Server Optimization and Optimal Routing to Parallel Servers

Bachelor's Term Project-II Report submitted to

Indian Institute of Technology Kharagpur

in partial fulfillment for the award of the degree of

Bachelor of Technology

in

Electronics and Electrical Communication Engineering

By

Rishikant Kashyap

(20EC39028)

Under the supervision of

Professor Rajarshi Roy

Department of Electronics and Electrical Communication Engineering

Indian Institute of Technology Kharagpur

Spring Semester, 2023-24

DECLARATION

I declare that

a. I worked with my supervisor to complete the tasks included in this report.
b. No other institute has received this work for any certification or degree.
c. I have complied with the standards and directives outlined in the Institute's Ethical Code of Conduct.
d. I have always provided proper acknowledgment of the sources of the data, theoretical analysis, figures, and text I have used by citing them in the thesis text and providing their specifics in the references. Additionally, where required, I have obtained consent from the sources' copyright holders.

Date: April 28, 2024
Place: Kharagpur

Rishikant Kashyap
20EC39028

DEPARTMENT OF ELECTRONICS AND ELECTRICAL COMMUNICATION ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR - 721302, INDIA

CERTIFICATE

This is to certify that the project report entitled "**Server Optimization and Optimal Routing to Parallel Servers**" submitted by **Rishikant Kashyap** (Roll No. 20EC39028) to IIT Kharagpur towards partial fulfillment of requirements for the award of degree of Bachelor of Technology in Electronics and Electrical Communication Engineering is a record of bona fide work carried out by him under my supervision and guidance during Spring Semester, 2023-24.

Dr. Rajarshi Roy
Professor

Department of Electronics and
Electrical Communication Engineering

Indian Institute of Technology, Kharagpur

Date: April 28, 2024
Place: Kharagpur

Acknowledgments

I would like to thank my Project Supervisor Prof. Rajarshi Roy for directing, guiding, and supporting me throughout this project.

I am thankful to my Faculty Advisor Prof. Gourab Dutta, Department of Electronics and Electrical Communication Engineering, Indian Institute of Technology, Kharagpur, for always being very accommodating of all of our requests and feedback.

I am grateful to my friends, who were constant support behind me and helped me to go through this work. At last, but not the least, I would like to thank my parents for their invaluable trust and support in all my choices.

Abstract

This report focuses on studying the problems to find suitable strategies to optimize the servers' resources by effectively routing incoming jobs to the servers to complete tasks in less time and maximize the utilities gained in the process.

Firstly, we address a simple problem that revolves around analyzing the distribution of file sizes in a system for downloading files and optimizing server resources accordingly. In this scenario, files are requested for download over a specified period, and their sizes vary. The distribution of file sizes follows a Gaussian distribution, with larger files tending to have higher sizes compared to smaller ones. The system comprises two types of servers: fast servers with high download speeds and slow servers with lower speeds. The objective is to efficiently schedule download requests to minimize the total time taken for processing, considering the characteristics of the files and the capabilities of the servers. By analyzing the distribution of file sizes and optimizing server resources based on file sizes, the proposed solution aims to enhance the overall performance and resource utilization of the system.

The second problem focuses on solving the optimal routing problem in a discrete-time system where a job dispatcher is connected to M parallel servers. At each time interval, the job dispatcher receives a random number of jobs, and it routes these jobs to the parallel servers. Each server has its service capacity and is linked to an unknown underlying utility. The objective is to create a routing policy that maximizes the total utility obtained at the end of a given time period while considering the stochastic nature of both the job arrival process and the service operations by the servers. This is further extended to the general case of multiple job dispatchers connected to multiple servers forming a bipartite network model.

By combining insights from stochastic queueing theory and stochastic multi-armed bandit problems, policies for both problems have been designed. Analytical bounds on regret are established, linking it to utilities and the length of server queues. Priority-K and Upper Confidence Priority-K policies are proposed for known and unknown utility cases, respectively, achieving logarithmic regret bounds. These results are extended to the general problem involving multiple job dispatchers and servers in a bipartite network.

