sigma]} = \widehat{h}^\pi,$$

where the inequality follows from (2.12) since under the assumptions in Section 2.2.2 inequality (2.12) becomes $E[V^{HR-VB}] \ge \mathbb{E}[V^\pi]$ since the sequence $\{V_k^\pi\}_k$ is stationary and ergodic for all $\pi \in \Pi_{FC}$.

2.3.3 Number of object hits and 0-1 knapsack problem

The setting and assumptions are still those of Section 2.2.1 but we now assume that objects are indivisible (IC). In particular, every object hit counts the same (*i.e.*, a hit for a large 1GB object and hit for a small 10B object both count as a "hit"). Denote by Π_{IC} the set of all IC policies. Recall the definition of H_k^π (1 if hit at T_k and 0 otherwise) and N_K^π (number of hits in the first K requests) under $\pi \in \Pi_{IC}$.

Fix $\pi \in \Pi_{IC}$. We have by using (2.6)

$$\begin{aligned}\mathbb{E}[H_k^\pi \,|\, \mathcal{H}_{T_k}] &= \sum_{i=1}^n \mathbb{E}[H_k^\pi \,|\, \mathcal{H}_{T_k}, \text{object } i \text{ is requested at } T_k] \times \frac{\lambda_i^*(T_k)}{\sum_{j=1}^n \lambda_j^*(T_k)} \\ &= \frac{1}{\sum_{j=1}^n \lambda_j^*(T_k)} \sum_{i=1}^n \mathbf{1}(i \in B_k^\pi)\lambda_i^*(T_k),\end{aligned}$$

where we recall that B_k^π is the set of objects in the cache just before T_k under π.

Hence, $\mathbb{E}[H_k^\pi \,|\, \mathcal{H}_{T_k}]$ can be maximized by solving the following 0-1 knapsack problem (KP),

$$\begin{aligned}\max \quad & \sum_{i=1}^n x_i \lambda_i^*(t) \\ \text{subject to} \quad & \sum_{i=1}^n s_i x_i \leq B \\ & x_i \in \{0,1\}, \quad i = 1, \cdots, n. \end{aligned} \tag{2.13a}$$

Solving KP is NP-hard. However, the solution to the corresponding relaxed problem where the constraints in (2.13a) are replaced by $x_i \in [0,1]$ for $i = 1, \ldots, n$, serves as an upper bound for $\sum_{i=1}^n x_i \lambda_i^*(t)$. The latter is achieved if the ratios $\{\lambda_i^*(t)/s_i\}_i$ are arranged in decreasing order, say $\lambda_{i_1}^*(t)/s_{i_1} \geq \cdots \geq \lambda_{i_n}^*(t)/s_{i_n}$ and $x_{i_j} = 1$ for $1 \leq j \leq a$ where a is defined in Section 2.3.1, $x_{i_{a+1}} = (B - s_{i_1} - s_{i_2} - \cdots - s_{i_a})/s_{i_{a+1}}$, and $x_{i_j} = 0$ for $j > a+1$ [44, Chapter 5.1].

Call HR-VC the rule which, at any time t, places in the cache the objects in decreasing order of the ratios $\{\lambda_i^*(t)/s_i\}_i$ until an object does not fit in the cache; if object i_{a+1} is the first one that cannot entirely fit in the cache and objects $i_1, \ldots, i_a$ are already in the cache, then with probability $p_{i_{a+1}} = (B - s_{i_1} - s_{i_2} - \cdots - s_{i_a})/s_{i_{a+1}}$

object i_{a+1} is cached. All subsequent objects according to this decreasing ordering are not cached. Note that HR-VC does not meet the cache size constraint as there is not enough room in the cache to fit entirely object i_{a+1}. However, as mentioned in Section 2.2, our goal is to upper bound $\mathbb{E}[H_k^{\pi}]$ and $\mathbb{E}[N_k^{\pi}]$. Let $x^* = (x_1^*, \dots, x_n^*)$ be the solution of (2.13). We have

$$\begin{aligned}\mathbb{E}[H_k^{HR-VC} \,|\, \mathcal{H}_{T_k}] &= \frac{1}{\sum_{j=1}^n \lambda_j^*(T_k)} \left[\sum_{j=1}^{a_k} \lambda_{i_j}^*(T_k) + p_{i_{a_k+1}} \lambda_{i_{a_k+1}}^*(T_k)\right] \\ &\geq \frac{1}{\sum_{j=1}^n \lambda_j^*(T_k)} \sum_{i=1}^n x_i^* \lambda_i^*(t) \\ &\geq \frac{1}{\sum_{j=1}^n \lambda_j^*(T_k)} \sum_{i=1}^n \mathbf{1}(i \in B_k^{\pi}) \lambda_i^*(T_k) = \mathbb{E}[H_k^{\pi} \,|\, \mathcal{H}_{T_k}],\end{aligned}$$

where a_k is the last job that can be entirely cached at time T_k- according to the decreasing ordering of the ratios $\{\lambda_i^*(T_k)/s_i\}_i$. Removing the conditioning on $\mathcal{H}_{T_k}$ yields $\mathbb{E}[H_k^{HR-VC}] \geq \mathbb{E}[H_k^{\pi}]$. Summing both sides of this inequality for $k = 1, \dots, K$ gives

$$\mathbb{E}[N_K^{HR-VC}] \geq \mathbb{E}[N_K^{\pi}]. \tag{2.14}$$

2.3.4 Upper Bound on Object Hit Probability for Stationary and Ergodic Object Arrival Processes

Using similar arguments as discussed in Section 2.2.2, one can define the object hit probability under HR-VC (cf. Section 2.3.3) as

$$h^{HR-VC} = \lim_{K\to\infty} \frac{1}{K} \sum_{k=0}^{K-1} \mathbb{E}[H_k^{HR-VC}] = \lim_{K\to\infty} \frac{\mathbb{E}[N_K^{HR-VC}]}{K}, \tag{2.15}$$

Now combining (2.14) and (2.15) we obtain

$$h^{\pi} = \lim_{K\to\infty} \frac{\mathbb{E}[N_K^{\pi}]}{K} \leq \lim_{K\to\infty} \frac{\mathbb{E}[N_K^{HR-VC}]}{K} = h^{HR-VC}.$$

2.4 Specific Request Arrival Processes

Below we consider four specific request processes for equal size objects.

2.4.1 Poisson Process

We consider the case where successive requests to object i ($i = 1, \dots, n$) occur according to a Poisson process with rate $\lambda_i > 0$ and these n Poisson processes are mutually independent. This is the standard Independence Reference Model (see Section 2.1.2) where references to all objects are independent rvs. Without loss of generality assume that $\lambda_1 \geq \cdots \geq \lambda_n$.

Under HR-E (see Section 2.2.1) objects $1, \dots, B$ are in the cache at all times. Therefore, the hit probability h_i^{HR-E} for object i is $h_i^{\text{HR-E}} = \mathbf{1}(i \leq B)$ and the hit rate $r_i^{\text{HR-E}}$ for object i is $r_i^{\text{HR-E}} = \lambda_i \mathbf{1}(i \leq B)$. The overall hit probability $h^{\text{HR-E}}$ and hit rate r^{HR-E} are given by

$$h^{\text{HR-E}} = \frac{1}{\Lambda} \sum_{i=1}^{B} \lambda_i, \quad r^{\text{HR-E}} = \sum_{i=1}^{B} \lambda_i,$$

where $\Lambda := \sum_{i=1}^{n} \lambda_i$.

2.4.2 On-Off Request Process

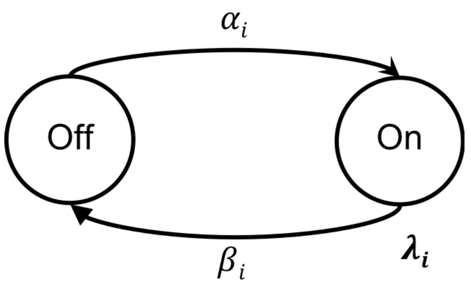


Figure 2.1: On-Off Request Process

The object popularity dynamics in caching systems can often be captured by using a stationary, on-off traffic model [41]. Specifically, we assume that successive requests to object i occur according to a Poisson process with rate $\lambda_i > 0$ when the underlying on-off process depicted in Figure 2.1 is in state 1 ($X_i = 1$) and that no request occurs when this process is in state 0 ($X_i = 0$). The stationary distribution of this on-off process is given by $\boldsymbol{\pi}_i := [\pi_{i,0}, \pi_{i,1}] = [\beta_i/(\alpha_i+\beta_i), \alpha_i/(\alpha_i+\beta_i)]$. We assume that these n on-off processes are mutually independent. Without loss of generality, assume that $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_n$ and define $\Lambda = \sum_{i=1}^{n} \lambda_i$. Below, we derive expressions for the hit rate and hit probability under HR-E.

Due to the way HR-E behaves, we can assume without loss of generality that object i is never in the cache when $X_i = 0$. With this, at any time at most B of the most popular objects are in the cache among all objects whose associated on-off process is in state 1. Therefore, if $i > B$ the hit probability $h_i^{\text{HR-E}}$ for object i is given by

$$\begin{aligned} h_i^{\text{HR-E}} &= \mathbb{P}(\text{at most } B-1 \text{ on-off processes are in state 1 among on-off processes } 1,\ldots,i-1) \\ &= \sum_{k=0}^{B-1} \mathbb{P}(\text{exactly } k \text{ on-off processes are in state 1 among on-off processes } 1,\ldots,i-1) \\ &= \sum_{k=0}^{B-1} \sum_{\substack{i_1,\ldots,i_k \in \{1,2,\ldots,i-1\} \\ i_1 < i_2 < \cdots < i_k}} \prod_{l=1}^{k} \pi_{i_l,1} \prod_{m \in \{1,\ldots,i-1\} \setminus \{i_1,\ldots,i_k\}} \pi_{i_m,0}, \end{aligned}$$

and $h_i^{\text{HR-E}} = 1$ if $i \leq B$. The hit rate $r_i^{\text{HR-E}}$ for object i is $r_i^{\text{HR-E}} = \lambda_i \pi_{i,1} h_i^{\text{HR-E}}$.

The overall hit probability $h^{\text{HR-E}}$ and the overall hit rate $r^{\text{HR-E}}$ are given by

$$h^{\text{HR-E}} = \sum_{i=1}^{n} \frac{\lambda_i \pi_{i,1}}{\sum_{j=1}^{n} \lambda_j \pi_{j,1}} h_i^{\text{HR-E}} \quad \text{and} \quad r^{\text{HR-E}} = h^{\text{HR-E}} \sum_{i=1}^{n} \lambda_i \pi_{i,1}. \tag{2.16}$$

Assume that $\pi_{i,1} = \rho$ for all i. This occurs, for instance, if all n on-off processes have the same transition rates with $\alpha_i = \alpha$ and $\beta_i = \beta$ or if $\alpha_i = \alpha\theta_i$ and $\beta_i = \beta\theta_i$ for all i. Then, for $i > B$,

$$h_i^{\text{HR-E}} = \sum_{k=0}^{B-1} \sum_{\substack{i_1,\ldots,i_k\in\{1,2,\ldots,i-1\} \\ i_1<i_2<\cdots<i_k}} \rho^k(1-\rho)^{i-1-k} = (1-\rho)^{i-1}\sum_{k=0}^{B-1}\left(\frac{\rho}{1-\rho}\right)^k \binom{i-1}{k},$$

so that

$$h^{\text{HR-E}} = \frac{\sum_{i=1}^{B}\lambda_i}{\Lambda} + \frac{1}{\Lambda}\sum_{i=B+1}^{n}\lambda_i(1-\rho)^{i-1}\sum_{k=0}^{B-1}\left(\frac{\rho}{1-\rho}\right)^k\binom{i-1}{k},$$

and $r^{\text{HR-E}} = \rho\sum_{i=1}^{B}\lambda_i + \rho\sum_{i=B+1}^{n}\lambda_i(1-\rho)^{i-1}\sum_{k=0}^{B-1}\left(\frac{\rho}{1-\rho}\right)^k\binom{i-1}{k}$.

We now propose a recursive approach for computing the hit probability and the hit rate with a much lower computational complexity than the general formulas in (2.16).

The recursions are based on available objects in the catalog, starting from the situation where only object 1 is available, moving to the situation where objects 1 and 2 are available, etc. up to the final situation where all n objects are available. Introduce the following variables,

$$p_{l,k} = \mathbb{P}[\text{cache occupancy is } k \,|\, \text{catalog is composed of the } l \text{ most popular objects}],$$
$$r_{l,k} = \text{Hit rate when cache occupancy is } k \text{ given catalog has the } l \text{ most popular objects }.$$

When $l = 1$ then $p_{1,0} = \pi_{1,0}$, $p_{1,1} = \pi_{1,1}$, $r_{1,0} = 0$, and $r_{1,1} = \lambda_1$ from our convention that object 1 is not in the cache when $X_1 = 0$. It is easy to verify that under HR-E the following recursions hold true for the occupancy probabilities,

$$\begin{aligned}
p_{l,0} &= p_{l-1,0}\pi_{l,0}, \quad l = 2,\cdots,n,\\
p_{k,k} &= p_{k-1,k-1}\pi_{k,1}, \quad k = 1,\ldots,B,\\
p_{l,k} &= p_{l-1,k-1}\pi_{l,1} + p_{l-1,k,}\pi_{l,0}, \quad 0 < k < \min(l,B), l = 1,\cdots,n,\\
p_{l,B} &= p_{l-1,B-1}\pi_{l,1} + p_{l-1,B}, \quad l = B+1,\cdots,n.
\end{aligned}$$

Similarly, the following recursions hold true for the hit rates,

$$
\begin{aligned}
r_{l,0} &= 0, \quad l = 2, \cdots, n, \\
r_{k,k} &= r_{k-1,k-1} + \lambda_k, \quad k = 2, \dots B, \\
r_{l,k} &= \frac{p_{l-1,k-1}\pi_{l,1}(r_{l-1,k-1} + \lambda_l) + p_{l-1,k}\pi_{l,0}r_{l-1,k}}{p_{l-1,k-1}\pi_{l,1} + p_{l-1,k}\pi_{l,0}}, \quad 0 < k < \min(l, B), l = 1, \cdots, n, \\
r_{l,B} &= \frac{p_{l-1,B-1}\pi_{l,1}(r_{l-1,B-1} + \lambda_l) + p_{l-1,B}r_{l-1,B}}{p_{l-1,B-1}\pi_{l,1} + p_{l-1,B}}, \quad l = B + 1, \cdots, n.
\end{aligned}
$$

Once the above recursions have been solved, the overall hit rate r^H and hit probability $h^{\text{HR-E}}$ under HR-E are given by

$$
r^{\text{HR-E}} = \sum_{k=1}^{B} p_{n,k} r_{n,k} \quad \text{and} \quad h^{\text{HR-E}} = \frac{r^{\text{HR-E}}}{\sum_{l=1}^{n} \lambda_l \pi_{l,1}}.
$$

2.4.3 Markov Modulated Poisson Process

Recall that $0 < T_{i,1} < T_{i,2} < \cdots < T_{i,k} < \cdots$ are the successive times when object i is requested. Let $\mathbf{Z} = \{Z(t), t \geq 0\}$ be a stochastic process taking values in a denumerable set $\mathcal{E}$.

Throughout we assume that, given $Z(t) = x$, the object request processes $\{T_{1,k}\}_k, \dots, \{T_{n,k}\}_k$ behave as independent Poisson processes with rate $\lambda_1(x), \dots, \lambda_n(x)$, respectively, until the next jump of the process $\mathbf{Z}$. Notice that this setting deviates from the setting in Section 2.2 as requests to different objects are now no longer independent.

We consider cache eviction policies that may know the state of the environment at any time. Under the HR-E rule, if $Z(t) = x$ then the B objects with the largest arrival rates $\{\lambda_i(x)\}_i$ are stored in the cache at time t. The following lemma is the equivalent of Lemma 1 in Section 2.2.1.

Lemma 1. *For any policy $\pi \in \Pi$,*

$$
\mathbb{E}[N_K^{\text{HR}-\text{E}}] \geq \mathbb{E}[N_K^{\pi}], \quad \forall K \geq 1.
$$

Proof. Proof is given in Appendix A.1.1. ∎

We further assume that $\mathbf{Z}$ is an ergodic Markov process, so that the request processes are correlated Poisson processes. Denote by $(\theta(x), x \in \mathcal{E})$ the stationary distribution of $\mathbf{Z}$.

From now on the rule HR-E will be denoted by HR-MMPP to reflect the nature of the request processes.

The next result is the equivalent of Theorem 2 in Section 2.2.2. It shows that the HR-MMPP rule gives an upper bound for the stationary hit probability under any policy in Π.

Theorem 3. *For any policy $\pi \in \Pi$,*

$$h^{\mathrm{HR-MMPP}} \geq h^{\pi}.$$

Proof. Under the assumptions made on the environment process (Markov process $\mathbf{Z}$) and the request processes (conditionally independent Poisson processes, modulated by $\mathbf{Z}$) the sequence $\{H_k^{\pi}\}_k$ (see (2.3)) is a stationary and ergodic sequence. The proof is then the same as that of Theorem 2 in Section 2.2.2. ∎

Let us now calculate $h^{HR-MMPP}$. To avoid unnecessary complications, we assume from now we assume that the set $\mathcal{E}$ is finite.

Define the set $I(x)$ by $I(x) = \{i_1, \ldots, i_B\}$ if $\lambda_{i_1}(x) \geq \ldots \geq \lambda_{i_n}(x)$ where $i_1, \ldots, i_n$ is a permutation of $1, \ldots, n$ (if two or more objects have the same rate ties are broken randomly).

Define $\gamma(x) = \lim_{k\to\infty} \mathbb{P}(Z(T_k) = x)$, the stationary probability that the Markov process $\mathbf{Z}$ is in state x when a request for an object is made; for the time being assume that this limit exists.

We have (cf. Section 2.2.1),

$$
\begin{aligned}
h^{HR-MMPP} &= \lim_{k\to\infty} \mathbb{P}(R_k \in B_k^{HR-MMPP}) \\
&= \lim_{k\to\infty} \sum_{x\in\mathcal{E}} \sum_{i=1}^{n} \mathbb{P}(R_k \in B_k^{HR-MMPP} \mid R_k = i, Z(T_k) = x)\mathbb{P}(R_k = i \mid X(T_k) = x)\mathbb{P}(Z(T_k) = x) \\
&= \lim_{k\to\infty} \sum_{x\in\mathcal{E}} \mathbb{P}(Z(T_k) = x) \sum_{i=1}^{n} \frac{\lambda_i(x)}{\sum_{j=1}^{n} \lambda_j(x)} \mathbb{P}(R_k \in B_k^{HR-MMPP} \mid R_k = i, Z(T_k) = x) \\
&= \lim_{k\to\infty} \sum_{x\in\mathcal{E}} \mathbb{P}(Z(T_k) = x) \sum_{i\in I(x)} \frac{\lambda_i(x)}{\sum_{j=1}^{n} \lambda_j(x)} = \sum_{x\in\mathcal{E}} \lim_{k\to\infty} \mathbb{P}(Z(T_k) = x) \sum_{i\in I(x)} \frac{\lambda_i(x)}{\sum_{j=1}^{n} \lambda_j(x)} \\
&= \sum_{x\in\mathcal{E}} \gamma(x) \sum_{i\in I(x)} \frac{\lambda_i(x)}{\sum_{j=1}^{n} \lambda_j(x)}, \qquad (2.17)
\end{aligned}
$$

where the interchange of the limit and the summation in (2.17) is justified by the finiteness of set $\mathcal{E}$.

It remains to calculate $\gamma(x)$. To this end, we use a standard Poisson uniformization technique which takes advantage of the fact that requests arrive according to a Poisson process with rate $\sum_{j=1}^{n} \lambda_j(x)$ when the Markov process $\mathbf{Z}$ is in state x. More specifically, let us sample $\mathbf{Z}$ according to a Poisson process with constant rate $\mu := \max_{x\in\mathcal{E}} \sum_{j=1}^{n} \lambda_j(x)$. Whenever there is an occurrence of the Poisson process and $\mathbf{Z}$ is in state x, this occurrence is selected with probability $\sum_{j=1}^{n} \lambda_j(x)/\mu$ and is not selected with the complementary probability. Therefore,

$$
\gamma(x) = \frac{\mu \times \sum_{i=1}^{n} \lambda_i(x)/\mu \times \theta(x)}{\sum_{y\in\mathcal{E}} \mu \times \sum_{j=1}^{n} \lambda_j(y)/\mu \times \theta(y)} = \frac{\theta(x) \sum_{j=1}^{n} \lambda_j(x)}{\sum_{y\in\mathcal{E}} \theta(y) \sum_{j=1}^{n} \lambda_j(y)},
$$

obtained as the ratio of the rate at which an occurrence of the Poisson process is selected when $\mathbf{Z}$ is in state x to the rate at which an occurrence of the Poisson process is selected.

Therefore, by (2.17),

$$
h^{HR-MMPP} = \sum_{x\in\mathcal{E}} \frac{\theta(x) \sum_{i\in I(x)} \lambda_i(x)}{\sum_{y\in\mathcal{E}} \theta(y) \sum_{j=1}^{n} \lambda_j(y)}, \quad \forall x \in \mathcal{E}.
$$

2.4.4 Shot Noise Model

Another traffic model, named Shot Noise Model (SNM) [92], has been proposed to capture the temporal locality observed in real traffic in caching systems *e.g.* in Video on Demand (VoD) systems. The primary idea of the SNM is to represent the overall request process as the superposition of many independent time inhomogeneous Poisson processes or shots, each referring to an individual object. In particular, the request process for object i is described by an inhomogeneous Poisson process of instantaneous rate

$$\lambda_i^{inst}(t) = V_i \lambda_i(t - \tau_i),$$

where τ_i is the time instant at which object i is first requested, V_i denotes the expected number of requests generated by object i and $\lambda_i(x)$ is the popularity profile of object i over time. It is easy to check that the instantaneous hazard rate associated with object i can be calculated as

$$\lambda_i^*(t) = \lambda_i^{inst}(t),$$

Note that, in the shot-noise model, there can potentially be an infinite objects in the catalog. However, at any time t, there is only a finite number of objects that are actively requested. Indeed, the number of active objects at time t is upper bounded by the number of occurrences in $[0, t)$ of the Poisson process $\{\tau_i\}_i$, which is an almost finite quantity.

Since object request processes are independent (as $\{\tau_i\}_i$ is a Poisson process) and inhomogeneous Poisson (and therefore regular) processes, Theorem 1 applies to the SNM. We conjecture that the limit in (2.9) exists and is a constant. We then use Theorem 2 and generate results for the SNM in the experimental Section 2.5.8.1.

2.5 Numerical Results

In this section we use simulations to compare the stationary object hit probabilities of various online policies (Section 2.5.1) to that of our proposed upper bound (*HR upper bound*), Bélády's upper bound (BELADY) and to a third bound (FOO, see Section 2.5.2).

The goals of this study are to (1) determine conditions under which HR provides tighter bound than other approaches, (2) to determine how close different non-anticipative policies come to the HR bound, and (3) the effect of the request process on these results. This study is done for a number of requests processes (Section 2.5.3), for equal and different size objects (Section 2.5.4) and for several cache sizes. We first present the experimental setup and then discuss the results.

2.5.1 Investigated online policies

Several caching policies have been used to generate Figures 2.2-2.7. The well-known LRU, FIFO, and RANDOM cache replacement policies discard the least recently used items first, evicts objects in the order they were added, and randomly selects an object and discards it to make space when necessary, respectively. The STATIC policy keeps forever in the cache the B objects that have the largest arrival rates. Notice that the HR based bound and the hit probability under STATIC are equal when successive requests for each object follow a Poisson process (Section 2.4.1). We also consider the Greedy-Dual-Size-Frequency (GDSF) policy [27] which combines recency with frequency and size to improve upon LRU. Last, the AdaptSize policy [20] uses an adaptive size threshold with admission control preferring admission of small sized objects.

2.5.2 Upper bounds on object hit probability

Besides our proposed HR based upper bound which applies to both equal and variable sized objects, two other upper bounds on the object hit probability proposed in literature are used, Bélády's offline upper bound (BELADY, Section 2.1.1) for equal sized objects and a flow based offline optimal (FOO) [19] for different sized objects. FOO upper bound is computed by representing caching as a min-cost flow problem.

2.5.3 Arrival process of object requests

In each plot in Figures 2.2-2.3, request processes for objects $i = 1, \ldots, n$ are independent renewal processes with inter-request time (IRT) distributions shown in Table 5.1. More specifically, in Figure 2.2(a) (resp. Figures 2.2(b)-2.2(f)) the request process for object $i = 1, \ldots, n$ has an exponential IRT (resp. Generalized Pareto, Uniform, Hyperexponential, Gamma, Erlang) with arrival rate λ_i drawn from a Zipf distribution with parameter 0.8 (see last column of Table 5.1); similarly, in Figure 2.3(a) (resp. Figures 2.3(b)-(c)) the IRT has an exponential (resp. Generalized Pareto, Uniform) distribution with arrival rate λ_i drawn from a Zipf distribution with parameter 0.8.

Inter-request time distribution (IRT)	Hazard Rate	$\mathbb{P}(IRT_i < t)$	Arr. rate λ_i $(= 1/\mathbb{E}[IRT_i])$ drawn from Zipf (0.8)
Exponential	CHR	$1 - e^{-\lambda_i t}$	λ_i
Generalized Pareto	DHR	$1 - (1 + \frac{k_i t}{\sigma_i})^{-\frac{1}{k_i}}$, $k_i = 0.48$	$\frac{1-k_i}{\sigma_i}$
Hyperexponential*	DHR	$1 - \sum_{j=1}^{2} p_{ji} e^{-\theta_{j,i} t}$ $p_{1,i} + p_{2,i} = 1$ $p_{1,i}/\theta_{1,i} = p_{2,i}/\theta_{2,i} := \nu_i$ $SCV_i = \mathrm{var}(IRT_i)/\mathbb{E}[IRT_i]^2 = 2$	$\frac{1}{2\nu_i}$
Uniform	IHR	$\frac{t}{b_i}$	$\frac{2}{b_i}$
Gamma	DHR $(k_i < 1)$	$\frac{1}{\Gamma(k_i)} \gamma(k_i, \frac{t}{\theta_i})$, $k_i = 0.5$	$\frac{2}{\theta_i}$
Erlang	IHR	$\frac{\gamma(k_i, \mu_i t)}{(k_i-1)!}$, $k_i = 0.2$	$\frac{\mu_i}{2}$

Table 2.1: Inter-request time (IRT) distributions of the renewal request arrival processes in Figures 2.2-2.3 and their properties (CHR = Constant hazard rate, IHR = Increasing hazard rate, DHR = Decreasing hazard rate). *Once arrival rate $\frac{1}{2\nu_i}$ is known one finds $p_{1,i} = (1 - \sqrt{(SCV_i - 1)/(SCV_i + 1)})/2$ under the constraints.

In Figure 2.4(a)-(b) the arrival request process for object i $(i = 1, \ldots, n)$ is generated via an on-off process (see Section 2.4.2) and these n on-off processes are mutually independent. The transition rates for on-off process i are $\alpha_i = 1/T_{OFF}$ and $\beta_i = 1/T_{ON}$, with $T_{ON} = 7$ (days) and $T_{OFF} = 9T_{ON}$. The arrival rate λ_i in the on-state is given by $\lambda_i = V/T_{ON}$, where V is drawn from a Pareto distribution with pdf $f_V(v) = \beta V_{min}^{\beta}/v^{1+\beta}$, $\mathbb{E}[V] = 10$, and $\beta = 2$ [41].

In Figure 2.5 requests for objects are generated according to a two-state MMPP (see Section 2.4.3). Without loss of generality (W.l.o.g.), call 1 and 2 these two states. Let α and β be the state transition rate from state 1 to 2 and from state 2 to 1, respectively. The stationary state probabilities are $\gamma(1) = \beta/(\alpha+\beta)$ and $\gamma(2) = \alpha/(\alpha+\beta)$. In the simulations, we took $\alpha = 2 \times 10^{-3}$ and $\beta = 1.6 \times 10^{-3}$. In state j, successive requests for object i are generated according to a Poisson process with rate $\lambda_i(j)$ for $j = 1, 2$. In state 1, we assume that object arrival rates $\lambda_i(1), \ldots, \lambda_n(1)$ each follows a Zipf distribution with parameter 0.8; W.l.o.g assume that $\lambda_1(1) > \lambda_2(1) > \cdots > \lambda_n(1)$. In state 2, we assume that object arrival rates are given by $\lambda_i(2) = \lambda_{n+1-i}(1)$ for $i = 1, \cdots, n$.

In Figure 2.6 requests for objects are generated by n independent shot noise processes (see Section 2.4.4). Objects belong to four different classes. Objects in class c ($c = 1, \ldots, 4$) become available in the system at times $\tau_1(c) < \cdots < \tau_{n_c}(c)$ of a homogeneous Poisson process with rate $\gamma_c = \mathbb{E}[V_c]/\mathbb{E}[L_c]$. The SNM associated with the ith object of class c, which becomes available at time $\tau_i(c)$, has intensity $\lambda_i(t) = (V_i/\alpha_c)e^{-(t-\tau_i(c))/\alpha_c}$, with $\alpha_c = \frac{0.5}{0.8}\mathbb{E}[L_c]$ and where V_i is chosen according to a Poisson distribution with rate $\mathbb{E}[V_c]$. Values of $\mathbb{E}[V_c]$ (expected number of requests for a class c object) and $\mathbb{E}[L_c]$ (expected lifespan of a class c object) are given in Table 2.2. This model has been obtained by the authors of [91] from their Trace 1, which contains $n = \sum_{c=1}^{4} n_c = 143871$ objects (cf. 4th column of Table 2.2).

Class id (c)	$\mathbb{E}[L_c]$	$\mathbb{E}[V_c]$	Catalog size (n_c)
Class 1	1.14	86.4	29481
Class 2	3.36	41.9	45570
Class 3	6.40	59.5	27435
Class 4	10.53	36.9	41385

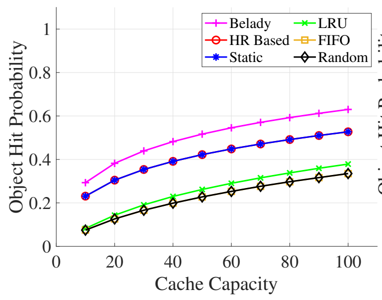
Table 2.2: Parameters of the shot-noise models in Figure 2.6.

In Figure 2.7 we use requests from a Web access trace collected from a gateway router at IBM research lab [101]. We filter the trace such that each object has been requested at least a hundred times. The filtered trace contains 3.5×10^6 requests with an object catalog of size $n = 5638$. Various parametric and non-parametric estimators have been developed in the literature to estimate the hazard rate [97, 87]. Here, we adopt a parametric estimator model and assume that the inter-request times for each object are independent and identically distributed non-negative random variables. Note that the Web and storage traffic inter-request times and access patterns are well modeled by heavy-tailed distributions [33, 46]. Hence, we fit the density of inter-request times of each object to a Generalized-Pareto distribution using the maximum likelihood estimation technique and estimate the hazard rate for each object accordingly.

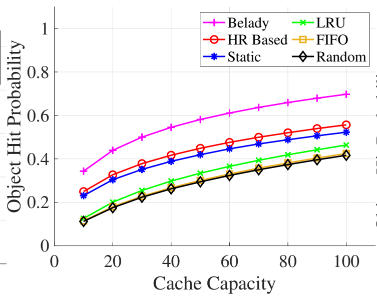
2.5.4 Size of objects

Both equal size and variable size objects are considered. In the former the size of each object is equal to 1 and in the latter the size of each object is drawn independently according to a bounded Pareto distribution with Pareto shape parameter 1.8, minimum object size of 5MB and maximum object size of 15MB. When all objects have same size the size of the cache is expressed in number of objects. It is expressed in MB when objects have different sizes. As a general comment, we note from Figure 2.2-2.7 that, as expected, the HR based upper bound serves as an upper bound on the hit probability among all online caching policies. Further comments are given below on each figure.

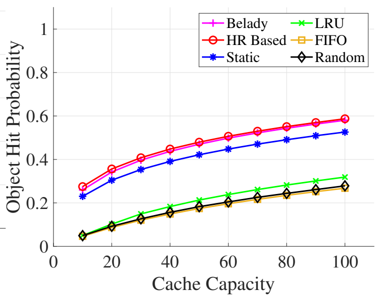
2.5.5 Renewal request processes and equal size objects



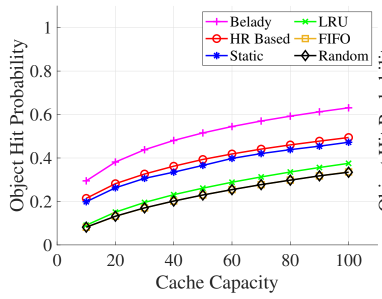
(a) Exponential (CHR)



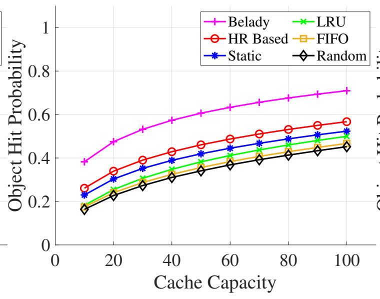
(b) Generalized-Pareto (DHR)



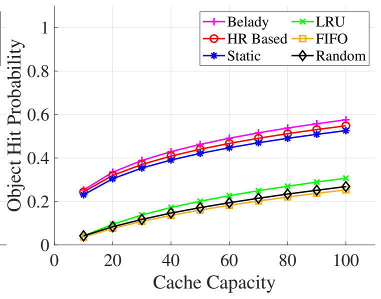
(c) Uniform (IHR)



(d) Hyperexponential (DHR)



(e) Gamma (DHR)

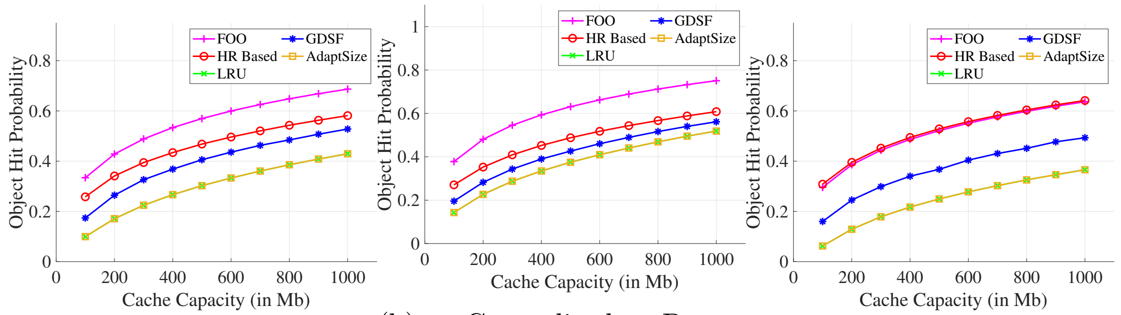


(f) Erlang (IHR)

Figure 2.2: Simulation results for HR based upper bound and various caching policies under different inter request arrival distributions ($n = 1000$, all objects of size 1).

Request processes used to generate plots in Figure 2.2 are presented in Section 2.5.3. These plots are obtained for 1000 objects where all objects have size 1. Observe (see discussion in Section 2.4.1) from Figure 2.2(a) that the STATIC policy and the HR based upper bound exhibit the same hit probabilities when IRTs are exponential distributed. We observe that when IRTs are either CHR or DHR, the HR based upper bound is much tighter than Bélády's upper bound and both bounds are close when IRTs are IHR. STATIC consistently yields the highest hit probability and is always close to the HR upper bound. For exponential IRTs or, equivalently for the independence reference model, the optimality of STATIC is well known [62].

2.5.6 Renewal request processes and variable size objects



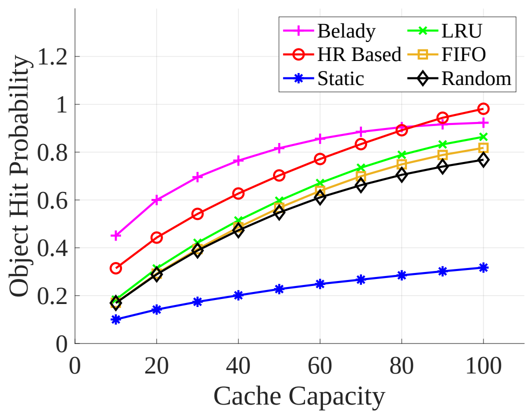
(a) Exponential (CHR) (b) Generalized Pareto (DHR) (c) Uniform (IHR)

Figure 2.3: Simulation Results for HR based upper bound and various caching policies under different inter request arrival distributions for variable object sizes ($n = 1000$).

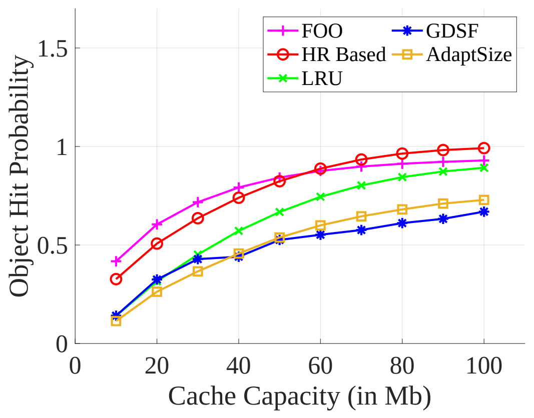
The request processes used to generate plots in Figure 2.3 are described in Section 2.5.3. Objects are variable in size (see Section 2.5.4) and there are 1000 objects. We observe that when IRTs have exponential or Generalized Pareto distributions the HR based upper bound is much tighter than the FOO upper bound (Figure 2.3(a)-(b)) and that both bounds are close when IRTs are uniformly distributed rvs (Figure 2.3(c)). For exponential and Generalized Pareto IRT distributions the GDSF policy performs well (close to HR); one way of interpreting the gap between HR (resp. FOO) and GDSF in Figure 2.3(c) is to say that there is room for improvement in caching policy performance when IRTs are uniformly distributed rvs.

2.5.7 On-off request arrivals and equal/variable size objects

The parameters of the on-off process used to generate arrival times of requests of object i ($i = 1, \ldots, n$) are given in Section 2.5.3. There are 1000 objects in the catalog for equal sized objects and 100 objects for variable sized objects. The average



(a) Equal Size ($n = 1000$)



(b) Variable Size ($n = 100$)

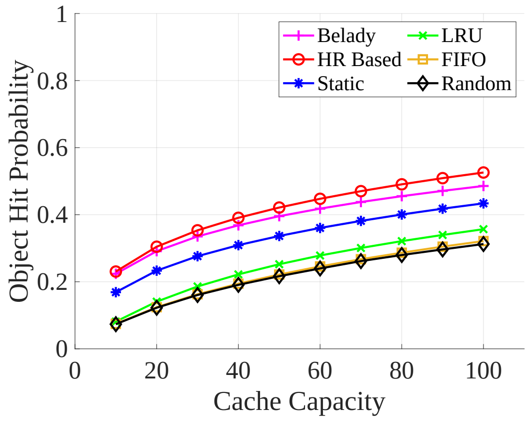
Figure 2.4: Performance comparison under on-off request process

arrival rate for object i is $\lambda_i\pi_{i,1}$ with $\pi_{i,1} = \alpha_i/(\alpha_i + \beta_i)$ (Section 2.4.2). The STATIC policy permanently stores in the cache the B objects in decreasing order of $\{\lambda_i\pi_{i,1}\}_i$.

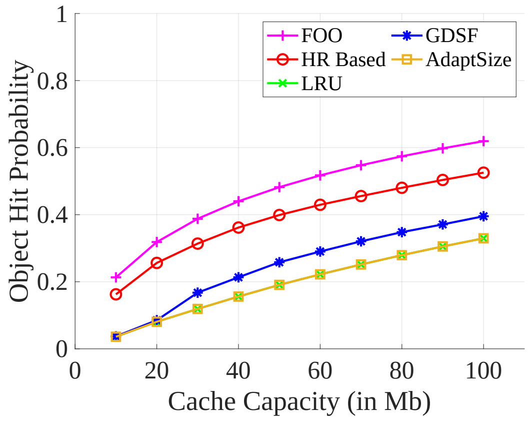
For equal size objects (resp. variable size objects) the HR bound is tighter than BELADY (resp. FOO) for small caches whereas BELADY (resp. FOO) becomes tighter for large cache sizes. LRU performs the best for both equal size and variable size objects and STATIC policy performs the worst for equal sized objects.

2.5.8 MMPP request arrivals and equal/variable size objects

The parameters of the two-state MMPP (states 1 and 2) used to generate arrival times of requests are given in Section 2.5.3. There are 1000 objects in the catalog for equal size objects and 100 objects for variable sized objects. The average arrival rates for object i is $\lambda_i(1)\gamma(1) + \lambda_i(2)\gamma(2) = (\lambda_i(1)\beta + \lambda_i(2)\alpha)/(\alpha + \beta)$. The STATIC caching policy permanently stores in the cache the B objects with the highest average arrival rates.

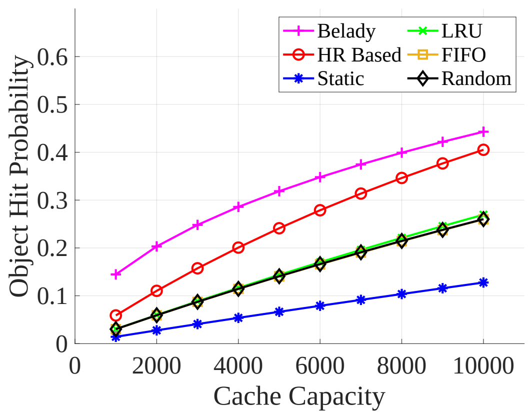


(a) Equal Size ($n = 1000$)

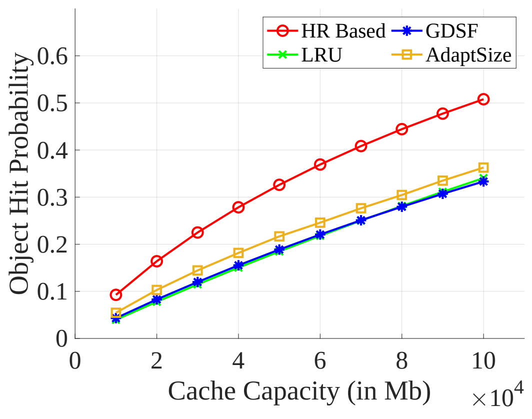


(b) Variable Size ($n = 100$)

Figure 2.5: Performance comparison under two-state MMPP request arrivals



(a) Equal Size



(b) Variable Size

Figure 2.6: Performance comparison under shot noise model ($n = 143871$).

