2018

Sequential Probing With a Random Start

Joshua Miller

Sequential Probing With a Random Start

Joshua Miller

Nick Pippenger, Advisor

Susan Martonosi, Reader

HARVEY MUDD COLLEGE

Department of Mathematics

May, 2018

Abstract

Processing user requests quickly requires not only fast servers, but also demands methods to quickly locate idle servers to process those requests. Methods of finding idle servers are analogous to open addressing in hash tables, but with the key difference that servers may return to an idle state after having been busy rather than staying busy. Probing sequences for open addressing are well-studied, but algorithms for locating idle servers are less understood. We investigate sequential probing with a random start as a method for finding idle servers, especially in cases of heavy traffic. We present a procedure for finding the distribution of the number of probes required for finding an idle server by using a Markov chain and ideas from enumerative combinatorics, then present numerical simulation results in lieu of a general analytic solution.

Contents

List of Figures

List of Tables

Acknowledgments

I'd like to thank Professor Pippenger for his patience throughout the last year, for offering valuable advice and feedback, and for being an incredible mentor and teacher. I'd like to thank Professor Martonosi for taking the time to read my work and for providing extremely helpful suggestions for improvements I could make, especially in the final stages of this research. Finally, I owe thanks to my friends for their never-ending support.

Chapter 1

Introduction

1.1 Motivation

When you submit any sort of request to a service like Google or WolframAlpha, your request is sent to a collection of servers which processes the request and returns the result to you. Naturally, these services would like to get the result to you as quickly as possible. The time it takes to return your result depends not only on the processing speed of the servers, but also on how quickly an idle server can be located to process your request.

The study of how to quickly find an idle server in our collection of servers is somewhat analogous to that of open addressing as a means of collision resolution in hash tables. In open addressing, we search or probe through our hash table in various manners in search for an empty bin to place an entry. Common probe-sequences described by Knuth (1998) include linear probing, which takes fixed-sized steps through the hash table, and double hashing, which makes use of a second hash function to determine the step size in the search.

The problem of finding an idle server differs slightly from that of open addressing in that contents are not removed from a hash table once entered, whereas servers do become idle again after processing a request. As such, prior research done on open addressing cannot be immediately adapted to apply to problems in server queueing.

Several algorithms for finding idle servers have been studied in the past. Eschenfeldt et al. (2014) studied sequential probing with a fixed starting point. Furthermore, extensive research has been done on open addressing algorithms in hash tables. Although such algorithms are for a slightly different problem, the methods of analyzing them and the results of such analysis can be used as guidance for studying server queueing.

1.2 Sequential Probing With a Random Start

Suppose we have a collection of N servers for processing requests. Each server can be in two possible states: busy or idle. When a server is processing a request, it is busy and cannot accept any additional requests. When a server is not processing anything, it is idle. We will think of the servers as being arranged in a circular manner so that each server has a "next" server to move to when we are searching for an idle server. This circular model is illustrated in Figure 1.1. Requests enter the system via a stationary Poisson point process with parameter λN where $0 \leq \lambda \leq 1$. On average, the rate of entry for each server is λ. When a request enters the system and all servers are busy, the request is dropped. This is know as *loss*. Each server processes requests in a single unit of time, so requests leave the system at a rate equal to the number of busy servers. With these rates, we expect around λN of the servers to be busy at any given time. When λ (known as the *load factor*) is close to 1, the equilibrium of the system is such that most of the servers are busy. In such cases, because idle servers are relatively scarce, it is important that we have an algorithm that can quickly find idle servers.

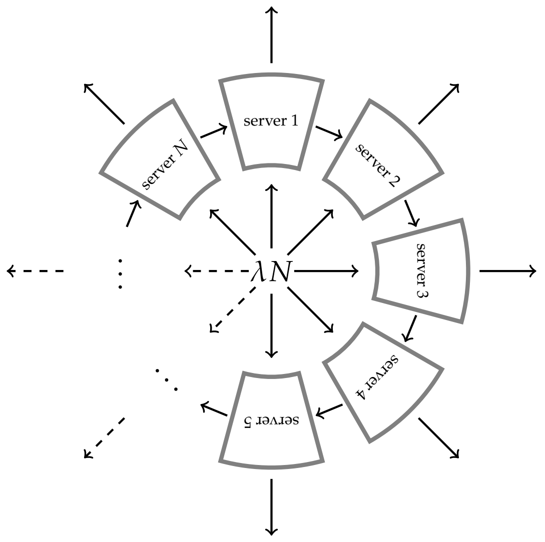


Figure 1.1 Circular model for a system of N servers.

The algorithm we will focus on in this paper is sequential probing with a random start. This is a cross between random probing and sequential probing with a fixed start. First, we make a uniform random probe into the collection of servers. If the initial probe is not successful, we will proceed via sequential probing. That is, we will step one-by-one through the collection of servers until an idle server is found. For simplicity, we will assume that the initial random probe and following sequential probing take place instantaneously. That is, no additional requests will enter the system while another request is still searching for an idle server. This algorithm is of particular interest because it has some advantages and disadvantages when compared to the well-studied random probing.

- **Advantage**: Sequential probing with a random start avoids the possibility of probing the same busy server twice. Especially in the case of a high load factor, random probing is likely to probe some busy servers more than once. By avoiding this, our algorithm improves on one of the shortcomings of random probing.

- **Disadvantage**: Sequential probing with a random start will perform badly when there exist long sequences of busy servers. For example, if our initial probe is at the beginning of a long chain of busy servers, then we must step through all of those servers before finding an idle server. Random probing does not get "stuck" in long sequences of busy servers like this.

Because it has both an advantage and disadvantage when compared to random probing, it is not immediately obvious which performs better. The purpose of this paper is to determine the performance of sequential probing with a random start. In particular, we will attempt to determine the probability distribution of the number of probes required to find an idle server, and we will compare numerical results regarding the performance of sequential probing with a random start to the performance of other algorithms.

We can attempt to make some early predictions of what the expected number of probes required to find an idle server might be. In open addressing, random probing is simply a Bernoulli process where the probability of finding an empty bin given a load factor of λ is $1-\lambda$. So the expected number of probes required to find an empty bin using random probing in hash tables is

$$\frac{1}{1-\lambda}.$$

According to Knuth (1963), the expected number of probes required to find an empty bin using linear probing in hash tables is approximately

$$\frac{1}{2}\left(1+\frac{1}{(1-\lambda)^2}\right).$$

Furthermore, Eschenfeldt et al. (2014) showed that the expected number of probes required to find an idle server when using sequential probing from a fixed starting point is approximately

$$\frac{\lambda}{2}.$$

Given these results of similar problems, we expect the average number of probes required when using sequential probing with a random start to be between $1/(1-\lambda)$ and $1/(1-\lambda)^2$.

1.3 Tools for Analysis

Before diving into the analytical approach to this problem (covered in Chapter 2), we should familiarize ourselves with several mathematical concepts.

1.3.1 Markov Chains

Suppose we have a sequence of random variables $X_0, X_1, \ldots$ on a set S where S is the set of states accessible to the system and X_k is the state of the system at time k. According to Serfozo (2009), such a sequence is considered to be a *Markov Chain* if it has the property that

$$P(X_{n+1}=j \mid X_0,\ldots,X_n)=P(X_{n+1}\mid X_n).$$

That is, the probability distribution of the state at time $n+1$ only depends on the state at time n. In particular, we will say that the probability of moving from state i to state j is denoted by

$$P(X_{n+1}=j \mid X_n=i)=p_{ij}.$$

We can take these *transition probabilities* and assemble them into a *transition matrix* P where $P_{i,j}=p_{i,j}$. Such a matrix is called *right stochastic*, and each of its rows sums to 1.

Suppose we have a row vector x which represents the initial distribution of states of the system we are modeling. Then the vector xP^k represents the probability distribution of states after k steps in the system. A *stationary probability vector* π is a row vector which does not change after any number of steps in the system. It can be thought of as an equilibrium, and is generally called the *steady state* of the system. Because π does not change with additional steps in the system, we have that $\pi P = \pi$. We will use this concept in Section 2.1.