Contents

List of Figures

Chapter 1

Server Optimization Problem

Consider a file-downloading system characterized by servers with variable speeds and two distinct file types differing significantly in size. The principal aim of this problem is to devise and assess a scheduling algorithm aimed at enhancing file retrieval efficiency within this system.

The file-downloading system comprises two primary components: files and servers.
Files are categorized into two types based on their size:

a. large files, with a size of 100MB (equivalent to 10^8 bits), and
b. small files, considerably smaller with a size of 1KB (equivalent to 10^3 bits).

Concurrently, the system features two server types:

a. fast servers, boasting a download speed of 1 MB/s, and
b. slow servers, operating at a slower pace with a download speed of 100 bits/s.

Upon receiving download requests from receivers, the system enqueues them based on the First Come First Serve (FCFS) principle. The frequency of download requests per unit of time adheres to a Poisson distribution, with the mean rate of requests being a modifiable parameter.

$$P(N(t) = n) = e^{-\lambda t}\frac{(\lambda t)^n}{n!}$$

$$E\big(N(t)\big) = \lambda t$$

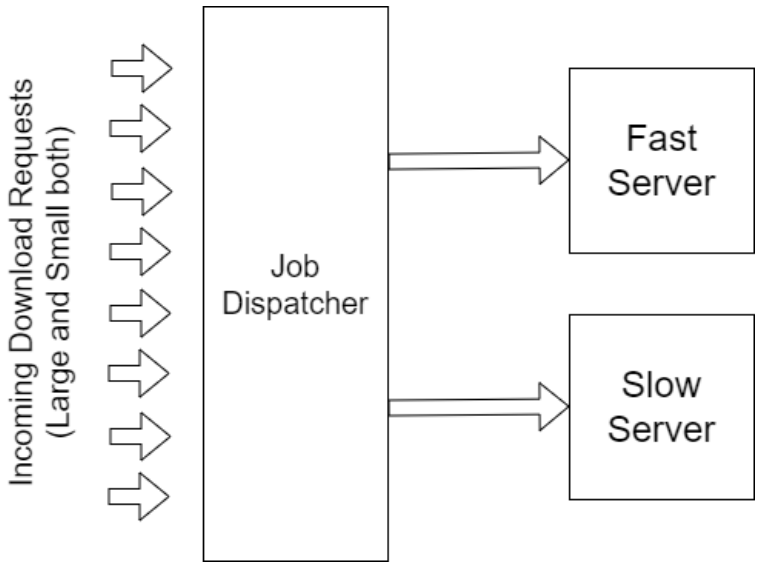

Figure 1: Server Optimization Problem

In our investigation of the file downloading system, we observed the considerable disparity in processing times between fast and slow servers when tasked with handling large files. This discrepancy underscored the necessity of optimizing the assignment of tasks to servers to minimize processing time and enhance overall system efficiency. To address this, we explored three distinct task assignment algorithms.

The initial algorithm, which prioritizes the assignment of tasks to the first available server, resulted in suboptimal performance. Slow servers often encountered large files, leading to significantly longer processing times and inefficient resource utilization.

Subsequently, we devised an algorithm that systematically assigns small tasks to slow servers and large tasks to fast servers. This strategy leverages the fast-processing capabilities of fast servers for large files while ensuring efficient utilization of slow servers for smaller tasks. Our simulation results indicated that this approach yielded satisfactory outcomes, given the defined parameters of server speeds and file sizes.

Motivated by the goal of further reducing server idle time, we proposed a third algorithm. In this approach, large files are exclusively processed by fast servers, while small files may be delegated to fast servers if they are available. This adaptive strategy effectively minimizes idle time across servers, contributing to improved system performance. However, the marginal time benefit gained from processing small files on fast servers was limited, as these files are inherently processed quickly.