Unlike Figures 2.2-2.4, BELADY is tighter than the HR based upper bound for equal size objects (Figure 2.5(a)) but the latter upper bound is tighter than the FOO upper bound for variable size objects (Figure 2.5(b)). STATIC performs the best among all online caching policies. Note that, in our simulations, $\gamma(1) = \beta/(\alpha + \beta)$ and $\gamma(2) = \alpha/(\alpha+\beta)$ are comparable. We postulate that the performance of STATIC will further improve when $\gamma(1) \gg \gamma(2)$ or $\gamma(1) \ll \gamma(2)$. For example, when $\gamma(1) \gg \gamma(2), \lambda_i^{STATIC} \sim \lambda_i(1)\gamma(1)$; in this case the STATIC policy will permanently store the popular objects in state 1, thus always getting a hit when the MMPP is in state 1.

2.5.8.1 Shot noise request arrivals and equal/variable size objects

The parameters of the SNM used to generate Figure 2.6 are given in Section 2.5.3. For equal size objects (Figure 2.6(a)) our proposed HR bound not only upper bounds the hit probability for existing online caching policies but also provides a tighter bound than the state-of-the-art BELADY. STATIC policy performs the worst while LRU performs the best among all online policies. For variable size objects (Figure 2.6(b)) AdaptSize performs the best and GDSF and LRU have similar performance. The difference in the object hit probability between the HR upper bound and AdaptSize suggests that there is room for improvement in caching policy performance.

2.5.9 Real-world trace

Characteristics of the real-world trace and its application to the production of Figure 2.7 are discussed in Section 2.5.3. Upper bounds on the object hit probability obtained with HR and BELADY are almost identical. LRU performs the best and STATIC the worst.

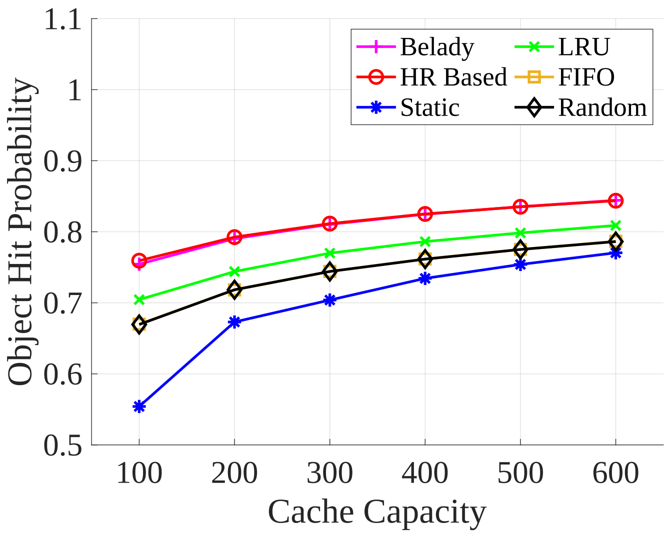


Figure 2.7: Performance comparison under real world data trace ($n = 5638$).

2.6 Summary

We began this chapter by asking: *Can we provide an upper bound on the cache hit probability for any feasible non-anticipative caching policy?* We have answered this question by developing a HR based upper bound on the cache hit probability for non-anticipative caching policies. We showed that for equal sized objects, hazard rate associated with the object arrival process can be used to provide this upper bound. Inspired by the results for equal size objects, we extended the HR based argument to obtain an upper bound on the byte and object hit probability for variable size objects solving a knapsack problem. We derived closed form expressions for the upper bound under some specific object request arrival processes. We showed that HR based upper bound is tighter for a variety of object arrival processes than those analyzed in the literature.

CHAPTER 3

A TTL-BASED APPROACH FOR FAIR RESOURCE ALLOCATION IN CACHE NETWORKS

In Chapter 2 we proposed an upper bound on overall cache hit probability for any online caching policy that operates on a single cache. We assumed all contents had the same quality of service requirements. In this chapter, we consider the problem of providing differentiated service to different content classes in a cache network. This chapter asks: *How do we build cache networks that provide differential services?*

3.1 Background

We consider a *Time-To-Live* (TTL) based cache network, where a set of network caches host a library of unique contents, and serve a set of users. We define each component of the TTL based cache network in greater detail in Sections 3.1.3 and 3.1.4. Figure 3.1 illustrates an example of such a network, which is consistent with the YouTube video delivery system [83, 86]. Each user can generate a request for a content, which is forwarded along a fixed *path* from the edge cache towards the server. Forwarding stops upon a cache hit, i.e., the requested content is found in a cache on the path. When such a cache hit occurs, the content is sent over the reverse path to the edge cache initializing the request. This raises the questions: *where to cache the requested content on the reverse path* and *what value should the timer take?* In this chapter our goal is to provide thorough and rigorous answers to these questions.

There is a rich literature on the design, modeling and analysis of cache networks, including TTL caches [85, 36, 37, 18], optimal caching [49, 61] and routing policies

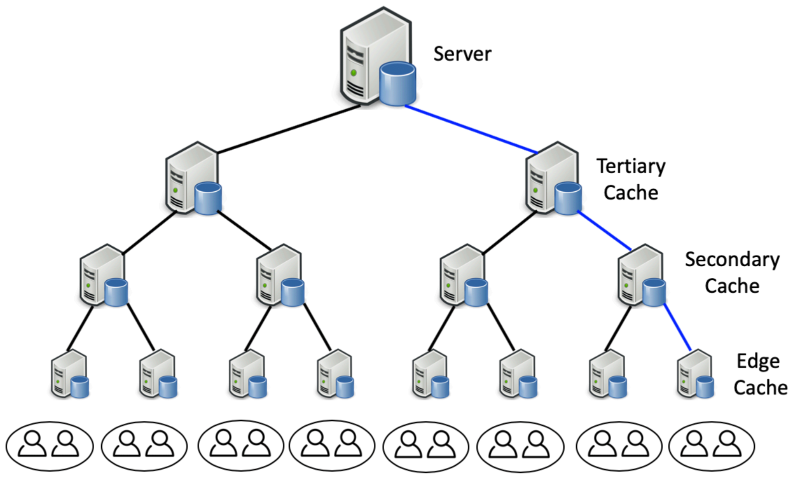


Figure 3.1: An cache network with a server holding all contents and three layers of caches. Each leaf/edge cache serves a set of users with different requests. The blue line illustrates a unique path between the leaf cache and the server.

[50]. In particular, Rodriguez et al. [85] analyzed the advantage of pushing content upstream, Berger et al. [18] characterized the exactness of TTL policy in a hierarchical topology. A unified approach to study and compare different caching policies is given in [40] and an optimal placement problem under a heavy-tailed demand has been explored in [34]. Dehghan et al. [31] as well as Abedini and Shakkottai [2] studied joint routing and content placement with a focus on a bipartite, single-hop setting. Both showed that minimizing single-hop routing cost can be reduced to solving a linear program. Ioannidis and Yeh [50] studied the same problem under a more general setting for arbitrary topologies. An adaptive caching policy for a cache network was proposed in [49], where each node makes a decision on which item to cache and evict. An integer programming problem was formulated by characterizing the content transfer costs. Both centralized and complex distributed algorithms were designed with performance guarantees. This work complements the work proposed in this chapter, as we consider TTL caches and use timer control settings to cache and maximize the sum of utilities over all contents across the network. [49] proposed approximate

algorithms while our timer-based models enable us to design optimal solutions since content occupancy can be modeled as a real variable (e.g. a probability).

3.1.1 Network Model

We represent a cache network by a graph $G = (V, E)$. See Figure 3.1 for an example. Let $\mathcal{D} = \{d_1, \cdots, d_n\}$ denote the set of contents. Each network cache $v \in V$ can store up to B_v contents to serve requests from users. We assume that each user first sends a request for the content to its local network cache, which may then route the request to other caches for retrieving the content. Without loss of generality, we assume that there is a fixed and unique path from the local cache towards a terminal cache that is connected to a server that always contains the content.

3.1.2 Content Request Process

Denote a request (v, i, p) by the cache, v, that initially receives the user request, the requested content, i, and the path, p, over which the request is routed. We denote a path p of length $|p| = L$ as a sequence $\{v_{1p}, v_{2p}, \cdots, v_{Lp}\}$ of nodes $v_{lp} \in V$ such that $(v_{lp}, v_{(l+1)p}) \in E$ for $l \in \{1, \cdots, L\}$, where $v_{Lp} = v$. Unless specified, we assume path p is loop-free and that terminal cache v_{1p} is the only cache on path p that accesses the server for content i. We assume that the request processes for distinct contents are described by independent Poisson processes with arrival rate λ_i for content $i \in \mathcal{D}$. Denote $\Lambda = \sum_{i=1}^{n} \lambda_i$. Then the popularity (request probability) of content i satisfies [12]

$$\rho_i = \frac{\lambda_i}{\Lambda}, \quad i = 1, \cdots, n.$$

3.1.3 Time-To-Live Caches

We consider reset based TTL caches in this thesis. In a reset based TTL cache, each content is associated with a timer value (say T) as shown in Figure 3.2. When a

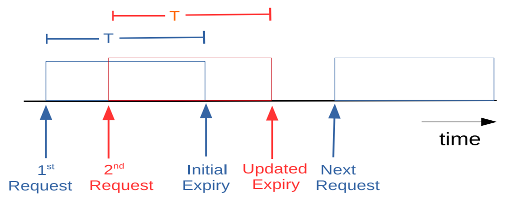


Figure 3.2: A reset TTL cache

content is requested, there are two cases: (i) if the content is not in the cache (miss), then the content is inserted into the cache and its timer is set to T; (ii) if the content is in the cache (hit), then the timer associated with the content is reset to T. The timer decreases at a constant rate and the content is evicted once its timer expires. We control the hit rate of each content by adjusting its timer value.

3.1.4 Cache Replication Strategy

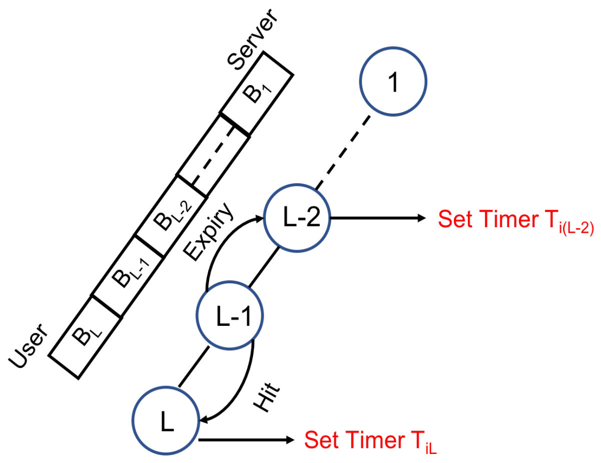


Figure 3.3: Move Copy Down with Push cache replication policy

Each content i is associated with a timer T_{ij} at cache j. Suppose content i is requested and routed along path p. There are two cases: (i) content i is not in any cache along path p, in which case it is fetched from the server and inserted into the terminal cache (denoted by cache 1)[1] and its timer is set to T_{i1}; (ii) if content i is in cache l along path p, content i is moved to cache $l+1$ preceding cache l in which i is found, and the timer at cache $l+1$ is set to $T_{i(l+1)}$. Whenever the timer for any content i at any cache l expires, it is pushed back to cache $l-1$ and the timer is set to $T_{i(l-1)}$ as shown in Figure 3.3. We call this policy Move Copy Down with Push (MCDP) [85]. Denote the *hit probability* of content i as h_i; then the corresponding *hit rate* is $\lambda_i h_i$.

3.1.5 Utility based caching

We first formulate a utility-driven caching framework for cache networks, where each content is associated with a utility and content is managed with a timer whose duration is set to maximize the aggregate utility for all contents over the cache network. In addition to the ability to represent costs and user satisfaction, utility based content caching can account for fairness. Fairness is important in many scenarios. For example, when content providers aggregate content from multiple sources, it is important that the benefits are spread fairly across caches [98]. A utility based framework allow a choice of utility functions to provide different notions of fairness, such as proportional or max-min fairness [88].

Utility functions can also capture the satisfaction perceived by a user after being served a content. We associate each content $i \in \mathcal{D}$ with a utility function $U_i : \mathbb{R} \to \mathbb{R}$ that is a function of hit rate $\lambda_i h_i$. $U_i(\cdot)$ is assumed to be increasing, continuously differentiable, and strictly concave. In particular, for our numerical studies, we focus on the widely used β-fair utility functions [88] given by

[1]Since we consider path p, for simplicity, we move the dependency on p and v, denote it as nodes $1, \cdots, L$ directly.

$$U_i(x) = \begin{cases} w_i \frac{x^{1-\beta}}{1-\beta}, & \beta \geq 0, \beta \neq 1; \\ w_i \log x, & \beta = 1, \end{cases} \tag{3.1}$$

where $w_i > 0$ denotes a weight associated with content i.

3.2 Special Case: Single Cache

We first begin our analysis by considering a single cache system. Note that, for a single cache system we only have one timer associated with each content. We consider a single cache of size B serving n distinct contents each of unit size.

3.2.1 Extension to Stationary Request Arrival Process

A *Hit-probability Based* utility driven caching framework under Poisson content arrival process was studied in Dehghan et.al. [30]. The objective was to maximize the sum of utilities under a cache capacity constraint when utilities are function of hit probabilities. While characterization of hit probability under Poisson content arrival process is valuable, real-world request processes exhibit changes in popularity and temporal correlations in requests [102, 26]. To account for these changes, we consider a very general traffic model where requests for distinct contents are described by mutually independent stationary and ergodic point processes.

We consider the content arrival process to be a stationary point process with cumulative inter-request time (*irt*) distribution function (c.d.f.) $F_i(t)$ for $i = 1, \cdots, n$. Denote by $\hat{F}_i(t)$ the c.d.f. of the age associated with the irt distribution for content i, satisfying ([12])

$$\hat{F}_i(t) = \mu_i \int_0^t (1 - F(x))dx.$$

Denote by h_i and h_i^{in} as the hit probability and *time-average probability* that content i is in the cache (i.e., *occupancy probability*) respectively. From the analysis of previous work [35], the hit probability and occupancy probability for a reset TTL cache can be computed as $h_i = F_i(t_i)$ and $h_i^{\text{in}} = \hat{F}_i(t_i)$ respectively. Observe that for Poisson arrival process [30], $F_i(t_i) = 1 - e^{-\lambda_i t_i}$ and $h_i = h_i^{\text{in}}$, based on the PASTA property [66]. Note that, we can represent h_i^{in} as a function of h_i as follows.

$$h_i^{\text{in}} = \hat{F}_i(F_i^{-1}(h_i)) \triangleq g_i(h_i),$$

Here we define $g_i(x)$ to be a one-to-one function with $g_i(x) = \hat{F}_i(F_i^{-1}(x))$.

3.2.2 Cache Utility Maximization

We are interested in optimizing the sum of utilities over all contents subject to cache capacity constraint.

$$\begin{aligned} \max_{\{h_1,\cdots,h_n\}} \quad & \sum_{i=1}^{n} U_i(\lambda_i h_i) \\ \text{s.t.} \quad & \sum_{i=1}^{n} g_i(h_i) \leq B, \qquad (3.2a) \\ & 0 \leq h_i \leq 1 \quad i = 1, \cdots, n. \end{aligned}$$

Again (3.2a) is a constraint on average cache occupancy. A related problem has been formulated in [34], where the authors formulated the optimization problem as a function of h_i^{in}. However, such a formulation may not be suitable for designing online algorithms since we need a closed form expression for $\hat{F}_i^{-1}$. Furthermore, [34] only considers linear utilities while we aim to characterize the impact of different utility functions on optimal TTL policies.

3.2.3 Effect of Hazard Rate

Now we consider the convexity of (3.2). Denote the density function associated with the content arrival process as $f_i(t)$. Let $\zeta_i(t) = f_i(t)/[1 - F_i(t)], \quad t \in [0, F_i^{-1}(1)]$

be the hazard rate function associated with $F_i(t)$. The behavior of $\zeta_i(t)$ plays a prominent role in solving (3.2). In particular, if $\zeta_i(t)$ is non-increasing (DHR), then it can be shown that $g'(h_i)$ is non-decreasing in h_i. Therefore, the feasible set in (3.2) is convex. Since the objective function is strictly concave and continuous, (3.2) is a convex optimization problem, and an optimal solution exists. In this chapter, we mainly focus on the case that $\zeta_i(t)$ is DHR, and refer the interested reader to [34] for discussions of other cases.

3.2.4 Online Dual Algorithm

In Section 3.2.2, we formulated an optimization problem with a fixed cache size under the assumption of a static known workload. However, system parameters (e.g. request processes) can change over time. Moreover, it is infeasible to solve the optimization problem offline and then implement the optimal strategy. Hence online algorithms are needed to implement the optimal strategy to adapt to these changes in the presence of limited information.

The Lagrange dual function for the optimization problem (3.2) is

$$D(\eta) = \max_{h_i} \left\{ \sum_{i=1}^{n} U_i(\lambda_i h_i) - \eta \left[\sum_{i=1}^{n} g_i\left(h_i\right) - B \right] \right\},$$

and the dual problem is $\min_{\eta \geq 0} \quad D(\eta)$.

Following standard *gradient descent algorithm* by taking the derivative of $D(\eta)$ w.r.t. η, the dual variable η should be updated as

$$\eta^{(k+1)} \leftarrow \max \left\{ 0, \eta^{(k)} + \gamma \left[\sum_{i=1}^{n} g_i(h_i) - B \right] \right\},$$

where k is the iteration number, $\gamma > 0$ is the step size at each iteration and $\eta \geq 0$ due to KKT conditions. Also, in order to achieve optimality, we must have

$$\eta^{(k)} = \frac{\lambda_i U_i'(\lambda_i h_i)}{g_i'(h_i)} \triangleq y_i(h_i), \text{ i.e., } h_i = y_i^{-1}(\eta^{(k)}).$$

Since $g_i(h_i)$ indicates the probability that content i is in the cache, $\sum_{i=1}^{n} g_i(h_i)$ represents the number of contents currently in the cache, denoted as B_{curr}. Therefore, the dual algorithm for a reset TTL cache is

$$\begin{aligned} &t_i^{(k)} = F_i^{-1}(y_i^{-1}(\eta^{(k)})), \\ &\eta^{(k+1)} \leftarrow \max\left\{0, \eta^{(k)} + \gamma(B_{\text{curr}} - B)\right\}, \end{aligned}$$

which is executed every time a request is made.

3.2.5 Online Poisson Approximation

From Section 3.2.4, it is clear that the implementation of the online algorithm involves solving $y_i^{-1}(x) = 0$ for x. It can be shown that under generalized Pareto, hyperexponential distributions and Markov modulated Poisson processes (MMPP), $y_i^{-1}(x)$ involves solving non-linear fixed point equations, that are computationally intensive. However, the Dual for the case of requests governed by Poisson processes is simple. Thus we apply the Dual designed for Poisson request processes to a workload where requests are described by a $n$on-Poisson stationary request processes. Such an algorithm does not require solving any non-linear equations and hence is computationally efficient. Moreover, we also use estimation techniques introduced in [30] to approximate request rates, which allows the use of these algorithms in an online fashion.

We consider the problem of estimating the arrival rate λ_i for content i adopting techniques used in [30] described as follows. Denote the remaining TTL time for content i as τ_i. This can be computed given t_i and a time-stamp for the last request time for content i. We approximate the mean inter-request time as $t_i - \tau_i$. Clearly

$t_i - \tau_i$ is an unbiased estimator of $1/\lambda_i$. Given this estimator and Dual (3.3), we propose the following *Poisson approximate online algorithm*

$$t_i^{(k)} = -\frac{1}{\hat{\lambda}_{iP}} \log\left(1 - \frac{1}{\hat{\lambda}_{iP}} U_i'^{-1}\left(\frac{\eta^{(k+1)}}{\hat{\lambda}_{iP}}\right)\right),$$

$$\eta^{(k+1)} \leftarrow \max\{0, \eta^{(k)} + \gamma(B_{\text{curr}} - B)\}.$$

3.3 Linear Cache Network

We now begin our analysis with a linear cache network, i.e., there is a single path between the user and the server, composed of $|p| = L$ caches labeled $1, \cdots, L$. A content enters the edge network via cache 1, and is promoted to a higher index cache whenever a cache hit occurs. In the following, we consider the MCDP replication strategy when each cache operates with a TTL policy.

3.3.1 Stationary Behavior

Requests for content i arrive according to a Poisson process with rate λ_i. Under TTL, content i spends a deterministic time in a cache if it is not requested, independent of all other contents. We denote the timer as T_{il} for content i in cache l on the path p, where $l \in \{1, \cdots, |p|\}$.

Denote by t_k^i the k-th time that content i is either requested or the timer expires. For simplicity, we assume that content is in cache 0 (i.e., server) when it is not in the cache network. We then define a discrete time Markov chain (DTMC) $\{X_k^i\}_{k\geq 0}$ with $|p|+1$ states, where X_k^i is the index of the cache that content i is in at time t_k^i. The event that the time between two requests for content i exceeds T_{il} occurs with probability $e^{-\lambda_i T_{il}}$; consequently we obtain the transition probability matrix of $\{X_k^i\}_{k\geq 0}$ and compute the stationary distribution. Details can be found in Appendix B.1. The timer-average probability that content i is in cache $l \in \{1, \cdots, |p|\}$ is

$$h_{i1} = \frac{e^{\lambda_i T_{i1}} - 1}{1 + \sum_{j=1}^{|p|}(e^{\lambda_i T_{i1}} - 1)\cdots(e^{\lambda_i T_{ij}} - 1)}, \tag{3.5a}$$

$$h_{il} = h_{i(l-1)}(e^{\lambda_i T_{il}} - 1),\ l = 2, \cdots, |p|, \tag{3.5b}$$

where h_{il} is also the hit probability for content i at cache l.

Remark 1. *The stationary analysis of MCDP is similar to a different caching policy LRU($\boldsymbol{m}$) considered in [43]. We refer interested readers to [43] for more detail.*

3.3.2 From Timer to Hit Probability

We consider a TTL cache network where requests for different contents are independent of each other and each content i is associated with a timer T_{il} at each cache $l \in \{1, \cdots, |p|\}$ on the path. Denote $\boldsymbol{T}_i = (T_{i1}, \cdots, T_{i|p|})$ and $\boldsymbol{T} = (\boldsymbol{T}_1, \cdots, \boldsymbol{T}_n)$. From (3.5), the overall utility on path p is given as

$$\sum_{i \in \mathcal{D}} \sum_{l=1}^{|p|} \psi^{|p|-l} U_i(\lambda_i h_{il}(\boldsymbol{T})),$$

where the utility function $U_i : [0, \infty) \to \mathbb{R}$ is assumed to be an increasing, continuously differentiable, and strictly concave function of content hit rate, and $0 < \psi \leq 1$ is a discount factor capturing the utility degradation along the request's routing direction. Since each cache is finite in size, we have the capacity constraint

$$\sum_{i \in \mathcal{D}} h_{il}(\boldsymbol{T}) \leq B_l, \quad l \in \{1, \cdots, |p|\}.$$

Therefore, the optimal TTL policy for content placement on path p is the solution of the following optimization problem

$$\begin{aligned} \max_{\boldsymbol{T}} \quad & \sum_{i \in \mathcal{D}} \sum_{l=1}^{|p|} \psi^{|p|-l} U_i(\lambda_i h_{il}(\boldsymbol{T})) \\ \text{s.t.} \quad & \sum_{i \in \mathcal{D}} h_{il}(\boldsymbol{T}) \leq B_l, \quad l \in \{1, \cdots, |p|\}, \end{aligned} \tag{3.6}$$

$$T_{il} \geq 0, \quad \forall i \in \mathcal{D}, \quad l = 1, \cdots, |p|,$$

where $h_{il}(\boldsymbol{T})$ is given in (3.5). However, (3.6) is a non-convex optimization with a non-linear constraint. Our objective is to characterize the optimal timers for different contents on path p.. To that end, it is helpful to express (3.6) in terms of hit probabilities. In the following, we discuss how to change the variables from timer to hit probability.

Since $0 \leq T_{il} \leq \infty$, it is easy to check that $0 \leq h_{il} \leq 1$ for $l \in \{1, \cdots, |p|\}$ from (3.5a) and (3.5b). Furthermore, it is clear that there exists a mapping between $(h_{i1}, \cdots, h_{i|p|})$ and $(T_{i1}, \cdots, T_{i|p|})$. By simple algebra, we obtain

$$\begin{aligned}
T_{i1} &= \frac{1}{\lambda_i} \log\left(1 + \frac{h_{i1}}{1 - \big(h_{i1} + h_{i2} + \cdots + h_{i|p|}\big)}\right), \\
T_{il} &= \frac{1}{\lambda_i} \log\left(1 + \frac{h_{il}}{h_{i(l-1)}}\right), \quad l = 2, \cdots, |p|.
\end{aligned}$$

Note that $h_{i1} + h_{i2} + \ldots + h_{i|p|} \leq 1$ must hold during the operation, which is always true for our caching policies.

3.3.3 Maximizing Aggregate Utility

With the change of variables discussed above, we can reformulate (3.6) as follows

$$\begin{aligned}
\max \quad & \sum_{i \in \mathcal{D}} \sum_{l=1}^{|p|} \psi^{|p|-l} U_i(\lambda_i h_{il}) \\
\text{s.t.} \quad & \sum_{i \in \mathcal{D}} h_{il} \leq B_l, \quad l = 1, \cdots, |p|, && \text{(3.7a)} \\
& \sum_{l=1}^{|p|} h_{il} \leq 1, \quad \forall i \in \mathcal{D}, && \text{(3.7b)}
\end{aligned}$$

$$0 \leq h_{il} \leq 1, \quad \forall i \in \mathcal{D}, \quad l = 1, \cdots, |p| \tag{3.7c}$$

where (3.7a) is the cache capacity constraint and (3.7b) is due to the variable exchanges under MCDP as discussed above.

Proposition 1. *The optimization problem defined in (3.7) under MCDP has a unique global optimum.*

3.3.4 Distributed Algorithm

In Section 3.3.3, we formulated convex utility maximization problems with a fixed cache size. However, system parameters (e.g. cache size and request processes) can change over time, so it is not feasible to solve the optimization offline and implement the optimal strategy. Thus, we need to design distributed algorithms to implement the optimal strategy and adapt to the changes in the presence of limited information.

Primal Algorithm: We aim to design an algorithm based on the optimization problem in (3.7), which is the primal formulation. Denote $\boldsymbol{h}_i = (h_{i1}, \cdots, h_{i|p|})$ and $\boldsymbol{h} = (\boldsymbol{h}_1, \cdots, \boldsymbol{h}_n)$. We first define the following objective function.

$$\begin{aligned} Z(\boldsymbol{h}) = & \sum_{i\in\mathcal{D}} \sum_{l=1}^{|p|} \psi^{|p|-l} U_i(\lambda_i h_{il}) - \sum_{l=1}^{|p|} C_l \left(\sum_{i\in\mathcal{D}} h_{il} - B_l \right) \\ & - \sum_{i\in\mathcal{D}} \tilde{C}_i \left(\sum_{l=1}^{|p|} h_{il} - 1 \right) - \sum_{i\in\mathcal{D}} \sum_{l=1}^{|p|} \hat{C}_{il}(-h_{il}), \end{aligned} \tag{3.8}$$

where $C_l(\cdot), \tilde{C}_i(\cdot)$ and $\hat{C}_{il}(\cdot)$ are convex and non-decreasing penalty functions denoting the cost for violating constraints (3.7a) and (3.7b).

Note that constraint (3.7b) ensures $h_{il} \leq 1 \ \forall i \in \mathcal{D}, \ l = 1, \cdots, |p|$, provided $h_{il} \geq 0$. One can assume that $h_{il} \geq 0$ holds in writing down (3.8). This would be true, for example, if the utility function is a β-fair utility function with $\beta > 0$ (Section 2.5[88]). For other utility functions, it is challenging to incorporate constraint (3.7c) since it introduces $n|p|$ additional price functions. For all cases evaluated across various system parameters we found $h_{il} \geq 0$ to hold true. Hence we ignore constraint (3.7c) in the primal formulation and define the following objective function

$$Z(\boldsymbol{h}) = \sum_{i\in\mathcal{D}}\sum_{l=1}^{|p|}\psi^{|p|-l}U_i(\lambda_i h_{il}) - \sum_{l=1}^{|p|}C_l\left(\sum_{i\in\mathcal{D}}h_{il} - B_l\right) - \sum_{i\in\mathcal{D}}\tilde{C}_i\left(\sum_{l=1}^{|p|}h_{il} - 1\right). \tag{3.9}$$

It is clear that $Z(\cdot)$ is strictly concave. Hence, a natural way to obtain the maximal value of (3.9) is to use the standard *gradient ascent algorithm* to move the variable h_{il} for $i \in \mathcal{D}$ and $l \in \{1, \cdots, |p|\}$ in the direction of gradient,

$$\frac{\partial Z(\boldsymbol{h})}{\partial h_{il}} = \lambda_i\psi^{|p|-l}U_i'(\lambda_i h_{il}) - C_l'\left(\sum_{j\in\mathcal{D}}h_{jl} - B_l\right) - \tilde{C}_i'\left(\sum_{m=1}^{|p|}h_{im} - 1\right),$$

where $U_i'(\cdot)$, $C_l'(\cdot)$, $\tilde{C}_i'(\cdot)$ denote partial derivatives w.r.t. h_{il}.

Since h_{il} indicates the probability that content i is in cache l, $\sum_{j\in\mathcal{D}}h_{jl}$ is the expected number of contents currently in cache l, denoted by $B_{\text{curr},l}$.

Therefore, the primal algorithm for MCDP is given by

$$T_{il}[k] \leftarrow \begin{cases} \frac{1}{\lambda_i}\log\left(1 + \frac{h_{il}[k]}{1-\left(h_{i1}[k]+h_{i2}[k]+\cdots+h_{i|p|}[k]\right)}\right), & l = 1; \\ \frac{1}{\lambda_i}\log\left(1 + \frac{h_{il}[k]}{h_{i(l-1)}[k]}\right), & l = 2, \cdots, |p|, \end{cases}$$

$$h_{il}[k+1] \leftarrow \max\Bigg\{0, h_{il}[k] + \zeta_{il}\Bigg[\lambda_i\psi^{|p|-l}U_i'(\lambda_i h_{il}[k]) - C_l'\left(B_{\text{curr},l} - B_l\right) - \tilde{C}_i'\left(\sum_{m=1}^{|p|}h_{im}[k] - 1\right)\Bigg]\Bigg\},$$

where $\zeta_{il} > 0$ is the step-size parameter, and k is the iteration number incremented upon each request arrival.

Theorem 4. *The primal algorithm (3.10) is locally asymptotically stable given a sufficiently small step-size parameter* ζ_{il}.

Proof. See appendix B.2. □

3.4 General Cache Network

Denote by $\mathcal{P}$ the set of all requests, and $\mathcal{P}^i$ the set of requests for content i. Suppose a network cache v (logically represented as v_i in path p_i for $i = 1, 2$) serves two requests (v_1, i_1, p_1) and (v_2, i_2, p_2), then there are two cases: (i) non-common requested content, i.e., $i_1 \neq i_2$; and (ii) common requested content, i.e., $i_1 = i_2$. In the following, we will focus on how to design optimal TTL policies for content placement in an edge cache network under these two cases.

3.4.1 Non-common Requested Contents

In this section, we consider the case where different users do not share content. Since there is no coupling between different requests (v, i, p), we can directly generalize the results for a particular path p in Section 3.3 to a tree network. Hence, given the utility maximization formulation in (3.7), we can directly formulate the optimization problem for MCDP as

$$
\begin{aligned}
\max \quad & \sum_{i\in\mathcal{D}} \sum_{p\in\mathcal{P}^i} \sum_{l=1}^{|p|} \psi^{|p|-l} U_{ip}(\lambda_{ip} h_{il}^{(p)}) \\
\text{s.t.} \quad & \sum_{i\in\mathcal{D}} \sum_{p:l\in\{1,\cdots,|p|\}} h_{il}^{(p)} \leq B_l, \quad \forall l \in V, \qquad (3.11\text{a}) \\
& \sum_{l=1}^{|p|} h_{il}^{(p)} \leq 1, \quad \forall i \in \mathcal{D}, p \in \mathcal{P}^i, \\
& 0 \leq h_{il}^{(p)} \leq 1, \quad \forall i \in \mathcal{D}, l \in \{1, \cdots, |p|\}, p \in \mathcal{P}^i.
\end{aligned}
$$

Proposition 2. *The optimization problem defined in (3.11) under MCDP has a unique global optimum.*

3.4.2 Common Requested Contents

Now consider the case where different users share the same content, e.g., there are two requests (v_1, i, p_1) and (v_2, i, p_2). Suppose that cache l is on both paths p_1 and p_2, where v_1 and v_2 request the same content i. If we cache separate copies on each path, results from the previous section apply. However, maintaining redundant copies in the same cache decreases efficiency. A simple way to deal with that is to only cache one copy of content i at l to serve both requests from v_1 and v_2. Though this reduces redundancy, it complicates the optimization problem.

In the following, we formulate a utility maximization problem for MCDP with TTL caches, where all users share the same requested contents $\mathcal{D}$.

$$\max \quad \sum_{i\in\mathcal{D}} \sum_{p\in\mathcal{P}^i} \sum_{l=1}^{|p|} \psi^{|p|-l} U_{ip}(\lambda_{ip} h_{il}^{(p)})$$

$$\text{s.t.} \quad \sum_{i\in\mathcal{D}} \left(1 - \prod_{p:j\in\{1,\cdots,|p|\}} (1 - h_{ij}^{(p)})\right) \le B_j, \quad \forall j \in V, \tag{3.12a}$$

$$\sum_{j\in\{1,\cdots,|p|\}} h_{ij}^{(p)} \le 1, \quad \forall i \in \mathcal{D}, p \in \mathcal{P}^i, \tag{3.12b}$$

$$0 \le h_{il}^{(p)} \le 1, \quad \forall i \in \mathcal{D}, j \in \{1, \cdots, |p|\}, p \in \mathcal{P}^i, \tag{3.12c}$$

where (3.12a) ensures that only one copy of content $i \in \mathcal{D}$ is cached at node j for all paths p that pass through node j. This is because the term $1 - \prod_{p:j\in\{1,\cdots,|p|\}}(1 - h_{ij}^{(p)})$ is the overall hit probability of content i at node j over all paths. (3.12b) is the cache capacity constraint and (3.12c) is the constraint from MCDP TTL cache policy as discussed in Section 3.3.2.

Example 1. *Consider two requests (v_1, i, p_1) and (v_2, i, p_2) with paths p_1 and p_2 intersecting at j. Let the corresponding path perspective hit probability be $h_{ij}^{(p_1)}$ and $h_{ij}^{(p_2)}$. Then the term inside outer summation of (3.12a) is $1-(1-h_{ij}^{(p_1)})(1-h_{ij}^{(p_2)})$, i.e., the hit probability of content i in j.*

Remark 2. *Note that we assume independence between different requests* (v, i, p) *in (3.12), e.g., in Example 1, if the insertion of content* i *in node* j *is caused by request* (v_1, i, p_1)*, when request* (v_2, i, p_2) *comes, it is not counted as a cache hit from its perspective. Our framework still holds if we follow the logical TTL MCDP on a path. However, in that case, the utilities will be larger than the one we consider here.*

Proposition 3. *The optimization problem (3.12) under MCDP is a non-convex optimization problem.*

In the following, we develop an optimization framework that handles the non-convexity issue in this optimization problem and provides a distributed solution. To this end, we first introduce the Lagrangian function

$$L(\boldsymbol{h}, \boldsymbol{\nu}, \boldsymbol{\mu}) = \sum_{i\in\mathcal{D}}\sum_{p\in\mathcal{P}^i}\sum_{l=1}^{|p|}\psi^{|p|-l}U_{ip}(\lambda_{ip}h_{il}^{(p)}) - \sum_{j\in V}\nu_j\Bigg(\sum_{i\in\mathcal{D}}\Bigg[1-\prod_{p:j\in\{1,\cdots,|p|\}}(1-h_{ij}^{(p)})\Bigg]-B_j\Bigg)$$
$$-\sum_{i\in\mathcal{D}}\sum_{p\in\mathcal{P}^i}\mu_{ip}\Bigg(\sum_{j\in\{1,\cdots,|p|\}}h_{ij}^{(p)}-1\Bigg),$$

where the Lagrangian multipliers (price vector and price matrix) are $\boldsymbol{\nu} = (\nu_j)_{j\in V}$, and $\boldsymbol{\mu} = (\mu_{ip})_{i\in\mathcal{D},p\in\mathcal{P}}$.

The dual function can be defined as

$$d(\boldsymbol{\nu}, \boldsymbol{\mu}) = \sup_{\boldsymbol{h}} L(\boldsymbol{h}, \boldsymbol{\nu}, \boldsymbol{\mu}),$$

and the dual problem is given as

$$\min_{\boldsymbol{\nu},\boldsymbol{\mu}} \quad d(\boldsymbol{\nu}, \boldsymbol{\mu}) = L(\boldsymbol{h}^*(\boldsymbol{\nu}, \boldsymbol{\mu}), \boldsymbol{\nu}, \boldsymbol{\mu}), \quad \text{s.t.} \quad \boldsymbol{\nu}, \boldsymbol{\mu} \geq \boldsymbol{0},$$

where the constraint is defined pointwise for $\boldsymbol{\nu}, \boldsymbol{\mu}$, and $\boldsymbol{h}^*(\boldsymbol{\nu}, \boldsymbol{\mu})$ is a function that maximizes the Lagrangian function for given $(\boldsymbol{\nu}, \boldsymbol{\mu})$, i.e.,

$$\boldsymbol{h}^*(\boldsymbol{\nu}, \boldsymbol{\mu}) = \arg\max_{\boldsymbol{h}} L(\boldsymbol{h}, \boldsymbol{\nu}, \boldsymbol{\mu}). \tag{3.13}$$

The dual function $d(\boldsymbol{\nu}, \boldsymbol{\mu})$ is always convex in $(\boldsymbol{\nu}, \boldsymbol{\mu})$ regardless of the convexity of the optimization problem (3.12) [21]. Therefore, it is always possible to iteratively solve the dual problem using

$$\begin{aligned} \nu_l[k+1] &= \nu_l[k] - \gamma_l \frac{\partial L(\boldsymbol{\nu}, \boldsymbol{\mu})}{\partial \nu_l}, \\ \mu_{ip}[k+1] &= \mu_{ip}[k] - \eta_{ip} \frac{\partial L(\boldsymbol{\nu}, \boldsymbol{\mu})}{\partial \mu_{ip}}, \end{aligned} \tag{3.14}$$

where γ_l and η_{ip} are the step sizes, and $\frac{\partial L(\boldsymbol{\nu}, \boldsymbol{\mu})}{\partial \nu_l}$ and $\frac{\partial L(\boldsymbol{\nu}, \boldsymbol{\mu})}{\partial \mu_{ip}}$ are the partial derivative of $L(\boldsymbol{\nu}, \boldsymbol{\mu})$ w.r.t. ν_l and μ_{ip}, respectively, satisfying

$$\begin{aligned} \frac{\partial L(\boldsymbol{\nu}, \boldsymbol{\mu})}{\partial \nu_l} &= -\left(\sum_{i \in \mathcal{D}} \left[1 - \prod_{p: l \in \{1, \cdots, |p|\}} (1 - h_{il}^{(p)}) \right] - B_l \right), \\ \frac{\partial L(\boldsymbol{\nu}, \boldsymbol{\mu})}{\partial \mu_{ip}} &= -\left(\sum_{j \in \{1, \cdots, |p|\}} h_{ij}^{(p)} - 1 \right). \end{aligned} \tag{3.15}$$

Sufficient and necessary conditions for the uniqueness of $\boldsymbol{\nu}, \boldsymbol{\mu}$ are given in [59]. The convergence of primal-dual algorithm consisting of (3.13) and (3.14) is guaranteed if the original optimization problem is convex. However, our problem is not convex. Nevertheless, we next show that the duality gap is zero, hence (3.13) and (3.14) converge to the globally optimal solution. To begin with, we introduce the following results

Theorem 5. *[94] (Sufficient Condition). If the price based function $\boldsymbol{h}^*(\boldsymbol{\nu}, \boldsymbol{\mu})$ is continuous at one or more of the optimal lagrange multiplier vectors $\boldsymbol{\nu}^*$ and $\boldsymbol{\mu}^*$, then* *the iterative algorithm consisting of (3.13) and (3.14) converges to the globally optimal solution.*

Theorem 6. *[94] If at least one constraint of (3.12) is active at the optimal solution, the condition in Theorem 5 is also a necessary condition.*

Hence, if we can show the continuity of $\boldsymbol{h}^*(\boldsymbol{\nu}, \boldsymbol{\mu})$ and that constraints (3.12) are active, then given Theorems 5 and 6, the duality gap is zero, i.e., (3.13) and (3.14) converge to the globally optimal solution.