1.3.2 Negative Hypergeometric Distribution

In Chapter 3, we will compare our algorithm to random probing without replacement; also known as *uniform probing*. It will be helpful to understand the negative hypergeometric distribution in order to understand the behavior of this method of probing.

Suppose we have an urn filled with N marbles, K of which are red or "unsuccessful," and $N - K$ of which are green or "successful." We'd like to draw marbles from the urn without replacement until we draw r green (or "successful") marbles, then answer how many red (or "unsuccessful") marbles we encountered. The number of unsuccessful draws within a sample of r successes will be denoted by k. Let X be a random variable which counts the number of unsuccessful draws. The negative hypergeometric distribution, a discrete probability distribution, has the probability mass function

$$P(X = k) = \frac{\binom{k+r-1}{k}\binom{N-r-k}{K-k}}{\binom{N}{K}}$$

for $k = 0, 1, \ldots, K$. The expected value of X is

$$E[X] = \frac{rK}{N - K + 1}.$$

In Chapter 3, we will think of idle servers as "successful" draws and busy servers as "unsuccessful" draws, then we'll be interested in how many busy servers are drawn before drawing one idle server. This quantity plus one will be the number of probes required to find an idle server.

1.3.3 Pólya's Enumeration Theorem

In a famous paper published in Acta Mathematica, Pólya (1937) re-discovered a theorem originally found by Redfield (1927) for counting the number of orbits of a group action on a set. This theorem, which is a generalization of Burnside's Lemma, is presented in its unweighted form below.

Let X be a finite set, let G be a permutation group which acts on X, and let Y be a finite set of states which can be used to describe X. For example, X may be a set of beads, Y may be a set of colors such that each bead may be colored by one of the elements of Y, and G may be a group of permutations of X. Let Y^X be the set of functions $X \to Y$. In the example of the colored beads, such functions would be different colorings of the beads. Thus, Y^X is the set of colorings of the beads. We use this notation because there are $|Y|^{|X|}$ possible colorings of the beads. The number of orbits under G of the colorings of X can be computed as

$$|Y^X/G| = \frac{1}{|G|} \sum_{g \in G} m^{c(g)}$$

where $m = |Y|$ and $c(g)$ is the number of cycles which make up g when thought of as a permutation on X.

In Section 2.2 we will use this theorem to determine the number of distinct server configurations up to rotation.

Chapter 2

An Analytical Approach

As we will see later on in Section 2.2, finding a general analytic solution for an arbitrary number of servers is quite difficult. Despite this, we can find solutions for small systems with relative ease, and we can describe a procedure for finding a solution for N servers.

2.1 A Complete Solution for a Small System

Suppose we have a system of 5 servers, denoted $s_1, s_2, \ldots, s_5$. Given which servers are busy and which are idle, we can draw a diagram which illustrates the current state of the system:

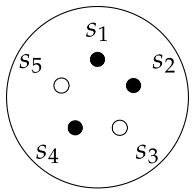


Figure 2.1 Diagrammatic representation of a configuration with two non-adjacent idle servers in a system of 5 servers.

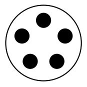
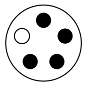
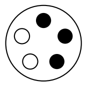
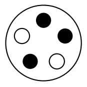
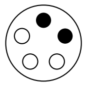
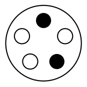
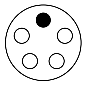
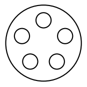
In the diagram above, empty circles denote idle servers and filled circles denote busy servers. This diagram represents when s_1, s_2 and s_4 are busy, and s_3 and s_5 are idle. In future diagrams, we will not label the servers with $s_1, \ldots, s_5$. Rather, it will be implied that we begin with s_1 at the topmost position and proceed clockwise.

Figure 2.1 actually represents an equivalence class of server configurations. We will consider configurations that are identical up to rotation to be equivalent. Figure 2.1 is the representative for configurations which consist of two adjacent busy servers and a third busy server which is not adjacent to the first two. Rather than describing configurations in this way, we will use 5-bit binary strings of the form $a_1a_2a_3a_4a_5$ where

$$a_k = \begin{cases} 1 & s_k \text{ is busy} \\ 0 & s_k \text{ is idle.} \end{cases}$$

For example, the configuration shown in Figure 2.1 is represented by the string 11010. Table 2.1 shows all the equivalence classes of configurations, along with the binary representations of each configuration in that class:

Table 2.1 Equivalence classes of configurations a 5-server system.

Representative Server Diagram	Binary Representations	Description
	11111	Five busy servers
	11110, 01111, 10111, 11011, 11101	Four busy servers
	11100, 01110, 00111, 10011, 11001	Three adjacent busy servers
	11010, 01101, 10110, 01011, 10101	Two non-adjacent idle servers
	11000, 01100, 00110, 00011, 10001	Two adjacent busy servers
	10100, 01010, 00101, 10010, 01001	Two non-adjacent busy servers
	10000, 01000, 00100, 00010, 00001	One busy server
	00000	No busy servers

In general, we will always draw the diagram which represents the first binary representation when arranged in reverse lexicographical order. Note that all

$2^5 = 32$ possible 5-bit binary strings are listed in this table, so every possible configuration of 5 servers is represented in one of these 8 equivalence classes

Now that we have a way to visualize each of the 8 classes of configurations, we can begin to examine the interactions between states. We will construct a state transition diagram for the system. Beginning in state `00000`, the only possible transition is into state `10000`. There are 5 ways to make this transition (any of the 5 idle servers could become busy), and because requests enter the system at a rate λ, we would label the `00000` $\rightarrow$ `10000` transition with 5λ. Similarly, if we're in state `10000` and the next event is a departure event, then we must move down into the `00000` state. Departures happen at a rate of 1, and there is only 1 way to move from `10000` to `00000`, so we label the `10000` $\rightarrow$ `00000` transition with 1. Following this logic, we create a state diagram to describe the rates of transitions between states. See Figure 2.2.

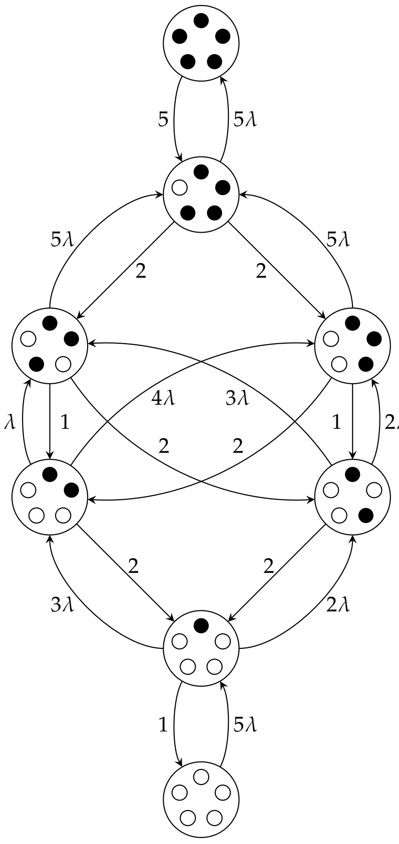


Figure 2.2 State diagram with rates of transitions for a 5-server system.

We'd like to be able to model this system using a Markov chain so that we can find out the likelihood of being in any given state in the long run. To do this, we must first modify the state diagram in Figure 2.2 so that each edge is labeled with the probability of that transition occurring, rather than the rate. The probability of a transition can be found by simply dividing the rate of that transition by the sum of rates of transitions departing from the same state. Figure 2.3 shows the resulting diagram.