We further design a simulation mechanism to gauge the efficacy of the proposed scheduling algorithms. The simulation workflow encompasses several pivotal steps. Initially, download requests are generated over a predefined time span, aligning with a Poisson distribution. Each request delineates the file type, whether large or small. Subsequently, the scheduling algorithm orchestrates the prioritized assignment of large files to fast servers and small files to slow servers. Requests are handled sequentially, under the FCFS paradigm.

The simulation results underscore the critical importance of optimizing the utilization of high-capacity servers in file-downloading systems. Idle time in these servers signifies the underutilization of their potential, leading to suboptimal performance. By minimizing idle time through strategic task prioritization and dynamic assignment based on server availability, overall system efficiency can be enhanced. This proactive approach not only reduces processing time but also maximizes the return on investment in high-performance server infrastructure, yielding better outcomes.

```python
def simulation_1(requests, fast_server, slow_server, threshold):
    total_time = []

    for t in range(len(requests)):
        for request in requests[t]:
            if fast_server.work_time <= slow_server.work_time:
                time_taken = fast_server.process_file(request)
            else:
                time_taken = slow_server.process_file(request)
        fast_server.work_time = max(fast_server.work_time, t+1)
        slow_server.work_time = max(slow_server.work_time, t+1)

    return total_time
def simulation_2(requests, fast_server, slow_server, threshold):
    total_time = []

    for t in range(len(requests)):
        for request in requests[t]:
            if request >= threshold:
                time_taken = fast_server.process_file(request)
            else:
                time_taken = slow_server.process_file(request)
        fast_server.work_time = max(fast_server.work_time, t+1)
        slow_server.work_time = max(slow_server.work_time, t+1)

    return total_time

def simulation_3(requests, fast_server, slow_server, threshold):
    total_time = []

    for t in range(len(requests)):
        for request in requests[t]:
            if request >= threshold or fast_server.work_time <= slow_server.work_time:
                time_taken = fast_server.process_file(request)
            else:
                time_taken = slow_server.process_file(request)
        fast_server.work_time = max(fast_server.work_time, t+1)
        slow_server.work_time = max(slow_server.work_time, t+1)

    return total_time
```