Taking the derivative of $L(\boldsymbol{h}, \boldsymbol{\nu}, \boldsymbol{\mu})$ w.r.t. $h_{il}^{(p)}$ for $i \in \mathcal{D}$, $l \in \{1, \cdots, |p|\}$ and $p \in \mathcal{P}^i$, we have

$$\frac{\partial L(\boldsymbol{h}, \boldsymbol{\nu}, \boldsymbol{\mu})}{\partial h_{il}^{(p)}} = \psi^{|p|-l} \lambda_{ip} U_{ip}'(\lambda_{ip} h_{il}^{(p)}) - \mu_{ip} - \nu_l \Bigg(\prod_{\substack{q: q \neq p, \\ j \in \{1, \cdots, |q|\}}} (1 - h_{ij}^{(q)}) \Bigg). \tag{3.16}$$

Setting (3.16) equal to zero, we obtain

$$U_{ip}'(\lambda_{ip} h_{il}^{(p)}) = \frac{1}{\psi^{|p|-l} \lambda_{ip}} \Bigg(\nu_l \Bigg(\prod_{\substack{q: q \neq p, \\ j \in \{1, \cdots, |q|\}}} (1 - h_{ij}^{(q)}) \Bigg) + \mu_{ip} \Bigg). \tag{3.17}$$

Consider the utility function $U_{ip}(\lambda_{ip} h_{il}^{(p)}) = w_{ip} \log(1 + \lambda_{ip} h_{il}^{(p)})$, then $U_{ip}'(\lambda_{ip} h_{il}^{(p)}) = w_{ip}/(1 + \lambda_{ip} h_{il}^{(p)})$. Hence, from (3.17), we have

$$h_{il}^{(p)} = \frac{w_{ip} \psi^{|p|-l}}{\nu_l \Bigg(\prod_{\substack{q: q \neq p, \\ j \in \{1, \cdots, |q|\}}} (1 - h_{ij}^{(q)}) \Bigg) + \mu_{ip}} - \frac{1}{\lambda_{ip}}. \tag{3.18}$$

Lemma 2. *Constraints (3.12a) and (3.12b) cannot be both non-active, i.e., at least one of them is active.*

Proof. We prove this lemma by contradiction. Suppose both constraints (3.12a) and (3.12b) are non-active, i.e., $\boldsymbol{\nu} = (\mathbf{0})$, and $\boldsymbol{\mu} = (\mathbf{0})$. Then the optimization problem (3.11) achieves its maximum when $h_{il}^{(p)} = 1$ for all $i \in \mathcal{D}$, $l \in \{1, \cdots, |p|\}$ and

$p \in \mathcal{P}^i$. If so, then the left hand size of (3.12a) equals $|\mathcal{D}|$ which is much greater than B_l for $l \in V$, which is a contradiction. Hence, constraints (3.12a) and (3.12b) cannot be both non-active. □

From Lemma 2, we know that the feasible region for the Lagrangian multipliers satisfies $\mathcal{R} = \{\nu_l \geq 0, \mu_{ip} \geq 0, \nu_l + \mu_{ip} \neq 0, \forall i \in \mathcal{D}, l \in \{1, \cdots, |p|\}, p \in \mathcal{P}^i\}$.

Theorem 7. *The hit probability $h_{il}^{(p)}$ given in (3.18) is continuous in ν_l and μ_{ip} for all $i \in \mathcal{D}$, $l \in \{1, \cdots, |p|\}$ and $p \in \mathcal{P}^i$ in the feasible region $\mathcal{R}$.*

Proof. See appendix B.3. □

Remark 3. *Note that similar arguments (by using Lemma 2) hold true for various other choices of utility functions such as: β- fair utility functions (Section 2[88]). Therefore, the primal-dual algorithm consisting of (3.13) and (3.14) converges to the globally optimal solution for a wide range of utility functions.*

Algorithm 1 Primal-Dual Algorithm

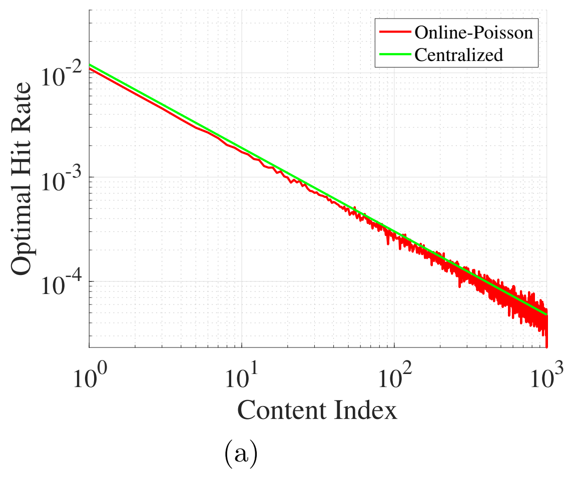
```
Input: $\forall \boldsymbol{\nu_0}, \boldsymbol{\mu_0} \in \mathcal{R}$ and $\boldsymbol{h}_0$
Output: The optimal hit probabilities $\boldsymbol{h}$
Step 0: $t = 0$, $\boldsymbol{\nu}[t] \leftarrow \boldsymbol{\nu}_0$, $\boldsymbol{\mu}[t] \leftarrow \boldsymbol{\mu}_0$, $\boldsymbol{h}[t] \leftarrow \boldsymbol{h}_0$
Step $t \geq 1$
while Equation (3.15) $\neq 0$ do
    First, compute $\boldsymbol{h}_{il}^{(p)}[t+1]$ for $i \in \mathcal{D}$, $l \in \{1, \cdots, |p|\}$ and $p \in \mathcal{P}^i$ through (3.18);
    Second, update $\nu_l[t+1]$ and $\mu_{ip}[t+1]$ through (3.14) given $\boldsymbol{h}[t+1]$, $\boldsymbol{\nu}[t]$ and
$\boldsymbol{\mu}[t]$ for $l \in V$, $i \in \mathcal{D}$ and $p \in \mathcal{P}^i$
```

Algorithm 1 summarizes the details of this algorithm.

3.5 Results and Discussion

3.5.1 Results for Single Cache

In our studies, we consider a Zipf popularity distribution with $\alpha = 0.8$, $n = 1000$ and $B = 100$.



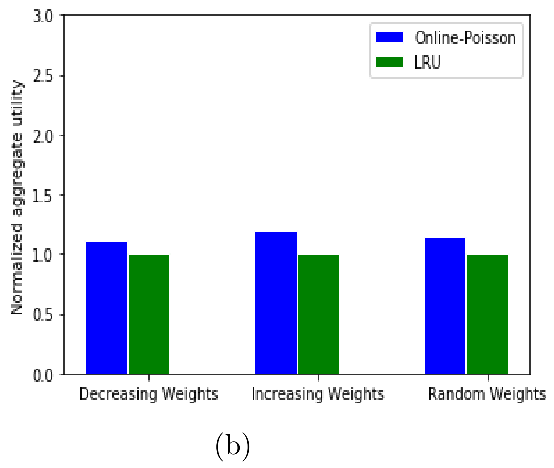


Figure 3.4: (a) Accuracy of Poisson online approximation for Generalized Pareto inter-requests. (b) Trace-driven utility comparison for Poisson online approximation and LRU caching policies.

Accuracy of Poisson Approximation: We first apply the online algorithm (3.4) to a workload where requests are described by a stationary request process under a generalized Pareto distribution with shape parameter $k_i = 0.48$. The performance is shown in Figure 3.4 (a) . It is clear that the approximation is accurate. Furthermore, it has been theoretically characterized in [99] that for any given generalized Pareto model with finite variance, the exponential approximation that minimizes the K-L divergence between these two distributions has the same mean as that of the generalized Pareto distribution, i.e. $\lambda_i = (1 - k_i)/\sigma_i$. The estimator we use in our online algorithm (3.4), i.e., $1/\hat{\lambda}_{iP}$, is an unbiased estimator of mean inter-request time of the generalized Pareto arrival process, thus explaining the better performance of our Poisson approximation in accordance with the theoretical results provided in [99]. Moreover, we observe that as k_i decreases, accuracy improves. However, this approximation performs poorly when $k_i > 0.5$ since the generalized Pareto distribution has infinite variance for $k_i > 0.5$. We also find the Poisson online algorithm to be accurate for other IRT distributions such as Weibull and 2-state MMPP.

Trace Driven Simulation: We compare the performance of LRU to that of Poisson approximate online algorithm through trace-driven simulation. We use requests from a web access trace collected from a gateway router at IBM research lab [101]. The trace contains 3.5×10^6 requests with a content catalog of size $n = 5638$. We consider a cache size $B = 1000$. The utility function involves content weights, w_i, associated with each content i. Classical cache replacement policies such as LRU are oblivious to content weights. However, the Poisson approximation based online algorithm updates the TTL timer by considering the content weight at each time step. Thus the Poisson approximation based online algorithm is more robust to variation in content weights. Figure 3.4 (b) compares the performance of online Poisson algorithm to that of LRU across different sets of content weights, i.e. we consider the following three cases: (a) $w_i = \lambda_i$ (decreasing weights and decreasing request rates) (b) $w_i = 1/\lambda_i$ (increasing weights and decreasing request rates) (c) $w_i = \texttt{rand}(0, 1)$ (random weights and decreasing request rates). Let U_P and U_L denote the aggregate content utility for online Poisson algorithm and for LRU policy, respectively. We normalize both utilities w.r.t. LRU policy as U_P/U_L and $U_L/U_L = 1$, respectively. From Figure 3.4 (b), it is clear that in each case online Poisson algorithm performs better than LRU, i.e. online Poisson algorithm achieves larger aggregate utility as compared to the LRU policy.

3.5.2 Results for Linear Cache Network

We validate our analytical results with simulations for MCDP. We consider a three-node path with cache capacities $B_l = 10$, $l = 1, 2, 3$. The total number of unique contents considered in the system is $n = 100$. We consider a Zipf popularity distribution with parameter $\alpha = 0.8$. W.l.o.g., we consider log based utility function[2]

[2]One can also choose $U_i(x) = \lambda_i \log x$. However, $U_i(x)$ evaluated at $x = 0$ becomes negative infinity, which may produce undesired results while comparing the performance of MCDP with other caching policies.

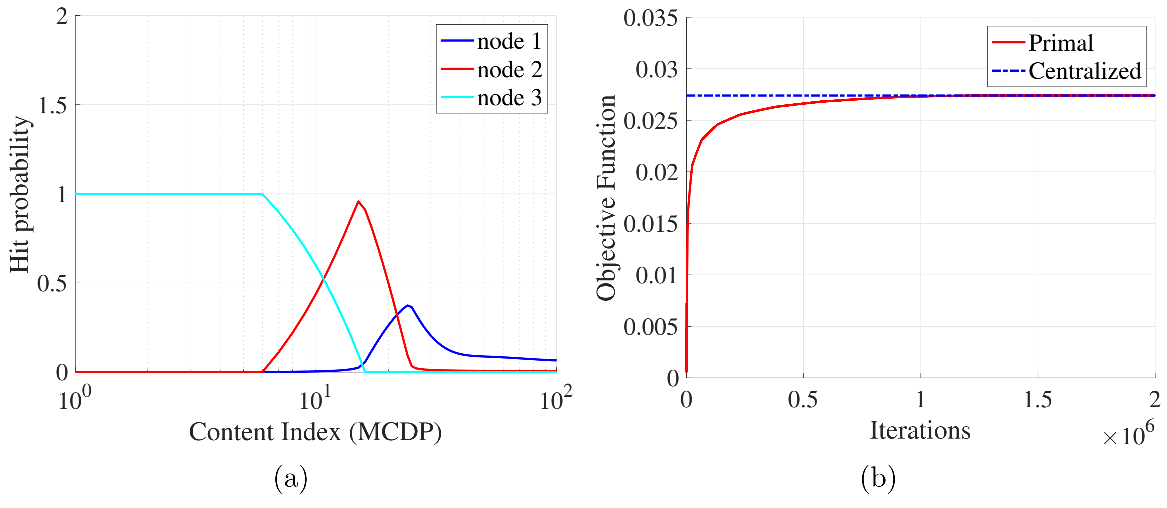


Figure 3.5: (a) Hit probability for MCDP in a three-node path; (b) Convergence of primal algorithm.

$U_i(x) = \lambda_i \log(1 + x)$ [95], and discount factor $\psi = 0.1$. We assume that requests arrive according to a Poisson process with aggregate request rate $\Lambda = 1$.

We first solve optimization problem (3.7) using a Matlab routine `fmincon`. From Figure 3.5 (a), we observe that popular contents are assigned higher hit probabilities at cache node 3, i.e. at the edge cache closest to the user as compared to other caches. The optimal hit probabilities assigned to popular contents at other caches are almost negligible. However, the assignment is reversed for moderately popular contents. For non-popular contents, the optimal hit probabilities at cache node 1 (closest to origin server) are the highest.

We then implement the primal algorithm given in (3.10), where we take the following penalty functions [88] $C_l(x) = \max\{0, x - B_l \log(B_l + x)\}$ and $\tilde{C}_i(x) = \max\{0, x - \log(1 + x)\}$. Figure 3.5 (b) shows that the primal algorithm successfully converges to the optimal solution.

We define an upper bound (UB) on optimal aggregate utility by removing (3.7b) and solving the optimization problem (3.7). We also compare the performance of MCDP to other policies such as K-LRU (K=3), K-LRU with big cache abstraction:

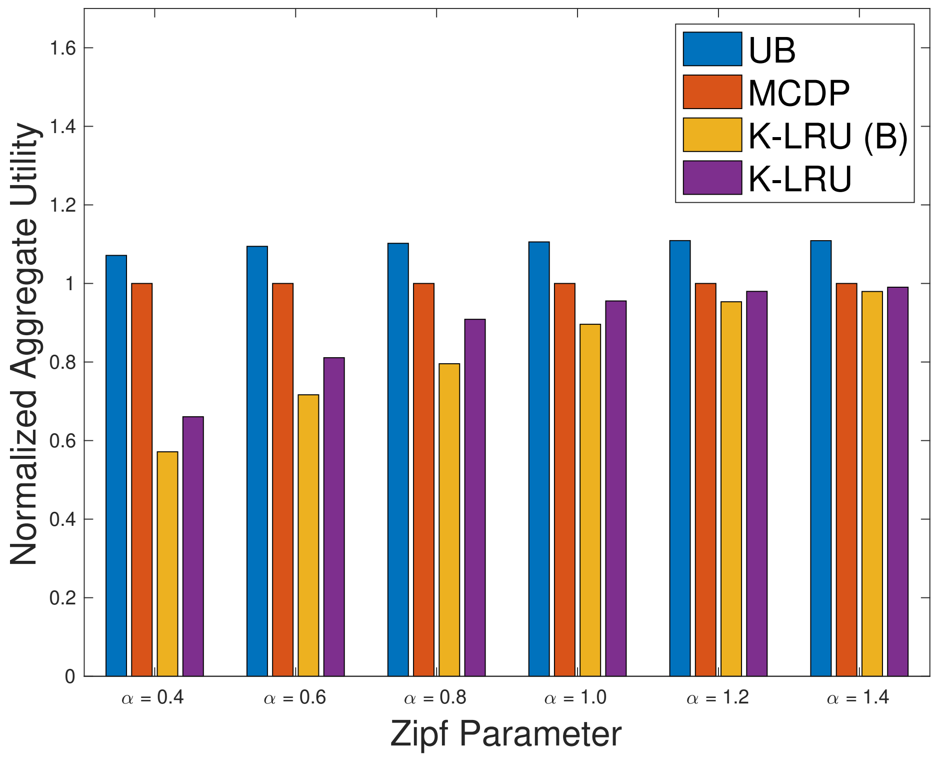


Figure 3.6: Normalized optimal aggregated utilities in a three-node path.

K-LRU(B) and the UB based bound. We plot the relative performance w.r.t. the optimal aggregated utilities of all above policies, normalized to that under MCDP shown in Figure 3.6. We observe that MCDP significantly outperforms K-LRU and K-LRU(B) for small and moderate values of Zipf parameter. Furthermore, the performance gap between UB and MCDP increases in the Zipf parameter.

3.5.3 Results for General Cache Network (Non-Common Requested Contents)

We consider a three-layer cache network shown in Figure 3.1 with node set $\{1, \cdots, 7\}$, which is consistent with the YouTube video delivery system [83, 86]. Nodes 1-4 are edge caches, and node 7 is tertiary cache. There exist four paths $p_1 = \{1, 5, 7\}$, $p_2 = \{2, 5, 7\}$, $p_3 = \{3, 6, 7\}$ and $p_4 = \{4, 6, 7\}$. Each edge cache serves requests for 100 distinct contents, and cache size is $B_v = 10$ for $v \in \{1, \cdots, 7\}$. Assume that content follows a Zipf distribution with parameter $\alpha_1 = 0.2$, $\alpha_2 = 0.4$, $\alpha_3 = 0.6$ and

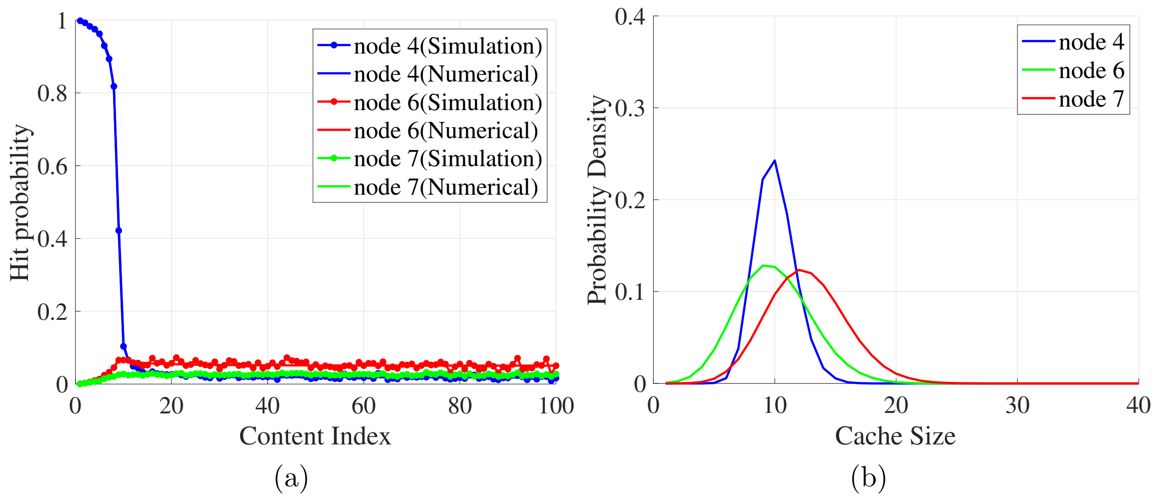


Figure 3.7: (a) Hit probability; (b) Cache size, of MCDP under three-layer edge network where each path requests distinct contents.

$\alpha_4 = 0.8$, respectively. We consider utility function $U_{ip}(x) = \lambda_{ip}\log(1+x)$, where λ_{ip} is the request arrival rate for content i on path p, and requests are described by a Poisson process with $\Lambda_p = 1$ for $p = 1, 2, 3, 4$. The discount factor $\psi = 0.1$.

Figure 3.7 shows results for path $p_4 = \{4, 6, 7\}$. From Figure 3.7 (a), we observe that our algorithm yields the exact optimal and empirical hit probabilities under MCDP. Figure 3.7 (b) shows the distribution of the number of contents in the cache[3] in the cache network. As expected, the density is concentrated around their corresponding cache sizes. Similar trends exist for paths p_1, p_2 and p_3, hence are omitted here.

3.5.4 Results for General Cache Network (Common Requested Contents)

We evaluate the performance of primal-dual Algorithm 1 on a three-layer binary tree cache network. We assume that there are 100 unique contents in the system requested across four paths. The cache size is $B_v = 10$ for $v = 1, \cdots, 7$. We consider

[3]The constraint (3.11a) in problem (3.11) is on average cache occupancy. However it can be shown that if $n \to \infty$ and B_l grows in sub-linear manner, the probability of violating the target cache size B_l becomes negligible [30].

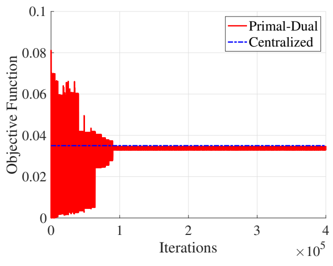


Figure 3.8: Convergence of Primal-Dual algorithm.

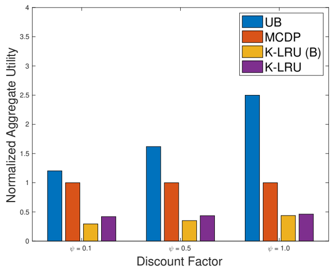


Figure 3.9: Optimal aggregated utilities under common requested contents.

Topology	α	MCDP	UB	% Gap
Grid	0.8	0.0923	0.1043	11.50
Grid	1.2	0.3611	0.4016	10.08
Lollipop	0.8	0.0908	0.1002	9.38
Lollipop	1.2	0.3625	0.4024	9.91

Figure 3.10: Optimal aggregate utilities under various network topologies.

the utility function $U_{ip}(x) = \lambda_{ip} \log(1+x)$, and the popularity distribution over these contents is Zipf with parameter $\alpha = 0.8$. W.l.o.g., the aggregate request arrival rate is one. The discount factor $\psi = 0.1$.

Convergence of Primal-Dual Algorithm 1: We solve the optimization problem in (3.12) using a Matlab routine `fmincon`. Then we implement our primal-dual algorithm given in Algorithm 1. Results for aggregate optimal utility are presented in Figure 3.8. It is clear that the primal-dual algorithm successfully converges to the optimal solution.

Comparison with Other Caching Policies: While classical cache eviction policies such as LRU provide good performance and are easy to implement, Garetto et al. [40] showed that K-LRU[4] provides significant improvement over LRU even for very small value of K. Furthermore, Ramadan et al. [83] proposed K-LRU with big cache abstraction (K-LRU(B)) to effectively utilize resources in a hierarchical network of cache servers. We also define an upper bound (UB) on optimal aggregate utility by removing (3.12b) and solving the optimization problem (3.12). We compare the performance of MCDP to K-LRU, K-LRU(B) and UB in Figure 3.9. We plot the relative performance w.r.t. the optimal aggregated utilities of all above policies,

[4]K-LRU adds $K-1$ meta-caches ahead of the real cache. Only "popular" contents (requested at least $K-1$ times) are stored in real cache.

normalized to that under MCDP. We again observe a huge gain of MCDP w.r.t. K-LRU and K-LRU(B) across all values of discount factor. However, the performance gap between MCDP and UB increases with an increase in the value of discount factor.

Different Network Topologies: We consider two other network topologies: Grid and lollipop. *Grid* is a two-dimensional square grid while a (a,b) *lollipop* network is a complete graph of size a, connected to a path graph of length b. Denote the network as $G = (V, E)$. For grid, we consider $|V| = 16$, while we consider a $(3, 4)$ lollipop topology with $|V| = 7$ and clique size 3. The library contain $|\mathcal{D}| = 100$ unique contents. Each node has access to a subset of contents in the library. We assign a weight to each edge in E, selected uniformly from the interval $[1, 20]$. Next, we generate a set of requests in G as described in [49]. To ensure that paths overlap, we randomly select a subset $\tilde{V} \subset V$ nodes to generate requests. Each node in $\tilde{V}$ can generate requests for contents in $\mathcal{D}$ following a Zipf distribution with parameter $\alpha = 0.8$. Requests are then routed over the shortest path between the requesting node in $\tilde{V}$ and the node in V that caches the content. Again, we assume that the aggregate request rate at each node in $\tilde{V}$ is one and the discount factor is $\psi = 0.1$.

We evaluate the performance of MCDP over the graphs across various Zipf parameter in Figure 3.10. It is clear that for both network topologies, aggregate utility obtained from our TTL-based framework with MCDP policy is larger for large zipf parameters as compared to small zipf parameters. With an increase in Zipf parameter, the difference between request rates of popular and less popular contents increases. The aggregate request rate over all contents is the same in both cases. Thus popular contents get higher request rates which in turn yields higher aggregate utility. However, the performance gap between UB and MCDP is around one tenth in both cases and is not affected much by the Zipf parameter.

3.6 Summary

We began this chapter by asking: *How do we build caching networks that implement differential services?* We have answered this question by developing optimal timer-based TTL polices for content placement in cache networks through a unified optimization approach. We formulated a general utility maximization framework, which is non-convex in general. We identified the nonconvexity issue and proposed efficient distributed algorithm to solve it. We proved that the distributed algorithms converge to the globally optimal solutions. We showed the efficiency of these algorithms through numerical studies.

CHAPTER 4
JOINTLY COMPRESSING AND CACHING DATA IN WIRELESS SENSOR NETWORKS

The theory we developed in Chapter 3 for cache networks can easily be extended to other settings such as a Wireless Sensor Network (*WSN*). We now consider a WSN comprised of a large number of sensors and routers. The end sensors usually generate data, and routers provide resources (such as caching and compression resources) on the route from end sensors to the sink node (where end users request for data). We represent the WSN as a directed graph $G = (V, E)$. An illustrative example for data generation and request propagation in a tree-structured WSN is depicted in Figure 4.1. Each router has a cache to store data for compression. Let $\mathcal{K} \subset V$ be the set of end sensor nodes generating data with $|\mathcal{K}| = K$. Furthermore, we assume that each node j that receives data i from an end node k can compress it with a reduction ratio[1] $\delta_{ij}^{(k)}$, where $0 < \delta_{ij}^{(k)} \leq 1$, $\forall k, j$. We assume that sensor $k \in \mathcal{K}$ continuously generates data, which will be active for a time interval W and may be requested by analysts (users). If there is no request for this data in that time interval, the generated data becomes inactive and is discarded from the system. The generated data is compressed and cached along the path between the end sensor and sink node when a request is made for active data. We consider TTL-routers in the WSN G, where each data has its own timer. We follow the MCDP cache replication strategy

[1]defined as the ratio of the volume of the output data to the volume of input data at any node. We consider the compression that only reduces the quality of data (e.g. remove redundant information), but the total number of distinct data in the system remains the same.

defined in Chapter 3 and also defined in Section 4.1.2. We associate each data with a utility function, a function of cache hit probability.

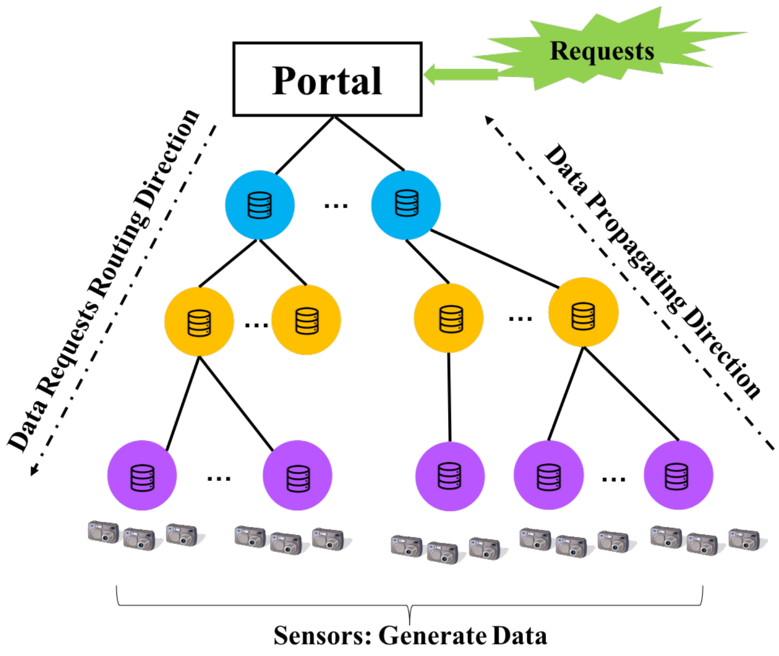


Figure 4.1: An illustrative example for data generating, request and propagating in a wireless sensor network.

4.1 System Model

We consider a WSN comprised a large number of sensors and routers. The end sensors usually generate data, and routers provide resources (such as caching and compression resources) on the route from end sensors to the sink node. We represent the WSN as a directed graph $G = (V, E)$. An illustrative example for data generation and request propagation in a tree-structured WSN is depicted in Figure 4.1.

4.1.1 TTL-Router

Each router has a cache to store data for compression. Denote by B_v the cache capacity at node $v \in V$. Let $\mathcal{K} \subset V$ be the set of end sensor nodes generating data with $|\mathcal{K}| = K$. Furthermore, we assume that each node j that receives data i from an end node k can compress it with a reduction ratio[2] $\delta_{ij}^{(k)}$, where $0 < \delta_{ij}^{(k)} \leq 1$, $\forall k, j$.

Consider the cache at router j. Each data i is associated with a timer T_{ij}. When we focus on router j, we omit the subscript j. Consider the event when data i is requested. There are two cases: (i) if data i is not in the cache, data i is inserted into the cache and its timer is set to T_i; (ii) if data i is in the cache, its timer is reset to T_i. The timer decreases at a constant rate and the data is evicted once its timer expires.

4.1.2 Data Generation and Requests

We assume that sensor $k \in \mathcal{K}$ continuously generates data, which will be active for a time interval W and may be requested by analysts (users). If there is no request for this data in that time interval, the generated data becomes inactive and is discarded from the system. The generated data is compressed and cached along the path between the end sensor and sink node when a request is made for active data. Thus the total number of paths is $|\mathcal{K}| = K$, hence, w.l.o.g., $\mathcal{K}$ is also used to denote the set of all paths.

We consider TTL-routers in the WSN G, where each data has its own timer. Suppose data i is requested and routed along path p. There are two cases: (i) data i is not in any cache along path p, in which case data i is generated from the end sensor and inserted into the first TTL-router (denoted by 1)[3] on the path. Its timer is set to T_{i1}; (ii) if data i is in TTL-router l along path p, we consider the following simple strategy [85]

[2]defined as the ratio of the volume of the output data to the volume of input data at any node. We consider the compression that only reduces the quality of data (e.g. remove redundant information), but the total number of distinct data in the system remains the same.

[3]Since we consider path p, for simplicity, we move the dependency on p and v, denote it as $1, \cdots, L$ directly.

- **Move Copy Down with Push (MCDP)**: data i is moved to TTL-router $l+1$ preceding TTL-router l in which i is found, and the timer at TTL-router $l+1$ is set to $T_{i(l+1)}$. If timer T_{il} expires, data i is pushed one TTL-router back to TTL-router $l-1$ and the timer is set to $T_{i(l-1)}$.

4.1.3 Utility Function

We associate with each data $i \in \mathcal{D}$ a utility function $U_i : [0,1] \to \mathbb{R}$ that is a function of hit probability h_i. $U_i(\cdot)$ is assumed to be increasing, continuously differentiable, and strictly concave. In particular, for our numerical studies, we focus on the widely used β-fair utility functions [88] given by

$$U_i(h) = \begin{cases} w_i \frac{h^{1-\beta}}{1-\beta}, & \beta \geq 0, \beta \neq 1; \\ w_i \log h, & \beta = 1, \end{cases} \tag{4.1}$$

where $w_i > 0$ denotes a weight associated with data i.

4.2 Optimization Formulation

In a WSN, each end sensor generates a sequence of data that analysts are interested in. Different end sensors may generate different types of data, i.e., there is no common data sharing between different end sensors.

W.l.o.g., we consider a particular end sensor k and denote the path from k to the sink as $p = (1, \cdots, |p|)$, where TTL-router $|p|$ is the sink node that serves the requests and TTL-router 1 is the end sensor that generates the data. Let the set of data generated by end sensor k be $\mathcal{D}^{(p)}$, where requests for data $i \in \mathcal{D}^{(p)}$ follow a Poisson process with rate λ_i.

Let $h_{ij}^{(p)}, T_{ij}^{(p)}$ denote the hit probability and TTL timer associated with data $i \in \mathcal{D}^{(p)}$ at node $j \in \{1, \cdots, |p|\}$, respectively. Let $\boldsymbol{h}_i^{(p)} = (h_{i1}^{(p)}, \cdots, h_{i|p|}^{(p)})$, $\boldsymbol{\delta}_i^{(p)} =$

$(\delta_{i1}^{(p)}, \cdots, \delta_{i|p|}^{(p)})$ and $\boldsymbol{T}_i^{(p)} = (T_{i1}^{(p)}, \cdots, T_{i|p|}^{(p)})$. Let $\boldsymbol{h} = (\boldsymbol{h}_i^{(p)})$, $\boldsymbol{\delta} = (\boldsymbol{\delta}_i^{(p)})$ and $\boldsymbol{T} = (\boldsymbol{T}_i^{(p)})$ for $i \in \mathcal{D}^{(p)}$ and $p \in \mathcal{K}$.

4.2.1 Utilities

The overall utility for data i fetched over path p is

$$\sum_{j=1}^{|p|} \psi^{|p|-j} U_i^{(p)}\left(h_{ij}^{(p)} \prod_{l=1}^{j} \delta_{il}^{(p)}\right),$$

where $0 < \psi \leq 1$ is a discount factor capturing the data utility degradation along the request route. Here utilities not only capture hit probabilities but also characterize data quality degradation due to compression along the path.

4.2.2 Costs

We consider cost, for example the delay to retrieve the data to a user, of routing the data along the path, which includes the cost to forward data to routers that caches it, the cost to search for the data along the path, and the cost to fetch cached data to analysts that sent the requests. Again, we assume that the per hop cost to transfer (search) data along the path is a function $c_f(\cdot)$ ($c_s(\cdot)$) of hit probabilities and compression ratios.

4.2.2.1 Forwarding Costs

Suppose a hit for data i occurs on TTL-router $j \in \{1, \cdots, |p|\}$, then the total cost to forward data i along p is

$$\sum_{j=1}^{|p|} \lambda_i \cdot j \cdot c_f\left(h_{ij}^{(p)} \prod_{l=1}^{j} \delta_{il}^{(p)}\right).$$

4.2.2.2 Search Costs

Given a hit for data i on TTL-router $j \in \{1, \cdots, |p|\}$, the total cost to search for data i along p is

$$\sum_{j=1}^{|p|} \lambda_i \cdot (|p| - j + 1) \cdot c_s(h_{ij}^{(p)}).$$

4.2.2.3 Fetching Costs

Upon a hit for data i on TTL-router $j \in \{1, \cdots, |p|\}$, the total cost to fetch data i along p is

$$\sum_{j=1}^{|p|} \lambda_i \cdot (|p| - j + 1) \cdot c_f\left(h_{ij}^{(p)} \prod_{l=1}^{j} \delta_{il}^{(p)}\right).$$

4.2.3 Hit Probability and Timer-based Policies

The mapping between hit probabilities and timers for different cache replication strategies was established in Chapter 3. Using results from Chapter 3, we obtain the following expressions for corresponding data timers at a sensor j along path p.

$$\begin{aligned} T_{i1}^{(p)} &= \frac{1}{\lambda_i} \log\left(1 + \frac{h_{i1}^{(p)}}{1 - \sum_{j\in\{1,\cdots,|p|\}} h_{ij}^{(p)}}\right), \\ T_{ij}^{(p)} &= \frac{1}{\lambda_i} \log\left(1 + \frac{h_{ij}^{(p)}}{h_{i(j-1)}^{(p)}}\right), \quad j = 2, \cdots, |p|. \end{aligned}$$

Note that

$$\sum_{j\in\{1,\cdots,|p|\}} h_{ij}^{(p)} \leq 1,$$

must hold during the mapping.

4.2.4 Optimization Formulation

Our objective is to determine a feasible TTL policy and compression ratio for data management in a WSN to maximize the difference between utilities and costs, i.e.,

$$
\begin{aligned}
F(\boldsymbol{h}, \boldsymbol{\delta}) = &\sum_{p\in\mathcal{K}} \sum_{i\in\mathcal{D}^{(p)}} \Bigg\{ \sum_{j=1}^{|p|} \psi^{|p|-j} U_i^{(p)}\Bigg(h_{ij}^{(p)} \prod_{l=1}^{j} \delta_{il}^{(p)}\Bigg) \\
&- \sum_{j=1}^{|p|} \lambda_i \cdot j \cdot c_f\Bigg(h_{ij}^{(p)} \prod_{l=1}^{j} \delta_{il}^{(p)}\Bigg) - \sum_{j=1}^{|p|} \lambda_i \cdot (|p|-j+1) \cdot c_s(h_{ij}^{(p)}) \\
&- \sum_{j=1}^{|p|} \lambda_i \cdot (|p|-j+1) \cdot c_f\Bigg(h_{ij}^{(p)} \prod_{l=1}^{j} \delta_{il}^{(p)}\Bigg)\Bigg\} \\
= &\sum_{p\in\mathcal{K}} \sum_{i\in\mathcal{D}^{(p)}} \Bigg\{ \sum_{j=1}^{|p|} \psi^{|p|-j} U_i^{(p)}\Bigg(h_{ij}^{(p)} \prod_{l=1}^{j} \delta_{il}^{(p)}\Bigg) \\
&- \Bigg[\sum_{j=1}^{|p|} \lambda_i (|p|+1) c_f\Bigg(h_{ij}^{(p)} \prod_{l=1}^{j} \delta_{il}^{(p)}\Bigg) \\
&+ \sum_{j=1}^{|p|} \lambda_i (|p|-j+1) c_s(h_{ij}^{(p)})\Bigg]\Bigg\}.
\end{aligned}
$$

Hence, the optimal TTL policy and compression ratio for MCDP should solve the following optimization problem:

$$
\begin{aligned}
\max \quad & F(\boldsymbol{h}, \boldsymbol{\delta}) \\
\text{s.t.} \quad & \sum_{p:l\in p} \sum_{i\in\mathcal{D}^{(p)}} h_{il}^{(p)} \prod_{j=1}^{\mathcal{I}(l,p)} \delta_{ij}^{(p)} \leq B_l, \quad \forall l \in V, \\
& c_c\Bigg(\sum_{i\in\mathcal{D}^{(p)}} \sum_{l=1}^{|p|} \prod_{j=1}^{l} \delta_{ij}^{(p)}\Bigg) \leq O^{(p)}, \quad \forall p \in \mathcal{K}, && (4.3a) \\
& \sum_{j\in\{1,\cdots,|p|\}} h_{ij}^{(p)} \leq 1, \quad \forall i \in \mathcal{D}^{(p)}, \forall p \in \mathcal{K}, && (4.3b) \\
& 0 \leq h_{ij}^{(p)} \leq 1, \quad \forall i \in \mathcal{D}^{(p)}, \forall p \in \mathcal{K}, j \in \{1, \cdots, |p|\}, \\
& 0 < \delta_{ij}^{(p)} \leq 1, \quad \forall i \in \mathcal{D}^{(p)}, \forall p \in \mathcal{K}, j \in \{1, \cdots, |p|\},
\end{aligned}
$$

where $\mathcal{I}(l,p)$ is the index of router j on path p and constraint (4.3a) is the energy available on path p to transmit the compressed data, and $c_c(\cdot)$ is the per unit energy consumption function for data transmission. Constraint (4.3b) is included in the formulation due to the mapping between hit probabilities and timers as discussed in Section 4.2.3.

It is easy to check that (4.3) is a non-convex problem. In the following, we transform (4.3) into a convex problem through Boyd's method (Section 4.5 [21]).

4.2.4.1 Convex Transformation

First, we define two new sets of variables for $i \in \mathcal{D}^{(p)}$, $l \in \{1, \cdots, |p|\}$ and $p \in \mathcal{K}$ as follows:

$$\log h_{ij}^{(p)} \triangleq \sigma_{ij}^{(p)}, \quad i.e., \quad h_{ij}^{(p)} = e^{\sigma_{ij}^{(p)}},$$
$$\log \delta_{ij}^{(p)} \triangleq \tau_{ij}^{(p)}, \quad i.e., \quad \delta_{ij}^{(p)} = e^{\tau_{ij}^{(p)}},$$

and denote $\boldsymbol{\sigma}_i^{(p)} = (\sigma_{i1}^{(p)}, \cdots, \sigma_{ip}^{(p)})$, $\boldsymbol{\tau}_i^{(p)} = (\tau_{i1}^{(p)}, \cdots, \tau_{ip}^{(p)})$ and $\boldsymbol{\sigma} = (\boldsymbol{\sigma}_i^{(p)})$, $\boldsymbol{\tau} = (\boldsymbol{\tau}_i^{(p)})$ for $i \in \mathcal{D}^{(p)}$ and $p \in \mathcal{K}$.

Then the objective function $F(\boldsymbol{h}, \boldsymbol{\delta})$ can be transformed into

$$\begin{aligned} F(\boldsymbol{\sigma}, \boldsymbol{\tau}) = \sum_{p \in \mathcal{K}} \sum_{i \in \mathcal{D}^{(p)}} \Bigg\{ & \sum_{j=1}^{|p|} \psi^{|p|-j} U_i^{(p)} \left(e^{\sigma_{ij}^{(p)} + \sum_{l=1}^{j} \tau_{il}^{(p)}} \right) \\ & - \Bigg[\sum_{j=1}^{|p|} \lambda_i (|p|+1) c_f \left(e^{\sigma_{ij}^{(p)} + \sum_{l=1}^{j} \tau_{il}^{(p)}} \right) \\ & + \sum_{j=1}^{|p|} \lambda_i (|p|-j+1) c_s \left(e^{\sigma_{ij}^{(p)}} \right) \Bigg] \Bigg\}. \end{aligned}$$

We transform the constraints in a similar manner. Then we obtain the following transformed optimization problem

$$\max \quad F(\boldsymbol{\sigma}, \boldsymbol{\tau})$$

$$\text{s.t.} \quad \sum_{p:l\in p}\sum_{i\in\mathcal{D}^{(p)}} e^{\sigma_{il}^{(p)}+\sum_{j=1}^{\mathcal{I}(l,p)}\tau_{ij}^{(p)}} \le B_l, \quad \forall l \in V,$$

$$c_c\left(\sum_{i\in\mathcal{D}^{(p)}}\sum_{l=1}^{|p|} e^{\sum_{j=1}^{l}\tau_{ij}^{(p)}}\right) \le O^{(p)}, \quad \forall p \in \mathcal{K}, \tag{4.4a}$$

$$\sum_{j\in\{1,\cdots,|p|\}} e^{\sigma_{ij}^{(p)}} \le 1, \quad \forall i \in \mathcal{D}^{(p)}, \forall p\in\mathcal{K}, \tag{4.4b}$$

$$\sigma_{ij}^{(p)} \le 0, \quad \forall i \in \mathcal{D}^{(p)}, \forall p \in \mathcal{K}, j\in\{1,\cdots,|p|\}, \tag{4.4c}$$

$$\tau_{ij}^{(p)} \le 0, \quad \forall i \in \mathcal{D}^{(p)}, \forall p \in \mathcal{K}, j\in\{1,\cdots,|p|\}, \tag{4.4d}$$

where $\mathcal{I}(l,p)$ is the index of router l on path p.

Lemma 3. $U_i\left(e^{\sum_{k=1}^{n}x_k}\right)$ *is a concave function for* $\beta \ge 1$ *where* $U_i(\cdot)$ *is defined in* (D.3).

Proof. See appendix C.1. □

Theorem 8. *The transformed problem in (4.4) is convex* $\boldsymbol{\sigma}$ *and* $\boldsymbol{\tau}$*, when we consider the* β*-utility function with* $\beta \ge 1$ *and increasing convex cost functions* $c_f(\cdot)$, $c_s(\cdot)$ *and* $c_c(\cdot)$.