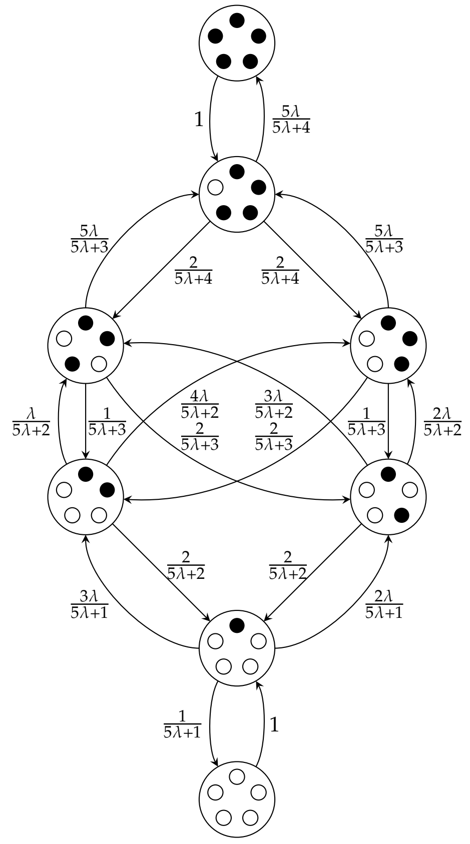

1
$\frac{5\lambda}{5\lambda+4}$
$\frac{5\lambda}{5\lambda+3}$
$\frac{2}{5\lambda+4}$
$\frac{2}{5\lambda+4}$
$\frac{5\lambda}{5\lambda+3}$
$\frac{\lambda}{5\lambda+2}$
$\frac{1}{5\lambda+3}$
$\frac{4\lambda}{5\lambda+2}$
$\frac{2}{5\lambda+3}$
$\frac{3\lambda}{5\lambda+2}$
$\frac{2}{5\lambda+3}$
$\frac{1}{5\lambda+3}$
$\frac{2\lambda}{5\lambda+2}$
$\frac{2}{5\lambda+2}$
$\frac{2}{5\lambda+2}$
$\frac{3\lambda}{5\lambda+1}$
$\frac{2\lambda}{5\lambda+1}$
$\frac{1}{5\lambda+1}$
1

Figure 2.3 State diagram with probabilities of transitions for a 5-server system.

From this diagram, we can write down the stochastic matrix which describes the system.

$$P = \begin{array}{c|cccccccc} & 00000 & 10000 & 10100 & 11000 & 11010 & 11100 & 11110 & 11111 \\ \hline 00000 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 10000 & \frac{1}{5\lambda+1} & 0 & \frac{2\lambda}{5\lambda+1} & \frac{3\lambda}{5\lambda+1} & 0 & 0 & 0 & 0 \\ 10100 & 0 & \frac{2}{5\lambda+2} & 0 & 0 & \frac{3\lambda}{5\lambda+2} & \frac{2\lambda}{5\lambda+2} & 0 & 0 \\ 11000 & 0 & \frac{2}{5\lambda+2} & 0 & 0 & \frac{\lambda}{5\lambda+2} & \frac{4\lambda}{5\lambda+2} & 0 & 0 \\ 11010 & 0 & 0 & \frac{2}{5\lambda+3} & \frac{1}{5\lambda+3} & 0 & 0 & \frac{5\lambda}{5\lambda+3} & 0 \\ 11100 & 0 & 0 & \frac{1}{5\lambda+3} & \frac{2}{5\lambda+3} & 0 & 0 & \frac{5\lambda}{5\lambda+3} & 0 \\ 11110 & 0 & 0 & 0 & 0 & \frac{2}{5\lambda+4} & \frac{2}{5\lambda+4} & 0 & \frac{5\lambda}{5\lambda+4} \\ 11111 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{array}$$

Entry P_{jk} is the probability of transitioning from state j to state k. In order to find the steady state of the system, we need to find the stationary probability vector π such that $\pi P = \pi$. In this case π is a left eigenvector of P with eigenvalue 1. To find π, we first find the cokernel of $P - I$ where I is the identity matrix. This can be found as the kernel of $(P - I)^T = P^T - I$. With some help from Maple, we find that

$$\operatorname{coker}(P - I) = \ker(P^T - I) = \operatorname{span}\left(\begin{bmatrix} \frac{24}{625\lambda^4} \\ \frac{24}{625}\frac{5\lambda+1}{\lambda^4} \\ \frac{6}{625}\frac{(125\lambda^2+100\lambda+24)(5\lambda+2)}{(25\lambda^2+23\lambda+6)\lambda^3} \\ \frac{6}{125}\frac{625\lambda^3+900\lambda^2+440\lambda+72}{(25\lambda^2+23\lambda+6)\lambda^3} \\ \frac{2}{125}\frac{(625\lambda^2+500\lambda+108)(5\lambda+3)}{(25\lambda^2+23\lambda+6)\lambda^2} \\ \frac{2}{125}\frac{(625\lambda^2+650\lambda+192)(5\lambda+3)}{(25\lambda^2+23\lambda+6)\lambda^2} \\ \frac{1}{5}\frac{5\lambda+4}{\lambda} \\ 1 \end{bmatrix}^T\right)$$

The only stochastic vector in this span is the steady state π. With further aid from Maple, we find π to be

$$\pi = \begin{bmatrix} \frac{12}{625\lambda^4+500\lambda^3+300\lambda^2+120\lambda+24} \\ \frac{12(5\lambda+1)}{625\lambda^4+500\lambda^3+300\lambda^2+120\lambda+24} \\ \frac{3\lambda(625\lambda^3+750\lambda^2+320\lambda+48)}{15625\lambda^6+26875\lambda^5+22750\lambda^4+12900\lambda^3+5160\lambda^2+1272\lambda+144} \\ \frac{3\lambda(625\lambda^3+900\lambda^2+440\lambda+72)}{15625\lambda^6+26875\lambda^5+22750\lambda^4+12900\lambda^3+5160\lambda^2+1272\lambda+144} \\ \frac{\lambda^2(3125\lambda^3+4375\lambda^2+2040\lambda+324)}{15625\lambda^6+26875\lambda^5+22750\lambda^4+12900\lambda^3+5160\lambda^2+1272\lambda+144} \\ \frac{\lambda^2(3125\lambda^3+5125\lambda^2+2910\lambda+576)}{15625\lambda^6+26875\lambda^5+22750\lambda^4+12900\lambda^3+5160\lambda^2+1272\lambda+144} \\ \frac{125}{2}\frac{\lambda^3(5\lambda+4)}{625\lambda^4+500\lambda^3+300\lambda^2+120\lambda+24} \\ \frac{625}{2}\frac{\lambda^4}{625\lambda^4+500\lambda^3+300\lambda^2+120\lambda+24} \end{bmatrix}^T.$$

Now that we know the steady state of the system, we must analyze each state to determine the distribution of number of required probes to find an idle server. Let's examine configuration 11010 as an example. For each idle server, there is a set of servers which, if initially probed, will result in us finding the associated idle server. We will call such sets of servers *catch regions*.

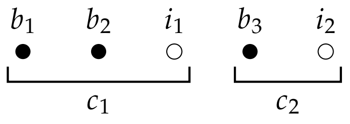


Figure 2.4 Catch regions of configuration 11010.

In Figure 2.4, i_1 and i_2 denote the two idle servers of the configuration and b_1, b_2, and b_3 denote the three busy servers of the configuration. We use c_1 and c_2 to denote the catch regions of i_1 and i_2 respectively. Catch region c_ℓ is the set of servers which, if initially probed into, will result in us finding i_ℓ if our algorithm is sequential probing with a random start. The size of catch region c_ℓ will be denoted by m_ℓ.

Let X be a random variable which counts the number of probes required to find an idle server using our search algorithm. Then we get the probability mass function seen in Table 2.2.

We get this because there are two places we could initially probe such that it only requires 1 probe to find an idle server; namely i_1 and i_2. Since

Table 2.2 Probability mass function for configuration 11010.

k	1	2	3
$P(X=k)$	$\frac{2}{5}$	$\frac{2}{5}$	$\frac{1}{5}$

the initial probe is uniform, there is a 2/5 chance of initially probing one of these two. Similarly, if we initially probe b_2 or b_3, then we require 2 probes to find an idle server. Again, the probability of initially probing one of these two servers is 2/5. Finally, there is only one place we could probe which will require us to make 3 probes to find an idle server, and that is b_1. There is a 1/5 chance of initially probing here.

We find the expected number of probes required for configuration 11010 to be

$$E[X] = 1\frac{2}{5} + 2\frac{2}{5} + 3\frac{1}{5} = \frac{9}{5} \text{ probes.}$$

We can repeat such calculations for each of the 8 configurations. Table 2.3 shows the PMF and expected value for each configuration.