Figure 2: Algorithm used for Server Optimization Problem

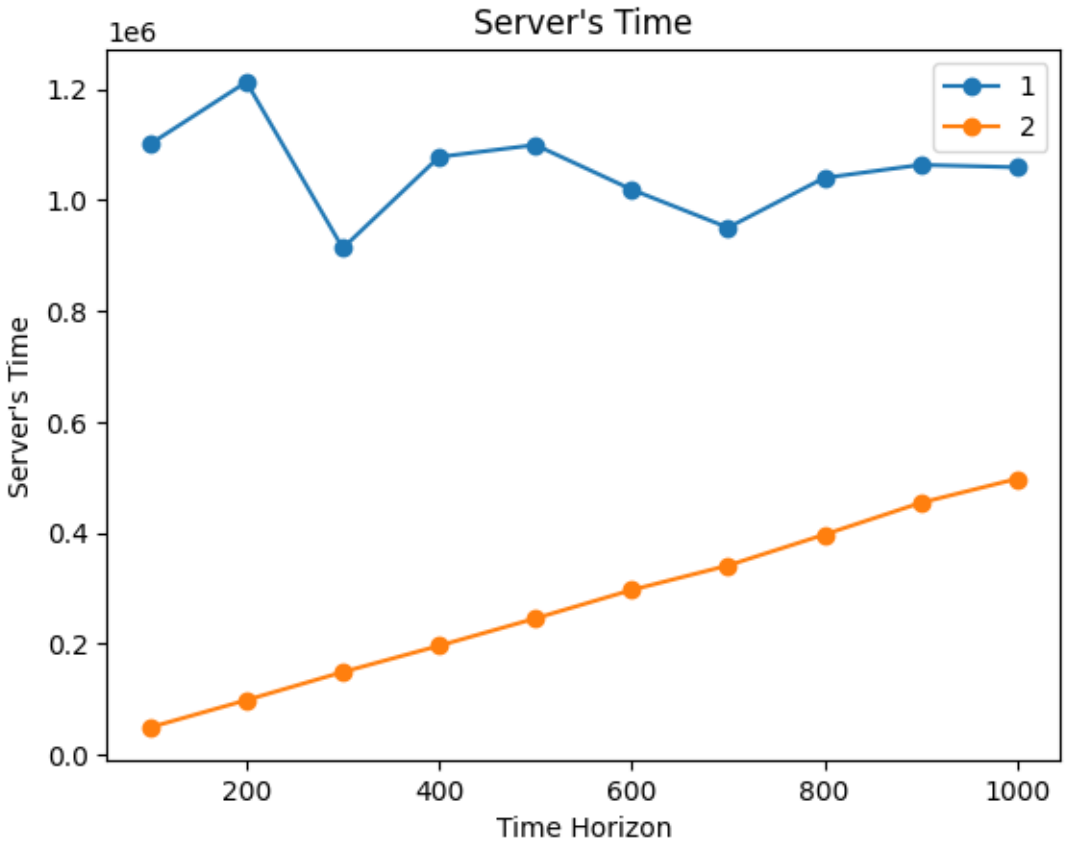

Figure 3: Simulation Results for Server Optimization Problem

Chapter 2

Optimal Routing to Parallel Servers

Now let us consider a system that includes a job dispatcher and several parallel servers connected to it. For each time interval t, the job dispatcher receives several incoming jobs and then it promptly routes them to the servers where jobs are stored in a queue for execution. The routing problem under this model has been the subject of much research. Numerous applications in manufacturing lines, web server farms, and communication networks are captured by this type of system paradigm and hence it is much needed to optimize such scheduling problems as it has a lot of practical implementations.

Here, we take an **optimization of the system utility** approach to the routing problem for parallel servers. In this system, whenever a server finishes a job, it gains a utility linked to that particular server. Our aim focuses on building a routing policy that optimizes the overall gained utility in this setting. The servers' utility isn't known at first, but it can be discovered via finishing tasks and processing feedback. A utility model of this kind can act as a representation for measurement of performance depending on the server, like energy consumption and quality of service, which are not available at first to the system operator but are available through feedback following job completions.

In particular, a discrete-time system that is linked to a group of M parallel servers via a job dispatcher is taken into consideration. At dispatcher, the jobs appear at an unknown arrival rate, which is stochastic in nature. During each time slot, every incoming job is routed by the dispatcher to one of the linked servers, where the execution process is carried on. A data storing queue on each server s_m holds incoming jobs in reserve. The total count of jobs that server s_m can finish at each time slot is the result of a stochastic process which is given by an unknown service rate. Every server (s_m) has an underlying utility v_m, and when s_m finishes a task, the corresponding utility is gained in the process.

Although the underlying utilities are unknown beforehand, each job finished at s_m results in a noisy observation $\widehat{v_m}$ with $E[\widehat{v_m}] = v_m$. For such a parallel server system, our objective is to create routing policies that maximize the utility attained at the end of a given period T. We choose to use regret as the metric of policy performance, which is obtained by taking the absolute difference between the expected total utility of the proposed policy and the supremum of the expectation of total utilities over all of the possible policies—including those which have access to network statistics and the unknown underlying utilities beforehand.

The main goal of this report is to study efficient routing policies to solve the problem of optimal routing by combining the methods from the stochastic multi-armed bandits and control of stochastic queueing networks. In particular, first, the problem lying in the use of static policies is explored. We find support for the result obtained in the previous task to minimize the idle time of the servers with high utilities. After that, a routing strategy known as Priority-K is discussed. It works by assigning the highest utility servers first to the incoming jobs, provided that the queue length of that server is not greater than a predetermined threshold value K. Upon determining the server order based on the underlying utility, it is found that Priority-K experiences a regret of $O(log\ T)$ which is much better than the static policies.

Next, the Upper Confidence Priority-K policy is explained for the case of unknown utility, which combines the utility ordering obtained after some feedback from the servers to the Priority-K policy. Under the Upper-Confidence Priority-K policy, we achieve a desirable trade-off between exploration and exploitation, leading to a regret value of $O(log^3\ T)$.