Proof. It is easy to check that the objective function in (4.4) is subject to convex inequality constraints. In particular, constraints (4.4c) and (4.4d) are affine convex functions. Inequality in constraint (4.4b) is convex due to convex composition under an affine mapping. Since the function $c_c(x)$ is convex and non-decreasing, by composition property (Section 3.2.4 [21]), constraint (4.4a) is also convex. Thus the feasible region in (4.4) is convex. A direct application of Lemma 3 yields the concavity condition for the objective $F(\boldsymbol{\sigma},\boldsymbol{\tau})$. □

4.3 Results and Discussion

First, we consider a binary tree network with seven nodes, where $\mathcal{K} = \{1,2,3,4\}$. There are 4 leaf nodes, each is connected to 30 sensors. We assume that each sensor

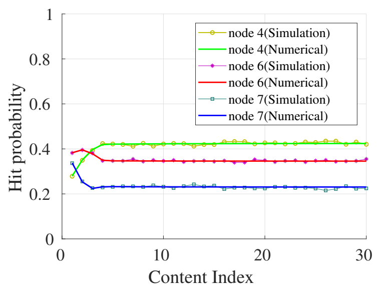


Figure 4.2: Hit probability of MCDP under seven-node tree WSN.

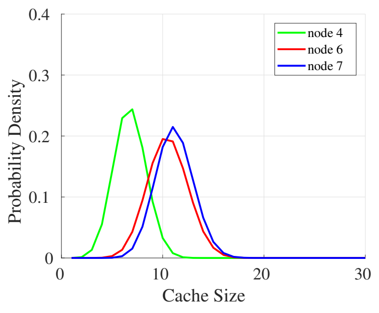


Figure 4.3: Cache size of MCDP under seven-node tree WSN.

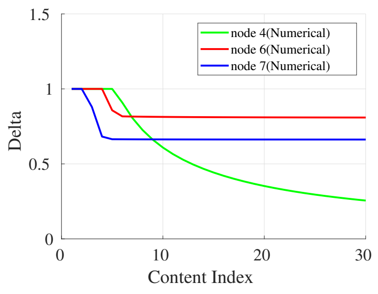


Figure 4.4: Compression ratio of MCDP under a seven-node tree WSN.

continuously generates content that are active for one time unit. Hence the paths are $p_1 = \{1, 5, 7\}$, $p_2 = \{2, 5, 7\}$, $p_3 = \{3, 6, 7\}$ and $p_4 = \{4, 6, 7\}$. Also let $B_v = 6$ for all leaf nodes $v \in \{1, \cdots, 4\}$, and $B_v = 10$ for nodes $v = 5, 6, 7$. Furthermore, for each leaf node, the content gathered from its sensors follows a Zipf distribution with parameters $\alpha_1 = 0.2$, $\alpha_2 = 0.4$, $\alpha_3 = 0.6$ and $\alpha_4 = 0.8$, respectively. For simplicity, we consider linear cost functions with coefficients 0.003 for $c_f(\cdot)$ and $c_s(\cdot)$, and 1 for $c_c(\cdot)$. The total energy constraint is set to $O = 40$ for all paths. We consider the log utility function $U_i^{(k)}(x) = \lambda_i^{(k)} \log x$, where $\lambda_i^{(k)}$ is the request arrival rate for content i from sensor k. W.l.o.g., we assume the total arrival rate at each leaf node is 1, hence $\lambda_i^{(k)}$ equals to the content popularity.

Results for path p_4 are shown in Figures 4.2, 4.3 and 4.4. Again, we observe that our algorithm yields the exact optimal and empirical hit probabilities under MCDP for seven-node WSN. The density of number of content in the network concentrates around their corresponding cache sizes. Furthermore, we notice that the compression ratio δ at node 4 is much smaller than the ratios at nodes 6 and 7. Thus data compression occurs at routers near to sensors so as to transmit less data long distances. This captures the trade-off between the costs of compression, communication and caching in our optimization framework. Similar observations can be made for the other three request paths and hence are omitted here.

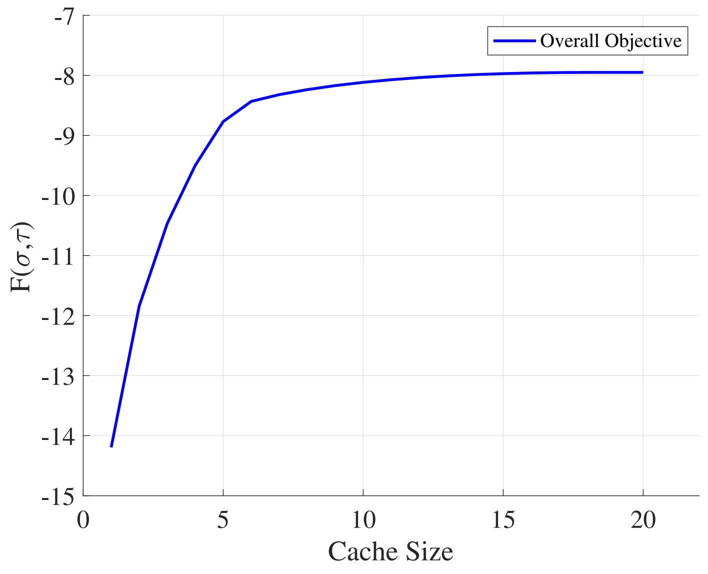


Figure 4.5: Storage vs. overall objective under a seven-node tree WSN.

We now focus on how cache capacity affects the overall objective $F(\sigma, \tau)$ as shown in Figure 4.5. With an increase in $B_l, \forall l \in V$, $F(\sigma, \tau)$ increases. Note that $F(\sigma, \tau)$ gradually converge to a value as cache capacity increases. As $B_l \rightarrow |\mathcal{D}|$, $F(\sigma, \tau)$ becomes insensitive to B_l.

4.4 Summary

In this chapter, we characterized the tradeoff among caching, compression and communication through our optimization framework by incorporating utilities of hit probability and costs of compression and communication. We identified the non-convexity issue and proposed a transformation technique to convert it into a convex problem. We showed the efficiency of our framework through numerical studies.

CHAPTER 5

A QUEUEING-THEORETIC MODEL FOR RESOURCE ALLOCATION IN ONE-DIMENSIONAL DISTRIBUTED SERVICE NETWORK

In Chapters 2-4 we characterized the fundamental limits and optimal content placement problem in a cache network. We now move our focus to a different service network where resources and users are located on a one-dimensional line.

5.1 Background

In many networked systems, for example, Internet of Things [10], a large number of computational and storage resources are widely distributed in the physical world. These resources are accessed by various end users/applications that are also distributed over the physical space. Assigning users or applications to resources efficiently is key to the sustained high-performance operation of the system.

In some systems, requests are transferred over a network to a server that provides a needed resource. In other systems, servers are mobile and physically move to the user making a request. Examples of the former type of service include accessing storage resources over a wireless network to store files and requesting computational resources to run image processing tasks; whereas an example of the latter type of service is the arrival of ride-sharing vehicles to the user's location over a road transportation network.

Not surprisingly, the spatial distribution of resources and users[1] in the network is an important factor in determining the overall performance of the service. A key measure of performance is *average request distance*, that is average distance between a user and its allocated resource/server (where distance is measured on the network). This directly translates to latency incurred by a user when accessing the service, which is arguably among the most important criteria in distributed service applications. Another important practical constraint in distributed service networks is *service capacity*. For example, in network analytics applications, a networked storage device can only support a finite number of concurrent users up to a fixed capacity; similarly, a computational resource can only support a finite number of concurrent processing tasks up to a fixed capacity. Likewise, in physical service applications like ride-sharing, a vehicle has a maximum capacity for passengers.

Therefore, a primary problem in such distributed service networks is to efficiently assign each user to a suitable resource so as to minimize average request distance and ensure no resource serves more users than its capacity. If the entire system is being managed by a single administrative entity such as a ride sharing service, or a datacenter network where analytics tasks are being assigned to available CPUs, there are economic benefits in minimizing the average request distance across all (user, resource) pairs, which is tantamount to minimizing the average delay in the system.

The general version of this capacitated assignment problem can be solved by modeling it as a *minimum cost flow* problem on graphs [7] and running the *network simplex algorithm* [71]. However, if the network has a low-dimensional structure and some assumptions about the spatial distributions of users and resources hold, more efficient methods can be developed.

[1]We use the terms "users" and "requesters" interchangeably and same holds true for the terms "resources" and "servers".

In this chapter, we consider two one-dimensional network scenarios that motivate the study of this special case of the user-to-resource assignment problem.

The first scenario is ride-hailing on a one-way street where vehicles move right to left. If the vehicles of a ride-sharing company are distributed along the street at a certain time, and users equipped with smartphone ride-hailing apps request service, the system attempts to assign vehicles with spare capacity located towards the right of the users so as to minimize average "pick up" distance. Abadi et al. [1] introduced this problem and presented a policy known as Unidirectional Gale-Shapley[2] matching (*UGS*) to minimize average pick up distance. In this policy, all users concurrently emit rays of light toward their right and each user is matched with the vehicle that first receives the emitted ray. While the well-known Gale-Shapley matching algorithm [39] matches user-resource pairs that are mutually nearest to each other, its unidirectional variant, UGS, matches a user to the nearest available resource on its right. Note that, this one-dimensional network setting also applies to vehicular wireless ad-hoc networks on a one-lane roadway [48, 60][3], where users are in vehicles and servers are attached to fixed infrastructure such as lamp posts. Users attempt to allocate their computation tasks over the wireless network to servers located to their right so that they can retrieve the results with little effort while driving by.

In this chapter, we propose another policy "Move to Right" policy (or *MTR*) which has the same "expected distance traveled by a request" (*request distance*) as UGS but has a lower variance. *MTR* sequentially allocates users to the geographically nearest available vehicle located to his/her right. When user and resource locations are modeled as statistical point processes the one-dimensional unidirectional space behaves similar to time and notions from queueing theory can be applied. In partic-

[2]We rename *queue matching* defined in [1] as Unidirectional Gale-Shapley Matching to avoid overloading the term *queue*.

[3]Furthermore, [48] confirms that vehicle location distribution on the streets in Central London can be closely approximated by a Poisson distribution.

ular, when user and vehicle locations are modeled by independent Poisson processes, average request distance can be characterized in closed form by considering inter-user and inter-server distances as parameters of a *bulk service* M/M/1 queue where the bulk service capacity denotes the maximum number of users that can be handled by a server. We equate request distance in the spatial system to the expected *sojourn time* in the corresponding queuing model[4]. This natural mapping allows us to use well-known results from queueing theory and in some cases to propose new queueing theoretic models to characterize request distances for a number of interesting situations beyond M/M/1 queues.

A natural extension to our spatial framework is to consider more general communication costs associated with each resource allocation. Assuming communication cost for each allocation is a function of request distance, we provide closed form expressions for the expected communication cost for specific user-server distributions and specific server capacities.

The second scenario involves a convoy of vehicles traveling on a one-dimensional space, for example, trucks on a highway or boats on a river. Some vehicles have expensive camera sensors (image/video) but have inadequate computational storage or processing power. On the other hand, cheap storage and processing is easily available on several other vehicles. The cameras periodically take photos/videos as they move through space and want them processed / stored. In such case, bidirectional assignment schemes are more suitable. Since no directionality restrictions are imposed on the allocation algorithms, computing the optimal assignment is not as simple as in the unidirectional case.

We explore the special structure of the one-dimensional topology to develop an optimal algorithm that assigns a set of requesters R to a set of resources S such that

[4]Sojourn time is the sum of waiting and service times in a queue.

the total assignment cost is minimized. This problem has been recently solved for $|R| = |S|$ [23]. However, we are interested in the case when $|R| < |S|$. We propose a dynamic Programming based algorithm which solves this case with time complexity $O(|R|(|S|-|R|+1))$. Note that other assignment algorithms in literature such as the Hungarian primal-dual algorithm and Agarwal's variant [5] have time complexities $O(|R|^3)$ and $O(|R|^{2+\epsilon})$ respectively and assume $|R| = |S|$ for general and Euclidean distance measures.

5.2 System Model

Consider a set of users R and a set of servers S. Each user makes a request that can be satisfied by any server. Assume that each server $j \in S$ has capacity $c_j \in \mathbb{Z}^+$ corresponding to the maximum number of requests that it can process. Suppose users and servers are located on a line $\mathcal{L}$. Formally, let $r : R \to \mathcal{L}$ and $s : S \to \mathcal{L}$ be the location functions for users and servers, respectively, such that a distance $d_{\mathcal{L}}(r, s)$ is well defined for all pairs $(r, s) \in R \times S$. Initially we assume that all servers have equal capacities i.e. $c_j = c \; \forall j \in S$. Later in Section 5.6.2 we extend our analysis to a case in which server capacities are integer random variables.

5.2.1 User and server spatial distributions

Let $0 \leq r_1 \leq r_2 \leq \cdots$ represent user locations and $0 \leq s_1 \leq s_2 \leq \cdots$ be the server locations. Let $X_j = s_j - s_{j-1}, j \geq 1, s_0 = 0$, denote the inter-server distances and $Y_i = r_i - r_{i-1}, i \geq 1, r_0 = 0$, the inter-user distances. We assume $\{X_j\}_{j\geq 1}$ to be a renewal process with cumulative distribution function (cdf)

$$\mathbb{P}(X_j \leq x) = F_X(x).$$

We also assume $\{Y_i\}_{i\geq 1}$ to be a renewal process with cdf $F_Y(x)$, i.e.,

$$\mathbb{P}(Y_i \leq x) = F_Y(x).$$

We denote $\alpha_X = 1/\mu$ and σ_X^2 to be the mean and variance associated with F_X. Similarly let $\alpha_Y = 1/\lambda$ and σ_Y^2 be the mean and variance associated with F_Y. We let $\rho = \lambda/\mu$ and assume that $\rho < c$. Denote by $F_X^*(s) = \int_0^\infty e^{-sx} dF_X(x)$ and $F_Y^*(s)$ the Laplace-Stieltjes transform (*LST*) of F_X and F_Y with $s \geq 0$.

In this chapter, we consider various inter-server and inter-user distance distributions, including exponential, deterministic, uniform and hyperexponential.

5.2.2 Allocation policies

One of our goals is to analyze the performance of various request allocation policies using expected request distance as a performance metric. We define various allocation policies as follows.

- **Unidirectional Gale-Shapley (*UGS*):** In UGS, each user simultaneously emits a ray to their right. Once the ray hits an unallocated server s, the user is allocated to s.

- **Move To Right (*MTR*):** In MTR, starting from the left, each user is allocated sequentially to the nearest available server to its right.

- **Nearest Neighbor (*NN*) [89]:** In this matching, starting from the left, each user is allocated sequentially to the nearest available server. This policy can be viewed as the bidirectional version of MTR policy.

- **Gale-Shapley (*GS*) [39]:** In this matching, each user selects the nearest server and each server selects its nearest user. Remove reciprocating pairs, and continue.

- **Optimal Matching:** This matching minimizes average request distance among all feasible allocation policies.

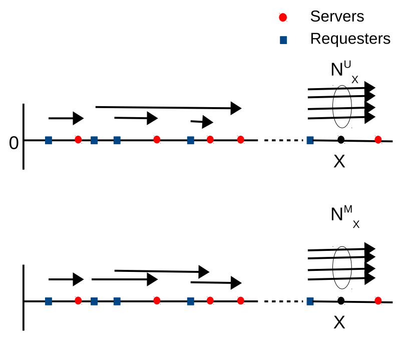


Figure 5.1: Allocation of users to servers on the one-dimensional network. Top: UGS, Bottom: MTR allocation policy.

5.3 Unidirectional Allocation Policies

In this Section, we establish the equivalence of UGS and MTR w.r.t number of requests that traverse a point and expected request distance. Define N_x^P and D_i^P to be random variables for the number of requests that traverse point $x \in \mathcal{L}$ and distance between user i and its allocated server under policy P, respectively. Thus N_x^U and N_x^M denote the number of requests that traverse point $x \in \mathcal{L}$ under UGS and MTR, respectively, as shown in Figure 5.1. Consider the following definition of busy cycle in a service network.

Define 1. *A busy cycle for a policy P is an interval $I = [a, b] \subset \mathcal{L}$ such that $\exists\ i, j$ with $r_i = a, s_j = b$ for which $N_x^P > 0, \forall x \in I$ and $N_x^P = 0$ for $x = a - \epsilon$ and $x = b + \epsilon$ with ϵ being an infinitesimal positive value.*

We have the following theorem.

Theorem 9. $N_x^U = N_x^M, x \geq 0$.

Proof. Due to the unidirectional nature of matching, both UGS and MTR have the same set of busy cycles. Denote $\mathcal{I}$ as the set of all busy cycles in the service network. In the case when $x \in \mathcal{L} \setminus \bigcup\limits_{I \in \mathcal{I}} I$ we already have $N_x^U = N_x^M = 0$. Let us now consider a busy cycle $I^U = [a^U, b^U]$ under UGS. Let $x \in I^U$. Let $L_{x,R}^U = |\{r_i | a^U \leq r_i \leq x\}|$ and $L_{x,S}^U = |\{s_j | a^U \leq s_j \leq x\}|$. $N_x^U = L_{x,R}^U - L_{x,S}^U$. Similarly define $L_{x,R}^M$ and $L_{x,S}^M$ for

MTR policy. Clearly $N_x^M = L_{x,R}^M - L_{x,S}^M$. As both policies have the same set of busy cycles we have $L_{x,R}^U = L_{x,R}^M$ and $L_{x,S}^U = L_{x,S}^M$. Thus we get

$$N_x^U = N_x^M,\ x \in \mathbb{R}^+,$$

$\square$

Corollary 1. $\mathbb{E}[D^U] = \mathbb{E}[D^M]$ *i.e. the expected request distances are the same for both UGS and MTR in steady state.*

Proof. In steady state both N_x^U and N_x^M converge to random variables. Applying Little's law we have $\mathbb{E}[D^U] = \mathbb{E}[D^M]$. $\square$

Remark 4. *Note that Theorem 9 applies to any inter-server or inter-user distance distribution. It also applies to the case where servers have capacity* $c > 1$.

Remark 5. *Although MTR and UGS are equivalent w.r.t. the expected request distance, MTR tends is fairer, i.e., has low variance*[5] *w.r.t. request distance.*

5.4 Unidirectional Poisson Matching

In this section, we characterize request distance statistics under unidirectional policies when both users and servers are distributed according to two independent Poisson processes. We first analyze MTR as follows.

5.4.1 MTR

Under this allocation policy, the service network can be modeled as a bulk service M/M/1 queue. A bulk service M/M/1 queue provides service to a group of c or fewer

[5]It is well known in queueing theory that among all service disciplines the variance of the waiting time is minimized under FCFS policy for Poisson arrivals and exponential service times [56]. In Section 5.4 we show that MTR maps to a temporal FCFS queue.

customers. The server serves a batch of at most c customers whenever it becomes free. Also customers can join an existing service if there is room which is an example of accessible batch. In Section 5.5 we describe the notion of accessible batches in greater detail. The service time for the group is exponentially distributed and customer arrivals are described by a Poisson process. The distance between two consecutive users in the service network can be thought of as inter-arrival time between customers in the bulk service M/M/1 queue. The distance between two consecutive servers maps to a bulk service time.

Having established an analogy between the service network and the bulk service M/M/1 queue, we now define the state space for the service network. Consider the definition of N_x as the number of requests[6] that traverse point $X \in L$ under MTR. In steady state, N_x converges to a random variable N provided $\lambda < c\mu$. Let π_k denote $\mathtt{Pr}[N = k]$ with $k \geq 0$.

Following the procedure in Section 4.2.1 of [47], we obtain the steady state probability vector $\pi = [\pi_i, i \geq 0]$. In the service network, request distance corresponds to the sojourn time in the bulk service M/M/1 queue. By applying Little's formula, we obtain the following expression for the expected request distance

$$\mathbb{E}[D] = \frac{r_0}{\lambda(1 - r_0)}, \tag{5.1}$$

where r_0 is the only root in the interval $(0, 1)$ of the following equation (with r as the variable)

$$\mu r^{c+1} - (\lambda + \mu)r + \lambda = 0. \tag{5.2}$$

[6]We drop the superscript (M) for brevity.

5.4.1.1 When server capacity is one ($c = 1$)

When $c = 1$, $r_0 = \rho$ is a solution of (5.2). Thus we can evaluate the expected request distance as

$$\mathbb{E}[D] = \frac{\rho}{\lambda(1-\rho)} = \frac{1}{\mu - \lambda}. \tag{5.3}$$

Note that, when server capacity is one, the service network can be modeled as an M/M/1 queue. In such a case, (5.3) is the mean sojourn time for an M/M/1 queue.

5.4.2 UGS

When both users and servers are Poisson distributed and servers have unit capacity, the request distance in UGS has the same distribution as the busy cycle in the corresponding Last-Come-First-Served Preemptive-Resume (*LCFS-PR*) queue having the density function [1]

$$f_{D^U}(x) = \frac{1}{x\sqrt{\rho}} e^{(\lambda+\mu)x} I_1(2x\sqrt{\lambda\mu}),\ x > 0,$$

where $\rho = \lambda/\mu$ and I_1 is the modified Bessel function of the first kind. Thus the expected request distance is equivalent to the average busy cycle duration in a LCFS-PR queue given by $1/(\mu - \lambda)$ [1].

When servers have capacities $c > 1$ it is difficult to characterize the expected request distance explicitly. However, by Theorem 9, the expected request distance under UGS is the same as that of MTR given by (5.1).

5.5 Unidirectional General Matching

We now derive expressions for the expected request distance when either users or servers are distributed according to a Poisson process and the other by renewal process. In Section 5.5.1 we map the service network to an exceptional service with

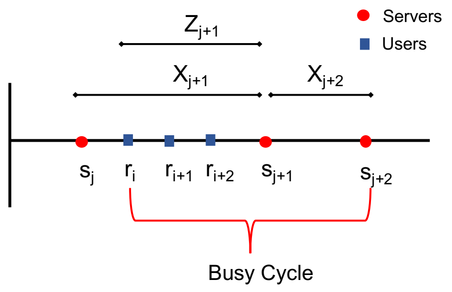


Figure 5.2: Allocation of users to servers under MTR policy.

accessible batches queueing model. In Section 5.5.2 we derive expression for expected request distance when servers are distributed according to a Poisson process. In Section 5.5.3 we consider the case when users are distributed according to a Poisson process.

5.5.1 Notion of exceptional service and accessible batches

We discuss the notion of exceptional service and accessible batches applicable to our service network as follows. Consider a service network with $c = 2$ as shown in Figure 5.2. Consider a user r_i. Let s_j be the server immediately to the left of r_i. We assume all users prior to r_i have already been allocated to servers $\{s_k, 1 \leq k \leq j\}$. MTR allocates both r_i and r_{i+1} to s_{j+1} and allocates r_{i+2} to s_{j+2}. We denote $[r_i, s_{j+2}]$ as a busy cycle of the service network. We have the following queueing theory analogy.

User r_i can be thought of as the first customer in a queueing system that initiates a busy period while r_{i+1} sees the system busy when it arrives. Because only r_i is in service at the arrival of r_{i+1}, r_{i+1} enters service with r_i and the two customers form a batch of size 2. and depart at time s_{j+1}. This is an example of an *accessible batch* [45]. An accessible batch admits subsequent arrivals, while the service is on, until the server capacity c is reached.

The service time for the batch, r_i, r_{i+1}, is described by the random variable Z_{j+1} which is different or *exceptional* when compared to service times of successive batches such as the one consisting of r_{i+2}. The service time for the second batch is X_{j+2}. Note that, Z_{j+1} only depends on X_{j+2} and Y_{i+2}. Thus when either X_{j+2} or Y_{i+2} is described by a Poisson process and the other by renewal process, Z_{j+1} converges to a random variable Z under steady state conditions. Denote $F_Z(x)$ and $f_Z(x)$ as the distribution and density functions for the random variable Z. Thus the service network can be mapped to an exceptional service with accessible batches queueing (*ESABQ*) model. We formally define ESABQ as follows.

ESABQ: *Consider a queueing system where customers are served in batches of maximum size c. A customer entering the queue and finding fewer than c customers in the system joins the current batch and enters service at once, otherwise it joins a queue. After a batch departs leaving k customers in the buffer,* $\min(c, k)$ *customers form a batch and enter service immediately. There are two different service times cdfs,* $F_Z(x)$ *(exceptional batch) with mean* $\alpha_Z = 1/\mu_Z$ *and* $F_X(x)$ *(ordinary batch) with mean* $\alpha_X = 1/\mu$*. A batch is exceptional if its oldest customer entered an empty system, otherwise it is a regular batch. When the service time expires, all customers in the server depart at once, regardless of the nature of the batch (exceptional or regular).*

5.5.1.1 Evaluation of the distribution function: $F_Z(x)$

Distribution	**Parameters**	$\boldsymbol{F_X(x)}$	$\boldsymbol{\mathcal{B}(x)}$
Exponential	μ: rate	$1 - e^{-\mu x}$	$\frac{1}{\lambda}\left[1 - e^{-\lambda x}\right] - \frac{1}{\lambda+\mu}\left[1 - e^{-(\lambda+\mu)x}\right]$
Uniform	b : maximum value	$x/b, \quad 0 \le x \le b$	$\frac{1}{\lambda^2 b}\left[1 - e^{-\lambda b}\right] - \frac{e^{-\lambda x}}{\lambda}$
Deterministic	d_0 : constant	$1, \quad x \ge d_0$	$\frac{e^{-\lambda d_0} - e^{-\lambda x}}{\lambda}$
Hyper -exponential	l: order; p_j : phase probability; μ_j : phase rate	$1 - \sum_{j=1}^{l} p_j e^{-\mu_j x}$	$\frac{1}{\lambda}\left[1 - e^{-\lambda x}\right] - \sum_{j=1}^{l} \frac{p_j}{\lambda+\mu_j}\left[1 - e^{-(\lambda+\mu_j)x}\right]$

Table 5.1: Properties of specific inter-server distance distributions.

In this Section, we compute explicit expressions for the distribution function $F_Z(x)$ applicable to our service network.

When $F_X(x) \sim$ `Expo`(μ): In this case, we invoke the memoryless property of the exponential distribution F_X. Thus the exceptional distribution, F_Z, is

$$F_Z(x) = F_X(x) = 1 - e^{-\mu x}, x \geq 0.$$

When $F_Y(x) \sim$ `Expo`(λ): Using the memoryless property of F_Y, F_Z can be computed as

$$F_Z(x) = \Pr(X - Y < x | Y < X) = \Pr(X - Y < x | X - Y > 0) = \frac{\Pr(X - Y < x) - \Pr(X - Y < 0)}{1 - \Pr(X - Y < 0)}$$
$$= \frac{D_{XY}(x) - D_{XY}(0)}{1 - D_{XY}(0)},\ x \geq 0, \tag{5.4}$$

where $D_{XY}(x)$ is the distribution of the random variable $X - Y$ (also known as difference distribution). $D_{XY}(x)$ can be expressed as

$$D_{XY}(x) = \Pr(X - Y \leq x) = \int_0^\infty \Pr(X - y \leq x)\Pr(Y = y)dy = \int_0^\infty F_X(x + y)\lambda e^{-\lambda y}dy$$
$$= \int_x^\infty F_X(z)\lambda e^{-\lambda(z-x)}dz = \lambda e^{\lambda x}\left[\int_0^\infty F_X(z)e^{-\lambda z}dz - \int_0^x F_X(z)e^{-\lambda z}dz\right] = \lambda e^{\lambda x}\left[\mathcal{A}(F_X) - \mathcal{B}(x)\right], \tag{5.5}$$

where $\mathcal{A}$ is the Laplace Transform operator on the function F_X and $\mathcal{B}(x)$ is denoted by

$$\mathcal{B}(x) = \int_0^x F_X(z)e^{-\lambda z}dz$$

Clearly $\mathcal{B}(0) = 0$. Thus combining (5.4) and (5.5) yields

$$F_Z(x) = \frac{\lambda e^{\lambda x}\left[\mathcal{A}(F_X) - \mathcal{B}(x)\right] - \lambda\mathcal{A}(F_X)}{1 - \lambda\mathcal{A}(F_X)},$$
$$f_Z(x) = \frac{\lambda^2 e^{\lambda x}\left[\mathcal{A}(F_X) - \mathcal{B}(x)\right] - \lambda F_X(x)}{1 - \lambda\mathcal{A}(F_X)},$$
$$\alpha_Z = \int_0^\infty x f_Z(x)dx,\ \sigma_Z^2 = \left[\int_0^\infty x^2 f_Z(x)dx\right] - \alpha_Z^2.$$

Expressions for $\mathcal{B}(x)$ are presented in Table 5.1. We can evaluate $\mathcal{A}(F_X)$ by setting $\mathcal{A}(F_X) = B(\infty)$. Detailed derivations are relegated to Appendix D.1.

5.5.2 General requests and Poisson distributed servers (*GRPS*)

From our discussion in Section 5.5.1.1, it is clear that when servers are distributed according to a Poisson process, the exceptional service time distribution equals the regular batch service time distribution.Thus we have the following queueing model.

Under GRPS, inter-arrival times are arbitrarily distributed and batch service times are exponentially distributed. Before initiating a service, a server finds the system in either of the following settings: (i) $1 \leq n \leq c-1$ or (ii) $n \geq c$. Here n is the number of customers in the waiting buffer. For (i) the server provides service to all n customers and admits subsequent arrivals until c is reached. For (ii) the server takes c customers with no admission for subsequent customers arriving within its service time.

ESABQ can directly be modeled as a special case of a renewal input bulk service queue with accessible and non-accessible batches proposed in [45] with parameter values $a = 1$ and $d = b = c$. Let N_s and N_q denote random variables for numbers of customers in the system and in the waiting buffer respectively for ESABQ under GRPS. We borrow the following definitions from [45].

$$P_{n,0} = \Pr[N_s = n]; 0 \leq n \leq c-1,\ P_{n,1} = \Pr[N_q = n]; n \geq 0.$$

Using results from [45] we obtain the following expressions for equilibrium queue length probabilities.

$$P_{0,1} = \frac{C}{\mu}\left[\frac{r_0^{c-1} - r_0^c}{1 - r_0^c} + \frac{1}{r_0} - 1\right],\ P_{n,1} = \frac{C r_0^{n-1}(1 - r_0)}{\mu(1 - r_0^c)}; n \geq 1,$$

where $0 < r_0 < 1$ is the real root of the equation $r = F_Y^*(\mu - \mu r^c)$ and C is the normalization constant[7] given by

$$C = \lambda \left[\frac{1-\omega^c}{1-\omega} + \frac{1}{1-r_0} - \frac{\omega(r_0 - F_Y^*(\mu))}{r_0^c(1-r_0\omega)} \left(\frac{1-r_0^c}{1-r_0} - r_0^{c-1} \frac{1-w^c}{1-w} \right) \right]^{-1}, \tag{5.6}$$

with $\omega = 1/F_Y^*(\mu)$. We then derive the expected queue length as

$$\mathbb{E}[N_q] = \sum_{n=0}^{\infty} nP_{n,1} = \sum_{n=1}^{\infty} n \frac{Cr_0^{n-1}(1-r_0)}{\mu(1-r_0^c)} = \frac{C(1-r_0)}{\mu(1-r_0^c)} \sum_{n=1}^{\infty} nr_0^{n-1} = \frac{C}{\mu(1-r_0^c)(1-r_0)}.$$

Applying Little's law and considering the analogy between our service network and ESABQ we obtain the following expression for the expected request distance.

$$\mathbb{E}[D] = \frac{C}{\lambda\mu(1-r_0^c)(1-r_0)} + \frac{1}{\mu}.$$

5.5.3 Poisson distributed requests and general distributed servers (*PRGS*)

As discussed in Section 5.5.1.1, if servers are placed on a 1-d line according to a renewal process with requests being Poisson distributed, the service time distribution for the first batch in a busy period differs from those of subsequent batches. Below we derive expressions for queue length distribution and expected request distance for ESABQ under PRGS.

5.5.3.1 Queue length distribution

We use a supplementary variable technique to derive the queue length distribution for ESABQ under PRGS as follows.

Let $L(t)$ be the number of customers at time $t \geq 0$, $R(t)$ the residual service time at time $t \geq 0$ (with $R(t) = 0$ if $L(t) = 0$), and $I(t)$ the type of service at time $t \geq 0$ with $I(t) = 1$ (resp. $I(t) = 2$) if exceptional (resp. ordinary) service time.

[7]The normalization constant C derived in [45] is incorrect. The correct constant for our case is given in (5.6).

Let us write the Chapman-Kolmorogov equations for the Markov chain $\{(L(t), R(t), I(t)),\ t \geq 0\}$.

For $t \geq 0,\ n \geq 1,\ x > 0,\ i = 1, 2$ define

$$p_t(n, x; i) = \mathbb{P}(L(t) = n, R(t) < x, I(t) = i) \quad \text{and} \quad p_t(0) = \mathbb{P}(L(t) = 0).$$

Also, define for $x > 0,\ i = 1, 2,$

$$p(n, x; i) = \lim_{t\to\infty} p_t(n, x; i) \quad \text{and} \quad p(0) = \lim_{t\to\infty} p_t(0).$$

By analogy with the analysis for the M/G/1 queue we get

$$\frac{\partial}{\partial t} p_t(0) = -\lambda p_t(0) + \sum_{k=1}^{c} \frac{\partial}{\partial x} p_t(k, 0; 1) + \sum_{k=1}^{c} \frac{\partial}{\partial x} p_t(k, 0; 2),$$

so that, by letting $t \to \infty$,

$$\lambda p(0) = \sum_{k=1}^{c} \left(\frac{\partial}{\partial x} p(k, 0; 1) + \frac{\partial}{\partial x} p(k, 0; 2) \right). \tag{5.7}$$

With further simplification (See Appendix D.2.1), for $n \geq 1, x > 0$ we get

$$\begin{aligned} \frac{\partial}{\partial x} g(n, x) - \lambda g(n, x) - \frac{\partial}{\partial x} g(n, 0) + \lambda g(n-1, x)\mathbf{1}(n \geq 2) + \lambda p(0) F_Z(x)\mathbf{1}(n = 1) \\ + F_X(x) \frac{\partial}{\partial x} g(n + c, 0) = 0, \end{aligned} \tag{5.8}$$

where $g(n, x) = p(n, x; 1) + p(n, x; 2)$ for $n \geq 1,\ x > 0$. Introduce

$$G(z, s) := \sum_{n\geq 1} z^n \int_0^\infty e^{-sx} g(n, x) dx \quad \forall |z| \leq 1,\ s \geq 0.$$

Denote by $F_Z^*(s) = \int_0^\infty e^{-sx} dF_Z(x)$ the LST of F_Z for $s \geq 0$. Note that

$$\int_0^\infty e^{-sx} F_{Z \text{or} X}(x) dx = \frac{F_{Z\text{or}X}^*(s)}{s}, \quad \forall s > 0.$$

Multiplying both sides of (5.8) by $z^n e^{-sx}$, integrating over $x \in [0, \infty)$ and summing over all $n \geq 1$, yields

$$s\left(\lambda(1-z) - s\right) G(z,s) = \lambda z p(0) F_Z^*(s) - \sum_{n \geq 1} z^n \frac{\partial}{\partial x} g(n,0) + F_X^*(s) \sum_{n \geq 1} z^n \frac{\partial}{\partial x} g(n+c,0)) \tag{5.9}$$

where $\lambda p(0) = \sum_{k=1}^c \frac{\partial}{\partial x} g(k,0)$ from (5.7). We have

$$\frac{1}{z^c} \sum_{n \geq 1} z^{n+c} \frac{\partial}{\partial x} g(n+c,0)) = \frac{1}{z^c} \sum_{n \geq 1} z^n \frac{\partial}{\partial x} g(n,0) - \frac{1}{z^c} H(z)$$

where $H(z) = \sum_{k=1}^c z^k a_k$ with $a_k := \frac{\partial}{\partial x} g(k,0)$, for $k = 1, \ldots, c$. Introducing the above into (5.9) gives

$$s\left(\lambda(1-z) - s\right) G(z,s) = \left(\frac{F_X^*(s)}{z^c} - 1 \right) \Psi(z) - F_X^*(s) \frac{H(z)}{z^c} + \lambda z p(0) F_Z^*(s) \tag{5.10}$$

where $\Psi(z) := \sum_{n \geq 1} z^n \frac{\partial}{\partial x} g(n,0)$. Since $G(z,s)$ is well-defined for $|z| \leq 1$ and $s \geq 0$, the r.h.s. of (5.10) must vanish when $s = \lambda(1-z)$. This gives the relation

$$\Psi(z) = \frac{z^c}{z^c - F_X^*(\theta(z))} \left[-F_X^*(\theta(z)) \frac{H(z)}{z^c} + \lambda z p(0) F_Z^*(\theta(z)) \right]$$

with $\theta(z) = \lambda(1-z)$ and $|z| \leq 1$. Introducing the above in (5.10) gives

$$\begin{aligned} s\left(\lambda(1-z) - s\right) G(z,s) = &- F_X^*(s) \frac{H(z)}{z^c} + \lambda z p(0) F_Z^*(s) \\ &+ \frac{F_X^*(s) - z^c}{z^c - F_X^*(\theta(z))} \left[\lambda z p(0) F_Z^*(\theta(z)) - F_X^*(\theta(z)) \frac{H(z)}{z^c} \right]. \end{aligned} \tag{5.11}$$

Let $N(z)$ be the z-transform of the stationary number of customers in the system. Integrating by part, we get for $n \geq 1$,

$$s \int_0^\infty e^{-sx} g(n,x)dx = \int_0^\infty e^{-sx} dg(n,x),$$

so that

$$\lim_{s\to\infty} s \int_0^\infty e^{-sx} g(n,x)dx = \lim_{s\to 0} \int_0^\infty e^{-sx} dg(n,x) = \int_0^\infty dg(n,x) = g(n,\infty), \quad (5.12)$$

where the interchange between the limit and the integral sign is justified by the bounded convergence theorem. Therefore,

$$\begin{aligned} N(z) &= \sum_{n\geq 1} z^n g(n,\infty) + p(0) = \sum_{n\geq 1} z^n \lim_{s\to\infty} s \int_0^\infty e^{-sx} g(n,x)dx \quad \text{from (5.12)} \\ &= \lim_{s\to 0} sG(z,s) + p(0), \qquad (5.13) \end{aligned}$$

where the interchange between the summation over n and the integral sign is again justified by the bounded convergence theorem. Letting now $s \to 0$ in (5.11) and using (5.13), gives

$$\theta(z)N(z) = \frac{1-z^c}{z^c - F_X^*(\theta(z))}\left[-F_X^*(\theta(z))\frac{H(z)}{z^c} + \lambda z p(0) F_Z^*(\theta(z))\right] - \frac{H(z)}{z^c} + \lambda p(0). \quad (5.14)$$

By noting that $\lambda p(0) = \sum_{k=1}^c a_k$ (cf. (5.7)), Eq. (5.14) can be rewritten as

$$N(z) = \frac{1}{\theta(z)}\left(\frac{z(1-z^c)}{z^c - F_X^*(\theta(z))}\sum_{k=1}^c a_k \left[F_Z^*(\theta(z)) - z^{k-c-1}F_X^*(\theta(z))\right] + \sum_{k=1}^c a_k(1-z^{k-c})\right). \quad (5.15)$$

The r.h.s. of (5.15) contains c unknown constants $a_1, \ldots, a_c$ yet to be determined. Define $A(z) = F_X^*(\theta(z))$. It can be shown that $z^c - A(z)$ has $c-1$ zeros inside and one on the unit circle, $|z|=1$ (See Appendix D.2.3). Denote by $\xi_1,\ldots,\xi_q$ the $1\le q\le c$ distinct zeros of $z^c-A(z)$ in $\{|z|\le 1\}$, with multiplicity $n_1,\ldots,n_q$, respectively, with $n_1+\cdots+n_q=c$. Hence,

$$z^c - F_X^*(k(z)) = \gamma \prod_{i=1}^{q} (z-\xi_i)^{n_i}.$$

Since $z^c - A(z)$ vanishes when $z=1$ and that $\frac{d}{dz}(z^c-A(z))|_{z=1} = c-\rho > 0$, we conclude that $z^c - A(z)$ has one zero of multiplicity one at $z=1$.

Without loss of generality assume that $\xi_q = 1$ and let us now focus on the zeros $\xi_1,\ldots,\xi_{q-1}$. When $z=\xi_i$, $i=1,\ldots,q-1$, the term $F_Z^*(\theta(z)) - z^{k-c-1}F_X^*(\theta(z))$ in (5.15) must have a zero of multiplicity (at least) n_i since $N(\xi_i)$ is well defined. This gives $c-1$ linear equations to be satisfied by $\xi_1,\ldots,\xi_q$. In the particular case where all zeros have multiplicity one (see Appendix D.2.2), namely $q=c$, these $c-1$ equations are

$$\sum_{k=1}^{c} a_k \left[F_Z^*(\theta(\xi_i)) - \xi_i^{k-c-1} F_X^*(\theta(\xi_i))\right] = 0,\ i=1,\ldots,c-1. \tag{5.16}$$

With $U(z) := F_Z^*(\theta(z))/F_X^*(\theta(z))$ (5.16) is equivalent to

$$\sum_{k=1}^{c} a_k \left[U(\xi_i) - \xi_i^{k-c-1})\right] = 0,\ i=1,\ldots,c-1,$$

since $F_X^*(\theta(\xi_i)) \neq 0$ for $i=1,\ldots,c-1$ ($F_X^*(\theta(\xi_i)) = 0$ implies that ξ_i=0 which contradicts that ξ_i a zero of $z^c - F_X^*(\theta(z))$ since $F_X^*(\theta(0)) = F_X^*(\lambda) > 0$). Eq. (5.15) can be rewritten as

$$N(z) = \frac{\sum_{k=1}^{c} a_k \left[z^c - z^k + z(1-z^c)F_Z^*(\theta(z)) - (1-z^k)F_X^*(\theta(z))\right]}{\theta(z)(z^c - F_X^*(\theta(z))}. \tag{5.17}$$

A c-th equation is provided by the normalizing condition $N(z) = 1$. Since the numerator and denominator in (5.17) have a zero of order 2 at $z=1$, differentiating twice the numerator and the denominator w.r.t z and letting $z=1$ gives

$$\sum_{k=1}^{c} a_k(c(1+\rho_z) - \rho k) = \lambda(c-\rho), \tag{5.18}$$

where $\rho_z = \lambda \alpha_Z$. We consider few special cases of the model in Appendix D.2.4 and verify with the expressions of queue length distribution available in the literature.