Table 2.3 PMF and expected value for each of the 8 configurations.

	k					
	1	2	3	4	5	$E[X]$
$P(X=k \mid 00000)$	1	0	0	0	0	1
$P(X=k \mid 10000)$	$\frac{4}{5}$	$\frac{1}{5}$	0	0	0	$\frac{6}{5}$
$P(X=k \mid 10100)$	$\frac{3}{5}$	$\frac{2}{5}$	0	0	0	$\frac{7}{5}$
$P(X=k \mid 11000)$	$\frac{3}{5}$	$\frac{1}{5}$	$\frac{1}{5}$	0	0	$\frac{8}{5}$
$P(X=k \mid 11010)$	$\frac{2}{5}$	$\frac{2}{5}$	$\frac{1}{5}$	0	0	$\frac{9}{5}$
$P(X=k \mid 11100)$	$\frac{2}{5}$	$\frac{1}{5}$	$\frac{1}{5}$	$\frac{1}{5}$	0	$\frac{11}{5}$
$P(X=k \mid 11110)$	$\frac{1}{5}$	$\frac{1}{5}$	$\frac{1}{5}$	$\frac{1}{5}$	$\frac{1}{5}$	3
$P(X=k \mid 11111)$	N/A	N/A	N/A	N/A	N/A	N/A

Together with the probability of being in each state in the long run (provided by π), we can have a complete probability distribution for the number of probes required to find an idle server. Note that in the case of all servers being busy (configuration 11111), an incoming request cannot find an idle server, so it does not make sense to talk about the probabilities of requiring various numbers of probes. This is why, in Table 2.3 the entries associated with 11111 are all "NA." When requests enter a system where all servers are busy, these requests are called *loss* and are discarded. We touch briefly again on loss at the end of this chapter.

2.2 A Procedure for a General Analytic Solution

Now that we have an understanding of how to solve for a small system, we can attempt to find a solution for n servers. The solution outlined in this section follows the same approach as the example in the previous section, but is more formal and general.

Let $X = \{s_1, s_2, \dots, s_n\}$ be a set of n servers, let $Y = \{\text{busy}, \text{idle}\}$ be the set of states available to each server, and let C_n be the cyclic group of order n. The action of C_n on G can be described as rotations of the vertices of an n-gon. Thus, by thinking of each server as being the vertex of an n-gon, the unique orbits of configurations of servers under C_n give us the unique server configurations up to rotation. Note that we do not wish to consider configurations equivalent up to reflections because the direction in which we probe is fixed. If M is the number of unique server configurations up to rotation, then by application of Pólya's enumeration theorem we can say

$$M = |Y^X/C_n| = \frac{1}{n} \sum_{g \in C_n} 2^{c(g)}.$$

As a quick check, we can see that this is consistent with what we found in the previous section. The elements of $C_5 = \{1, r, r^2, r^3, r^4\}$ written as permutations of $X = \{s_1, \dots, s_5\}$ are shown in Table 2.4. Using the information

Table 2.4 Elements of C_5 as permutations in S_5.

Element of D_{10}	As element of S_5	Number of cycles
1	(1)(2)(3)(4)(5)	5
r	(12345)	1
r^2	(13524)	1
r^3	(14253)	1
r^4	(15432)	1

from this table, we find the number of distinct server configurations to be

$$\frac{1}{5}\left(2^5 + 4(2^1)\right) = 8.$$

As Table 2.1 shows, we did indeed have 8 distinct configurations of servers.

Moving on with the general solution, our next task is to try to fill the stochastic matrix P for the system. We will use the word *state* to mean an equivalence class of configurations that are the same up to action of C_n. We have M states, so the stochastic matrix P for the system is $M \times M$.

Consider two states S_a and S_b such that S_a has α busy servers, S_b has β busy servers, and $\beta - \alpha = 1$. In general, S_a and S_b are connected in the state transition diagram if any of the configurations in S_b can be obtained by changing any of the idle servers of any of the configurations in S_a to be busy. If S_a and S_b are not connected, then entries P_{ab} and P_{ba} are zero. If S_a and S_b are connected, then there will be arrows denoting transitions from S_a to S_b and vice versa. The transition probability from S_a to S_b is

$$P_{ab} = \frac{U_{a,b}\lambda}{n\lambda + \alpha}$$

where $U_{a,b}$ is the number of ways to set an idle server in a configuration in S_a busy to obtain a configuration in S_b. Similarly, the transition probability from S_b to S_a is

$$P_{ba} = \frac{D_{b,a}}{n\lambda + \beta}$$

where $D_{b,a}$ is the number of ways to set a busy server in a configuration in S_b idle to obtain a configuration in S_a. Part of the difficulty in finding an analytical solution to this problem is finding a way to generalize if two states are connected, and if they are, finding U and D.

The next step once we have P is to compute its steady state π such that $\pi P = \pi$. This is another part of the difficulty in solving this problem analytically. While there are ways to find the steady state of a Markov chain, it is difficult to say what the general form of the solution will be, especially considering the difficulty in writing what P looks like, in general.

We must now consider each state and come up the PMF for the number of probes required in each state, as well as an expression for the expected number of probes. Recall the type of model portrayed by Figure 2.4. Each idle server i_ℓ has a catch region c_ℓ of size m_ℓ associated with it. In general, the probability of k probes being required is the number of catch regions of at least size k, divided by the number of servers n. The expected value of the number of probes can be expressed more elegantly. For any given catch region c_ℓ, the expected number of probes required in that region is

$$\frac{1 + 2 + \cdots + m_\ell}{m_\ell}.$$

Then we sum this value over all the catch regions, scaling each by its size relative to the whole collection (m_ℓ/n). If there are a total of h idle servers in configuration S, then

$$E[X \mid S] = \sum_{1\le\ell\le h} \frac{m_\ell}{n}\left(\frac{1+2+\cdots+m_\ell}{m_\ell}\right) = \frac{1}{2n}\sum_{1\le\ell\le h} m_\ell(m_\ell+1).$$

After finding the steady state π, along with the PMF for each state, we will have a complete solution to the PMF for the number of probes required to find an idle server. Additionally, using the steady state π and the expected value $E[X \mid S]$ we just found, we can give an expression for the expected number of probes required.

Finally, although we could use the steady state π to help us determine the probability of loss, an alternative way of determining the loss in the system if we're not interested in anything else is by using the *Erlang B formula*. The formula is

$$P_{\text{loss}} = \frac{\frac{(\lambda n)^n}{n!}}{\sum_{i=0}^{n}\frac{(\lambda n)^i}{i!}},$$

where n is the number of servers in the system and λ is the parameter of the Poisson process governing arrivals of requests in the system.

Chapter 3

A Simulation Approach

Although there are difficulties in finding an analytic solution for a system with N servers, we can still get a number of results via simulation.

We have written a *discrete event simulation*, which loops through a sequence of events, each of which occurs with a certain probability. In our case, we have two events:

- **Arrival events** are events in which a new request enters the system and finds an idle server if available. This event handles both the initial random probe and the sequential probing portions of the search. If there are N servers, K busy servers, and the rate of arrival is λ, then the probability of the next event being an arrival event is

$$\frac{\lambda N}{\lambda N + K}.$$

- **Departure events** are events in which a single request leaves the system, thus making a previously busy server idle again. If there are N servers, K busy servers, and the rate of arrival is λ, then the probability of the next event being a departure event is

$$\frac{K}{\lambda N + K}.$$

We get these probabilities based on the rates of arrivals and departures. The rate of arrival in the whole system is λN because there is an arrival rate of λ to each of the N servers. The departure rate is K because the rate of processing is 1 for each of the K busy servers.

The simulation is set to run until a specified number of requests have successfully been processed. In each iteration, either an arrival or a departure occurs based on the probabilities presented above. Each time a request successfully finds an idle server, we record the number of probes that were required to find that idle server.

Before we present results of this simulation, it's important to discuss system equilibriums. Two in particular are important to consider:

- equilibrium in the number of busy and idle servers and
- equilibrium in structure.