Ultimately, the findings are expanded to a broader context in which the jobs originate from several job dispatchers, forming a general bipartite graph alongside parallel servers and it is concluded that the optimal solution for the discussed problem can be achieved in logarithmic time. The results have also been proved by simulation and the findings of the simulation have been attached in the report.

Chapter 3

System Model and Problem Formulation

A. System Model

Let us now examine a group of M parallel servers linked with one job dispatcher operating in a discrete-time system. For each time interval t, let us define:

$a(t)$ as the number of jobs arriving at the job dispatcher at time t, which is an independent random variable

$\lambda = E[a(t)]$, as arrival rate, i.e., the expected value of $a(t)$

Each job received at the dispatcher is then routed to a server where they are stored in a queue corresponding to each server.

For any generic server s_m, let us define:

$c_m(t)$, as the offered service at time t, which is also an independent random variable

$\mu_m = E[c_m(t)]$**,** as service rate i.e., the maximum count of jobs that s_m finished at time t.

After finishing a job j at server s_m, utility v_m is gained and we observe a noisy utility v_m^j, which is given by $v_m^j = v_m + \epsilon_j$, where the observation noise is represented as ϵ_j, as a sub-Gaussian independent random variable with $E[\epsilon_j] = 0.$,

We can notice that only v_m^j of each job j can be observed and not the underlying utility v_m, as the value for it is unknown.

Now let us assume that the offered service rates and realized arrivals, or $a(t)$'s and $c_m(t)$'s, are almost certainly all bounded on the upper side by any constant C. To not consider any superfluous nuances in the analysis and description of our shown results, we also assume that $a(t) \geq 1, c_m(t) \geq 1.$

Lastly, let $Q_m(t)$ represent s_m's queue length at time t, and $a_m(t)$ represents the total number of jobs the job dispatcher sent to server s_m at that time. We express the evolution of queue length for a server s_m as:

$Q_m(t+1) \coloneqq [Q_m(t) + a_m(t) - c_m(t)]^+$, and $[.]+ = \max\{., 0\}$.

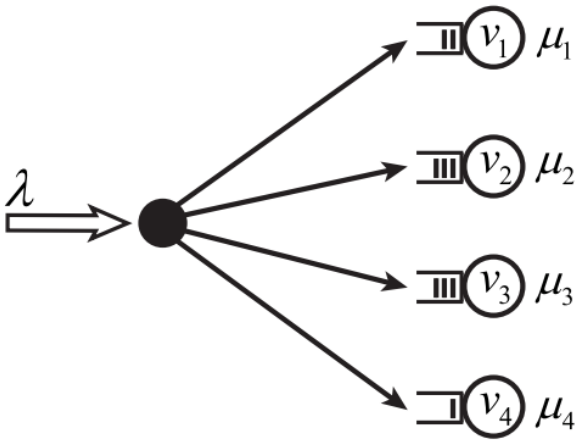

Figure 4: Illustration of the routing problem

*[**Source:** X. Fu and E. Modiano, "Optimal Routing to Parallel Servers With Unknown Utilities—Multi-Armed Bandit With Queues," in IEEE/ACM Transactions on Networking, vol. 31, no. 5, pp. 1997-2012]*

B. Problem Formulation

Now let us examine the issue of creating a strategy for routing that maximizes utility. Our objective is to create a policy that routes, or assigns each received job at the dispatcher to a server, in a way to maximizes the expected value of the utility at the end of the time period T.

For a generic policy π in consideration, let us first define its expected utility in order to give the problem some concreteness. Let ω be a trial path following the policy π. Assuming that server s_m completes $C_m^{\pi}(\omega, T)$ jobs, under the policy π, on the trial path ω, by the ending of the time period T.

Thus $\sum_{m=1}^{M} v_m C_m^{\pi}(\omega, T)$ is the utility gained under the policy π on the trial path ω.