5.5.3.2 Expected request distance

From (5.17) the expected queue length is

$$\begin{aligned}
\overline{N} &= \frac{d}{dz}N(z)\Big|_{z=1} \\
&= \frac{1}{2\lambda(c-\rho)^2}\sum_{k=1}^{c} a_k \Bigg[\quad \lambda^2\sigma_Z^{(2)}c(c-\rho) + \lambda^2\sigma_X^{(2)}c(1+\rho_z-k) \\
&\qquad\qquad + (ck(c-k) + k(k-1)\rho - c(c-1))\rho + 2c^2\rho_z - c(c+1)\rho_z\rho \Bigg],
\end{aligned}$$

where $\sigma_Z^{(2)}$ and $\sigma_X^{(2)}$ are the second order moments of distributions F_Z and F_X respectively. Again by applying Little's law and considering the analogy between our service network and ESABQ we get the following expression for the expected request distance.

$$\mathbb{E}[D] = \overline{N}/\lambda.$$

5.6 Discussion of Unidirectional Allocation Policies

In this section we describe generalizations of models and results for unidirectional allocation policies. We first consider the case when inter-user and inter-server distances both have general distributions.

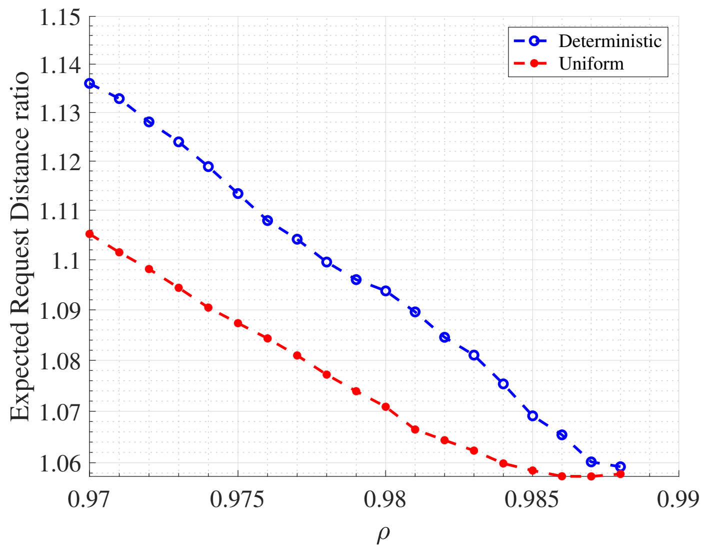


Figure 5.3: The plot shows the ratio $\mathbb{E}[D]/\overline{D}_s$ for deterministic and uniform inter-server distance distributions.

5.6.1 Heavy traffic limit for general request and server spatial distributions

Consider the case when the inter-user and inter-server distances each are described by general distributions. We assume server capacity, $c = 1$. As $\rho \to 1$, we conjecture that the behavior of MTR approaches that of the G/G/1 queue. One argument in favor of our conjecture is the following. As $\rho \to 1$, the busy cycle duration tends to infinity. Consequently, the impact of the exceptional service for the first customer of the busy period on all other customers diminishes to zero as there is an unbounded increasing number of customers served in the busy period.

It is known that in heavy traffic waiting times in a G/G/1 queue are exponentially distributed and the mean sojourn time is given by $\alpha_X + [(\sigma_X^2 + \sigma_Y^2)/2\alpha_Y(1-\rho)]$ [47]. We expect the expected request distance to exhibit similar behavior. Thus we have the following conjecture.

Conjecture 1. *At heavy traffic i.e. as $\rho \to 1$, the expected request distance for the G/G/1 spatial system with $c = 1$ is given by*

$$\mathbb{E}[D] = \alpha_X + \frac{\sigma_X^2 + \sigma_Y^2}{2\alpha_Y(1-\rho)}.$$

Denote by $\overline{D}_s$ the average request distance as obtained from simulation. We plot the ratio $\mathbb{E}[D]/\overline{D}_s$ across various inter-request and inter-server distance distributions in Figure 5.3. It is evident that as $\rho \to 1$, the ratio $\mathbb{E}[D]/\overline{D}_s$ converges to 1 across different inter-server distance distributions.

5.6.2 Heterogeneous server capacities under PRGS

We now proceed to analyze a setting where server capacity is a random variable. Assume server capacity $\mathcal{C}$ takes values from $\{1, 2, \ldots, c\}$ with distribution $\texttt{Pr}(\mathcal{C} = j) = p_j, \forall j \in \{1, 2, \ldots, c\}$, s.t. $\sum_{j=1}^{c} p_j = 1$ and $p_c > 0$. We also assume the stability condition $\rho < \overline{\mathcal{C}}$ where $\overline{\mathcal{C}}$ is the average server capacity. Denote H as the random variable associated with number of requests that traverse through a point just after a server location[8].

5.6.2.1 Distribution of H

Let V denote the number of new requests generated during a service period with $k_v = \texttt{Pr}(V = v), \forall v \geq 0$. According to the law of total probability, it holds that

$$k_v = \int_0^\infty \texttt{Pr}(V = v | X = \nu) f_X(\nu) = \frac{1}{v!} \int_0^\infty e^{-\lambda\nu} (\lambda\nu)^v dF_X(\nu).$$

Then the corresponding generating function $K(z)$ is denoted by

$$K(z) = \sum_{v=0}^{\infty} k_v z^v = F_X^*(\lambda(1-z)).$$

[8]An analysis for the distribution of number of requests that traverse through any random location would involve the notions of exceptional service and accessible batches.

We now consider an embedded Markov chain generated by H. Denote the corresponding transition matrix as M. Then we have

$$M_{m,l} = \begin{cases} \sum_{i=0}^{c-m} k_i P_{i+m}, & 0 \le m \le c, l = 0; \\ \sum_{i=0}^{c} k_{i+l-m} p_i, & 0 \le m \le l, l \neq 0; \\ \sum_{i=m-l}^{c} k_{i+l-m} p_i, & l+1 \le m \le c+l, l \neq 0; \\ 0, & o.w., \end{cases} \tag{5.19}$$

where $P_i = \sum_{j=i}^{c} p_j$ and $p_0 = 0$. Let $\pi = [\pi_j, j \geq 0]$ and $N(z) = \sum_{j \geq 0} \pi_j z^j$ denote the steady state distribution and its z-transform respectively. π is obtained out by solving

$$\pi_l = \sum_{m=0}^{\infty} \pi_m M_{m,l}, l = 0, 1, \ldots.$$

Thus we have for $l \in \mathbb{N}$,

$$\pi_0 = \sum_{m=0}^{c} \pi_m \sum_{i=0}^{c-m} k_i P_{i+m},\ \pi_l = \sum_{m=0}^{l} \pi_m \sum_{i=0}^{c} k_{i+l-m} p_i + \sum_{m=l+1}^{c+l} \pi_m \sum_{i=m-l}^{c} k_{i+l-m} p_i.$$

Multiplying by z^l and summing over l gives

$$N(z) = E_\pi + v_1(z) + v_2(z) \tag{5.20}$$

$$E_\pi = \pi_0 \sum_{i=0}^{c-1} k_i P_{i+1} + \sum_{m=1}^{c-1} \pi_m \sum_{i=m}^{c-1} k_{i-m} P_{i+1}$$

$$v_1(z) = \sum_{l=0}^{\infty} z^l \sum_{m=0}^{l} \pi_m \sum_{i=0}^{c} k_{i+l-m} p_i \tag{5.21}$$

$$v_2(z) = \sum_{l=0}^{\infty} z^l \sum_{m=l+1}^{c+l} \pi_m \sum_{i=m-l}^{c} k_{i+l-m} p_i. \tag{5.22}$$

The expressions for $v_1(z)$ and $v_2(z)$ can be further simplified (see Appendix D.3) to

$$v_1(z) = N(z) \left\{ \sum_{i=0}^{c} p_i z^{-i} \left[K(z) - \sum_{j=0}^{i} k_j z^j \right] + \sum_{i=0}^{c} k_i z^i \right\} \tag{5.23}$$

$$v_2(z) = \left[\sum_{m=0}^{c} z^{-m} \sum_{i=m}^{c} k_{i-m} p_i \left\{ N(z) - \sum_{j=0}^{m-1} \pi_j z^j \right\} \right] - N(z) \sum_{i=0}^{c} k_i z^i. \tag{5.24}$$

Combining (5.20), (5.23) and (5.24) yields

$$N(z) = E_\pi + N(z)\left\{K(z)\sum_{i=0}^{c} p_i z^{-i}\right\} - \sum_{j=0}^{c-1}\pi_j \sum_{m=1}^{c-j} z^{-m} \sum_{i=m+j}^{c} k_{i-(m+j)}p_i.$$

Thus we obtain

$$N(z) = \frac{E_\pi - \sum_{j=0}^{c-1}\pi_j \sum_{m=1}^{c-j} z^{-m} \sum_{i=m+j}^{c} k_{i-(m+j)}p_i}{1 - K(z)\sum_{i=0}^{c} p_i z^{-i}}. \tag{5.25}$$

Multipying numerator and denominator by z^c yields

$$N(z) = \frac{z^c E_\pi - \sum_{j=0}^{c-1}\pi_j \sum_{m=1}^{c-j} z^{c-m} \sum_{i=m+j}^{c} k_{i-(m+j)}p_i}{z^c - K(z)\sum_{i=0}^{c} p_{c-i} z^{i}}. \tag{5.26}$$

To determine $N(z)$, we need to obtain the probabilities $\pi_i, 0 \le i \le c-1$. It can be shown that the denominator of (5.26) has $c-1$ zeros inside and one on the unit circle, $|z| = 1$ (See Appendix D.3.2). As $N(z)$ is analytic within and on the unit circle, the numerator must vanish at these zeros, giving rise to c equations in c unknowns.

Let $\xi_q : 1 \le q \le c$ be the zeros of $z^c - K(z)\sum_{i=0}^{c} p_{c-i}z^i$ in $\{|z| \le 1\}$. W.l.o.g let $\xi_c = 1$. We have the following $c-1$ equations.

$$E_\pi - \sum_{j=0}^{c-1}\pi_j \sum_{m=1}^{c-j} \xi_q^{-m} \sum_{i=m+j}^{c} k_{i-(m+j)}p_i = 0,\ i = 1,\ldots,c-1,$$

A c-th equation is provided by the normalizing condition $\lim_{z\to1} N(z) = 1$. In the particular case where all zeros have multiplicity one, it can be shown that these c equations are linearly independent[9]. Once the parameters $\{\pi_i, 0 \le i \le c-1\}$ are known, $\mathbb{E}[H]$ can be expressed as

$$\mathbb{E}[H] = \overline{H} = \lim_{z\to1} N'(z).$$

[9]For all cases evaluated across uniform, deterministic and hyperexponential distributions we found the set of c equations to be linearly independent.

5.6.2.2 Expected Request Distance

To evaluate the expected request distance we adopt arguments from [14]. Consider any interval of length ν between two consecutive servers. There are on average $\overline{H}$ requests at the beginning of the interval , each of which must travel ν distance. New users are spread randomly over the interval and there are on an average $\lambda\nu$ new users. The request made by each new user must travel on average $\nu/2$. Thus we have

$$\mathbb{E}[D] = \frac{1}{\rho}\int_0^\infty (\overline{H}\nu + \frac{1}{2}\lambda\nu^2)dF_X(\nu) = \frac{1}{\rho}\left[\frac{\overline{H}}{\mu} + \frac{\lambda}{2}\left(\sigma_X^2 + \frac{1}{\mu^2}\right)\right]. \tag{5.27}$$

5.6.3 Uncapacitated request allocation

An interesting special case of the unidirectional general matching is the uncapacitated scenario. Consider the case where servers do not have any capacity constraints, i.e. $c = \infty$. In such a case, all users are assigned to the nearest server to their right.

GRPS: When $c \to \infty$ and given $0 < r_0 < 1$, $r_0 = F_Y^*(\mu - \mu r_0^c) = F_Y^*(\mu)$. Setting $\omega = 1/F_Y^*(\mu) = 1/r_0$ in (5.6) and simplifying yields

$$C \to 0, \text{ as } c \to \infty, \implies \mathbb{E}[D] \to \frac{1}{\mu} \text{ as } c \to \infty.$$

PRGS: Under PRGS, when $c \to \infty$ there exists no request allocated to a server other than the nearest server to its right. Again using Bailey's method as in [14] and setting $\overline{H} = 0$ in (5.27) we get

$$\mathbb{E}[D] \to \frac{\mu}{2}\left(\sigma_X^2 + \frac{1}{\mu^2}\right) \text{ as } c \to \infty.$$

5.6.4 Cost models

Consider the following generalization of the service network. We define cost of an allocation as the communication cost associated with an allocated request-server pair. Consider communication cost as a function $\mathcal{T}$ of the request distances. Then the expected communication cost across the service network is given as

$$\overline{T} = E[\texttt{cost}] = \int_{d=0}^{\infty} \mathcal{T}(d) dW(d),$$

where W is the request distance distribution. One such cost model widely used in wireless ad hoc networks is [32]

$$\mathcal{T}(d) = t_0 d^{\beta},$$

where β is the path loss exponent typically $2 \leq \beta \leq 4$ and t_0 is a constant. Below we derive the expected communication cost for the scenario when $c = 1$.

5.6.4.1 GRPS with $c = 1$

In this case the service network directly maps to a temporal G/M/1 queue. Thus W can be expressed as the sojourn time distribution of the corresponding G/M/1 queue. Hence $W \sim \text{Expo}(\mu(1 - r_0))$ with r_0 as defined in Section 5.5.2. We have

$$\overline{T} = \int_{d=0}^{\infty} t_0 d^{\beta} dW(d) = \frac{t_0}{\mu^{\beta}(1 - r_0)^{\beta}} \Gamma(\beta + 1),$$

where $\Gamma(x) = \int_0^{\infty} y^{x-1} e^{-y} dy$ is the gamma function.

5.6.4.2 PRGS with $c = 1$

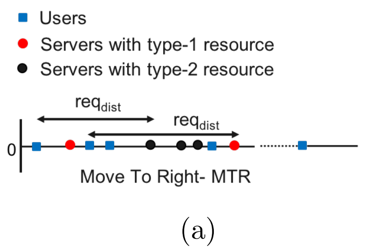
In this case, the service network can be modeled as a temporal M/G/1 queue with first customer having exceptional service [100]. Denote $W^*(s)$ as the LS transform of W. Using results from [100]

$$W^*(s) = \frac{(1-\rho)\left\{\lambda\Big[F_Z^*(s) - F_X^*(s)\Big] - sF_Z^*(s)\right\}}{(1-\rho+\rho_Z)\Big[\lambda - s - \lambda F_X^*(s)\Big]}.$$

When β is an integer,

$$\overline{T} = t_0(-1)^\beta \frac{d^{(\beta)}}{ds} W^*(s)|_{s=0}.$$

5.6.5 Extension to two resources



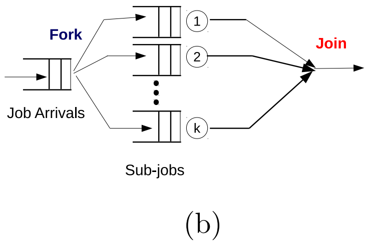


Figure 5.4: Two resource scenario with $c = 1$ (a) Depiction of request distances and (b) Mapping to Fork-join queues.

Now consider the following scenario where each user requests two resources which reside on different servers as shown in Figure 5.4(a). Let the corresponding servers be distributed according to a Poisson process with densities μ_1 and μ_2. Let the users be distributed according to a Poisson process. The service network, in this case, can be modeled as a fork-join queueing system as shown in Figure 5.4(b) [69]. In such a queue, each incoming job is split into two sub-jobs each of which is served on one of the two servers. After service, each sub-job waits until the other sub-job has been processed. They then merge and leave the system. In the service network as well, each request forks two sub-requests one for each resource type. A request is said to be completed only if it has retrieved both the resources, thus mapping it to a fork-join queue. We define the *overall request distance* to be the maximum value among the request distances across all resource types and denote it as the random variable D_{max}.

5.6.5.1 Identical service rates ($\mu_1 = \mu_2 = \mu$ and $c = 1$)

The approximated expected request distance for this scenario is obtained from the expression for the expected sojourn time of a fork join queue with homogeneous servers as [69]:

$$\mathbb{E}[D_{max}] = \frac{12\mu - \lambda}{8\mu(\mu - \lambda)},$$

Note that, the corresponding expected request distance in case of single resource is given by Equation (5.3) $\mathbb{E}[D] = 1/(\mu - \lambda)$. Clearly,

$$\mathbb{E}[D_{max}] = \frac{12\mu - \lambda}{8\mu(\mu - \lambda)} = [1.5 - 0.125\rho]\,\frac{1}{\mu - \lambda} > \frac{1}{\mu - \lambda} = \mathbb{E}[D], \tag{5.28}$$

Thus we have $\mathbb{E}[D_{max}] > \mathbb{E}[D]$.

5.7 Bidirectional Allocation Policies

Both UGS and MTR minimize expected request distance among all unidirectional policies. In this section we present the bi-directional allocation policy that minimizes expected request distance. Let $\eta : R \to S$ be any mapping of users to servers. Our objective is to find a mapping $\eta^* : R \to S$, that satisfies

$$\begin{aligned} \eta^* &= \arg\min_{\eta} \sum_{i \in R} d_{\mathcal{L}}(r_i, s_{\eta(i)}) \\ s.t. \quad &\sum_{i \in R} \mathbf{1}_{\eta(i)=j} \leq c, \forall j \in S \end{aligned} \tag{5.29}$$

W.l.o.g, let $r_1 \leq r_2 \leq \cdots \leq r_i \leq \cdots \leq r_{|R|}$ be locations of requests and $s_1 \leq s_2 \leq \cdots \leq s_i \leq \cdots \leq s_{|S|}$ be locations of servers. We first focus on the case when $c = 1$.

Algorithm 2 Optimal Assignment by Dynamic Programming

```
Input: $r_1 \leq \cdots \leq r_{|R|}$;  $s_1 \leq \cdots \leq s_{|S|}$
Output: The optimal assignment $\pi$
procedure OptDP($r, s$)
    $d_{|R| \times |S|} = $ ComputePairwiseDistances($r, s$)
    $C = \{\infty\}_{|R| \times |S|}$
    for $i = 1, \cdots, |R|$ do
        $C[i, i] = $ TrivialAssignment($i, d$)
    $A[|R|, |R|] = |R|$
    $nearest = 0$
    $nearestcost = C[1, 1]$
    for $j = 2, \cdots, |S| - |R| + 1$ do
        if $d[1, j] < nearestcost$ then
            $nearestcost = d[1, j]$
            $nearest = j$
        $C[1, j] = nearestcost$
        $A[1, j] = nearest$
    for $i = 2, \cdots, |R|$ do
        for $j = i + 1, \cdots, i + |S| - |R|$ do
            if $C[i, j-1] < d[i, j] + C[i-1, j-1]$ then
                $C[i, j] = C[i, j-1]$
                $A[i, j] = A[i, j-1]$
            else
                $C[i, j] = d[i, j] + C[i-1, j-1]$
                $A[i, j] = j$
    return ReadOptAssignment($A$)
procedure TrivialAssignment($n, d$)
    $Cost = 0$
    for $i = 1, \cdots, n$ do
        $Cost = Cost + d[i, i]$
    return $Cost$
procedure ReadOptAssignment($A$)
    $|R|, |S| = $ Dimensions($A$)
    $s = |S|$
    for $i = |R|, \cdots, 1$ do
        $\pi[i] = A[i, s]$
        $s = A[i, s] - 1$
    return $\pi$
```

We consider the following two scenarios.

Case 1: $|R| = |S|$

When $|R| = |S|$, an optimal allocation strategy is given by the following theorem [23].

Theorem 10. *When $|R| = |S|$, an optimal assignment is obtained by the policy: $\eta^*(i) = i, \ \forall i \in \{1, \cdots, |R|\}$ i.e. allocating the i^{th} request to the i^{th} server and the average request distance is given by*

$$\mathbb{E}[D] = \frac{1}{|R|} \sum_{i=1}^{|R|} |s(i) - r(i)|.$$

Case 2: $|R| < |S|$ This is the case where there are fewer requesters than servers. In this case, a Dynamic Programming (*DP*) based algorithm (Algorithm 2) obtains the optimal assignment.

Let $C[i, j]$ denote the optimal cost (i.e., sum of distances) of assigning the first i requests (counting from the left) located at $r_1 \leq r_2 \leq \ldots \leq r_i$ to the first j servers (also counting from the left) located at $s_1 \leq s_2 \leq \ldots \leq s_j$. If $j == i$, the optimal assignment is trivial due to Theorem 10 and $C[i, i]$ is computed easily for all $i \leq |R|$ by summing pairwise distances $d[1, 1], d[2, 2], \ldots, d[i, i]$ (Lines 6–7). For the base case, $i = 1, j > 1$, only the first user needs to be assigned to its nearest server (Lines 9–16). For the general dynamic programming step, consider $j > i$. Then $C[i, j]$ can be expressed in terms of the costs of two subproblems, i.e., $C[i - 1, j - 1]$ and $C[i, j - 1]$ (Lines 19–24). In the optimal solution, two cases are possible: either request i is assigned to server j, or the latter is left unallocated. The former case occurs if the first $i - 1$ requests are assigned to the first $j - 1$ servers at cost $C[i - 1, j - 1]$, and the latter case occurs when the first i requests are assigned to the first $j - 1$ servers at cost $C[i, j-1]$. This is a consequence of the no-crossing lemma (Lemma 4). The optimal $C[i, j]$ is chosen depending on these two costs and the current distance $d[i, j]$.

Lemma 4. *In an optimal solution, η^*, to the problem of matching users at $r_1 \leq r_2 \leq \ldots \leq r_{|R|}$ to servers at $s_1 \leq s_2 \leq \ldots \leq s_{|S|}$, where $|S| \geq |R|$, there do not exist indices i, j such that $\eta^*(i) > \eta^*(i')$ when $i' > i$.*

Proof. See Appendix D.4. □

The dynamic programming algorithm fills cells in an $|R| \times |S|$ matrix C whose origin is in the north-west corner. The lower triangular portion of this matrix is invalid since $|R| \leq |S|$. The base cases populate the diagonal and the northernmost row, and in the general DP step, the value of a cell depends on the previously computed values in the cells located to its immediate west and diagonally north-west. As an optimization, for a fixed i, the j-th loop index needs to run only from $i+1$ through $i+|S|-|R|$ (Lines 11 and 18) instead of from $i+1$ through $|S|$. This is because the first request has to be assigned to a server s_j with $j \leq |S|-|R|+1$ so that the rest of the $|R|-1$ requests have a chance of being placed on unique servers[10]. The optimal average request distance is given by $C[|R|, |S|]$.

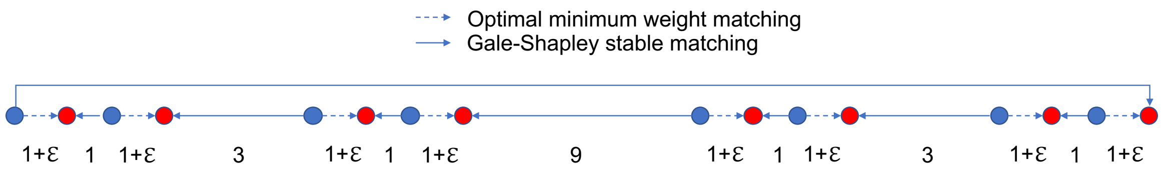


Figure 5.5: Worst case scenario for Gale-Shapley.

The time complexity of the main DP step is $O(|R| \times (|S| - |R| + 1))$. Note that this assumes that the pairwise distance matrix d of dimension $|R| \times |S|$ has

[10]Note that in this exposition, we consider server capacity $c = 1$. If $c > 1$, we simply add c servers at each prescribed server location, and requests will still be placed on unique servers.

been precomputed. The optimization applied above can be similarly applied to this computation and hence the overall time complexity of Algorithm 2 is $O(|R| \times (|S| - |R| + 1))$. Therefore, if $|S| = O(|R|)$, the worst case time complexity is quadratic in $|R|$. However, if $|S| - |R|$ grows sub-linearly with $|R|$, the time complexity is sub-quadratic in $|R|$.

Note that retrieving the optimal assignment requires more book-keeping. An $|R| \times |S|$ matrix A stores key intermediate steps in the assignment as the DP algorithm progresses (Lines 8, 16, 21, 24). The optimal assignment vector π can be retrieved from matrix A using procedure ReadOptAssignment.

Another bidirectional assignment scheme is the Gale-Shapley algorithm [39], which produces stable assignments, though in the worst case it can yield an assignment that is $O(|R|^{\ln 3/2}) \approx O(|R|^{0.58})$ times costlier than the optimal assignment yielded by Algorithm 2, where $|R|$ is the number of users [84]. The worst case scenario is illustrated in Figure 5.5, with $|R| = 2^{t-1}$, where t is the number of clusters of users and servers; and the largest distance between adjacent points is 3^{t-2}. However at low/moderate loads for the cases evaluated in Section 5.8, we find its performance to be not much worse than optimal.

5.8 Numerical Experiments

In this section, we examine the effect of various system parameters on expected request distance under MTR policy. We also compare the performance of various greedy allocation strategies along with the unidirectional policies to the optimal strategy.

5.8.1 Experimental setup

In our experiments, we consider a mean requester rate $\lambda \in (0, 1)$. We consider various inter-server distance distributions with density one. In particular, (i) for exponential distributions, the density is set to $\mu = 1$; (ii) for deterministic distributions, we assign parameter $d_0 = 1$, (iii) for second order hyper-exponential distribution (H_2), denote p_1 and p_2 as the phase probabilities. Let μ_1 and μ_2 be the corresponding phase rates. We assume $p_1/\mu_1 = p_2/\mu_2$. We express H_2 parameters in terms of the squared coefficient of variation, c_v^2, and mean inter-server distance, α_X, i.e. we set $p_1 = (1/2)\big(1 + \sqrt{(c_v^2 - 1)/(c_v^2 + 1)}\big), p_2 = 1 - p_1, \mu_1 = 2p_1/\alpha_X$ and $\mu_2 = 2p_2/\alpha_X$. Unless specified, for H_2 we take $c_v^2 = 4$ with $c = 2$. Also if not specified, users are distributed according to a Poisson process and servers a according to a renewal process.

We consider a collection of 10^5 users and 10^5 servers, i.e. $|R| = |S| = 10^5$. We assign users to servers according to MTR. Let $R_M \subseteq R$ be the set of users allocated under MTR. Clearly $|R_M| \leq |R|$. We then run optimal and other greedy policies on the set R_M and S. For each of the experiments, the expected request distance for the corresponding policy is averaged over 50 trials.

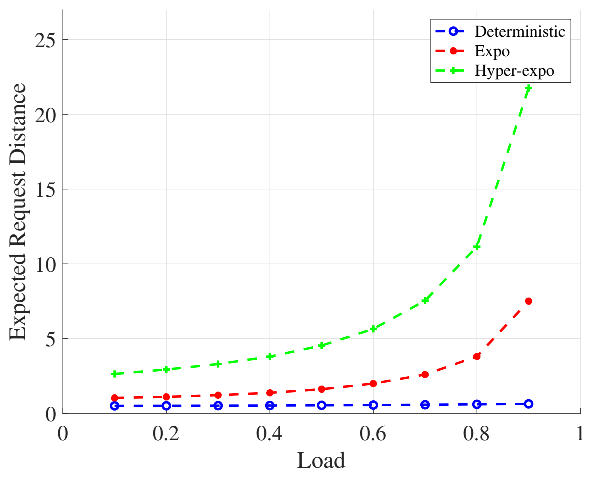
5.8.2 Sensitivity analysis

5.8.2.1 Expected request distance vs. load

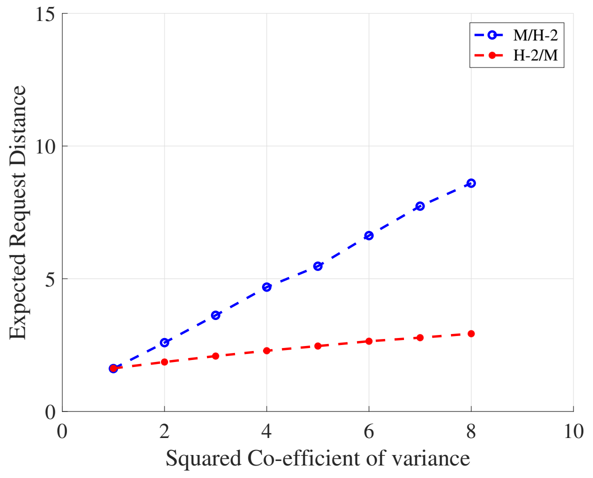
We first study the effect of load ($= \lambda/c\mu$) on $\mathbb{E}[D]$ as shown in Figure 5.6(a). Clearly $\mathbb{E}[D]$ increases as a function of load. Note that H_2 distribution exhibits the largest expected request distance and the deterministic distribution, the smallest because the servers are evenly spaced. While for H_2, c_v^2 is larger than for the exponential distribution. Consequently servers are clustered, which increases $\mathbb{E}[D]$.

5.8.2.2 Expected request distance vs. squared co-efficient of variation

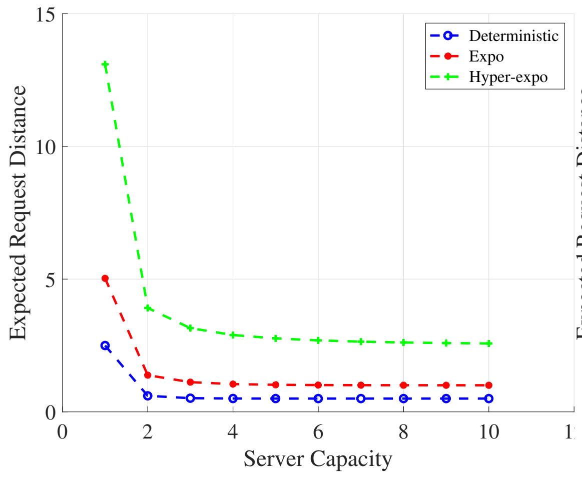
We now examine how c_v^2 affects $\mathbb{E}[D]$ when ρ is fixed. We compare two systems: a general request with Poisson distributed servers (H_2/M) and a Poisson request with general distributed servers (M/H_2) where the general distribution is a H_2 distribution with the same set of parameters, i.e. we fix $\lambda = \mu = 1$ with $c = 2$. The results are shown in Figure 5.6(b). Note that, when $c_v^2 = 1$ H_2 is an exponential distribution and



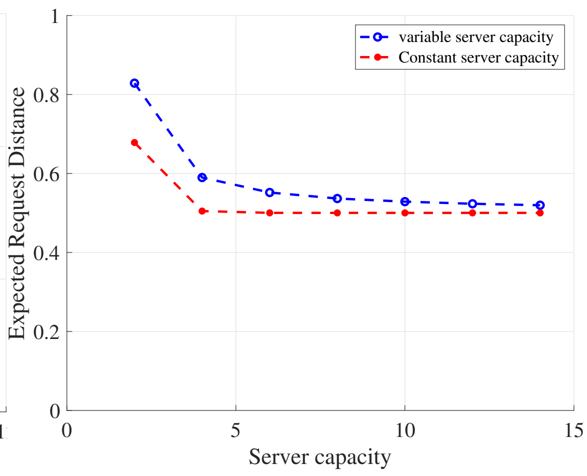
(a)



(b)



(c)



(d)

Figure 5.6: Sensitivity analysis of MTR/UGS policy. (a) Effect of load on expected request distance with $c = 2$. (b) Effect of squared coefficient of variation on expected request distance with $\lambda = \mu = 1$ and $c = 2$. (c) Effect of server capacity on expected request distance with $\rho = 0.8$. (d) Effect of variability in server capacity on expected request distance for Deterministic distribution with $\rho = 0.8$.

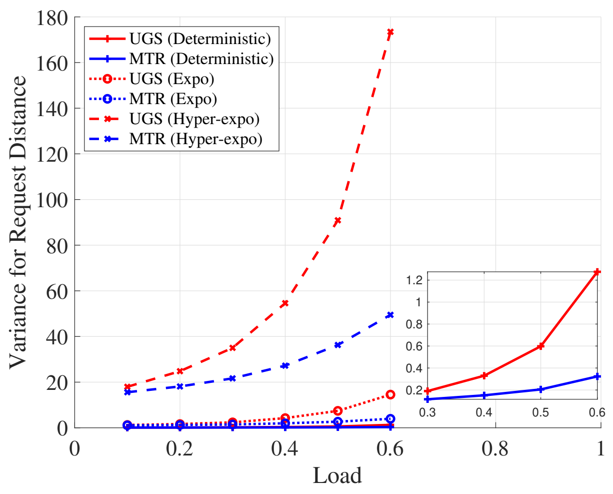
both H_2/M and M/H_2 are identical M/M/1 systems. As discussed in the previous graph, performance of both systems decreases with increase in c_v^2 due to increase in the variability of user and server placements. However, from Figure 5.6(b) it is clear that performance is more sensitive to server placement as compared to the corresponding user placement.

5.8.2.3 Expected request distance vs. server capacity

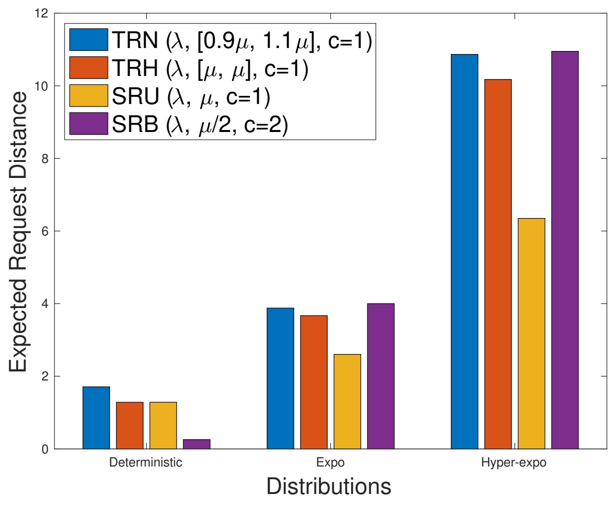
We now focus on how server capacity affects $\mathbb{E}[D]$ as shown in Figure 5.6(c). We fix $\rho = 0.8$. With an increase in c, while keeping ρ fixed, $\mathbb{E}[D]$ decreases. This is because queuing delay decreases. Note that $\mathbb{E}[D]$ gradually converges to a constant value as server capacity increases. Theoretically, this can be explained by our discussion on uncapacitated allocation in Section 5.6.3. As $c \to \infty$ the contribution of queuing delay to $\mathbb{E}[D]$ vanishes and $\mathbb{E}[D]$ becomes insensitive to c.

5.8.2.4 Expected request distance vs. capacity moments

We investigate the heterogeneous capacity scenario as discussed in Section 5.6.2. Consider the plot shown in Figure 5.6(d). We fix $\rho = 0.8$. For the variable server capacity curve we choose the server capacity of each server uniformly at random from the set $\{1, 2, \ldots, 2c\}$. For the constant server capacity curve we deterministically assign server capacity c to each server. We observe better performance for constant server capacity curve at lower values of c under deterministically distributed servers. Variability in constant server case is zero, thus explaining its better performance. Both curves exhibit similar performance under H_2 distribution as well.



(a)



(b)

Figure 5.7: (a) Effect of load on variance of request distance with $\boldsymbol{c = 2}$ across MTR and UGS. (b) Comparison of expected request distance under Two Resource Non-homogeneous (TRN), Two Resource Homogeneous (TRH), Single Resource Unit-service (SRU) and Single Resource Bulk-service (SRB) scenario across various server distributions with $\boldsymbol{\lambda = 0.6, \mu = 1}$.

5.8.2.5 Variance vs. load

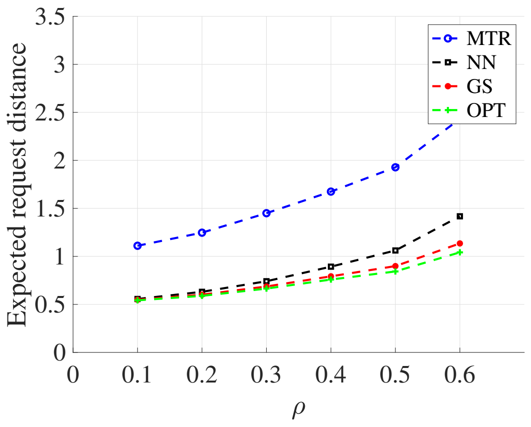
We now study the effect of load on the variance of request distance as shown in Figure 5.7(a). Clearly variance increases as a function of load. Also note that UGS has a higher variance as compared to MTR across all values of load and across various inter-server distance distributions. Provable results exist (from queueing theory) that among all service disciplines the variance of the request distance (or sojourn time in queueing terminology) is minimized under MTR (a FCFS based policy) for Poisson request arrivals and exponential inter-server distances (or service times) [56]. However, these results do not generalize to other inter-server distance distributions in an exceptional service accessible batch queueing discipline. Our simulation based results in Figure 5.7(a) thus bolster our observation in Remark 5 mentioned in Section 5.3. Again, a deterministic equidistant placement of servers produce the least variance for request distance among all other placements.

5.8.2.6 Comparison of two resource and single resource policies

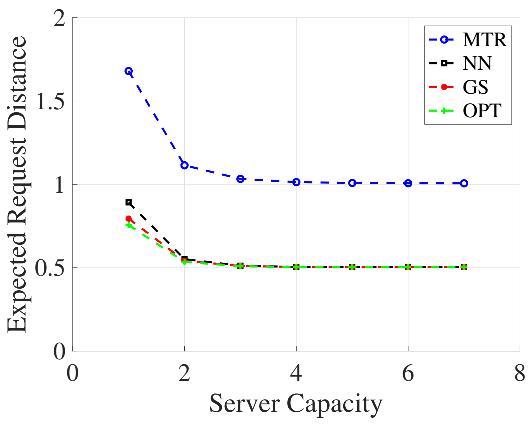
We compare the performance of MTR under various two resource (TR) and single resource (SR) settings as shown in Figure 5.7(b). For a two resource setting, denote $[\mu_1, \mu_2]$ as the server densities associated with resources of types 1 and 2 respectively as described in Section 5.6.5. Denote c as the server capacity associated with each resource type. We define a Two Resource Homogeneous (TRH) system to be a two resource setting with $\mu_1 = \mu_2 = \mu$. We define a Two Resource Non-homogeneous (TRN) system to be a two resource setting with $\mu_1 \neq \mu_2$. For simulation purpose, we chose $\mu_1 = \mu + \epsilon$ and $\mu_2 = \mu - \epsilon$ such that the effective server density remains μ. We also choose $c = 1$. A Single Resource Unit-service (SRU) system is a single resource system with server density μ and $c = 1$. A Single Resource Bulk-service (SRB) system is also a single resource system with server density $\mu/2$ and $c = 2$. Note that the request density and effective server densities ($c\mu$) are same in all settings. From Figure 5.7(b), it is clear that TRH performs better than TRN across all server distributions. This advocates for maintaining similar densities for each resource type in a two resource system. As expected, a deterministic equidistant placement of servers produce the least expected request distance for each system among all other choice of placements. SRB in deterministic server placement scenario performs the best among all other settings. However, it does not perform well with other server distributions. Also, note that, TRH has a higher expected request distance as compared to SRU across all server distributions. Thus (5.28) in Section 5.6.5 holds true even under non-markovian setting.

5.8.3 Comparison of different allocation policies

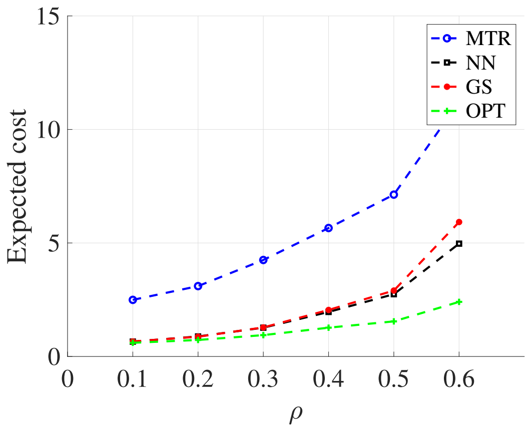
We consider the case in which both users and servers are distributed according to Poisson processes. From Figure 5.8 (a), we observe that due to its directional nature MTR has a larger expected request distance compared to other policies while



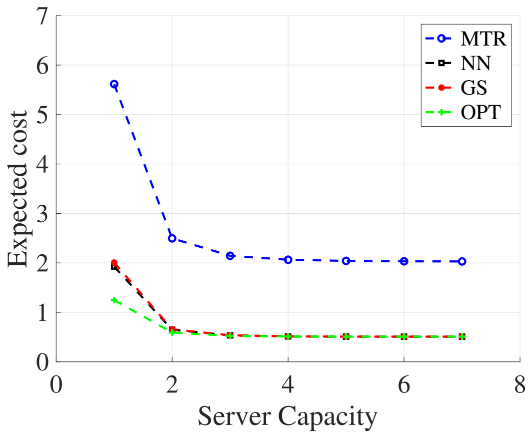
(a)



(b)



(c)



(d)

Figure 5.8: Comparison of different allocation policies: (a) ρ vs $\mathbb{E}[D]$ with $c = 1$, (b) c vs. $\mathbb{E}[D]$ with $\rho = 0.4$, (c) ρ vs $\overline{T}$ with $\beta = 2, t_0 = 1, c = 1$ and (d) c vs. $\overline{T}$ with $\beta = 2, t_0 = 1, \rho = 0.4$.

GS provides near optimal performance. At low loads i.e. when $\rho \ll 1$, the Nearest Neighbor policy policy performs similar to the optimal policy. But as $\rho \to 1$, the NN policy perform worse.

In Figure 5.8 (b), we compare the performance of allocation policies across different server capacities. The expected request distance decreases with increase in server capacities across all policies. NN, GS and the optimal policy converge to the same value as c gets higher.