Equilibrium in the number of busy and idle servers is simply when we have the expected number of busy servers. That is, when roughly λN servers are busy. Once we have an equilibrium number of busy servers, it may take some time for the configuration of servers to become stable.

In order to get consistent results about the performance of this algorithm in the long run, we wish to only record results when the system has entered both of these equilibriums. We can ensure we're in the first equilibrium by starting the simulation on a system which begins with λN servers busy. To ensure we're in the second equilibrium, we can keep track of the expected value of the number of required probes and start reporting results once it settles down. Figure 3.1 shows a plot of the expected value of the required number of probes over time and illustrates when we reach structural equilibrium.

Once we are in the desired equilibriums, we can plot a histogram of the number of probes required to find an idle server and calculate the expected value and standard deviation for the number of required probes. The histogram is show in Figure 3.2.

In addition to the histogram displayed, we were able to calculate that the expected number of probes was 95.45 with a standard deviation of 141.37. Despite the large value of the standard deviation here, repeated simulations consistently show that the expected number of probes is around 95. Note that this is close to and slightly less than $1/(1-\lambda) = 100$ which we predicted in Section 1.2. This seems to indicate that, at least for $\lambda = 0.99$, sequential probing with a random start has a better performance than random probing. Later on we will see results that show this is not the case for all λ.

Figure 3.2 shows a distribution which is similar in shape to the exponential distribution. To see if we actually have exponential decay here, we can make a semi-log plot of the same results. See Figure 3.3. We see that we don't

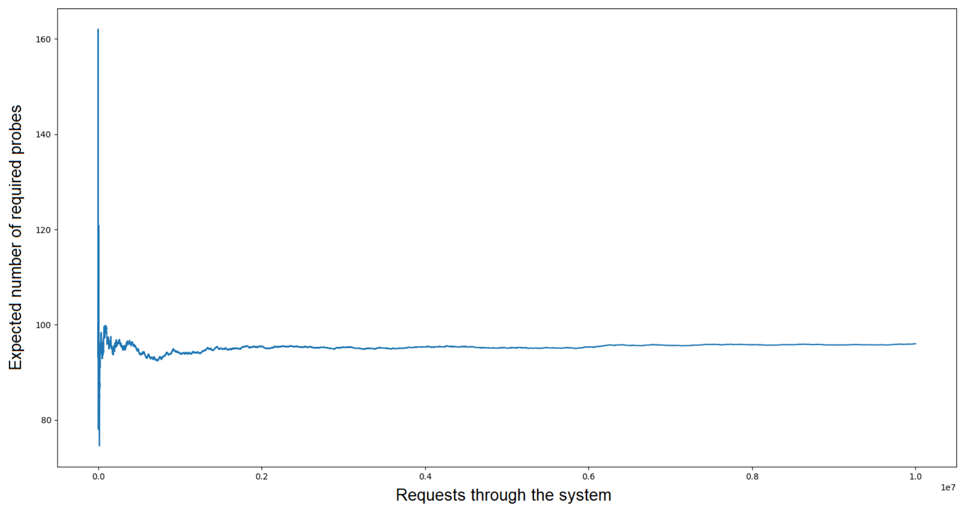


Figure 3.1 A plot of the expected value of the number of required probes over time. This is the result after 10 million requests through a system of 1000 servers with $\lambda = 0.99$.

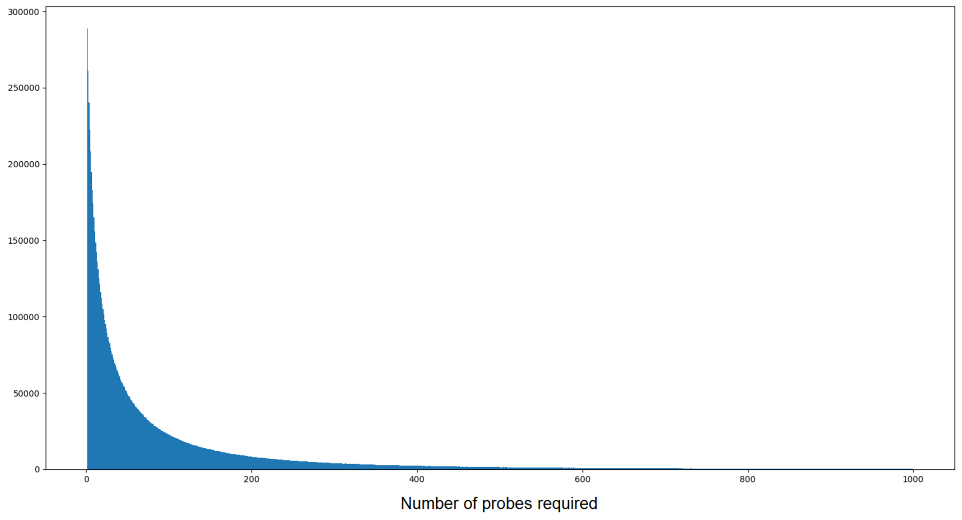


Figure 3.2 Histogram of the number of probes required to find an idle server in a system with 1000 servers, $\lambda = 0.99$, and beginning from expected equilibrium. This is the result after 10 million requests went through the system.

quite have exponential decay, as is evidenced by the lack of linearity on the left end of the semi-log plot. This indicates qualitative difference between sequential probing with a random start and random probing, the results for which we would expect to follow a geometric distribution. The fact that the left end of the distribution trends upward from a linear fit indicates that sequential probing performs better than random probing; at least for $\lambda = 0.99$.

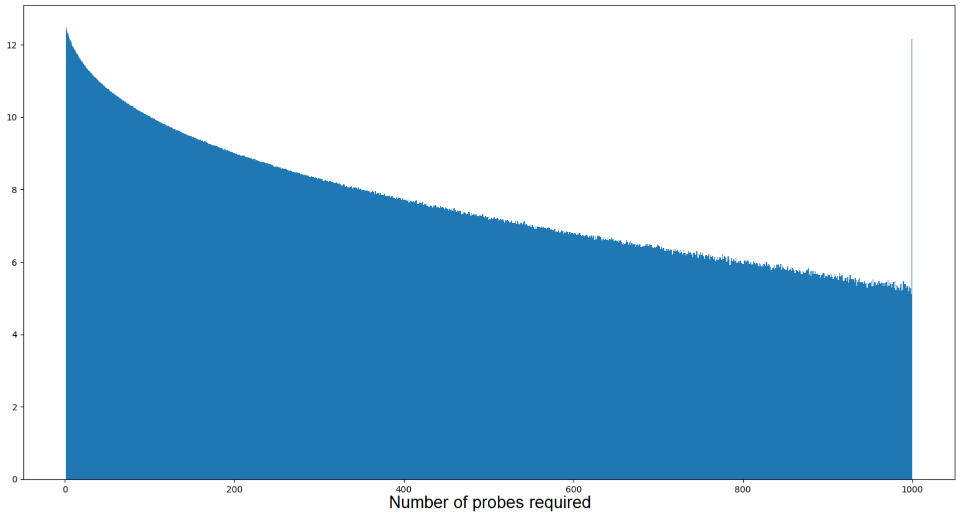
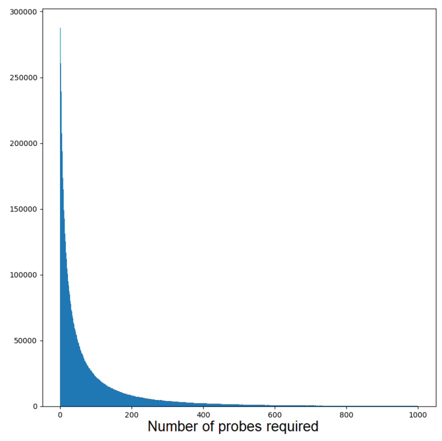


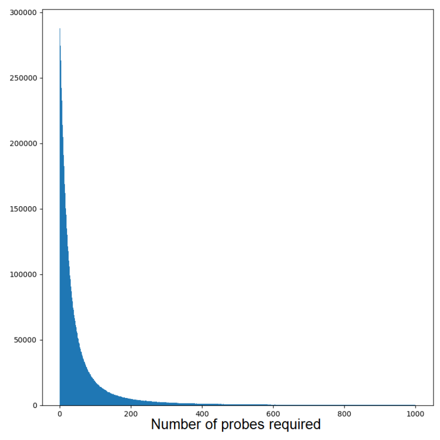
Figure 3.3 Semi-log plot of the histogram shown in Figure 3.2.