Now, the expected value of the utility of policy π over the complete time period T can be defined as:

$$U_T(\pi) = E\left[\sum_{m=1}^{M} v_m C_m^{\pi}(\omega, T)\right]$$

Now, the value of regret of the considered policy π can be defined as:

$$R_T(\pi) = sup_{\pi^* \in \prod^*} U_T(\pi^*) - U_T(\pi)$$

where, $\prod^*$ is the set of all the policies and $sup_{\pi^* \in \prod^*} U_T(\pi^*)$ represents the supremum of all policies

The regret value of any policy π represents the gap between the supremum over all policies in $\prod^*$ and the expected value of the utility of path π.

1. **Assumptions on the Unknown Statistics:**
   a. No two considered servers have the same underlying utility, i.e., $\forall m, m', v_m \neq v_{m'}$. Hence, servers are now ordered with respect to decreasing underlying utility as $v_1 > v_2 > ... > v_M$.
   b. We define L as the critical number of servers complying with the given conditions: $\sum_{m=1}^{L} \mu_m < \lambda < \sum_{m=1}^{L+1} \mu_m$, i.e., for the first L servers, their total service rate is strictly less than the arrival rate, and for the first L + 1 servers, their service rate is strictly greater than the arrival rate. This again highlights the need to decrease the idle time of top L servers to achieve lower regret as pointed out in the previous sections.

2. **Static Policies:**

P: $\max_{\{x_m\}} \sum_{m=1}^{M} v_m x_m$ **s.t.** $\sum_{m=1}^{M} x_m = \lambda, 0 \leq x_m \leq \mu_m$

P represents a simplified form of the discussed optimal routing problem, and the optimization variables $\{x_m\}$ signify the steady-state rate of job assignments to each of the servers. Given the unique structure of this problem, the optimal solution $\{x_m^*\}$ is straightforward: $x_m^* = \mu_m$ for servers 1 to L, $x_{L+1}^* = \lambda - \sum_{m=1}^{L} \mu_m$, and $x_m^* = 0$ for servers $L + 2$ to M. Hence, an effective policy for this problem should aim to maximize utilization of the first L servers, allocate any remaining tasks to server s_{L+1}, and minimize usage of servers s_{L+2} to s_M.

The mathematical results from reference [1] suggest that:

Static policies can be shown to be sub-optimal for the case of optimal routing problem and this argument:

   a. differentiates the optimal routing problem from the static one
   b. highlights the need to consider the supremum value over all possible policies and not of only the best one to define the regret.
   c. rejects the existing methods to solve the unknown utility optimization problems which focuses on converging the solution with the optimal solution for the case of the static one, as the proposed new technique is guaranteed to achieve a better performance than these policies

Chapter 4

Routing Policies

A. The case of Known Utility Ordering

Let us consider the scenario in which we know the underlying utilities' ordering. We know from the previous discussion that we need to decrease the top L servers' idle time in order to achieve low regret. Since the utilities are well-known, it makes sense to route incoming jobs through the top L servers first.

This concept, however, is impractical in real scenarios because we lack prior knowledge about the critical count of the servers (L) and the arrival and service rates. But here we first discuss a policy assuming that we know the utility ordering of the servers and establish that we can achieve logarithmic regret for this setting. Going forward, it is demonstrated that logarithmic regret can be achieved by applying the priority routing policy based on utility, without requiring knowledge of arrival and service rates or need-to-know L.

Let us discuss the policy in consideration assuming that we know the utility ordering of the servers. This policy is referred to as the Priority-K policy. The input for the considered policy is a single parameter K. It checks each server's queue length at each time slot in the decreasing order of utility, that is, from s_1 to s_M, and then updates its queue length by assigning a maximum number of possible tasks to these servers such that the queue length of none of the servers exceeds the value K. All incoming jobs are channeled to the final server, s_M if the queue length on all the servers equals K.