We now consider the expected communication cost as the performance metric. We use a cost model described in Section 5.6.4 with the parameter $\beta = 2$ and $t_0 = 1$. From Figure 5.8 (c), we observe that while at low loads i.e. when $\rho \ll 1$, GS and NN perform similar to the optimal policy, as ρ increases both GS and NN perform worse. Note that, NN has a higher expected request distance than GS at high load as shown in Figure 5.8 (a). However, the performance is reversed with $\beta = 2$, i.e. NN has a lower expected cost than GS at high load as shown in Figure 5.8 (c). This depicts the effect of β on the performance of various allocation policies. In Figure 5.8 (d), we observe that NN, GS and the optimal policy converge to the same value as c gets higher.

We observe similar trends in the case of deterministic inter-server distance distributions. However, under equal densities, all the policies produce smaller expected request distance as compared to their Poisson counterpart. This advocates for placing equidistant servers in a bidirectional system with Poisson distributed requesters to minimize expected request distance.

5.9 Conclusion

In this Chapter, we introduced a queuing theoretic model for analyzing the behavior of unidirectional policies to allocate tasks to servers on the real line. We showed the equivalence of UGS and MTR w.r.t the expected request distance and presented results associated with the case when either requesters or servers were Poisson distributed. In this context, we analyzed a new queueing theoretic model: ESABQ, not previously studied in queueing literature. We also proposed a dynamic programming based algorithm to obtain an optimal allocation policy in a bi-directional system. We performed sensitivity analysis for unidirectional system and compared the performance of various greedy allocation strategies along with the unidirectional policies to that of optimal policy.

CHAPTER 6

PROXIMITY AWARE LOAD BALANCING POLICIES IN TWO DIMENSIONAL SPATIAL NETWORKS

In Chapter 5 we designed resource allocation policies for a one-dimensional spatial network. This chapter addresses the question: *How should we design proximity aware randomized load balancing policies in a two-dimensional spatial network?*

6.1 Background

The past few years have witnessed an increased interest in the use of large-scale parallel and distributed systems for database and commercial applications. An important design goal of such a system is to distribute service requests or jobs among servers or distributed resources as evenly as possible. While the optimal server[1] selection problem can be solved centrally, due to scalability concerns, it is often preferred to adopt distributed randomized load balancing strategies to distribute these jobs among servers. This leads to the formulation of a randomized *load balancing* problem for the distributed systems with the goal to make the overall user-to-server assignments as fair as possible. Many previous works [4, 67] have used randomization effectively to develop simple and efficient load balancing algorithms in non-geographic settings. A randomized load balancing algorithm can be described as a classical balls and bins problem as follows.

In the classical balls-and-bins model of randomized load balancing, m balls are placed sequentially into n bins. Each ball samples d bins uniformly at random and is

[1]We use the terms "Servers" and "Resources" interchangeably.

allocated to the bin with the least number of balls, ties broken arbitrarily. It is well known that when $d = 1$ and $m = n$, this assignment policy results in a maximum load of $O(\log n/ \log\log n)$ with high probability [11]. However, if $d = 2$, then the maximum load is $O(\log\log n)$ w.h.p. [11]. Thus, there is an exponential improvement in performance from $d = 1$ to $d = 2$. This policy with $d = 2$ is widely known as *Power of Two* (POT) choices and the improvement in maximum load behavior is known as *POT benefits* [67]. Many subsequent works have studied assignment policies that generalize POT policy to account for correlated and non-uniform sampling strategies [17], [24], [96].

6.1.1 Spatial Load Balancing

While classical balls and bins based randomized load balancing can directly be used for user/job to server assignment in a geographic setting, it is oblivious to the spatial distribution of servers and users. We define the cost of moving jobs/results to/from their allocated servers as the *implementation cost* associated with a given policy. The implementation cost generally increases with the Euclidean distance between the user and its allocated server, also known as *request distance*.

For example in a wireless network, signal attenuation is strongly coupled to request distance, therefore developing allocation policies that minimize request distance can help reduce energy consumption. Thus the following natural question arises.

How should we design proximity aware load balancing policies that also reduce average implementation cost?

In this chapter, we aim to answer this question. To this end we propose a spatially motivated POT policy: *spatial POT* (sPOT) in which each user is allocated to the least loaded server among its two geographically nearest servers. We assume both users and servers are placed in a two-dimensional Euclidean plane. When both servers and users are placed uniformly at random in the Euclidean plane, we map sPOT to a classical balls and bins allocation policy with bins corresponding to the Voronoi regions associated with the second order Voronoi diagram of the set of servers. We show that sPOT performs better than POT in terms of average request distance. However, a lower bound analysis on the asymptotic expected maximum load for sPOT suggests that POT load balancing benefits are not achieved by sPOT.

Inspired by the analysis of sPOT, we further propose two assignment policies and empirically show that these policies are able to substantially reduce both request distance and maximum load. We first propose a server proximity aware policy, *Unif-POT(k)*, as follows. For each job, two servers u and v are sampled uniformly at random from its k-nearest servers. The job is then allocated to the server with the smallest load among u and v. Since a POT optimally balances load (by stochastic majorization argument), we compare the load distribution of Unif-POT(k) to that of POT. We also propose another proximity based load balancing policy: *InvSq-POT(k)* as follows. For each job, two servers u and v are sampled from its k-nearest servers with probabilities proportional to the inverse square of the Euclidean distances between the user and the corresponding server. The job is then allocated to the server with the smallest load among u and v. Through extensive simulations we verify that such a simple modification in the sampling technique, produces a load distribution behavior very similar to that of a POT policy while drastically reducing the average implementation cost across a variety of network topologies.

6.2 System Model

In this section, we introduce the system model used in the rest of the chapter. We have a set of users/jobs[2] R with $|R| = m$. Similarly, S denotes the set of servers with $|S| = n$. Let $\pi : R \to S$, denote a load balancing policy for assigning users/jobs to servers.

[2]We use the terms "users" and "jobs" interchangeably.

We consider a service network where users and servers are located on a two-dimensional Euclidean plane $\mathcal{D} \subset \mathbb{R}^2$. We assume users are placed on $\mathcal{D}$ uniformly at random. In the service network, each user is assigned to a server from the server set S. We consider two cases for placing the servers on a two-dimensional euclidean plane, (i) Grid Placement: servers are placed on a square grid topology (ii) Uniform placement: servers are placed uniformly at random in $\mathcal{D}$.

Denote $d(j, v)$ as the euclidean distance between user j and server v. We define the following geometric structures that are useful constructs for analyzing various load balancing policies on a plane.

Definition 1. *Voronoi Diagram: A Voronoi cell around a server $s \in S$ is the set of points in $\mathcal{D}$ that are closer to s than to any other server in $S \setminus \{s\}$ [15]. The Voronoi diagram V_S of S is the set of Voronoi cells of servers in S.*

Definition 2. *Delaunay Graph: The Delaunay graph, $G_S(S, E)$, is associated with the set of servers S. Here $(u, v) \in E$ iff the Voronoi cells of $u, v \in S$ are adjacent.*

Definition 3. *Higher order Voronoi diagram: A p^{th} order Voronoi diagram, $H_S^{(p)}$, is defined as partition of $\mathcal{D}$ into regions such that points in each region have the same p closest servers in S.*

In this chapter, our goal is to analyze the performance of several load balancing policies on a plane including the two classic policies.

- **Power of One (*POO*):** This policy assigns each user to one of the servers chosen uniformly at random from S.
- **Power of Two (*POT*):** In this policy, sequentially each user samples two servers uniformly at random from S and is allocated to the least loaded server.

In addition we propose new policies to reduce both maximum load and expected request distance. We define them as follows.

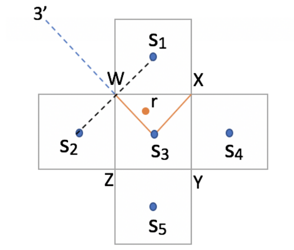


Figure 6.1: Second nearest region for user r

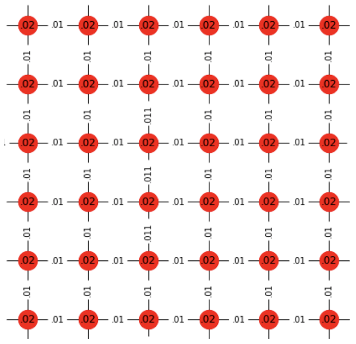


Figure 6.2: Delaunay Graph associated with grid based server placement

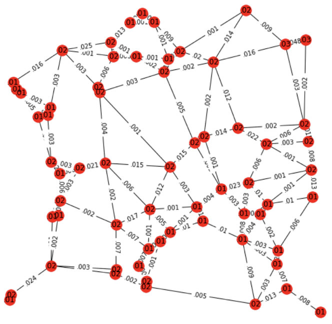


Figure 6.3: Delaunay Graph associated with uniform server placement

- **Unif-POT(k):** Each user samples two servers uniformly at random from a candidate set consisting of its k geographically nearest servers and is assigned to the least loaded server.

- **InvSq-POT(k):** In this policy each user j samples two servers from a candidate set consisting of its k geographically nearest servers (without replacement), each with probability proportional to $1/d(j,v)^2$. The user is then assigned to the least loaded server.

For our analysis we consider special cases of Unif-POT(k) policies with $k = 2$ and $k = 1$. We call them Spatial Power of Two ($sPOT$) and Spatial Power of One ($sPOO$) policies respectively for brevity. To be precise these policies are defined as

- **Spatial Power of Two ($sPOT$):** Each user is sequentially allocated to the least loaded server among its two geographically nearest servers.

- **Spatial Power of One ($sPOO$):** This policy assigns each user to its geographically nearest server.

6.2.1 Performance Metrics

To evaluate and characterize the performance of various load balancing policies, we define the performance metrics for both plane and graph based system as follows.

Denote $x^\pi(t) = [x_i^\pi(t), i \in \{1, \cdots, m\}]$ as the state of the system immediately after the t^{th} job is assigned under policy π. Here $x_i^\pi(t)$ denotes the fraction of servers with exactly i jobs immediately after t^{th} job is assigned. Denote $x^\pi(m)$ as the load distribution under policy π after all of m jobs are assigned.

Definition 4. *Maximum Load: The maximum load for policy π is defined as*

$$ML^\pi = i \text{ with } x_i^\pi(m) \neq 0 \text{ and } x_j^\pi(m) = 0 \text{ for } j = i+1, \cdots, m.$$

Definition 5. *Total Variation Distance: The total variation distance between two load distributions $x^{\pi_1}(m)$ and $x^{\pi_2}(m)$ is*

$$TV^{\pi_1\pi_2} = \frac{1}{2}\sum_{i=1}^{m} |x_i^{\pi_1}(m) - x_i^{\pi_2}(m)|.$$

$TV^{\pi_1\pi_2}$ takes values in $[0, 1]$. The closeness of two load distributions under two different policies can be measured by the total variation distance, i.e. the smaller the total variation distance the closer the two distributions are to each other.

Definition 6. *Average Request Distance: The average request distance for policy π is the average distance (or number of hops) between a random user (or its origin server) and its allocated server under π, i.e.*

$$RD^\pi = \frac{1}{m}\sum_{r \in R} d(r, \pi(r)).$$

Since POT is oblivious to inter server distances, RD^{POT} is generally large compared to other proximity based load balancing policies.

6.3 Spatial Power of Two policy on a plane

We now analyze the load behavior of sPOT policy for various server placements on a plane. We assume users are placed uniformly at random on $\mathcal{D}$.

6.3.1 sPOT with Grid based server placement

Consider the case where servers are placed on a two dimensional square grid: $\sqrt{n} \times \sqrt{n}$ on $\mathcal{D}$ with wrap-around. Let $B(\{s_1, s_2\}, r)$ be the event that the two nearest servers of r are in $\{s_1, s_2\}$. We establish the following result.

Lemma 5. *Let $G_S(X, E)$ denote the Delaunay graph associated with S when servers are placed on a two-dimensional square grid. Then*

$$\Pr[B(\{s_i, s_j\}, r)] = \begin{cases} \frac{1}{|E|}, & (s_i, s_j) \in E; \\ 0, & \textit{otherwise.} \end{cases} \tag{6.1}$$

Proof. See appendix E.1. □

We make use of the following lemma presented in [53].

Lemma 6. *Given a Δ-regular graph with n nodes representing n bins, if n balls are thrown into the bins by choosing a random edge and placing into the smaller of the two bins connected by the edge, then the maximum load is at least $\Omega(\log\log n + \frac{\log n}{\log(\Delta \log n)})$ with high probability of $1 - 1/n^{\Omega(1)}$.*

We now prove the following theorem.

Theorem 11. *Suppose servers are placed on a two dimensional square grid : $\sqrt{n}\times\sqrt{n}$ on $\mathcal{D}$ with wrap-around. Let users be placed independently and uniformly at random on $\mathcal{D}$. Under sPOT, the maximum load over all servers is at least $\Omega(\frac{\log n}{\log\log n})$ with probability $1 - 1/n^{\Omega(1)}$.*

Proof. Suppose we map the set of servers to the bins and the users to the balls. The Delaunay graph G_S is 4-regular. Let $e = (s_i, s_j)$ be an edge in G_S. From Lemma 5, it is clear that each user (ball) selects an edge e with probability $1/|E|$ (i.e. uniformly at random) and is allocated to the server (bin) connected by e with the least number of users under sPOT. Thus a direct application of Lemma 6 with $\Delta = 4$ proves the theorem. □

We verify the results in Lemma 5 through simulation for a 2D square grid under sPOT as shown in Figure 6.2. We set $n = 64$ and empirically compute $\Pr[B(\{s_i, s_j\}, r)]$ and denote it as edge probability on edge e on the Delaunay graph. We also verify $\Pr[A(s, r, 1)] = 1/|S|\ \forall s \in S$ expression and denote it as vertex probability on the Delaunay graph. It is clear from Figure 6.2 that the edge probabilities are almost all equal and so are the vertex probabilities.

Remark 6. *Note that, Theorem 11 concludes that we do not get POT benefits when servers are placed on a two dimensional square grid.*

Remark 7. *Note that Theorem 11 applies to other grid graphs such as a triangular grid, i.e. we do not get POT benefits when servers are placed on a two dimensional triangular grid. The Delaunay graph corresponding to a triangular grid based server placement is* 6- *regular.*

6.3.2 sPOT with Uniform server placement

We now consider the case where both users and servers are placed uniformly at random on $\mathcal{D}$. We can no longer invoke Lemma 6. This is due to the fact that the Delaunay graph associated with the servers is no longer regular. Also, the edge sampling probabilities $\Pr[B(s_i, s_j, r)]$ are no longer equal. This is evident from our simulation results on the corresponding Delaunay graph as shown in Figure 6.3. We have $n = 64$ servers placed randomly in a 2D square and empirically compute $\Pr[B(\{s_i, s_j\}, r)]$ and denote it as edge probability on edge e on the Delaunay graph. Note that the edge probabilities, i.e. $\Pr[B(\{s_i, s_j\}, r)]$, differ from each other. Also the Delaunay graph is not regular. Thus we resort to using a second order Voronoi diagram to analyze the maximum asymptotic load behavior.

6.3.2.1 Majorization Basics

We present a few definitions and basic results associated with majorization theory that we will use to analyze sPOT.

Definition 7. *The vector x is said to majorize the vector y (denoted $x \succ y$) if*

$$\sum_{i=1}^{k} x_{[i]} \geq \sum_{i=1}^{k} y_{[i]},\ k = 1, \cdots, n-1,$$
$$\textit{and} \sum_{i=1}^{n} x_{[i]} = \sum_{i=1}^{n} y_{[i]}$$

where $x_{[i]}(y_{[i]})$ is the i^{th} largest element of $x(y)$.

Definition 8. *A function $f : R^n \to R$ is called Schur-convex if*

$$x \succ y \implies f(x) \geq f(y)$$

Consider the following proposition (Chapter 11, [64])

Proposition 1. *Let X be a random variable having the multinomial distribution*

$$\Pr[X = x] = \binom{n}{x_1, \cdots, x_n} \prod_{i=1}^{n} p_i^{x_i}$$

where $x = (x_1, ..., x_n) \in \{z : z_i$ are nonnegative integers, $\sum z_i = n\}$. If δ is a Schur-convex function of X, then $\psi(p) = E_p \delta(X)$ is a Schur-convex function of p.

6.3.2.2 Loss of POT benefits under sPOT

Consider the second order Voronoi diagram: $H_S^{(2)}$ associated with the set of servers S. We have the following Lemma [Chapter 3.2, [70]].

Lemma 7. *The number of Voronoi cells in $H_S^{(2)}$ under uniform server placement is upper bounded by $O(3n)$.*

We also have the following Lemma.

Lemma 8. *Consider the following modified version of balls and bin problem. Suppose there are n balls and n bins. Each ball is thrown into one of the bins according to a probability distribution $p = (p_1, \cdots, p_n)$ with p_i being the probability of each ball falling into bin i, in an independent manner. Denote Z to be the random variable associated with the maximum number of balls in any bin. Then we have*

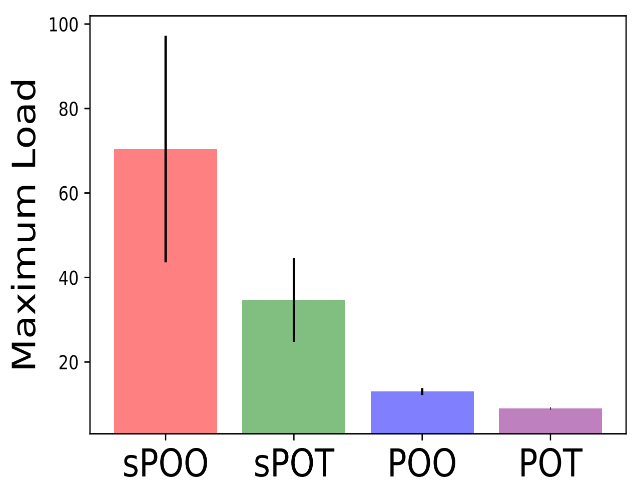
$$E_p[Z] \geq k_0 \frac{\log n}{\log \log n} \text{ as } n \to \infty.$$

where k_0 is a scalar constant.

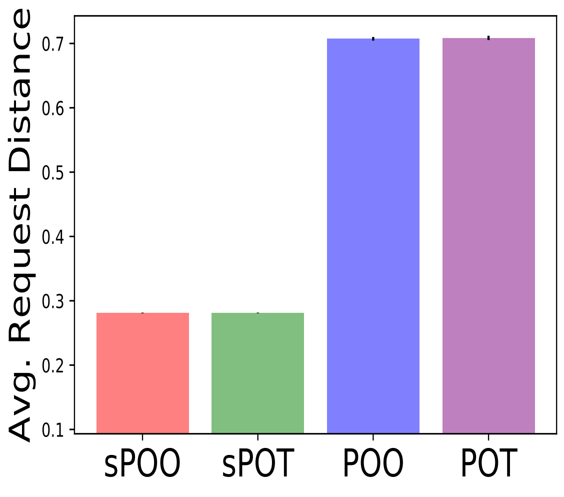
Proof. See appendix E.2. □

Theorem 12. *Suppose both users and servers are placed independently and uniformly at random on $\mathcal{D}$. Under sPOT, the expected maximum load over all servers is at least $\Omega(\frac{\log n}{\log \log n})$ with probability of $1 - 1/n^{\Omega(1)}$, i.e., we do not get POT benefits.*

Proof. See appendix E.3. □



(a) Maximum load



(b) Expected request distance

Figure 6.4: Performance comparison of basic allocation policies wrt (a) maximum load and (b) expected request distance for $n = 10000$ servers.

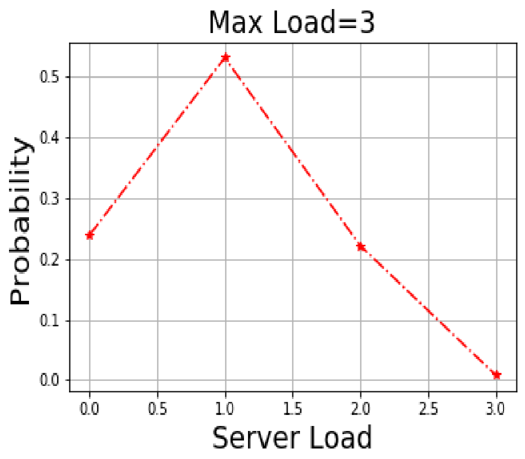
6.3.3 Tradeoff between Load and Request Distance

In this section, we discuss the inherent tradeoff between maximum load and expected request distance metric among different allocation policies. We evaluate the performance of sPOT and compare it to that of other allocation policies. We consider $n = 10000$ servers and an equal number of users placed uniformly at random on a unit square. We ran 10 trials for each policy. We compare the performance of various allocation policies in Figure 6.4.

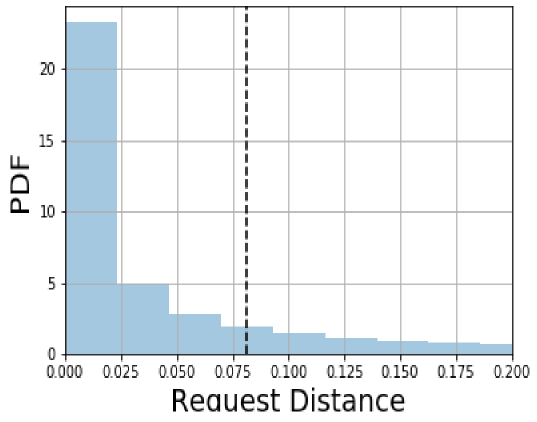
First, note that with respect to maximum load, the spatial based policies perform worse compared to their classical counterparts, POO and POT. Note that the introduction of spatial considerations into a policy increases its maximum load. For example, sPOT performs worse than both POO and POT as shown in Figure 6.4 (a). Since the maximum asymptotic load for POO is $O(\log n/\log\log n)$ with high probability, Figure 6.4(a) validates our lower bound results obtained for sPOT in Theorem 12.

However, the expected request distance is smallest for sPOO and almost similar to that of sPOT. Also, both POT and POO have very large and similar expected request distances as shown in Figures 6.4(b). Both results, shown in Figures 6.4(a) and (b) combined, illustrate the tradeoff between maximum load and expected request distance metric.

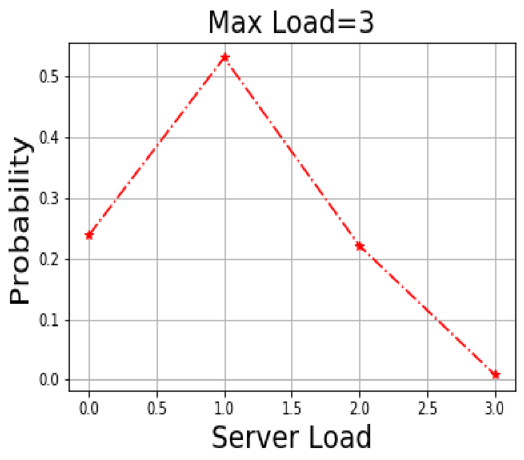
6.4 Improving Performance of sPOT



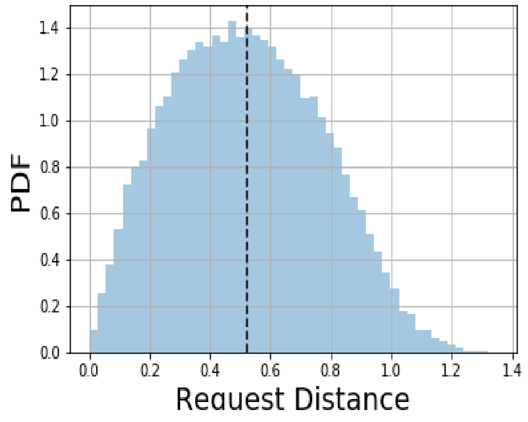
(a) InvSq-POT(n) Load distribution



(b) InvSq-POT(n) distance distribution



(c) POT Load distribution



(d) POT distance distribution

Figure 6.5: Performance comparison of allocation policies wrt InvSq-POT(n) for $n = 50000$ servers. (a) and (b) plots are for InvSq-POT(n) while (c) and (d) for POT.

In Section 6.3, we showed that for both grid and uniform based server placement, sPOT does not provide POT benefits. As POT is oblivious to the spatial locations of users and servers, it performs worse with respect to the expected request distance metric. Thus there exists a tradeoff between maximum load and expected request distance among different allocation policies.

Note that for each user, once its arrival location is known, the choice of two servers by sPOT is deterministic while it is completely random for POT. This random sampling over the entire set of servers results in better load behavior for POT than for sPOT. However, since random sampling is oblivious to the distances of servers from the particular user, POT exhibits a large expected request distance. Thus if one can design a policy with random and distance dependent sampling of servers, such a policy should provide benefits of both POT and sPOT in terms of maximum load and expected request distance. Below we propose and evaluate two such policies that obtain benefits of both POT and sPOT. We empirically show that they achieve both POT like load benefits while having a request distance profile similar to that of sPOT.

6.4.1 InvSq-POT(k)

Consider the allocation of a random user j in the service network. We propose InvSq-POT(k) to allocate j as follows. Under InvSq-POT(k), j samples two servers from a candidate set consisting of its k geographically nearest servers (without replacement), each with probability proportional to $1/d(j,v)^2$. Here, $d(j,v)$ denotes the euclidean distance between user j and server v. User j is then allocated to the least loaded server among the two sampled servers. This rule is similar to one used in small world routing [58]. Note that, since the probability of sampling a server is inversely proportional to its distance from the user, InvSq-POT(k) incurs a smaller expected request distance compared to POT. Surprisingly, InvSq-POT(k) achieves similar load behavior to that of POT. We compare the performance of InvSq-POT(k) to sPOT and POT as follows.

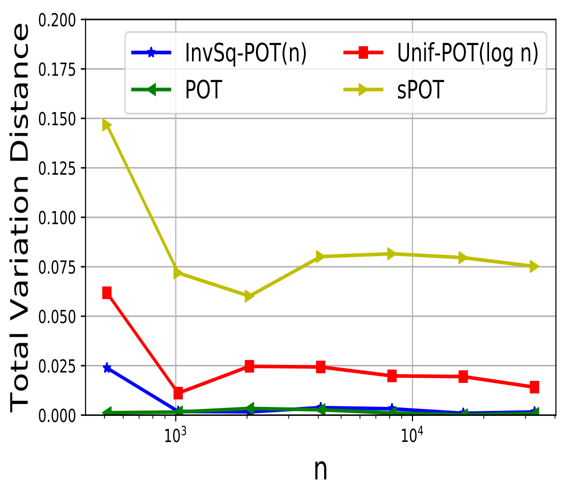
We perform a single simulation run for each of the policies: InvSq-POT(n), sPOT, POT. We define the load associated with a server to be the number of users assigned to it. We measure the load distribution across all servers and the request distance distribution. Figure 6.5(a) shows the load distribution and Figure 6.5(b) shows the request distance distribution for InvSq-POT(n). We plot the load distribution for POT and request distance distribution for sPOT in Figure 6.5(c) and (d) respectively.

First we focus on the server loads in Figures 6.5(a) and (c). Interestingly, the load distributions are almost identical for InvSq-POT(n) and POT. Similarly, sPOT performs better than InvSq-POT(n) in terms of request distance distribution as shown in Figures 6.5(b) and (d) since they significantly favor closer nodes. However, compared to POT (as shown in Figure 6.5(e)), InvSq-POT(n) performs significantly better in terms of request distances. Thus InvSq-POT(n) achieves the best of both worlds, i.e., small maximum load and small request distances.

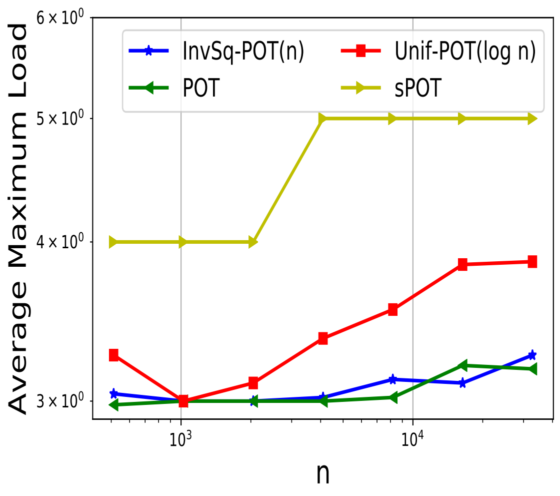
6.4.2 Unif-POT(k)

We now propose a policy that improves the load behavior of sPOT. We define C_k to be the candidate set (of size k) consisting of the k nearest servers to a particular user. Under Unif-POT(k), the user selects two servers uniformly at random from C_k and assigns itself to the least loaded one. Note that, random sampling of two servers within the candidate set helps to balance load and reduce overall maximum load. Clearly sPOT and POT are two extremes of the policy Unif-POT(k) with $k = 2$ and $k = n$ respectively. Below, we discuss the effect of k on maximum load and expected request distance and compare it to other policies.

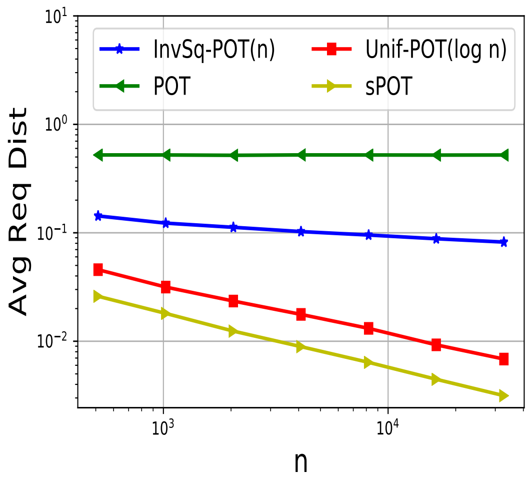
We present total variation distance between load distributions of proximity based policies and POT as a function of number of servers in Figure 6.6 (a). We also plot the



(a)



(b)



(c)

Figure 6.6: Performance comparison of Unif-POT(k) and InvSq-POT(n) with respect to (a) total variation distance to POT (b) average maximum load and (c) average request distance when servers placed uniformly at random on a plane.

total variation distance between load distributions of two independent runs of POT which quantifies the noise or variation in load distribution of POT due to randomness. Both Unif-POT($\log n$) and InvSq-POT(n) achieve total variation distances as low as 0.02 across a wide range of values of n. Also note that, sPOT achieves the load distribution farthest from POT while InvSq-POT(n) achieves the closest. Due to load-implementation cost trade-off, a very local policy sPOT, achieves larger variation distance.

Figure 6.6(b) shows average maximum load as n varies. We observe that both InvSq-POT(n) and POT perform the best. Unif-POT(k) with $k = \log n$ performs quite well compared to sPOT. Also, we have observed through simulation that the average maximum load profiles are very similar for Unif-POT(k) and sPOT when $k = O(1)$. Based on these results, we present the following conjecture.

Conjecture 2. *If the candidate set in Unif-POT(k) does not grow with n, no POT benefit is expected.*

Figure 6.6(c) shows how the average request distance drops as n increases (since the node density increases). We observe that, not surprisingly, sPOT outperforms the other policies. However, Unif-POT(k) with $k = \log n$ performs well. Thus Unif-POT(k) with $k = O(\log n)$ achieves good performance for both load and request distance.

Remark 8. *Among the proximity aware POT policies on a plane, InvSq-POT(n) selects servers through distance based sampling. Thus even a minor change in positions of servers theoretically requires choosing a new set of sampling distributions. However, sampling in Unif-POT($\log n$) depends on the local neighborhood of the user and thus involves less frequent updates for server sampling distributions.*

6.5 Conclusion

In this chapter we considered a class of proximity aware power of two choices based allocation policies where both servers and users are located on a two-dimensional plane. We analyzed the sPOT policy and provided expressions for the lower bound on the asymptotic maximum load on the resources. We claim that for both grid and uniform based resource placement, sPOT does not provide POT benefits. We proposed two non-uniform euclidean distance based server sampling policies that achieved the best load and request distance behavior.

CHAPTER 7

PROXIMITY AWARE LOAD BALANCING POLICIES ON GRAPHS

In Chapter 6 we designed proximity aware load balancing policies for a two-dimensional distributed service network. In this chapter we design proximity aware load balancing policies for the case when resources/servers are connected as generic graph.

7.1 Background

In Chapter 6, we studied the problem of balancing load. Load balancing is also important in other settings such as arbitrary graphs. This has application in many fields including bike-sharing systems, World Wide Web, peer-to-peer networks, vehicular wireless ad-hoc networks. Also, load balancing can prolong the lifetimes of battery-powered wireless sensor and actuator networks where energy is a limited resource [54].

Load balancing algorithms for certain fixed-degree deterministic graphs, such as ring topologies, have been studied in [42], [93] and have applications in bike-sharing systems. While many complex networks like the World Wide Web or peer-to-peer networks can be modeled as scale free or random regular graphs [28], little is known about load balancing policies on such random networks. Moreover, other complex systems such as transportation and mobility networks are often best represented as spatial graphs where nodes and edges are embedded in Euclidean space. For example, the communication network resulting from radio transmitters and wireless devices can be described by a random geometric graph [81]. Such networks have natural notions of distance and the cost of assigning a job scales with distance. Hence it is important to account for this geographical aspect when designing load balancing policies.

In the graph model, servers are represented as vertices of an arbitrary graph $G(S, E)$. When a job arrives at a server u (origin server), it is assigned to the server with the least load among server u and $d-1$ servers sampled uniformly at random from its one-hop neighborhood in G [55]. In order to make the graph model more tractable for theoretical analysis, many simplified assumptions have been made about the graph structure. For example, Kenthapadi et al. [55] studied the scaling of maximum load for the case $d = 2$ by allowing G to be regular or almost regular with degree n^{ϵ}. However, in practice, real world networks are highly irregular. Moreover, previous work lacks a comprehensive study on characterizing the implementation cost associated with load balancing policies on real world networks.

More often previous work only consider developing theoretical framework to characterize the scaling of maximum load behavior. In some applications, the load distribution may be more important than the maximum load since it yields a better resolution into the load characteristics of the network. Similarly, while designing proximity aware load balancing policies, one may try to balance two performance metrics: load and implementation cost. Therefore, while the maximum load can end up being higher in such policies than classical POT, if the distribution is nearly the same as POT one should consider a policy with significantly lower implementation cost as better than POT, and almost as good as POT in the sense of load balancing. These discussions raise the following research questions.

1. How should one evaluate the performance of proximity aware load balancing policies for non-regular graph models, such as random, scale free or spatial graph structures?

2. What is a good performance metric to characterize the implementation cost associated with real world complex networks?

3. What is the effect on load and implementation cost if servers are sampled from a k-hop neighborhood instead of a one hop neighborhood with $k \geq 2$?

4. How close can the performance of a proximity based policy be made to that of POT with respect to load distribution instead of maximum load metric?

The primary motivations behind this chapter is to address these questions. The key challenge in developing theoretical frameworks to answer these questions is that the notion of neighborhood for each job heavily depends on the choice of graph topology. Thus even asymptotic results for load balancing policies under generic graph model are scarce and techniques like witness tree methods [55] are not applicable. There is little hope of analyzing the graph model in this generality and generating analytical insights seems difficult to achieve. For this reason we investigate this model through detailed and extensive computer simulations across a variety of graph topologies in Section 7.3.

Similar to Chapter 6, we first propose a server proximity aware policy, *Unif-POT(k)*, as follows. For each job, a server v is sampled uniformly at random from the k-hop neighborhood of the origin server u. The job is then allocated to the server with the smallest load among u and v. Since a POT policy optimally balances load (by the stochastic majorization argument), we compare the load distribution of Unif-POT(k) policy to that of POT policy. We also propose another proximity based load balancing policy: *InvSq-POT(k)* as follows. For each job, a server v is sampled from the k-hop neighborhood of origin server u with probability proportional to the inverse square of the shortest path distance measured in number of hops between

u and v. The job is then allocated to the server with the smallest load among u and v. Through extensive simulations we verify that such a simple modification to the sampling technique, produces load distribution behavior very similar to that of a POT policy while drastically reducing the average implementation cost across a variety of network topologies. Our simulations demonstrate a total variation distance as low as $0.002 - 0.005$ between load distributions of classical POT and proposed proximity based policies while achieving a significant reduction in implementation cost on the order of $20\% - 99\%$ for proposed proximity based policies when compared to classical POT.

7.2 System Model

In this section, we introduce the system model used in the rest of the chapter. We denote the users/jobs[1] in the system as the set R with $|R| = m$. Similarly, denote S as the set of servers with $|S| = n$. Let $\pi : R \to S$ denote a load balancing policy for assigning users/jobs to servers.

We assume servers in the network are nodes of a connected graph $G(S, E)$ with $|S| = n$ and E a set of edges connecting the servers. We explore various random, deterministic and spatial graph structures for graph based load balancing systems. We assume that jobs arrive at one of the servers uniformly at random. Denote u as the arrival (origin) server for a job.

Next we define several network attributes that will be useful in analyzing simulation results obtained for different load balancing policies later. We denote $d(u, v), u, v \in S$ as the shortest path distance measured in number of hops between nodes u and v in the network.

[1]We use the terms "users" and "jobs" interchangeably.

Definition 9. *k-hop Neighborhood: The k-hop neighborhood of a node $u \in S$ is defined as*

$$\mathcal{N}_k(u) = \{w | 1 \leq d(u,w) \leq k\}.$$

Definition 10. *Graph Density: The graph density of an undirected graph $G(S,E)$ is*

$$\rho_G = \frac{|E|}{\binom{n}{2}} = \frac{2|E|}{n(n-1)}.$$

Definition 11. *Average Path Length: The average path length of an undirected graph $G(S,E)$ is*

$$l_G = \frac{1}{n(n-1)} \sum_{u \neq v} d(u,v).$$

We now introduce the three load balancing policies that we study. Suppose a job arrives at origin server $u \in V$. Denote $P_u = [p_{uv}, v \in V]$ as the server sampling distribution for the job where p_{uv} is the probability u queries server v for its load information with $p_{uu} = 0$. The first policy is the well known POT policy and the next two are newly proposed proximity based load balancing policies on a graph G.

- **Power of Two (POT):** If a job arrives at server u, then

$$p_{uv} = \begin{cases} \frac{1}{n-1}, & \text{if } v \neq u, \\ 0, & \text{otherwise .} \end{cases}$$

  That is, server v is sampled uniformly at random from the remaining $n-1$ servers. The job is then allocated to the server with the smallest load among u and v.

- **Unif-POT(k):** According to this policy, if a job arrives at server u, then

$$p_{uv} = \begin{cases} \frac{1}{|\mathcal{N}_k(u)|}, & \text{if } v \in \mathcal{N}_k(u), \\ 0, & \text{otherwise }. \end{cases}$$

  That is, a server v is sampled uniformly at random from the k-hop neighborhood of u[2]. The job is then allocated to the server with the smallest load among u and v.

- **InvSq-POT(k):** According to this policy, if a job arrives at server u, then

$$p_{uv} = \begin{cases} \frac{\left(\frac{1}{d(u,v)^2}\right)}{\sum\limits_{w \in \mathcal{N}_k(u)} \left(\frac{1}{d(u,w)^2}\right)}, & \text{if } v \in \mathcal{N}_k(u), \\ 0, & \text{otherwise }. \end{cases}$$

  That is, a server $v \in \mathcal{N}_k(u)$ is sampled with probability proportional to the inverse square of the distance to u. The job is then allocated to the server with the smallest load among u and v.

Remark 9. *Observe that Unif-POT(k) and InvSq-POT(k) are identical for $k = 1$. Similarly, POT and Unif-POT(k) are identical for $k = n$.*

7.2.1 Performance Metrics

To evaluate and characterize the performance of various load balancing policies, we use Maximum Load, Total Variation Distance and Average Request Distance as performance metrics. These performance metrics are defined in Chapter 6.

[2]While one can sample v from k nearest servers as done for policies on a plane, choosing from the k-hop neighborhood simplifies the graph based model and does not require tie breaking mechanism to determine the sampling space.

7.3 Proximity Aware POT policies on Graphs

In this section we present extensive simulation results to illustrate the effectiveness of both Unif-POT(k) and InvSq-POT(k) policies in graph based load balancing systems. Our study also provides insight into the choice of a load balancing policy under different load conditions and for different network topologies.

We implemented the proposed policies in Python to study their performance in a simulated environment. To make the performance comparisons between the algorithms meaningful, a number of simulation runs were conducted for each algorithm with different parameter values (e.g., system size, average degree etc.) for different graph topologies. We assume the topology remains fixed during the simulation. If not specified, we assume $n = 10000$ servers interconnected through a graph G. Also, $m = 10000$ jobs each arriving sequentially to one of the servers chosen uniformly at random and is allocated to a server according to different proximity based POT policies. We report the average of 10 simulation runs. Usually we set $k = \log n$.

We consider a wide range of network topologies such as: deterministic, random, scale-free and spatial networks. In our study, we evaluate the proposed schemes using total variation distance, average request distance and average maximum load as performance metrics. The results of the simulation experiments are presented in the following sections.

7.3.1 Maximum Load vs Request Distance Tradeoff

We first discuss the inherent tradeoff between average maximum load and average request distance for different values of k in Unif-POT(k) and InvSq-POT(k) policies. We perform a simulation experiment with $n = 1000$ servers connected through a line graph. We assume $m = 1000$ jobs each arriving sequentially to one of the servers chosen uniformly at random and is allocated to a server according to Unif-POT(k) policy. We report the average of 10 simulation runs. We plot both average maximum

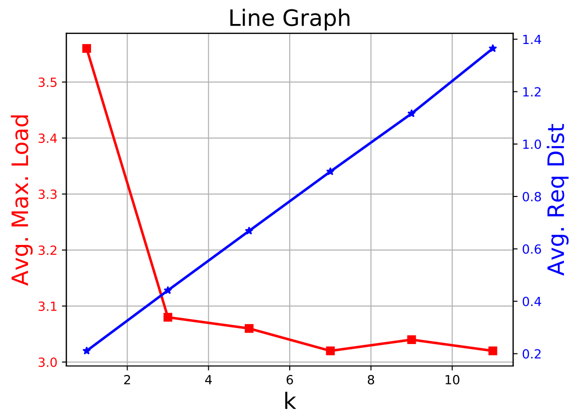


Figure 7.1: Trade-off between average maximum load and average request distance for servers on a Line graph with $m = n = 1000$ for Unif-POT(k) policy under static load balancing system.