We mentioned back in Section 1.2 that sequential probing with a random start has both advantages and disadvantages in comparison to random probing. Because of this, it is difficult to quantify the algorithm's disadvantage by comparing it to random probing. The solution is to compare to random probing without replacement. That is, we will consider an algorithm that randomly probes the collection of servers in search of an idle server, but does so without replacement as to not probe the same place twice. Such an algorithm is called *uniform hashing* in the related problem of open addressing Yao (1985), so we will refer to it as *uniform probing* here. Relative to uniform probing, sequential probing with a random start only has the disadvantage mentioned in Section 1.2 and no advantage. By comparing these two, we can isolate the effects of this disadvantage.

In order to compare these two algorithms, we ran both algorithms in parallel. That is, both went through the same sequence of arrival and departure events at the same time. We collected the results for each algorithm individually. As before, we can plot the histogram of the number of probes required to find an idle server, as well as calculate the expected value and standard deviation of the number of required probes.



a. Sequential probing with a random start



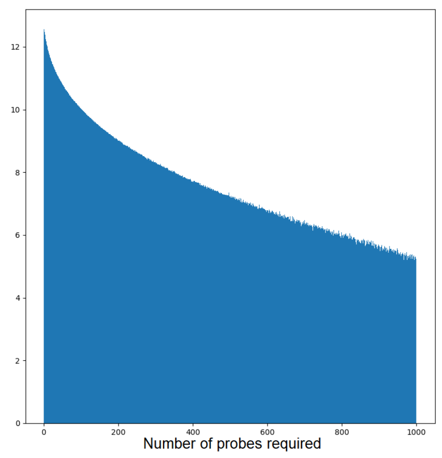
b. Uniform probing

Figure 3.4 Histograms of the number of probes required to find an idle server in a system of 1000 servers $\lambda = 0.99$, and beginning from expected equilibrium. These are the results after 10 million requests went through the system.

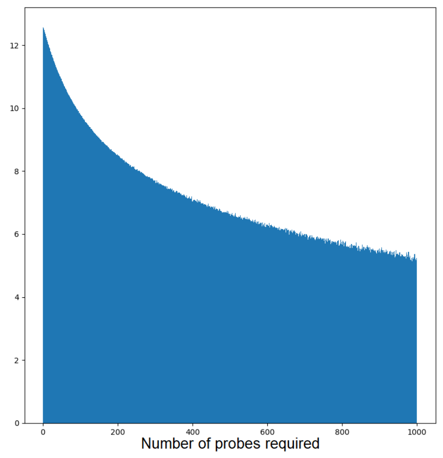
Figure 3.4 compares the histograms for both sequential probing with a random start and uniform probing when simulated in parallel. Both share similar shapes, although uniform probing appears to decay more quickly. As before, we can also examine semi-log plots to compare how close to an exponential distribution these histograms are. See Figure 3.5.

We note that uniform probing is farther from exponential than sequential probing with a random start. In particular, due to the more severe deviation from a linear fit, uniform appears to perform better than sequential probing with a random start, as expected. This is also consistent with the expected values for the number of required probes we calculated. From this parallel simulation we found that the expected number of probes for sequential probing with a random start was 95.22 with a standard deviation of 140.94, while uniform probing had an expected value of 68.77 with a standard deviation of 118.94.

Up until this point, we have only been looking at simulations with a



a. Sequential probing with a random start



b. Uniform probing

Figure 3.5 Semi-log plots of the histograms shown in Figure 3.4.

load factor of $\lambda = 0.99$. The reason for this is that we are interested in the performance when the system is under a heavy load. However, to get a bigger picture of the performance of this algorithm, especially relative to other methods of probing, we should examine results for other values of λ.

To begin, we can look at a simple plot of the expected number of probes required vs λ. This is shown in Figure 3.6.

This gives us an idea of how the expected number of probes required grows with λ, and even gives us a way to compare to the expected performance of other probing methods such as random probing and uniform probing. Recall that the expected performance of random probing is

$$\frac{1}{1-\lambda}.$$

If we overlay this curve on the same plot as the data seen in Figure 3.6, we can see how the two algorithms compare, on average, at different values of λ. See Figure 3.7. This figure shows our empirical data about the performance of sequential probing with a random start (the points) compared to the theoretical performance of random probing for a fixed load factor of λ (represented by the curve).

It would be ideal to try to fit a curve to this data and have a formula for expected performance of sequential probing with a random start based on λ.

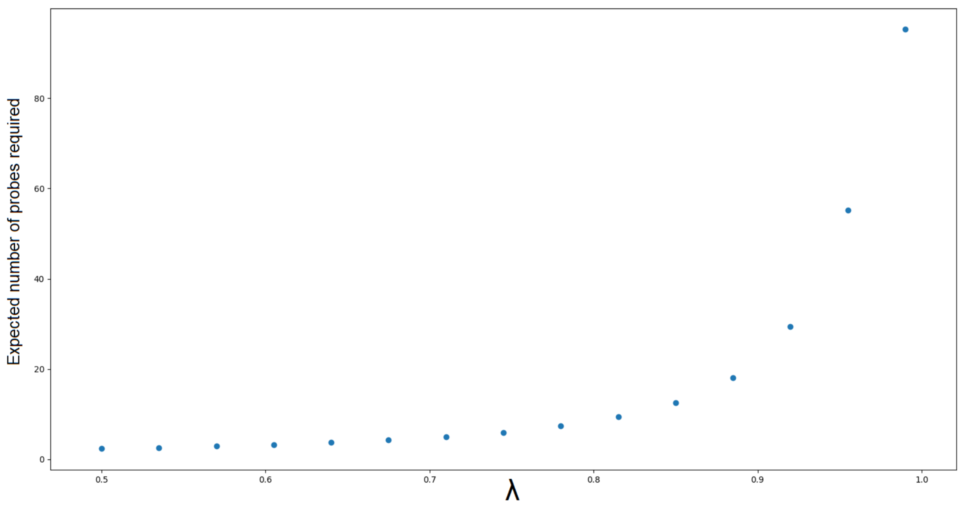


Figure 3.6 The expected number of probes required to find an idle server in a system of 1000 servers for λ ranging from 0.5 to 0.99 in steps of size 0.035.

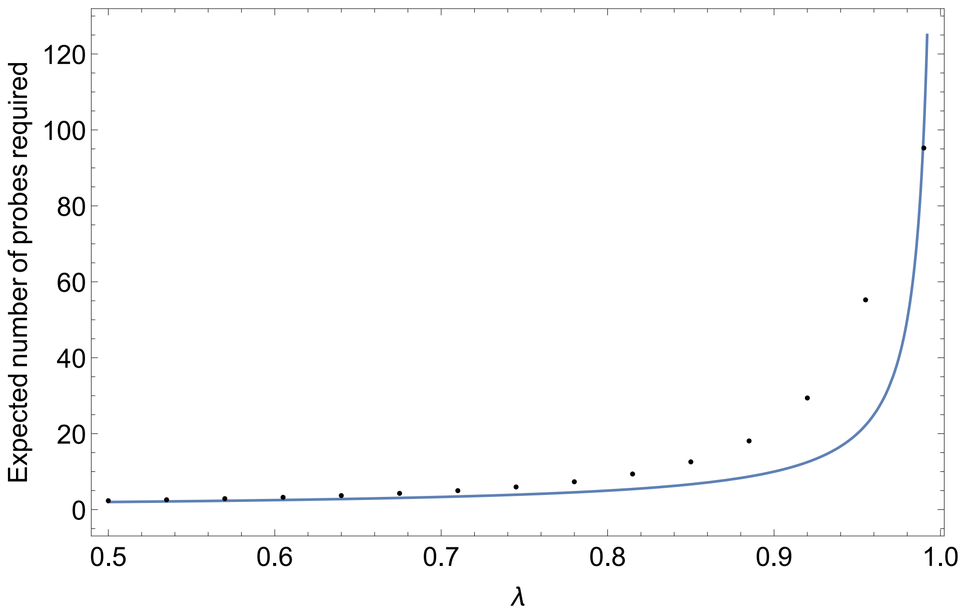


Figure 3.7 The data from Figure 3.6 along with the curve $1/(1-\lambda)$.