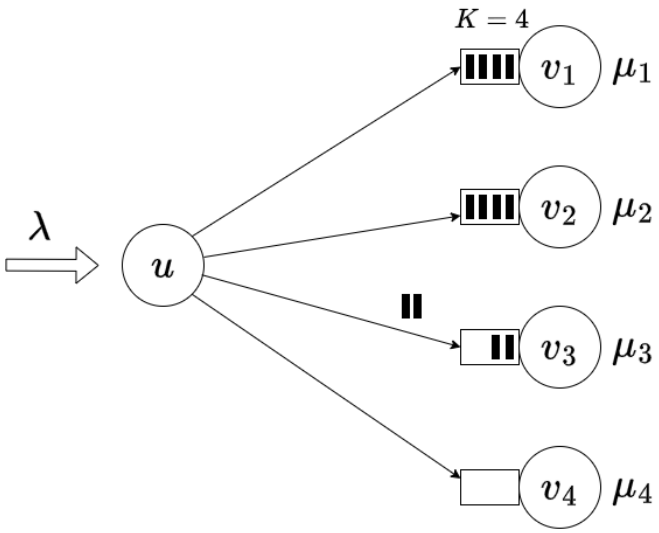

Figure 5: The Priority-K Policy

Priority-K is formally a routing policy based on the priority with a threshold value K for the length of the queue at each server, as its name implies. For determining the priority, we consider the underlying utilities associated with the servers.

This method helps to significantly reduce the idleness of high underlying utility servers. It is to be noted that if we don't consider the service capability for priority-based policy, then it can cause severe backlogs at the queues of servers at the finishing of the time interval T. If we choose a very small K, then we again have more idleness of the high utility servers leading to suboptimal solutions. If the K chosen is very large, then we face overloading of the high utility servers, again resulting in a suboptimal solution.

Hence, it is very necessary to choose the appropriate K so that we can avoid server overloading as well as high utility servers' idleness.

```
def priority_k(k, jobs, Ordering, servers):
    M = len(servers)
    for i in range(M):
        if jobs <= 0:
            break

        m = Ordering[i] - 1
        if servers[m].queue_length < k:
            available = k - servers[m].queue_length
            if available < jobs:
                servers[m].queue_length = k
                jobs -= available
            elif available >= jobs:
                servers[m].queue_length += jobs
                jobs = 0

    if jobs > 0:
        servers[M - 1].queue_length += jobs
        jobs = 0
```

Figure 6: Algorithm: The Priority-K Policy

B. The case of Unknown Utility Ordering

Let us now go back to the initial configuration of this optimal routing problem, in which it is unknown what the underlying utilities, $v_1, v_2, ..., v_M$ are arranged in. Now, we study a routing strategy with no prior knowledge of utility. To begin, we still consider servers to be arranged so that $v_1 > v_2 > ... > v_M$ for analytical purposes.

First, let us outline the approach to the optimal routing problem in the case of unknown utilities. Let us broaden the approach of the Priority-K policy by first carrying out the exploration process only, wherein $\widehat{N}$ jobs are sent to each server initially. Then we formulate the ordering of the servers based on the results obtained from exploration and then the Priority-K is carried out for the remainder of the time horizon. Let us call this approach the Upper-Confidence-Priority-K policy (UCPK).