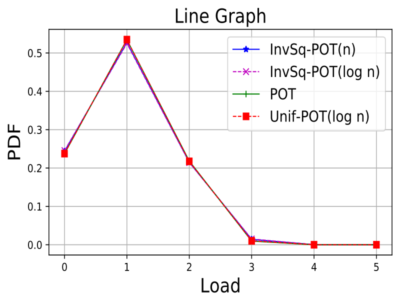
load and average request distance as a function of neighborhood parameter k as shown in Figure 7.1. We obtain similar results for the case when allocation is done according to InvSq-POT(k) policy.

It is clear from Figure 7.1 that average maximum load value decreases as k increases. This is because the size of k-hop neighborhood of an origin server increases as k increases. Thus the load is distributed among a larger group of servers and the behavior of Unif-POT(k) resembles more and more that of POT policy for large values of k.

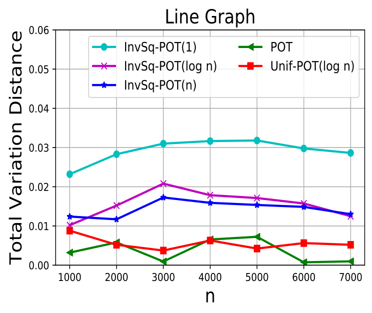
However, an increase in k results in larger values of average request distance. When k is small, the sampled servers remain close to the origin server. However, as k increases, the size of the k-hop neighborhood grows. One is more likely to sample a far away server thereby increasing the average request distance. Thus one needs to be careful in choosing the correct value of k according to the performance metric of interest.

7.3.2 Performance Comparison for Deterministic Graphs

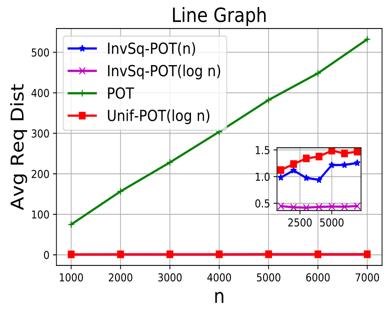
In this section, we analyze the performance of fixed degree deterministic graphs: Line and Ring. The results are presented in Figure 7.2. First we plot the load



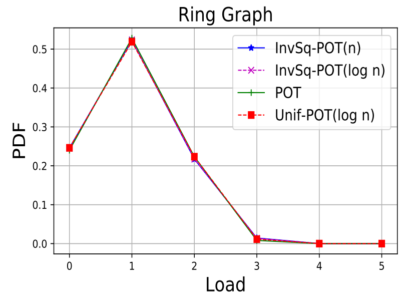
(a)



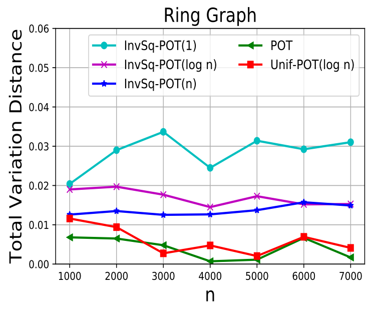
(b)



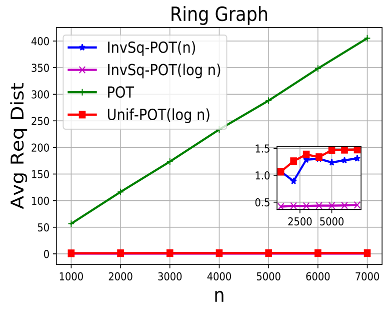
(c)



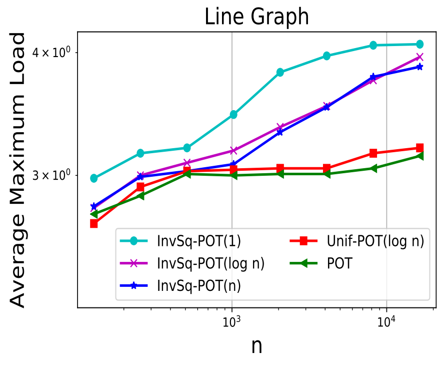
(d)



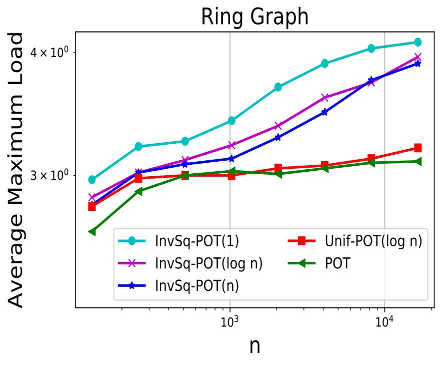
(e)



(f)



(g)



(h)

Figure 7.2: Simulation Results for Unif-POT(k) and InvSq-POT(k) for line and ring graphs.

distribution for Line and Ring topologies in Figures 7.2(a) and (d). We compare the load distributions for POT, Unif-POT($\log n$), InvSq-POT($\log n$) and InvSq-POT(n) policies. Surprisingly, the load distributions of Unif-POT($\log n$), InvSq-POT($\log n$) and InvSq-POT(n) almost exactly match to that of POT for both Line and Ring topologies.

Next, we compare the proximity based policies to POT with respect to total variation distance and present graphs for both Line and Ring topologies. We plot the total variation distance between load distributions of proximity based policies and POT as a function of number of servers for Line graph in Figure 7.2(b). We also plot the total variation distance between load distributions of two independent runs of POT which quantifies the noise or variation in load distribution of POT due to randomness. Again to our surprise, all proximity based policies with $k = \log n, n$ achieve total variation distances as low as 0.02 across a wide range of values of n. Also note that, InvSq-POT(1) achieves a load distribution farthest from POT while Unif-POT($\log n$) achieves the closest. Due to its uniform way of sampling, Unif-POT($\log n$) achieves the smallest total variation distance. However, due to a bias towards closest severs, both InvSq-POT($\log n$) and InvSq-POT(n) achieve higher variation distances. Due to load-implementation cost trade-off, a very local policy InvSq-POT(1), achieves even higher variation distance. Both InvSq-POT($\log n$) and InvSq-POT(n) appear to converge to a constant variation distance as n gets large. We obtain similar results for the case when servers are connected through a ring graph. The results are presented in Figure 7.2(e).

We plot average request distance as a function of number of servers in Figures 7.2(c) and (f). The average path length can be thought of as an upper bound on average request distance under POT. Larger values of average path length imply larger values of average request distance under POT. Surprisingly, proximity based policies significantly decrease average request distance ($\sim 99\%$ reduction) for large values of n. Since average path length for both line and ring graphs scale as $O(n)$, the average request distance under POT also drastically increases as n increases. Also note that, InvSq-POT($\log n$) achieves the lowest average request distance compared to other policies.

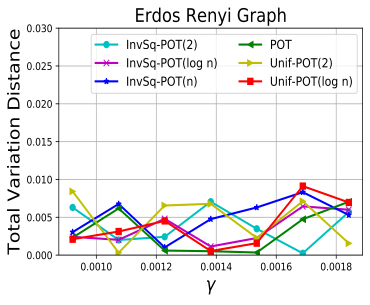
Finally Figures 7.2(g) and (h) show the growth in average maximum load as n increases. We observe that both Unif-POT($\log n$) and POT produce the smallest average maximum load. The average maximum load of InvSq-POT($\log n$) and InvSq-POT(n) are similar but larger compared to Unif-POT($\log n$). As expected InvSq-POT(1) exhibits the largest average maximum load since the policy distributes load only among immediate neighbors of the origin server.

7.3.3 Performance Comparison for Random Graphs

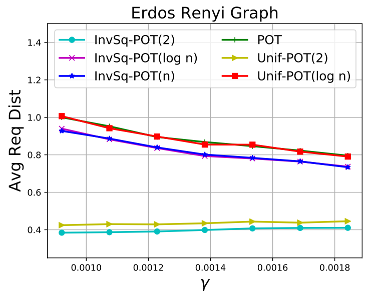
We now study the impact of proximity based policies on random graphs. In particular, we compare the performance of Unif-POT(k) and InvSq-POT(k) policies for $k = 2, \log n$ and n to that of POT. We consider the three random graphs: Erdos Reny (ER), Random Regular (RR) and Linear Preference (LP). To avoid ambiguity we define each of the above mentioned random graphs as follows.

Graph Type	n	m	Parameters
Erdos Reny (n, γ)	10000	10000	$\gamma : [\log n/n, \cdots, 2\log n/n]$
Random Regular (n, β)	10000	10000	$\beta : [5, 6, \cdots, 11]$
Barabasi Albert (n, α)	10000	10000	$\alpha : [1, 2, \cdots, 7]$

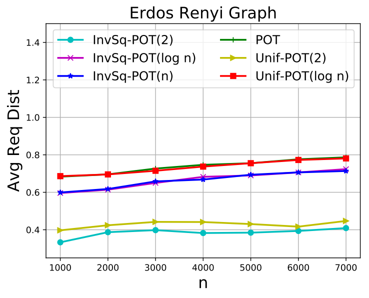
Table 7.1: Simulation parameters for random graph topologies.



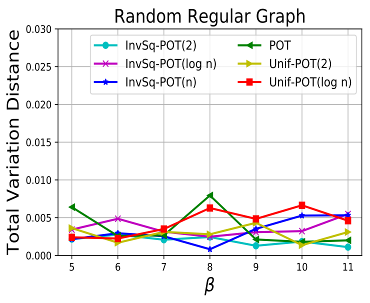
(a)



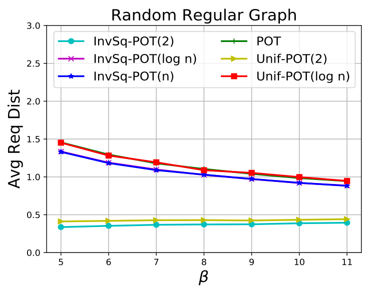
(b)



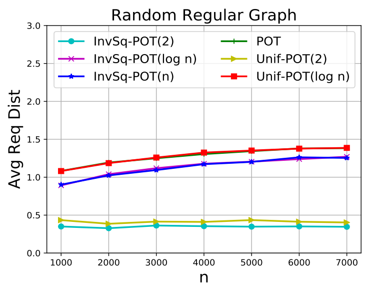
(c)



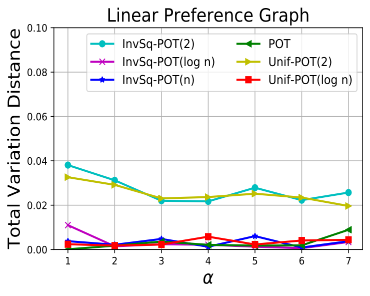
(d)



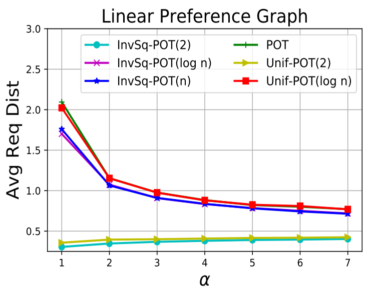
(e)



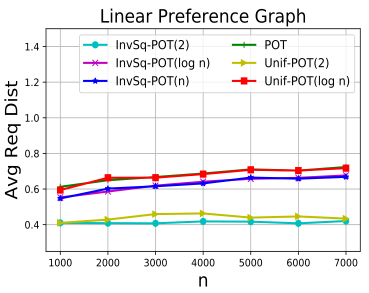
(f)



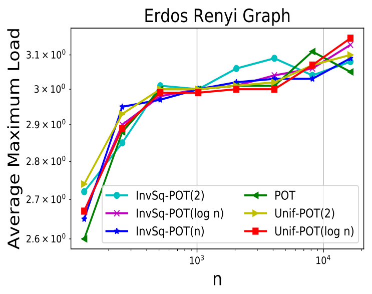
(g)



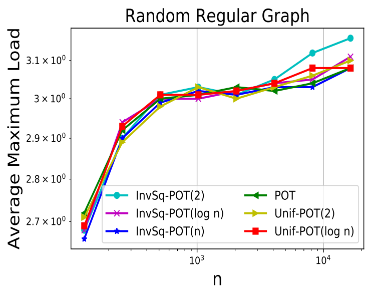
(h)



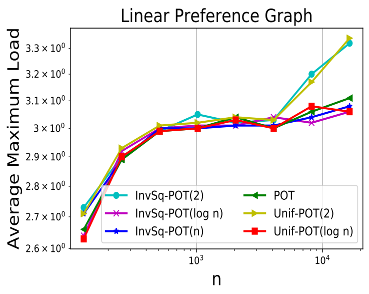
(i)



(j)



(k)



(l)

Figure 7.3: Simulation Results for Unif-POT(k) and InvSq-POT(k) with $n = 10000$ and $k = 2, \log n, n$ for random graphs.

Linear Preference Graph- *LP* (n, α)**:**

An LP (n, α) graph consists of n nodes grown by adding new nodes each with α edges attached to existing nodes with probability proportional to the node degree. This has been shown to yield a power-law degree distribution.

Random Regular Graph- *RR* (n, β)**:**

A β-regular graph RR (n, β) sampled from the probability space of all β-regular graphs on n vertices uniformly at random with $n\beta$ being even. For $\beta \geq 3$, a random β-regular graph of large size is asymptotically almost surely β-connected. In our simulations, $\beta \geq 3$.

Erdos-Renyi Graph- *ER* (n, γ)**:**

The ER (n, γ) graph is generated by choosing each of the $[n(n-1)]/2$ possible edges with probability γ. $\gamma = \log n/n$ is a sharp threshold for the connectedness of $ER(n, \gamma)$. Also as $n \to \infty$, the probability that $ER(n, \gamma)$ with $\gamma = 2\log n/n$ is connected, tends to 1. In all of our simulations, we assume $\gamma \geq \log n/n$.

We present the system and network parameters used in the simulation in Table 7.1. The results are presented in Figure 7.3. We first plot the total variation distance between load distributions of proximity based policies and POT as a function of ER edge probability parameter γ as shown in Figure 7.3(a). Note that, for all values of $k = 2, \log n, n$, both proximity based policies produce a variation distance as low as 0.5%. This is surprising since, when $k = O(1) = 2$ we only sample the two hop neighborhood of the origin server. But we are able to produce load distribution behavior almost identical to that of POT, which samples from the entire set of servers. We observe similar trends for RR graphs, as shown in Figure 7.3(d). However, we observe different results for the LP graph as shown in Figure 7.3(g). For LP graphs, we observe that when $k = 2$, both proximity based policies produce larger total variation distances than when $k = \log n, n$. The variation distance is still small when $k = 2$ and fluctuates around 0.033. Note that, an increase in k should decrease variation distance since the sampling set size increases with k. Observe that the variation distance of policies under a Line or Ring topology is larger than that of any random topology with fixed degree (Ex: RR topology). Higher graph densities in random topologies yield smaller variation distances compared to Line or Ring topologies.

We now study the effect of network parameters on the average request distances of the proximity based policies as shown in Figures 7.3(b), (e), (h). First observe that increases in the values of network parameters α, β and γ increase the graph densities of the corresponding graphs (LP, RR and ER) and hence connectedness. This results in a decrease in average request distance. Also observe the insensitivity of proximity based policies with $k = 2$ to the network size n. As expected, proximity policies with $k = 2$ produce very small request distances when compared to the case $k = \log n$.

Next, we study the scalability of average request distance with respect to network size as shown in Figures 7.3(c), (f) and (i). Note that the average path length of ER and LP exhibits small ($\log n$) and ultra small world ($\log n / \log \log n$) behavior respectively [38]. Due to small world behavior, the observed average request distances are small for LP and ER topologies across all policies when compared to similar size Line and Ring topologies. Again as expected, proximity policies with $k = 2$ are insensitive to changes in network size and produce the smallest average request distances. Also, observe that between Unif-POT(k) and InvSq-POT(k) for every k, InvSq-POT(k) policies produce smaller average request distances for similar size networks.

We finally study how average maximum load increases as a function of network size as shown in Figures 7.3(j), (k) and (l). For ER, for $k = 2, \log n, n$, both proximity based policies produce similar average maximum load values. For RR and LP we observe that all policies produce similar average maximum loads. However as network size increases, local policies InvSq-POT(2) and Unif-POT(2) produce larger maximum load values compared to InvSq-POT(n), Unif-POT($\log n$), and POT.

Graph Type	n	m	Parameters
Random Geometric (n, r)	10000	10000	$r : [\sqrt{\log n/\pi n}, \cdots, \sqrt{\sqrt{n}/\pi n}]$
Spatial Line (n, L_{max})	$[1000, \cdots, 7000]$	$[1000, \cdots, 7000]$	$L_{max} : [1000, \cdots, 7000]$
Spatial Ring (n, R)	$[1000, \cdots, 7000]$	$[1000, \cdots, 7000]$	$R : 1$

Table 7.2: Simulation parameters for spatial graph topologies.

7.3.4 Performance Comparison for Spatial Graphs

In this Section, we evaluate the performance of proximity aware POT policies for three spatial graphs: Random Geometric (RG), Spatial Line (SL) and Spatial Ring (SR) graphs defined as follows.

2-D Random Geometric Graph-*RG* (n, r)

A 2-D random geometric graph RG (n, r) is an undirected graph with n nodes uniformly sampled from a 2-dimensional Euclidean space $[0, 1)^2$. Two vertices: $a, b \in V$ share an edge iff the Euclidean distance between these two servers is less than r, excluding any loops. RG (n, r) possesses a sharp threshold for connectivity at $r \sim \sqrt{\log n/\pi n}$. In all our simulations we consider $r \geq \sqrt{\log n/\pi n}$.

Spatial Line Graph-*SL*(n, L_{max})

Locations of servers are uniformly sampled from a one-dimensional euclidean space $[0, L_{max})$.

Spatial Ring Graph-*SR*(n, R)

We assume servers are placed uniformly at random on a circle of radius R.

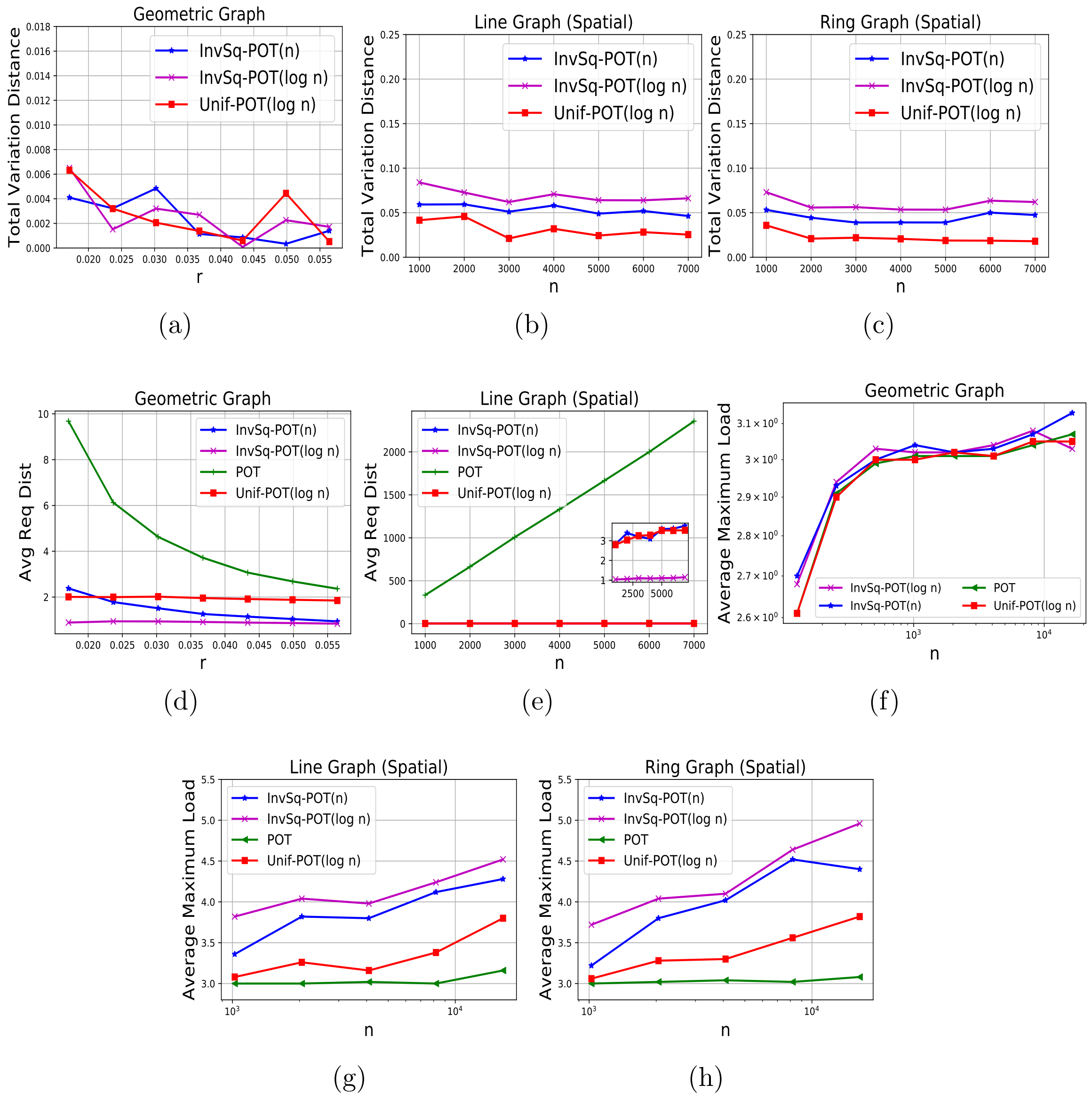


Figure 7.4: Simulation Results for Unif-POT(k) and InvSq-POT(k) with $n = 10000$ and $k = \log n, n$ for spatial graphs.

We present the network parameters used in our simulations in Table 7.2. Note that the radius parameters for RG are chosen such that the graph remains asymptotically almost surely connected. We first plot total variation distance as a function of radial parameter r of the RG topology as shown in Figure 7.4(a). Again to our surprise, for all values of $k = \log n, n$, both the proximity based policies produce a variation distance as low as 0.006. Note that for SL and SR topologies, we adopt a different job arrival model to incorporate the spatial nature of job request pattern. To be precise we assume both jobs and servers are placed uniformly at random on a one dimensional line $[0, L_{max})$ and on a circle of radius R for SL and SR topologies respectively. We plot variation distance as a function of network size for SL and SR as shown in Figures 7.4(b), (c). We observe a clear trend that Unif-POT($\log n$) and InvSq-POT($\log n$) policy produce the smallest and largest variation distances for both SL and SR with InvSq-POT($\log n$) producing a variation distance of around 0.08. These variation distances are insensitive to network size. Also, note that, with the introduction of the spatial dimension, the variation distances increased by five fold compared to their non-spatial counterparts (Figures 7.2(b) and (e)) for the same network size.

We next plot average request distance as a function of r for RG topology as shown in Figure 7.4(d). First note that the proximity aware policies are almost insensitive to r. As r increases, the graph density for an RG increases thereby reducing average path length of the network. Thus we observe a decrease in average request distance for POT with an increase in r. As observed before, InvSq-POT(k) produces lower average request distances as compared to their Unif-POT(k) counterpart. InvSq-POT(k) produces the smallest average request distance for SL, which is almost insensitive to system size as shown in Figure 7.4(e). However, as expected, POT produces a large request distance that increases linearly with system size.

We study the average maximum load behavior of the policies across various spatial networks in Figures 7.4(f), (g), (h). For low values of n for RG topology, Unif-

POT($\log n$) and POT exhibit similar average maximum loads. However, the corresponding average maximum load values for InvSq-POT($\log n$) and InvSq-POT(n) are larger. When n is large, all policies exhibit similar behavior. For SL and SR topologies, InvSq-POT($\log n$) is the worst while POT is the best policy. Unif-POT($\log n$) is closer to POT while InvSq-POT(n) is closer to InvSq-POT($\log n$) with respect to average maximum load.

7.4 Summary

In this chapter we designed a class of proximity aware power of two choices based allocation policies for the case when servers are interconnected as an arbitrary graph. We performed extensive simulations over a wide range of network topologies. To our surprise, with few simple modifications in the server sampling process, we observed a drastic reduction in the overall system wide implementation cost while obtaining a similar load distribution profile as that of POT policy.

CHAPTER 8
SUMMARY AND FUTURE WORK

The central topic of this thesis is the allocation of resources in distributed service networks— specifically, how to optimally allocate caching resources to provide differential services and how to distribute user requests to resources in a spatial setting.

We first proposed a new method to compute an upper bound on cache hit probability for all non-anticipative caching policies. We then developed a utility based caching framework that implemented differential services.

We designed new resource allocation policies for both one-dimensional and two-dimensional spatial networks. Finally, we developed proximity aware load balancing policies when resources are placed on the vertices of an arbitrary graph.

8.1 Future Work

Below we point out some potential future work and general directions that we wish to explore further.

- In Chapter 2 we developed a hazard rate based upper bound on cache hit probability for non-anticipative caching policies under fairly weak statistical assumptions. Going further, we aim to build a real-time trace driven version of our proposed bound by estimating the hazard rates and fitting the inter-request time distributions in an online manner.

- The utility based framework developed in Chapters 3 and 4 assumed Poisson request arrival process for contents. We would like to consider a more general request arrival process in future. Our results from Section 3.5.1 advocates for the accuracy of Poisson online approximation policy for a single cache system. It would be interesting to compare the performance of Poisson online approximation with other cache eviction policies for general request arrival process in an arbitrary generic cache network.

- A natural extension of the one dimensional unidirectional spatial framework developed in Chapter 5 is to consider more general communication costs associated with each resource allocation. Assuming communication cost for each allocation is a function of request distance, one can derive expressions for the expected communication cost from the request distance distribution. Thus one of our future goals would be to derive the request distance distribution associated with the MTR policy for various inter-resource and inter-user distance distributions.

- It would be interesting to analyze the impact of user dynamics on the performance of spatial resource allocation policies developed in Chapters 6 and 7. For example, user requests may arrive on the spatial network according to a temporal Poisson process. The user can then be assigned a spatial location uniformly at random on the network. The servers can be distributed over the network according to some spatial process. Each server will have a queue and the service time to process each request can be exponentially distributed. Requests can be served in FIFO order. Our goal would then be to analyze the expected maximum queue length across all servers or the expected request distance associated with a specific allocation policy.

APPENDIX A

ADDITIONAL PROOFS FOR CHAPTER 2

A.1 Proof of Equation (2.6)

We drop the argument t in $k_i(t)$ (see definition at the beginning of Section 2.2.1 as no confusion may occur.). We have

$$p_i(t) = \mathbb{P}\left(T_{i,k_i} < T_{j,k_j}, \forall j \neq i \,|\, \mathcal{H}_t, \min_{j=1,\ldots,n} T_{j,k_j} = t\right).$$

For $h > 0$

$$\begin{aligned}
&\mathbb{P}\left(T_{i,k_i} \in (t, t+h), T_{j,k_j} > t+h, \forall j \neq i \,\middle|\, \mathcal{H}_t, \min_{j=1,\ldots,n} T_{j,k_j} \in (t, t+h)\right) \\
&= \frac{\mathbb{P}\left(T_{i,k_i} \in (t, t+h), T_{j,k_j} > t+h, \forall j \neq i \,|\, \mathcal{H}_t\right)}{\mathbb{P}\left(\min_{j=1,\ldots,n} T_{j,k_j} \in (t, t+h) \,|\, \mathcal{H}_t\right)} \\
&= \frac{\mathbb{P}(T_{i,k_i} \in (t, h) \,|\, \mathcal{H}_{i,t}) \prod_{\substack{1 \leq j \leq n \\ j \neq i}} \mathbb{P}(T_{j,k_j} > t+h \,|\, \mathcal{H}_{j,t})}{\mathbb{P}\left(\min_{j=1,\ldots,n} T_{j,k_j} \in (t, t+h) \,|\, \mathcal{H}_t\right)},
\end{aligned} \tag{A.1}$$

from the conditional independence assumption in (2.1). Let us focus on the denominator in (A.1). It can be written as

$$\mathbb{P}\left(\min_{j=1,\ldots,n} T_{j,k_j} \in (t, t+h) \,|\, \mathcal{H}_t\right) = \sum_{j=1}^{n} \mathbb{P}(T_{j,k_j} \in (t, t+h) > t+h, \forall l \neq j \,|\, H_t) + f(h),$$

with $f(h) \to 0$ as $h \to 0$, since as $h \to 0$ there can be at least one random variables (rvs) located in $(t, t+h)$ among the rvs $T_{1,k_1}, \ldots, T_{n,k_n}$ since these rvs are continous. Therefore,

$$\begin{aligned}
&\mathbb{P}\Bigg(T_{i,k_i} \in (t,t+h), T_{j,k_j} > t+h, \forall j \neq i \,\Bigg|\, \mathcal{H}_t, \min_{j=1,\ldots,n} T_{j,k_j} \in (t,t+h)\Bigg) \\
&= \frac{\mathbb{P}(T_{i,k_i} \in (t,h) \,|\, \mathcal{H}_{i,t}) \prod_{\substack{1\le j\le n \\ j\neq i}} \mathbb{P}(T_{j,k_j} > t+h \,|\, \mathcal{H}_{j,t})}{\sum_{j=1}^n \mathbb{P}(T_{j,k_j} \in (t,d+t), T_{l,k_l} > t+h, \forall l \neq j \,|\, H_t) + f(h)} \\
&= \mathbb{P}(T_{i,k_i} \in (t,h) \,|\, \mathcal{H}_{i,t}) \prod_{\substack{1\le j\le n \\ j\neq i}} \mathbb{P}(T_{j,k_j} > t+h \,|\, \mathcal{H}_{j,t}) \\
&\times \left(\sum_{j=1}^n \mathbb{P}(T_{j,k_j} \in (t,t+h) \,|\, \mathcal{H}_{l,t}) \times \prod_{\substack{1\le l\le n \\ l\neq j}} \mathbb{P}(T_{l,k_l} > t+h \,|\, \mathcal{H}_{l,t}) + f(h) \right)^{-1} \qquad \text{(A.2)} \\
&= \frac{\frac{\mathbb{P}(T_{i,k_i} \in (t,t+h) \,|\, \mathcal{H}_{i,t})}{\mathbb{P}(T_{i,k_i} > t+h \,|\, \mathcal{H}_{i,t})}}{\sum_{j=1}^n \frac{\mathbb{P}(T_{j,k_j} \in (t,t+h) \,|\, \mathcal{H}_{l,t})}{\mathbb{P}(T_{j,k_j} > t+h \,|\, \mathcal{H}_{l,t})} + f(h)} = \frac{\lambda_i^*(t+h)}{\sum_{j=1}^n \lambda_j^*(t+h) + f(h)}, \qquad \text{(A.3)}
\end{aligned}$$

where (A.2) and (A.3) follow from (2.1) and (2.2). Letting $h \to 0$ in (A.3) gives (2.6).

A.1.1 Proof of Lemma 1

The proof mimicks that of Lemma 1. Fix $k \geq 1$ and denote by $Z(T_k-)$ the state of the process $\mathbf{Z}$ just before time T_k, namely, just before the kth request for an object is made. For all $\pi \in \Pi$ and $x \in \mathcal{E}$, we have

$$\begin{aligned}
&\mathbb{E}\left[H_k^{HR-E} \,|\, Z(T_k-) = x\right] = \mathbb{P}\left(R_k \in B_k^{HR-E} \,|\, Z(T_k-) = x\right) \\
&= \sum_{i=1}^n \mathbb{P}\left(R_k \in B_k^{HR-E} \,|\, R_k = i, Z(T_k-) = x\right) \mathbb{P}(R_k = i \,|\, Z(T_k-) = x) \\
&= \sum_{i \in B_k^{HR-E}} \mathbb{P}(R_k = i \,|\, Z(T_k-) = x) \\
&= \sum_{i \in B_k^{HR-E}} \frac{\lambda_i(x)}{\sum_{j=1}^n \lambda_j(x)} \geq \sum_{i \in B_k^\pi} \frac{\lambda_i(x)}{\sum_{j=1}^n \lambda_j(x)} = \mathbb{E}\left[H_k^\pi \,|\, Z(T_k-) = x\right].
\end{aligned}$$

Removing the conditioning gives $\mathbb{E}\left[H_k^{HR-E}\right] \geq \mathbb{E}\left[H_k^\pi\right]$.

APPENDIX B
ADDITIONAL PROOFS FOR CHAPTER 3

B.1 Stationary Behaviors of MCDP

[43] considered a caching policy LRU($\boldsymbol{m}$). Though the policy differ from MCDP, the stationary analysis is similar. We present our result here for completeness.

Under IRM model, the request for content i arrives according a Poisson process with rate λ_i. As discussed earlier, for TTL caches, content i spends a deterministic time in a cache if it is not requested, which is independent of all other contents. We denote the timer as T_{il} for content i in cache l on the path p, where $l \in \{1, \cdots, |p|\}$.

Denote t_k^i as the k-th time that content i is either requested or moved from one cache to another. For simplicity, we assume that content is in cache 0 (i.e., server) if it is not in the cache network. Then we can define a discrete time Markov chain (DTMC) $\{X_k^i\}_{k\geq 0}$ with $|p|+1$ states, where X_k^i is the cache index that content i is in at time t_k^i. Since the event that the time between two requests for content i exceeds T_{il} happens with probability $e^{-\lambda_i T_{il}}$, then the transition matrix of $\{X_k^i\}_{k\geq 0}$ is given as

$$\mathbf{P}_i^{\text{MCDP}} = \begin{bmatrix} 0 & 1 & & & \\ e^{-\lambda_i T_{i1}} & 0 & 1-e^{-\lambda_i T_{i1}} & & \\ & \ddots & \ddots & \ddots & \\ & & e^{-\lambda_i T_{i(|p|-1)}} & 0 & 1-e^{-\lambda_i T_{i(|p|-1)}} \\ & & & e^{-\lambda_i T_{i|p|}} & 1-e^{-\lambda_i T_{i|p|}} \end{bmatrix}.$$

Let $(\pi_{i0}, \cdots, \pi_{i|p|})$ be the stationary distribution for $\mathbf{P}_i^{\text{MCDP}}$, we have

$$\pi_{i0} = \frac{1}{1+\sum_{j=1}^{|p|} e^{\lambda_i T_{ij}} \prod_{s=1}^{j-1}(e^{\lambda_i T_{is}}-1)},$$

$$\pi_{i1} = \pi_{i0} e^{\lambda_i T_{i1}},$$

$$\pi_{il} = \pi_{i0} e^{\lambda_i T_{il}} \prod_{s=1}^{l-1} (e^{\lambda_i T_{is}} - 1),\ l = 2, \cdots, |p|.$$

Then the average time that content i spends in cache $l \in \{1, \cdots, |p|\}$ can be computed as

$$\mathbb{E}[t^i_{k+1} - t^i_k | X^i_k = l] = \int_0^{T_{il}} \left(1 - \left[1 - e^{-\lambda_i t}\right]\right) dt = \frac{1 - e^{-\lambda_i T_{il}}}{\lambda_i}, \tag{B.2}$$

and $\mathbb{E}[t^i_{k+1} - t^i_k | X^i_k = 0] = \frac{1}{\lambda_i}$.

Given (B.1) and (B.2), the timer-average probability that content i is in cache $l \in \{1, \cdots, |p|\}$ is

$$h_{i1} = \frac{e^{\lambda_i T_{i1}} - 1}{1 + \sum_{j=1}^{|p|} (e^{\lambda_i T_{i1}} - 1) \cdots (e^{\lambda_i T_{ij}} - 1)},$$
$$h_{il} = h_{i(l-1)} (e^{\lambda_i T_{il}} - 1),\ l = 2, \cdots, |p|,$$

where h_{il} is also the hit probability for content i at cache l.

B.2 Proof of Theorem 4: Convergence of Primal Algorithm

Since $U_i(\cdot)$ is strictly concave, $C_l(\cdot)$ and $\tilde{C}_i(\cdot)$ are convex, then (3.9) is strictly concave, hence there exists a unique maximizer. Denote it as $\boldsymbol{h}^*$.

Any differentiable function $f(x)$ can be linearized around a point x^* as $L(x) = f(x^*) + f'(x^*)(x - x^*)$. Denote $\forall i, l$

$$f(h_{il}) = h_{il} + \zeta_{il} \left[\lambda_i \psi^{|p|-l} U_i'(\lambda_i h_{il}) - C_l' \left(\sum_{j \in \mathcal{D}} h_{jl} - B_l \right) - \tilde{C}_i' \left(\sum_{m=1}^{|p|} h_{im} - 1 \right) \right], \tag{B.3}$$

with $f : \mathbb{R}^+ \to \mathbb{R}$. We have $f(h^*_{il}) = h^*_{il}$. Under linearization,

$$h_{il}[k+1] = h^*_{il} + f'(h^*_{il})(h_{il}[k] - h^*_{il}). \tag{B.4}$$

Denote $h_{il}^{\delta}[k] = h_{il}[k] - h_{il}^{*}$ as deviation from h_{il}^{*} at k^{th} iteration. Hence we have

$$h_{il}^{\delta}[k+1] = f'(h_{il}^{*})h_{il}^{\delta}[k] = [f'(h_{il}^{*})]^{k}\, h_{il}^{\delta}[0].$$

Thus (B.4) is locally asymptotically stable if

$$|f'(h_{il}^{*})| < 1. \tag{B.5}$$

Computing $f'(h_{il}^{*})$ from (B.3) and substituting in (B.5) yields

$$\zeta_{il} < \frac{2}{C_l''\left(\sum\limits_{j\in\mathcal{D}} h_{jl}^{*} - B_l\right) + \tilde{C}_i''\left(\sum\limits_{m=1}^{|p|} h_{im}^{*} - 1\right) - \lambda_i^2 \psi^{|p|-l} U_i''(\lambda_i h_{il}^{*})}. \tag{B.6}$$

Note that, since the functions $C_l, \tilde{C}_i$ and U_i are strictly convex, strictly convex and strictly concave functions respectively, $C_l''(x) < 0, \tilde{C}_i''(x) < 0$ and $U_i''(x) < 0\ \forall x \in \mathbb{R}^{+}$. Hence the r.h.s of (B.6) is strictly positive and for a sufficiently small positive step-size parameter ζ_{il}, (B.6) always holds. Thus the update rule (3.9) converges to $\boldsymbol{h}^{*}$ as long as $h_{il}^{(0)}$ is sufficiently close to h_{il}^{*} for all $i \in \mathcal{D}$ and $l = 1, 2, \cdots, |p|$.

B.3 Proof of Theorem 7: Convergence of Primal-dual Algorithm

From Lemma 2, we know at least one of ν_l and μ_{ip} is non-zero, for all $i \in \mathcal{D}$, $l \in \{1, \cdots, |p|\}$ and $p \in \mathcal{P}^i$. Hence there are three cases, (i) $\nu_l \neq 0$ and $\mu_{ip} = 0$; (ii) $\nu_l = 0$ and $\mu_{ip} \neq 0$; and (iii) $\nu_l \neq 0$ and $\mu_{ip} \neq 0$.

For case (i), we have

$$h_{il}^{(p)} = \frac{w_{ip}\psi^{|p|-l}}{\nu_l\left(\prod\limits_{\substack{q:q\neq p,\\ j\in\{1,\cdots,|q|\}}} (1 - h_{ij}^{(q)})\right)} - \frac{1}{\lambda_{ip}},$$

which is clearly continuous in ν_l, for all $i \in \mathcal{D}$, $l \in \{1, \cdots, |p|\}$ and $p \in \mathcal{P}^i$.

Similarly for case (ii), we have

$$h_{il}^{(p)} = \frac{w_{ip}\psi^{|p|-l}}{\mu_{ip}} - \frac{1}{\lambda_{ip}},$$

which is also clearly continuous in μ_{ip}, for all $i \in \mathcal{D}$, $l \in \{1, \cdots, |p|\}$ and $p \in \mathcal{P}^i$.

For case (iii), from (3.18), it is obvious that $h_{il}^{(p)}$ is continuous in ν_l and μ_{ip} for all $i \in \mathcal{D}$, $l \in \{1, \cdots, |p|\}$ and $p \in \mathcal{P}^i$.

Therefore, we know that $h_{il}^{(p)}$ is is continuous in ν_l and μ_{ip} for all $i \in \mathcal{D}$, $l \in \{1, \cdots, |p|\}$ and $p \in \mathcal{P}^i$.

APPENDIX C

ADDITIONAL PROOFS FOR CHAPTER 4

C.1 Proof of Lemma 3

We consider the following two cases, i.e., when $\beta = 1$ and $\beta \neq 1$.

Case 1($\beta = 1$): The utility function is $U_i(h) = w_i \log(h)$. Thus we have

$$U_i\left(e^{\sum_{k=1}^{n} x_k}\right) = w_i \log\left(e^{\sum_{k=1}^{n} x_k}\right) = w_i \sum_{k=1}^{n} x_k,$$

which is an affine function and thus concave as well.

Case 2($\beta \neq 1$): The utility function is $U_i(h) = w_i h^{1-\beta}/(1-\beta)$. Thus we have

$$U_i\left(e^{\sum_{k=1}^{n} x_k}\right) = w_i \frac{e^{(1-\beta)\sum_{k=1}^{n} x_k}}{1-\beta},$$

and the corresponding Hessian matrix is

$$H_i = (1-\beta) w_i e^{(1-\beta)\sum_{k=1}^{n} x_k} \begin{bmatrix} 1 & 1 \cdots & 1 \\ 1 & 1 \cdots & 1 \\ \vdots & \vdots & \vdots \\ 1 & 1 \cdots & 1 \end{bmatrix}.$$

Note that, the unit matrix with all ones has eigenvalues n with multiplicity 1, and 0 with multiplicity $n-1$. The terms $e^{(1-\beta)\sum_{k=1}^{n} x_k}$ and w_i are always positive. Hence H_i is negative semi-definite, i.e., has non-positive eigenvalues, only when $1-\beta < 0$. Combining both cases, $U_i\left(e^{\sum_{k=1}^{n} x_k}\right)$ is a concave function for $\beta \geq 1$.