However, because the performance blows up at $\lambda = 1$, it is difficult to find an accurate fit. To help alleviate this, we can transform the the data so that the expected performance of the algorithm we're comparing to is a represented as a straight line which does not blow up in the region of $\lambda \in [0, 1]$.

For random probing, since we expect the performance to be $E[X] = 1/(1 - \lambda)$, we solve for λ and find

$$\lambda = 1 - \frac{1}{E[X]}$$

and plot $1 - 1/E[X]$ on the vertical axis and λ on the horizontal axis. Then the performance of random probing is the 45° line, and the data for sequential probing with a random start can still be overlain on to make a comparison.

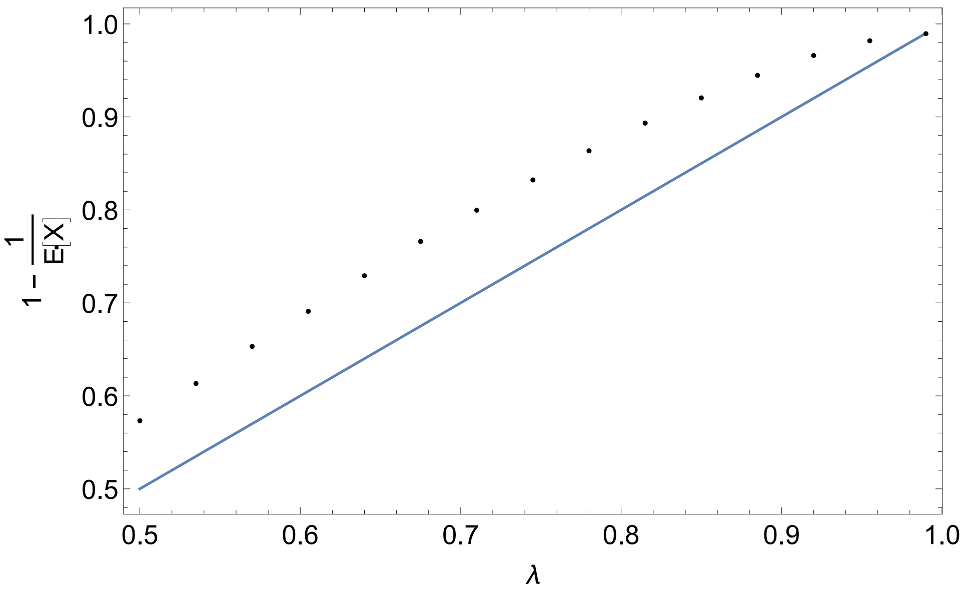


Figure 3.8 Data and curve from Figure 3.7 transformed to plot $1 - \dfrac{1}{E[X]}$ vs λ.

Figure 3.8 shows the results we've been discussing transformed in this way. The line represents the expected performance of random probing, and the points represent the results of our simulations. When transformed in this way, we can find a polynomial fit to the data rather than trying to find a fit the form $1/(1 - \lambda)$ or $1/(1 - \lambda)^2$. The shape of the data suggests that a polynomial fit of degree 2 would be appropriate here.

Using Mathematica, we find a degree-2 polynomial fit of the form $a\lambda + b\lambda^2$. We leave off a constent term in order to meet the condition that when $\lambda = 0$, we have $1 - 1/E[X] = 0$. The fit is

$$1 - \frac{1}{E[X]} = 1.34325\lambda - 0.321585\lambda^2.$$

Table 3.1 shows some information about the goodness of this fit. In addition, the R^2 value is 0.99981, and the adjusted R^2 is 0.999783. Transforming the

Table 3.1 Goodness of fit: $1 - 1/E[X]$ vs. λ.

	Estimate	Std. Error	t-Statistic	P-Value
a	1.34325	0.0230279	58.3316	4.07063×10^{-18}
b	−0.321585	0.0282078	−11.4006	1.80048×10^{-8}

fit allows us to obtain a fit for the pre-transformed data. Doing so yields

$$E[X] = \frac{1}{1 + 0.321585\lambda^2 - 1.34325\lambda}.$$

This is the formula which tells us the expected performance of sequential probing with a random start as a function of λ. Note that this formula shares some similarities to what we predicted the results might be back in Chapter 1.

We would like to compare the performance of sequential probing with a random start to that of uniform probing in this manner. If we follow the same procedure, the transformation we'll make is based on the formula for the expected value of the negative hypergeometric distribution (see Section 1.3.2), but with +1 added since we're interested in also counting the probe which finds an idle server. Thus, we would use the formula

$$E[X] = \frac{\lambda N}{(1-\lambda)N + 1} + 1.$$

Here we're taking $K = \lambda N$ to be the number of busy servers and $r = 1$ to be the number of idle servers we encounter before stopping. Solving for λ yields

$$\lambda = \frac{(N+1)(E[X]-1)}{NE[X]}.$$

Normally we would plot the quantity on the right side of this equation on the vertical axis and λ on the horizontal axis to get a plot akin to what we have for random probing in Figure 3.8. However, this time we should be more careful in how we make our comparison. Note that the formula for the expected value of the negative hypergeometric distribution depends on the number of busy servers. If we plug in λN for this quantity, we fail to account for instances where there are more or fewer busy servers. In our simulation, we don't always have λN servers busy, so it would be inaccurate to compare the simulation results to a line which assume a constant number of busy servers. In order to account for this, we should compute the likelihood of having a certain number of servers busy, then weight the formula for the expected performance of uniform probing accordingly. Note that this is different from calculating the probability of having certain *configurations* of servers, which we discussed in Chapter 2. Here we only need to consider the number of busy servers, not their arrangement.

Let P_K denote the probability of having K busy servers. Allen (1990) gives the following formula for the steady state probability of having K busy servers:

$$P_K = \frac{(\lambda N)^K}{K!}\left(\sum_{i=0}^{N} \frac{(\lambda N)^i}{i!}\right)^{-1}$$

This formula holds regardless of the probing method. However, as N grows large and λ grows close to 1, using this formula to compute probabilities becomes more difficult. As a result, we turn to a recursive method of computation. Recall that, given N servers, K of which are busy, and an individual arrival rate of λ, system-wide rates of arrival and departure are λN and K respectively. Based on these rates, we can establish the following relationship between the probabilities of having K and $K+1$ servers busy:

$$\lambda N P_K = (K+1)P_{K+1}.$$

Then if we assign P_N to be some constant C, we can recursively calculate the relative probabilities of having any number of servers busy. These should sum to 1 if C was picked correctly, but if not, we can normalize these probabilities by what they sum to since this method correctly finds the relative probabilities of these events. Computing probabilities in this way yields the same results as those given by the formula presented by Allen (1990), but this method avoids dealing with very large numbers.

We can now use these probabilities in conjunction with the expected value formula for the negative hypergeometric distribution to get a more precise result for the expected performance of uniform probing with parameter λ.

In particular, we use the formula

$$E[X] = \sum_{K=0}^{N} P_K E[X \mid K],$$

where

$$E[X \mid K] = \frac{K}{N - K + 1} + 1$$

is the expected number of probes required to find an idle server using uniform probing when there are K busy servers.

It's difficult to solve for λ and make the same transformation as we did for random probing, so here we present untransformed results comparing sequential probing with a random start to uniform probing. Figure 3.9 shows this comparison where the data for uniform probing was computed using the method described above. The plot shows that uniform probing is consistently better than sequential probing with a random start, as expected. However, the performance of SPRS follows the same general shape as that of uniform probing, and generally isn't too much worse. The difference between the points for uniform probing and sequential probing with a random start illustrates negative impact of the disadvantage of poor performance in long sequences of busy servers.