APPENDIX D

ADDITIONAL PROOFS FOR CHAPTER 5

D.1 Derivation of F_Z for various inter-server distance distributions

D.1.1 $F_X(x) \sim$ Exponential(μ)

In this case, both X and Y are exponentially distributed. Thus the difference distribution is given by

$$D_{XY}(x) = 1 - \frac{\lambda}{\lambda+\mu} e^{-\mu x}, \text{when } x \geq 0 \tag{D.1}$$

Combining (5.4) and (D.1), we get

$$F_Z(x) = \frac{1 - \frac{\lambda}{\lambda+\mu} e^{-\mu x} - 1 + \frac{\lambda}{\lambda+\mu}}{\frac{\lambda}{\lambda+\mu}} = 1 - e^{-\mu x}.$$

Thus we obtain $F_X(x) = F_Z(x) \sim$ Exponential(μ).

D.1.2 $F_X(x) \sim$ Uniform($0, b$)

The c.d.f. for uniform distribution is

$$F_X(x) = \begin{cases} \frac{x}{b}, & 0 \leq x \leq b; \\ 1, & x > b, \end{cases} \tag{D.2}$$

where b is the uniform parameter. Thus we have

$$D_{XY}(x) = \int_0^\infty F_X(x+y)\lambda e^{-\lambda y}dy = \left[\int_0^{b-x} \frac{x+y}{b}\lambda e^{-\lambda y}dy\right] + \left[\int_{b-x}^\infty 1\ \lambda e^{-\lambda y}dy\right]$$
$$= \frac{\lambda x - e^{-\lambda(b-x)} + e^{-\lambda b}}{b\lambda + e^{-\lambda b} - 1}$$

Taking $k_\lambda = 1/(b\lambda + e^{-\lambda b} - 1)$ and using Equation (5.4) we have

$$F_Z(x) = k_\lambda\left[\lambda x + e^{-\lambda b}(1-e^{\lambda x})\right] \quad \text{and} \quad f_Z(x) = \lambda k_\lambda\left[1 - e^{-\lambda b}e^{\lambda x})\right].$$

Taking $\alpha_Z = \int_0^b x f_Z(x)dx$ and $\sigma_Z^2 = [\int_0^b x^2 f_Z(x)dx] - \alpha_Z^2$ we have

$$\alpha_Z = \frac{b^2\lambda}{2}k_\lambda - \frac{1}{\lambda}, \quad \sigma_Z^2 = \frac{b^3\lambda}{3}k_\lambda - \frac{k_\lambda}{\lambda}\left[b(b\lambda - 2) + \frac{2}{\lambda}(1-e^{-\lambda b})\right] - \alpha_Z^2,$$
$$\alpha_X = b/2, \quad \sigma_X^2 = b^2/12.$$

D.1.3 $F_X(x) \sim$ Deterministic(d_0)

Another interesting scenario is when servers are equally spaced at a distance d_0 from each other i.e. when $F_X(x) \sim$ Deterministic(d_0). The c.d.f. for deterministic distribution is

$$F_X(x) = \begin{cases} 0, & 0 \le x < d_0; \\ 1, & x \ge d_0, \end{cases} \tag{D.3}$$

where d_0 is the deterministic parameter. A similar analysis as that of uniform distribution yields

$$F_Z(x) = c_\lambda\left[e^{-\lambda(d_0-x)} - e^{\lambda d_0}\right];\ f_Z(x) = \lambda c_\lambda\left[e^{-\lambda(d_0-x)}\right],$$

where $c_\lambda = 1/(1-e^{-\lambda d_0})$. Thus we have

$$\alpha_Z = c_\lambda\frac{d_0\lambda + e^{-\lambda d_0} - 1}{\lambda}, \quad \sigma_Z^2 = \frac{c_\lambda}{\lambda}\left[d_0(d_0\lambda - 2) + \frac{2}{\lambda}(1-e^{-\lambda d_0})\right] - \alpha_Z^2,$$
$$\alpha_X = d_0, \quad \sigma_X^2 = 0.$$

D.2 ESABQ under PRGS

D.2.1 Chapman-Kolmorogov equations

Let us write the Chapman-Kolmorogov equations for the Markov chain $\{(L(t), R(t), I(t)),\ t \geq 0\}$ defined in Section 5.5.3.1.

For $n \geq 2$ and $x > 0$ we get

$$\begin{aligned}
\frac{\partial}{\partial t}p_t(n,x;1) &= \frac{\partial}{\partial x}p_t(n,x;1) - \lambda p_t(n,x;1) - \frac{\partial}{\partial x}p_t(n,0;1) + \lambda p_t(n-1,x;1)\\
\frac{\partial}{\partial t}p_t(n,x;2) &= \frac{\partial}{\partial x}p_t(n,x;2) - \lambda p_t(n,x;2) - \frac{\partial}{\partial x}p_t(n,0;2) + \lambda p_t(n-1,x;2)\\
&\quad + F_X(x)\frac{\partial}{\partial x}p_t(n+c,0;1) + F_X(x)\frac{\partial}{\partial x}p_t(n+c,0;2).
\end{aligned}$$

Letting $t \to \infty$ yields

$$\begin{aligned}
0 &= \frac{\partial}{\partial x}p(n,x;1) - \lambda p(n,x;1) - \frac{\partial}{\partial x}p(n,0;1) + \lambda p(n-1,x;1) \qquad \text{(D.4)}\\
0 &= \frac{\partial}{\partial x}p(n,x;2) - \lambda p(n,x;2) - \frac{\partial}{\partial x}p(n,0;2) + \lambda p(n-1,x;2)\\
&\quad + F_X(x)\frac{\partial}{\partial x}p(n+c,0;1) + F_X(x)\frac{\partial}{\partial x}p(n+c,0;2).
\end{aligned}$$

For $n = 1$, $x > 0$

$$\begin{aligned}
\frac{\partial}{\partial t}p_t(1,x;1) &= \frac{\partial}{\partial x}p_t(1,x;1) - \lambda p_t(1,x;1) - \frac{\partial}{\partial x}p_t(1,0;1) + \lambda p_t(0)F_Z(x)\\
\frac{\partial}{\partial t}p_t(1,x;2) &= \frac{\partial}{\partial x}p_t(1,x;2) - \lambda p_t(1,x;2) - \frac{\partial}{\partial x}p_t(1,0;2)\\
&\quad + F_X(x)\frac{\partial}{\partial x}p(1+c,0;1) + F_X(x)\frac{\partial}{\partial x}p_t(1+c,0;2).
\end{aligned}$$

Letting $t \to \infty$ yields

$$\begin{aligned}
0 &= \frac{\partial}{\partial x}p(1,x;1) - \lambda p(1,x;1) - \frac{\partial}{\partial x}p(1,0;1) + \lambda p(0)F_Z(x)\\
0 &= \frac{\partial}{\partial x}p(1,x;2) - \lambda p(1,x;2) - \frac{\partial}{\partial x}p(1,0;2)\\
&\quad + F_X(x)\left(\frac{\partial}{\partial x}p(1+c,0;1) + \frac{\partial}{\partial x}p(1+c,0;2)\right), x > 0. \qquad \text{(D.5)}
\end{aligned}$$

We can collect the results in (D.4)-(D.5) as follows: for $n \geq 1$, $x > 0$,

$$\begin{aligned}
0 =& \frac{\partial}{\partial x}p(n,x;1) - \lambda p(n,x;1) - \frac{\partial}{\partial x}p(n,0;1) + \lambda p(n-1,x;1)\mathbf{1}(n \geq 2) \\
&+ \lambda p(0)F_Z(x)\mathbf{1}(n=1) \qquad \text{(D.6)} \\
0 =& \frac{\partial}{\partial x}p(n,x;2) - \lambda p(n,x;2) - \frac{\partial}{\partial x}p(n,0;2) + \lambda p(n-1,x;2)\mathbf{1}(n \geq 2) \\
&+ F_X(x)\left(\frac{\partial}{\partial x}p(n+c,0;1) + \frac{\partial}{\partial x}p(n+c,0;2)\right). \qquad \text{(D.7)}
\end{aligned}$$

Define $g(n,x) = p(n,x;1) + p(n,x;2)$ for $n \geq 1$, $x > 0$. Summing (D.6) and (D.7) gives

$$\begin{aligned}
0 =& \frac{\partial}{\partial x}g(n,x) - \lambda g(n,x) - \frac{\partial}{\partial x}g(n,0) + \lambda g(n-1,x)\mathbf{1}(n \geq 2) + \lambda p(0)F_Z(x)\mathbf{1}(n=1) \\
&+ F_X(x)\frac{\partial}{\partial x}g(n+c,0), \forall n \geq 1, x > 0.
\end{aligned}$$

D.2.2 Multiplicity of roots of $z^c - F_X^*(\lambda(1-z))$

Assume that $F_X(x) = 1 - e^{-\mu x}$ (regular batch service times are exponentially distributed). Then,

$$z^c - F_X^*(\lambda(1-z)) = \frac{-\rho z^{c+1} + (1+\rho)z^c - 1}{1 + \rho(1-z)}.$$

$z^c - F_X^*(\lambda(1-z)) = 0$ for $|z| \leq 1$ iff $Q(z) := -\rho z^{c+1} + (1+\rho)z^c - 1 = 0$. The derivative of $Q(z)$ is $Q'(z) = z^{c-1}((1+\rho)c - \rho(c+1)z)$. It vanishes at $z = 0$ and at $z = \frac{(1+\rho)c}{\rho(c+1)} > 1$ under the stability condition $\rho < c$. Since $z = 0$ is not a zero of $Q(z)$, we conclude that all zeros of $z^c - F_X^*(\lambda(1-z))$ in $\{|z| \leq 1\}$ have multiplicity one.

More generally, it is shown in [14] that all zeros of $z^c - F_X^*(\lambda(1-z))$ in $\{|z| \leq 1\}$ have multiplicity one if F_X is a χ^2-distribution with an even number $2p$ of degrees of freedom, i.e. $dF_X(x) = \frac{a^p}{\Gamma(p)}x^{p-1}e^{-ax}dx$ so that $1/\mu = p/a$.

D.2.3 Roots of $A(z)$

Define $A(z) = F_X^*(\theta(z))$. If $A(z)$ has a radius of convergence larger than one (i.e. $A(z)$ is analytic for $|z| \leq \nu$ with $\nu > 1$) and $A'(1) < c \in \{1, 2, \ldots\}$ a direct application of Rouché's theorem shows that $z^c - A(z)$ has c zeros in the unit disk $\{|z| \leq 1\}$(see e.g. [3]). If the radius of convergence of $A(z)$ is one, $A(z)$ is differentiable at $z = 1$, $A'(1) < c$, and $z^c - A(z)$ has period p, then $z^c - A(z)$ has exactly $p \leq s$ zeros on the unit circle and $s - p$ zeros inside the unit disk $\{|z| < 1\}$ [3, Theorem 3.2]. Assume that the stability condition $\frac{d}{dz}A(z)|_{z=1} = \rho < c$ holds. $A(z)$ has a radius of convergence larger than one when F_X is the exponential/Erlang/Gamma/ etc probability distributions.

D.2.4 Special Cases

One easily checks that (5.17) gives the classical Pollaczek-Khinchin formula for the M/G/1 queue when $c = 1$ and $F_Z = F_X$.

Let now $c = 1$ in (5.17) with F_Z and F_X arbitrary. Then,

$$N(z) = \frac{a_1}{\lambda}\left(\frac{F_X^*(\lambda(1-z)) - zF_Z^*(\lambda(1-z))}{F_X^*(\lambda(1-z)) - z}\right)$$

gives the z-transform of the stationary number of customers in a M/G/1 queue with an exceptional first customer in a busy period. The constant a_1/λ is obtained from the identity $N(1) = 1$ by application of L'Hopital's rule, which gives[1] $a_1/\lambda = (1 - \rho)/(1 - \rho + \rho_Z)$. This gives

$$N(z) = \frac{1-\rho}{1-\rho+\rho_Z}\left(\frac{F_X^*(\lambda(1-z)) - zF_Z^*(\lambda(1-z))}{F_X^*(\lambda(1-z)) - z}\right).$$

The above is a known result [100].

[1]Note that we retrieve this result by letting $c = 1$ in (5.18).

If $F_Z^* = F_X^* := F^*$, then

$$N(z) = \frac{\sum_{k=1}^{c} a_k \left[(z^c - z^k)z^c + ((1-z^c)z - (1-z^k))F^*(\theta(z))\right]}{\theta(z)(z^c - F^*(\theta(z))}.$$

D.3 Results for Section 5.6.2

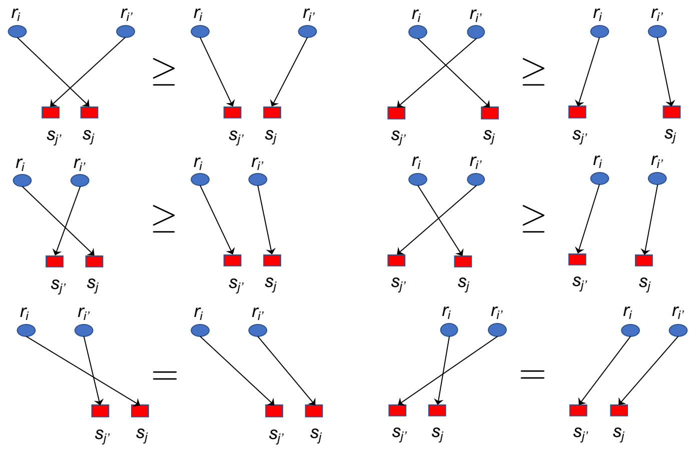


Figure D.1: Uncrossing an assignment either reduces request distance or keeps it unchanged.

D.3.1 Derivation of $v_1(z)$ and $v_2(z)$

$v_1(z)$ in (5.21) can further be simplified to

$$\begin{aligned} v_1(z) &= \sum_{l=0}^{\infty} z^l \sum_{m=0}^{l} \pi_m \sum_{i=0}^{c} k_{i+l-m} p_i = \sum_{m=0}^{\infty} \pi_m \sum_{l \geq m} z^l \sum_{i=0}^{c} k_{i+l-m} p_i \\ &= \sum_{m=0}^{\infty} \pi_m z^m \sum_{l \geq m} z^{l-m} \sum_{i=0}^{c} k_{i+l-m} p_i = \sum_{m=0}^{\infty} \pi_m z^m \sum_{j=0}^{\infty} z^j \sum_{i=0}^{c} k_{i+j} p_i \\ &= N(z) \sum_{i=0}^{c} p_i z^{-i} \sum_{j=0}^{\infty} z^{i+j} k_{i+j} = N(z) \sum_{i=0}^{c} p_i z^{-i} \left[K(z) - \sum_{j=0}^{i} k_j z^j + k_i z^i\right] \\ &= N(z) \left\{ \sum_{i=0}^{c} p_i z^{-i} \left[K(z) - \sum_{j=0}^{i} k_j z^j\right] + \sum_{i=0}^{c} k_i z^i \right\}. \end{aligned}$$

Similarly $v_2(z)$ in (5.22) can further be simplified to

$$v_2(z) = \sum_{l=0}^{\infty} z^l \sum_{m=l+1}^{c+l} \pi_m \sum_{i=m-l}^{c} k_{i+l-m} p_i = \Bigg[\sum_{l=0}^{\infty} z^l \sum_{m=l}^{c+l} \pi_m \sum_{i=m-l}^{c} k_{i+l-m} p_i\Bigg] - N(z) \sum_{i=0}^{c} k_i z^i$$
$$= \Bigg[\sum_{m=0}^{c} z^{-m} \sum_{i=m}^{c} k_{i-m} p_i \sum_{l=0}^{\infty} z^{m+l} \pi_{m+l}\Bigg] - N(z) \sum_{i=0}^{c} k_i z^i$$
$$= \Bigg[\sum_{m=0}^{c} z^{-m} \sum_{i=m}^{c} k_{i-m} p_i \Big\{ N(z) - \sum_{j=0}^{m-1} \pi_j z^j \Big\}\Bigg] - N(z) \sum_{i=0}^{c} k_i z^i.$$

D.3.2 Roots of $A(z)$

Denote $A(z) = K(z) \sum_{i=0}^{c} p_{c-i} z^i$. Clearly, $A(z)$ is also a probability generating function (*pgf*) for the non-negative random variable $V + \tilde{\mathcal{C}}$ where $\tilde{\mathcal{C}}$ is a random variable on $\{0, \ldots, c-1\}$ with distribution $\texttt{Pr}(\tilde{\mathcal{C}} = j) = p_{c-j}, \forall j \in \{0, 1, \ldots, c-1\}$. Also we have

$$A'(1) = K'(1) + \sum_{i=0}^{c} p_{c-i} i = \rho + \sum_{i=1}^{c} p_i (c-i) = \rho + \sum_{i=1}^{c} p_i c - \sum_{i=1}^{c} i p_i = \rho + c - \overline{\mathcal{C}}$$

From our stability condition we know that $\rho < \overline{\mathcal{C}}$. Thus $A'(1) < c$. Since $A(z)$ is a pgf and $A'(1) < c$, by applying the arguments from [3, Theorem 3.2] we conclude that the denominator of equation (5.26) has $c-1$ zeros inside and one on the unit circle, $|z| = 1$.

D.4 Proof of Lemma 4

Proof. It can be observed that if such a 4-tuple (i, j, i', j') exists, the cost can be reduced by assigning i to j' and i' to j, hence we arrive at a contradiction. To show this, consider the six possible cases of relative ordering between $r_i, r_{i'}, s_j, s_{j'}$ which obey $r_i < r_{i'}$ and $s_j > s_{j'}$. We give a pictorial proof in Figure D.1[2]. It is easy to see

[2]For ease of exposition, the requesters and servers are shown to be located along two separate horizontal lines, although they are located on the same real-line.

that in each of the cases, the request distance of the *uncrossed* assignment is either smaller or remains unchanged. □

APPENDIX E

ADDITIONAL PROOFS FOR CHAPTER 6

E.1 Proof of Lemma 5

Let $A(s, r, l)$ denote the event that a random user $r \in R$ is l^{th} closest to $s \in S$ among all servers in S. Denote $NN(r)$ as the geographically nearest server of r. Thus we have

$$\begin{aligned}\Pr[B(\{s_i, s_j\}, r)] =& \Pr[A(s_i, r, 1)] \Pr[A(s_j, r, 2) | NN(r) = s_i] \\ &+ \Pr[A(s_j, r, 1)] \Pr[A(s_i, r, 2) | NN(r) = s_j]. \qquad \text{(E.1)}\end{aligned}$$

It is not difficult to show that all Voronoi cells in V_S have equal areas. As $\Pr[A(s, r, 1)]$ is proportional to the area of the Voronoi cell surrounding s, we have

$$\Pr[A(s, r, 1)] = 1/|S| \; \forall \; s \in S. \qquad \text{(E.2)}$$

Without loss of generality (*W.l.o.g.*) consider a user r placed uniformly at random on $\mathcal{D}$ as shown in Figure 6.1. Denote $\triangle ABC$ as the triangle associated with vertices A, B and C. Let $NN(r) = s_3$. We now evaluate $\Pr[A(s_1, r, 2) | NN(r) = s_3]$. Clearly, $\Pr[A(s_1, r, 2) | NN(r) = s_3] \propto \text{Area}(\triangle WXs_3)$. We also have

$$\begin{aligned}\text{Area}(\triangle WXs_3) &= \text{Area}(\triangle WZs_3) = \text{Area}(\triangle YXs_3) \\ &= \text{Area}(\triangle ZYs_3),\end{aligned}$$

Therefore $\Pr[A(s_i, r, 2)|NN(r) = s_3]$, for $i \in \{1, 2, 4, 5\}$ are all equal. Let $NG(s)$ be the set of neighboring servers of a server $s \in S$ on the square grid. Thus, we have

$$\Pr[A(s_j, r, 2)|NN(r) = s_i] = \begin{cases} \frac{1}{4}, & s_j \in NG(s_i); \\ 0, & \text{otherwise}, \end{cases} \tag{E.3}$$

Note that when $s_j \in NG(s_i)$, the Voronoi cells corresponding to s_i and s_j share an edge. In this case, by definition $(s_i, s_j) \in E$. Combining (E.2) and (E.3) and substituting in (E.1) yields

$$\Pr[B(\{s_i, s_j\}, r)] = \begin{cases} \frac{1}{2|S|}, & (s_i, s_j) \in E; \\ 0, & \text{otherwise}, \end{cases} \tag{E.4}$$

Also, when servers are placed on a square grid, $G_S(X, E)$ is 4- regular. Thus the total number of edges is $|E| = 2|X| = 2|S|$. Substituting $|S| = |E|/2$ in Equation (E.4) yields (6.1) and completes the proof.

E.2 Proof of Lemma 8

Denote X_i as the random variable associated with the load for bin i. Clearly $X = [X_1, X_2, \cdots, X_n]$ follows multinomial distribution

$$\Pr[X = x] = \binom{n}{x_1, \cdots, x_n} \prod_{i=1}^{n} p_i^{x_i}$$

We have $Z = \max(X_1, \cdots, X_n) = \delta(X)$. Clearly, $\delta(x) = \max(x)$ is a schur convex function since $\max(x) = x_{[1]}$ and if $x \succ y$ then $x_{[1]} \geq y_{[1]}$. Also, we have (Chapter 1, [64]): $(p_1, p_2, \cdots, p_n) \succ (1/n, 1/n, \cdots, 1/n)$. whenever $p_i \geq 0$ with $\sum_{i=1}^{n} p_i = 1$. Thus applying Proposition 1 yields $E_p[Z] \geq E_{(1/n,\cdots,1/n)}[Z] \geq k_0 \frac{\log n}{\log \log n}$.

E.3 Proof of Theorem 12

Consider the second order Voronoi diagram: $H_S^{(2)}$ associated with the set of servers S. W.l.o.g. consider a cell $\{s_i, s_j\}$ in $H_S^{(2)}$. The probability that a user selects the server pair $\{s_i, s_j\}$ as its two nearest servers is proportional to the area of the cell $\{s_i, s_j\}$. However, the area distribution of cells in $H_S^{(2)}$ is non-uniform (say with probability distribution p). We can invoke classical balls and bins argument on $H_S^{(2)}$ as follows inspired by the discussion in [53]. We treat each cell in $H_S^{(2)}$ as a bin. Thus by Lemma 7, there are $O(3n)$ bins (or cells). Each ball (or user) choses a bin (or a cell) from a distribution p and let L_p denote the expected maximum asymptotic load across the bins. Let L_U denote the expected maximum asymptotic load across the bins when $O(n)$ balls are assigned to $O(3n)$ bins with each ball choosing a bin uniformly at random. From classical balls and bins theory, $L_U = O(\log n / \log\log 3n) = O(\log n / \log\log n)$. Clearly, by Lemma 8, we have $L_p \geq L_U = O(\log n / \log\log n)$. Since a cell consists of a server pair, one of the server pair corresponding to the maximum load would have load at least $(1/2)L_p$. Thus the maximum load across all servers would be at least $(1/2)L_p \geq O(\log n / \log\log n)$.

BIBLIOGRAPHY

[1] Abadi, H. K., and Prabhakar, B. Stable Matchings in Metric Spaces: Modeling Real-World Preferences using Proximity. *arXiv:1710.05262* (2017).

[2] Abedini, N., and Shakkottai, S. Content Caching and Scheduling in Wireless Networks with Elastic and Inelastic Traffic. *IEEE/ACM Transactions on Networking 22*, 3 (2014), 864–874.

[3] Adan, I. J. B. F., Van Leeuwaarden, J. S. H., and Winands, E. M. M. On the Application of Rouché's Theorem in Queueing Theory. *Operations Research Letters 34* (2006), 355–360.

[4] Adler, Micah, Chakrabarti, Soumen, Mitzenmacher, Michael, and Rasmussen, Lars. Parallel randomized load balancing. *Random Structures and Algorithms 13*, 2 (1998), 159–188.

[5] Agarwal, P.K., Efrat, A., and Sharir, M. Vertical Decomposition of Shallow Levels in 3-Dimensional Arrangements and Its Applications. *SOCG* (1995).

[6] Aho, A. V., Denning, P. J., and Ullman, J. D. Principles of Optimal Page Replacement. *J. ACM 18*, 1 (1971), 80–93.

[7] Ahuja, R.K., Magnanti, T.L., and Orlin, J.B. *Network Flows: Theory, Algorithms, and Applications.* Prentice-Hall, Inc, 1993.

[8] Almeida, Virgilio, Bestavros, Azer, Crovella, Mark, and de Oliveira, Adriana. Characterizing Reference Locality in the WWW. In *4th IEEE Conf. on Parallel and Distributed Information Systems (PDSI'96)* (December 18-20 1996), pp. 92–103.

[9] Arlitt, Martin, Cherkasova, Ludmila, Dilley, John, Friedrich, Rich, and Jin, Tai. Evaluating content management techniques for web proxy caches. In *ACM SIGMETRICS 2000* (2000), vol. 27, pp. 3–11.

[10] Atzori, L., Iera, A., and Morabito, G. The Internet of Things: A Survey. *Computer Networks 54*, 15 (2010), 2787–2805.

[11] Azar, Yossi, Broder, Andrei Z., Karlin, Anna R., and Upfal, Eli. Balanced allocations. *SIAM Journal on Computing 29*, 1 (1999), 180–200.

[12] Baccelli, F., and Brémaud, P. *Elements of Queueing Theory: Palm Martingale Calculus and Stochastic Recurrences*, vol. 26. Springer Science & Business Media, 2013.

[13] Baccelli, François, and Brémaud, Pierre. *Elements of Queueing Theory.* Springer, 2003.

[14] Bailey, N. T. J. On Queueing Processes with Bulk Service. *J. R. Stat. SOCE. 16* (1954), 80–87.

[15] Bash, B. A., and Desnoyers, P. J. Exact Distributed Voronoi Cell Computation in Sensor Networks. In *IPSN* (2007).

[16] Beckmann, Nathan, Chen, Haoxian, and Cidon, Asaf. LHD : Improving Cache Hit Rate by Maximizing Hit Density Relative Size. In *NSDI'18* (2018), pp. 389–404.

[17] Berenbrink, Petra, Czumaj, Artur, Steger, Angelika, and Vöcking, Berthold. Balanced allocations: The heavily loaded case. *SIAM Journal on Computing 35*, 6 (2006), 1350–1385.

[18] Berger, D. S., Gland, P., Singla, S., and Ciucu, F. Exact Analysis of TTL Cache Networks. *Performance Evaluation 79* (2014), 2–23.

[19] Berger, Daniel S., Beckmann, Nathan, and Harchol-Balter, Mor. Practical Bounds on Optimal Caching with Variable Object Sizes. *POMACS 2*, 2 (2018), 1–32.

[20] Berger, Daniel S., Sitaraman, Ramesh K., and Harchol-Balter, Mor. AdaptSize: Orchestrating the Hot Object Memory Cache in a Content Delivery Network. In *NSDI'17* (2017), pp. 483–498.

[21] Boyd, S., and Vandenberghe, L. *Convex Optimization.* Cambridge University Press, 2004.

[22] Breslau, Lee, Cao, Pei, Fan, Li, Phillips, Graham, and Shenker, Scott. Web caching and zipf-like distributions: Evidence and implications. In *IEEE INFOCOM 1999* (1999), vol. 1, pp. 126–134.

[23] Bukac, J. Matching On a Line. *arXiv:1805.00214* (2018).

[24] Byers, J., Considine, J., and Mitzenmacher, M. Geometric Generalizations of the Power of Two Choices. In *SPAA* (2004).

[25] Cao, Pei, and Irani, Sandy. Cost-aware WWW proxy caching algorithms. In *USENIX Symposium on Internet Technologies and Systems (USITS'97)* (1997), no. December, p. 18.

[26] Cha, Meeyoung, Kwak, Haewoon, Rodriguez, Pablo, Ahn, Yong-Yeol, and Moon, Sue. I Tube, You Tube, Everybody Tubes: Analyzing the World's Largest User Generated Content Video System. In *ACM IMC* (2007).

[27] Cherkasova, Ludmila. Improving WWW proxies performance with Greedy-Dual-Size-Frequency caching policy. *HP Laboratories Technical Report*, 98 -69 (1998).

[28] Cooper, Colin, Dyer, Martin, and Greenhill, Catherine. Sampling regular graphs and a peer-to-peer network. *Combinatorics Probability and Computing 16*, 4 (2007), 557–593.

[29] Daley, D. J., and Vere-Jones, D. An Introduction to the Theory of Point Processes: Elementary Theory and Methods. *Springer* (2003).

[30] Dehghan, M., Massoulie, L., Towsley, D., Menasche, D., and Tay, Y.C. A Utility Optimization Approach to Network Cache Design. In *Proc. of IEEE INFOCOM* (2016).

[31] Dehghan, M., Seetharam, A., Jiang, B., He, T., Salonidis, T., Kurose, J., Towsley, D., and Sitaraman, R.h. On the Complexity of Optimal Routing and Content Caching in Heterogeneous Networks. In *Proc. of IEEE INFOCOM* (2015), pp. 936–944.

[32] Doshi, S., and Bhandare, S. An On-demand Minimum Energy Routing Protocol for a Wireless ad-hoc Network. In *ACM Mobile Computing and Communications Review* (2002).

[33] Downey, Allen B. Lognormal and pareto distributions in the internet. *Computer Communications 28*, 7 (2005), 790 – 801.

[34] Ferragut, A., Rodríguez, I., and Paganini, F. Optimizing TTL Caches under Heavy-tailed Demands. In *Proc. of ACM SIGMETRICS* (2016).

[35] Fofack, N. C., Dehghan, M., Towsley, D., Badov, M., and Goeckel, D. L. On the Performance of General Cache Networks. In *VALUETOOLS* (2014).

[36] Fofack, N. C., Nain, P., Neglia, G., and Towsley, D. Analysis of TTL-based Cache Networks. In *VALUETOOLS* (2012).

[37] Fofack, N. C., Nain, P., Neglia, G., and Towsley, D. Performance Evaluation of Hierarchical TTL-based Cache Networks. *Computer Networks 65* (2014), 212–231.

[38] Fronczak, Agata, Fronczak, Piotr, and Hołyst, Janusz A. Average path length in random networks. *Physical Review E - Statistical Physics, Plasmas, Fluids, and Related Interdisciplinary Topics 70*, 5 (2004), 7.

[39] Gale, D., and Shapley, L. College Admissions and Stability of Marriage. *Amer. Math. Monthly 69* (1962), 9–15.

[40] Garetto, M., Leonardi, E., and Martina, V. A Unified Approach to the Performance Analysis of Caching Systems. *ACM TOMPECS 1*, 3 (2016), 12.

[41] Garetto, Michele, Leonardi, Emilio, and Traverso, Stefano. Efficient analysis of caching strategies under dynamic content popularity. In *IEEE INFOCOM 2015* (2015), pp. 2263–2271.

[42] Gast, N. The power of two choices on graphs: the pair-approximation is accurate. In *In Proc. MAMA workshop 2015* (2015), pp. 69–71.

[43] Gast, N., and Houdt, B. V. Asymptotically Exact TTL-Approximations of the Cache Replacement Algorithms LRU(m) and h-LRU. In *ITC 28* (2016).

[44] Goodrich, Michael T., and Tamassia, Roberto. *Algorithm Design: Foundations, Analysis, and Internet Examples.* John Wiley & Sons, 2002.

[45] Goswami, V., and Laxmi, P. V. A Renewal Input Single and Batch Service Queues with Accessibility to Batches. *International Journal of Management Science and Engineering Management* (2011), 366–373.

[46] Gracia-Tinedo, Raúl, Tian, Yongchao, Sampé, Josep, Harkous, Hamza, Lenton, John, García-López, Pedro, Sánchez-Artigas, Marc, and Vukolic, Marko. Dissecting UbuntuOne: Autopsy of a global-scale personal cloud back-end. In *Internet Measurement Conference (IMC'15)* (October 2015), pp. 155–168.

[47] Gross, D., and Harris, C.M. Fundamentals of Queueing Theory. *Wiley Series in Probability and Statistics* (1998).

[48] Ho, I. W. H., Leung, K. K., and Polak, J. W. Stochastic Model and Connectivity Dynamics for VANETs in Signalized Road Systems. *IEEE/ACM Transactions on Networking 19*, 1 (2011), 195–208.

[49] Ioannidis, S., and Yeh, E. Adaptive Caching Networks with Optimality Guarantees. In *Proc. of ACM SIGMETRICS* (2016), pp. 113–124.

[50] Ioannidis, S., and Yeh, E. Jointly Optimal Routing and Caching for Arbitrary Network Topologies. In *Proc. of ACM ICN* (2017), pp. 77–87.

[51] Jaleel, Aamer, Theobald, Kevin B., Steely, Simon C., and Joel, Jr. High Performance Cache Replacement Using Re-Reference Interval Prediction (RRIP). In *37th Int. Symposium on Computer Architecture (ISCA'10)* (Saint Malo, France, June 19-23 2010).

[52] Jiang, Song, and Zhang, Xiaodong. LIRS: An efficient low inter-reference recency set replacement policy to improve buffer cache performance. In *ACM SIGMETRICS 2002* (2002), vol. 30, pp. 31–42.

[53] K., Kenthapadi, and Panigrahy, R. Balanced Allocation on Graphs. In *SODA* (2006).

[54] Kacimi, Rahim, Dhaou, Riadh, and Beylot, André Luc. Load balancing techniques for lifetime maximizing in wireless sensor networks. *Ad Hoc Networks 11*, 8 (2013), 2172–2186.

[55] Kenthapadi, K., and Panigrahy, R. Balanced Allocation on Graphs.

[56] Kingman, F.C. The Effect of Queue Discipline on Waiting Time Variance. *Math. Proc. Cambridge Phil. Soc. 58* (1962), 163–164.

[57] Kingman, J. F. C. The Ergodic Theory of Subadditive Stochastic Processes. *Journal of the Royal Statistical Society: Series B (Methodological) 30*, 3 (1968), 499–510.

[58] Kleinberg, Jon M. Navigation in a small world. *Nature 406*, 6798 (2000), 845.

[59] Kyparisis, J. On Uniqueness of Kuhn-Tucker Multipliers in Nonlinear Programming. *Mathematical Programming 32*, 2 (1985), 242–246.

[60] Leung, K. K., Massey, W. A., and Whitt, W. Traffic Models for Wireless Communication Networks. *IEEE Journal on Selected Areas in Communications 12*, 8 (1994), 1353–1364.

[61] Li, J., Phan, T. K., Chai, W. K., Tuncer, D., Pavlou, G., Griffin, D., and Rio, M. DR-Cache: Distributed Resilient Caching with Latency Guarantees. In *Proc. IEEE INFOCOM* (2018).

[62] Liu, Zhen, Nain, Philippe, Niclausse, Nicolas, and Towsley, Don. Static Caching of Web Servers. In *Multimedia Computing And Networking (MCNC'98)* (San Jose, CA, USA, January 1998), SPIE Press, Ed.

[63] Maggi, Lorenzo, Gkatzikis, Lazaros, Paschos, Georgios, and Leguay, Jérémie. Adapting caching to audience retention rate. *Computer Communications 116* (2018), 159–171.

[64] Marshall, A. W., and Olkin, I. Inequalities: Theory of Majorization and its Applications. In *In: Academic Press* (1979).

[65] Megiddo, Nimrod, and Modha, Dharmendra S. ARC: A Self-Tuning, Low Overhead Replacement Cache. In *FAST'03: 2nd USENIX Conference on File and Storage Technologies* (2003), pp. 115–130.

[66] Meyn, S. P, and Tweedie, R. L. *Markov Chains and Stochastic Stability.* Springer Science & Business Media, 2012.

[67] Mitzenmacher, M. D. The Power of Two Choices in Randomized Load Balancing. In *Ph.D. Dissertation, Harvard University* (1996).

[68] Nain, Philippe, Panigrahy, Nitish K, Basu, Prithwish, and Towsley, Don. One-dimensional service networks and batch service queues. *Queueing Systems 98*, 1 (2021), 181–207.

[69] Nelson, R., and Tantawi, A.N. Approximate Analysis of Fork/join Synchronization in Parallel Queues. *IEEE Transactions on Computers 37*, 6 (1988), 739–743.

[70] Okabe, A., Boots, B., and Sugihara, K. Spatial Tessellations Concepts and Applications of Voronoi Diagrams. In *New York: Wiley* (1992).

[71] Orlin, J.B. A Polynomial Time Primal Network Simplex Algorithm for Minimum Cost Flows. *Mathematical Programming 78* (1997), 109–129.

[72] Panigrahy, Nitish K., Basu, Prithwish, Nain, Philippe, Towsley, Don, Swami, Ananthram, Chan, Kevin S., and Leung, Kin K. Resource allocation in one-dimensional distributed service networks with applications. *Performance Evaluation 142* (2020), 102110.

[73] Panigrahy, Nitish K., Basu, Prithwish, Towsley, Don, Swami, Ananthram, and Leung, Kin K. On the analysis of spatially constrained power of two choice policies. *SIGMETRICS Perform. Eval. Rev. 48*, 3 (Mar. 2021), 51–56.

[74] Panigrahy, Nitish K., Li, Jian, Towsley, Don, and Hollot, C.V. Network cache design under stationary requests: Exact analysis and poisson approximation. *Computer Networks 180* (2020), 107379.

[75] Panigrahy, Nitish K., Li, Jian, Zafari, Faheem, Towsley, Don, and Yu, Paul. Jointly compressing and caching data in wireless sensor networks. In *2019 IEEE International Conference on Smart Computing (SMARTCOMP)* (2019), pp. 57–62.

[76] Panigrahy, Nitish K., Li, Jian, Zafari, Faheem, Towsley, Don, and Yu, Paul. A ttl-based approach for content placement in edge networks, 2020.

[77] Panigrahy, Nitish K., Nain, Philippe, Neglia, Giovanni, and Towsley, Don. A new upper bound on cache hit probability for non-anticipative caching policies. *SIGMETRICS Perform. Eval. Rev. 48*, 3 (Mar. 2021), 138–143.

[78] Panigrahy, Nitish K., Nain, Philippe, Neglia, Giovanni, and Towsley, Don. A new upper bound on cache hit probability for non-anticipative caching policies, 2021.

[79] Panigrahy, Nitish K., Vasantam, Thirupathaiah, Basu, Prithwish, and Towsley, Don. Proximity based load balancing policies on graphs: A simulation study, 2020.

[80] Paschos, Georgios S., Destounis, Apostolos, Vigneri, Luigi, and Iosifidis, George. Learning to Cache with No Regrets. In *Proceedings - IEEE INFOCOM* (2019), pp. 235–243.

[81] Penrose, Mathew. *Random Geometric Graphs.* 2007.

[82] Phillips, Peter C.B. Lectures on stationary and nonstationary times series. `http://korora.econ.yale.edu/phillips/teach/notes/1988-lectures.pdf`, 1992.

[83] Ramadan, E., Narayanan, A., Zhang, Z.-L., Li, R., and Zhang, G. Big Cache Abstraction for Cache Networks. In *Proc. Of IEEE ICDCS* (2017).

[84] Reingold, E. M., and Tarjan, R. E. On a Greedy Heuristic for Complete Matching. *SIAM Journal on Computing 10*, 4 (1981), 676–681.

[85] Rodríguez, I., Ferragut, A., and Paganini, F. Improving Performance of Multiple-level Cache Systems. In *SIGCOMM* (2016).

[86] Sasikumar, Archana, Zhao, Tao, Hou, I, Shakkottai, Srinivas, et al. Cache-version selection and content placement for adaptive video streaming in wireless edge networks. *arXiv preprint arXiv:1903.12164* (2019).

[87] Singpurwalla, Nozer D., and Wong, Man Yuen. Kernel estimators of the failure-rate function and density estimation: An analogy. *Journal of the American Statistical Association 78*, 382 (1983), 478–481.

[88] Srikant, R., and Ying, L. *Communication Networks: an Optimization, Control, and Stochastic Networks Perspective.* Cambridge University Press, 2013.

[89] Stuart, E.A. Matching Methods for Causal Inference: a Review and a Look Forward. *Stat. Sci. 25* (2010), 1–21.

[90] Tanenbaum, Andews S. *Modern Operating Systems.* Prentice Hall Press, 2001.

[91] Traverso, Stefano, Ahmed, Mohamed, Garetto, Michele, Giaccone, Paolo, Leonardi, Emilio, and Niccolini, Saverio. Temporal locality in today's content caching: Why it matters and how to model it. *Computer Communication Review 43*, 5 (2013), 5–12.

[92] Traverso, Stefano, Ahmed, Mohamed, Garetto, Michele, Giaccone, Paolo, Leonardi, Emilio, and Niccolini, Saverio. Unravelling the impact of temporal and geographical locality in content caching systems. *IEEE Transactions on Multimedia 17*, 10 (2015), 1839–1854.

[93] Turner, Stephen R.E. The effect of increasing routing choice on resource pooling. *Probability in the Engineering and Informational Sciences 12*, 1 (1998), 109–124.

[94] Tychogiorgos, G., Gkelias, A., and Leung, K. K. A Non-Convex Distributed Optimization Framework and its Application to Wireless Ad-Hoc Networks. *IEEE Transactions on Wireless Communications 12*, 9 (2013), 4286–4296.

[95] Vecer, J. Dynamic Scoring: Probabilistic Model Selection Based on Utility Maximization. In *Available at SSRN* (2018).

[96] Vöcking, Berthold. How asymmetry helps load balancing. *Journal of the ACM 50*, 4 (2003), 568–589.

[97] Wang, Jane-Ling. Smoothing Hazard Rate. *Encyclopedia of Biostatistics (2nd ed.) 7* (2005), 4986–4997.

[98] Wang, L., Tyson, G., Kangasharju, J., and Crowcroft, J. FairCache: Introducing Fairness to ICN Caching. *In Proc. IEEE 24th Int. Conf. Netw. Protocols (ICNP)* (2016), 1–10.

[99] Weinberg, G.V. Kullback Leibler Divergence and the Pareto Exponential Approximation. *SpringerPlus 5* (2016).

[100] Welch, P.D. On a Generalized m/g/1 Queuing Process in Which The First Customer of Each Busy Period Receives Exceptional Service. *Operations Research 12* (1964), 736–752.

[101] Zerfos, P., Srivatsa, M., Yu, H., Dennerline, D., Franke, H., and Agrawal, D. Platform and Applications for Massive-scale Streaming Network Analytics. *IBM Journal for Research and Development: Special Edition on Massive Scale Analytics 57*, 136 (2013), 1–11.

[102] Zink, M., Suh, K., Gu, Y., and Kurose, J. Watch Global, Cache Local: YouTube Network Traffic at a Campus Network: Measurements and Implications. In *Electronic Imaging* (2008).