To get a closer look at the comparison between these two probing algorithms for high load factors, we can look at an additional plot where λ ranges from 0.95 to 0.995. This comparison can be seen in Figure 3.10. Even in heavy traffic, we see that sequential probing with a random start has similar performance to uniform probing.

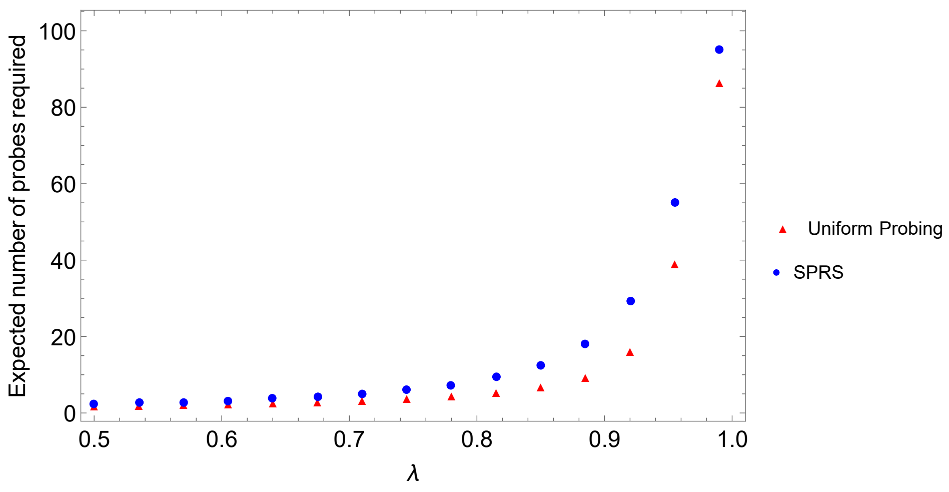


Figure 3.9 A comparison of the expected number of probes required to find an idle server using sequential probing with a random start (SPRS) and uniform probing. These results are for a system of 1000 servers with λ ranging from 0.5 to 0.99 in steps of size 0.035.

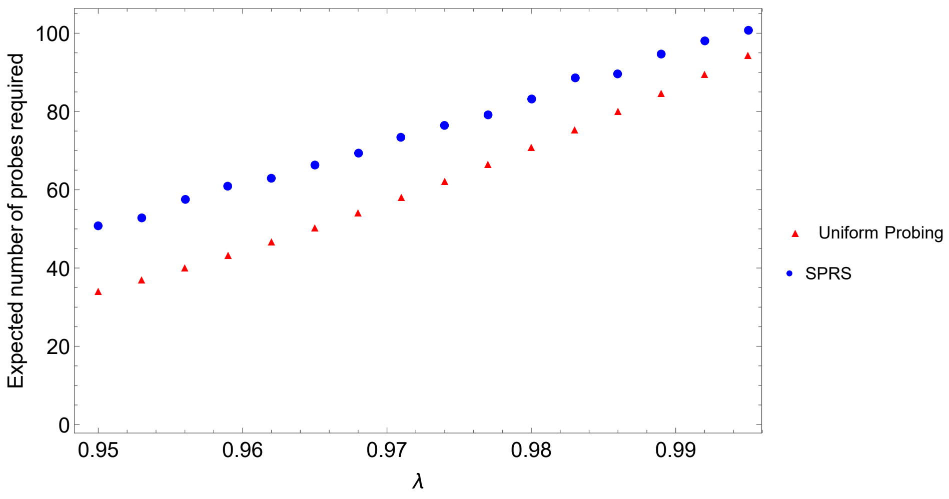


Figure 3.10 The same comparison as in Figure 3.9, but with λ ranging from 0.95 to 0.995 in steps of size 0.003.

Chapter 4

Conclusions and Future Work

We have provided an analytical solution for the probability distribution of the number of required probes for a 5-server system and have outlined a procedure for determining the solution in the case of N servers. Writing down the solution in the general case is difficult, so we have also written a simulation which can provide results for large systems that we would otherwise not try to solve analytically. Aside from providing numerical results for statistics like the expected performance of our algorithm, the simulations allow us to look at a number of histograms and other plots which help illustrate the shape of the probability distribution which arises from sequential probing with a random start and compare our algorithm to others.

Results indicate that with $\lambda \geq 0.99$, sequential probing with a random start performs slightly better than random probing. However, looking at a wider range of values of λ reveals that sequential probing with a random start is not always better than random probing. Regardless, a key finding is that these two do seem to share similar performance for many values of λ. Sequential probing with a random start is more similar to random probing than linear probing, even though the randomness of SPRS is limited to a single initial probe.

Aside from finding that SPRS performs more similarly to random probing than to linear probing, we also compared SPRS to uniform probing in an attempt to understand the disadvantageous aspects of our algorithm. Random probing has an advantage and disadvantage when compared to sequential probing with a random start, but uniform probing does not have any disadvantage in such a comparison. Thus, comparing our algorithm to uniform probing helps to highlight the disadvantageous behavior of our algorithm; namely that it does not perform well when there are long chains of busy servers. A careful comparison of the two resulted in Figures 3.9 and 3.10 which provide a visual representation of the difference between SPRS and uniform probing. Future research could attempt to model the difference between the numerical results in this comparison as a fraction of the performance of uniform probing. This would give quantitative insight into the severity of the disadvantage of poor performance in long chains of busy servers.

Future research could investigate this disadvantage in another way which does not involve comparing SPRS to other algorithms. Simulations could keep track of the lengths of chains of busy servers as well as the results regarding how many probes were required to find an idle server. The number of required probes could be modeled as a function of the number of and sizes of such chains.

This could also be a worthwhile approach for the analytical solution. Rather than using configurations of servers as states in a Markov model, there may be a way to define states based on the number of and sizes of chains of busy servers. This may simplify the problem slightly and help simplify the procedure for finding the solution, or even allow us to write down a general solution.

Bibliography

Allen, Arnold O. 1990. *Probability, Statistics, and Queueing Theory: With Computer Science Applications*. Academic Press, Inc., 2nd ed.

Bolch, Gunter, Hermann de Meer, Kishor Shridharbhai Trivedi, and Stefan Greiner. 2006. *Queueing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science Applications*. John Wiley and Sons. URL https://doi.org/10.1002/0471791571.ch6.

Eschenfeldt, Patrick, Ben Gross, and Nicholas Pippenger. 2014. Stochastic service systems, random interval graphs and search algorithms. *Random Structures and Algorithms* 45(3):421–442.

Harary, Frank, and Edgar M. Palmer. 1973. *Graphical Enumeration*. Academic Press, Inc.

Knuth, Donald. 1963. Notes on open addressing.

———. 1998. *The Art of Computer Programming, Volume 3: (2nd Ed.) Sorting and Searching*. Addison Wesley Longman Publishing Co., Inc.

Pólya, G. 1937. Kombinatorische Anzahlbestimmungen für Gruppen, Graphen und chemische Verbindungen. *Acta Mathematica* 68(1):145–254. doi:10.1007/BF02546665. URL https://doi.org/10.1007/BF02546665.

Pólya, G., and R.C. Read. 1987. *Combinatorial Enumeration of Groups, Graphs, and Chemical Compounds*. Springer-Verlag New York. doi: 10.1007/978-1-4612-4664-0. URL https://doi.org/10.1007/978-1-4612-4664-0.

Redfield, J. Howard. 1927. The theory of group-reduced distributions. *American Journal of Mathematics* 49(3):433–455. URL http://www.jstor.org/stable/2370675.

Schuster, Eugene F., and William R. Sype. 1987. On the negative hypergeometric distribution. *International Journal of Mathematical Education in Science and Technology* 18(3):453–459. URL https://doi.org/10.1080/0020739870180316.

Serfozo, Richard. 2009. *Probability and Its Applications: Basics of Applied Stochastic Processes*. Springer.

Yao, Andrew C. 1985. Uniform hashing is optimal. *J ACM* 32(3):687–693. doi:10.1145/3828.3836. URL http://doi.acm.org/10.1145/3828.3836.