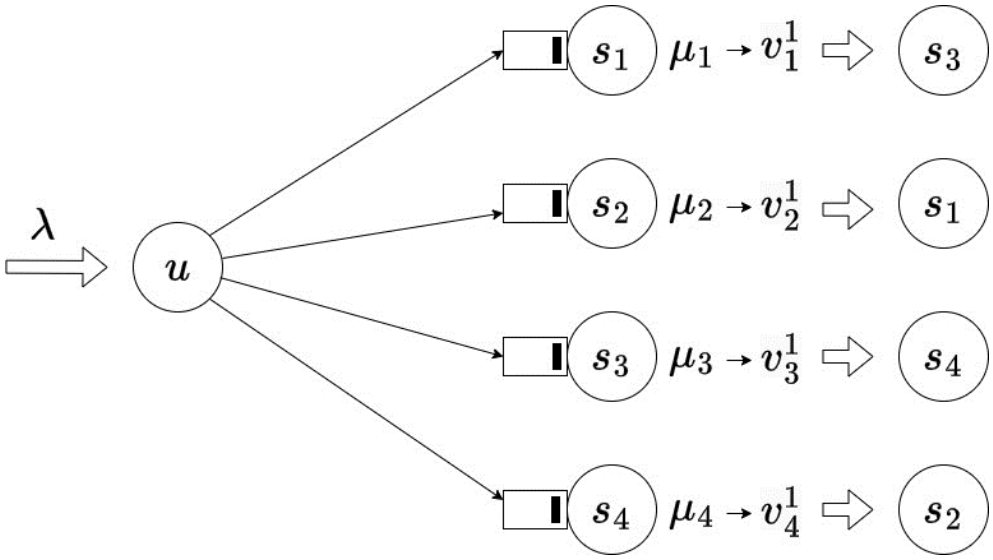

Figure 7: Reordering the servers based on observed utility

UCPK can be divided into two modes of working. Starting with an arbitrary initial ordering O, we keep following Priority-K policy if an update of upper-confidence bounds does not cause any change in O. However, after a change in order is seen, we again go through an exploration phase where we send the next M arrivals to all the servers in a round-robin manner and keep updating our ordering based on the feedback.

So, this policy also works similarly to the Priority-K policy. The only difference here is that we have to order the servers based on the utility feedback we receive from them.

```
def upper_confidence_priority_k(k, Jobs, Ordering_UCPK, servers, T, Service_rates):
    Ordering_copy = [0] * len(Ordering_UCPK)
    M = len(servers)

    t = 0
    while t<T:
        if Ordering_UCPK == Ordering_copy:
            jobs = Jobs[t]
            priority_k(k, jobs, Ordering_UCPK, servers)
            for m in range(M):
                service_rate = random.choice(Service_rates)
                servers[m].process_files(service_rate)
            Ordering_UCPK = get_new_Ordering(servers)
            t = t + 1
        else:
            Ordering_copy = Ordering_UCPK
            m = 0
            jobs = Jobs[t]

            for m in range(M):
                if(m == M-1):
                    servers[m].queue_length += jobs % M
                else:
                    servers[m].queue_length += jobs // M
                service_rate = random.choice(Service_rates)
                servers[m].process_files(service_rate)

            t = t + 1
            Ordering_UCPK = get_new_Ordering(servers)
```

Figure 8: Algorithm: The Upper-Confidence Priority-K Policy

Chapter 5

Simulation Setup

Here we present the simulation results for a network with 20 servers. The servers have been assigned underlying utilities from 1 to 20. The job dispatcher has an arrival rate of 100, and the servers have a random service rate from the set of numbers {10, 8, 6}.

The comparison of the results for three discussed policies has been shown below:

a. Static policy (Static),
b. Priority-K (PK) and
c. Upper-Confidence Priority-K (UCPK)

The simulation also shows the logarithmic dependence for our case and thus it supports the proof for the optimal routing problem as proposed.

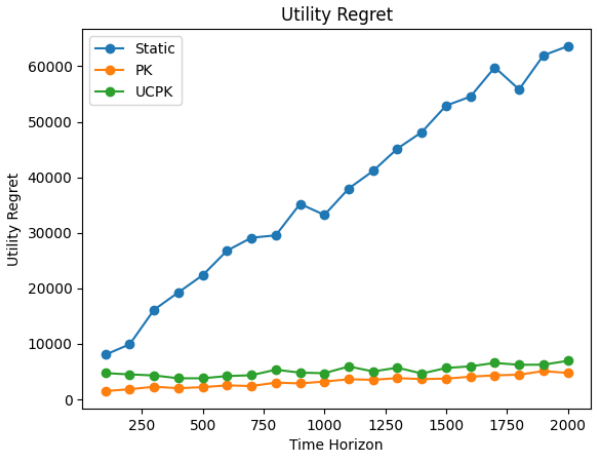

Figure 9: Utility Regret of Different Policies

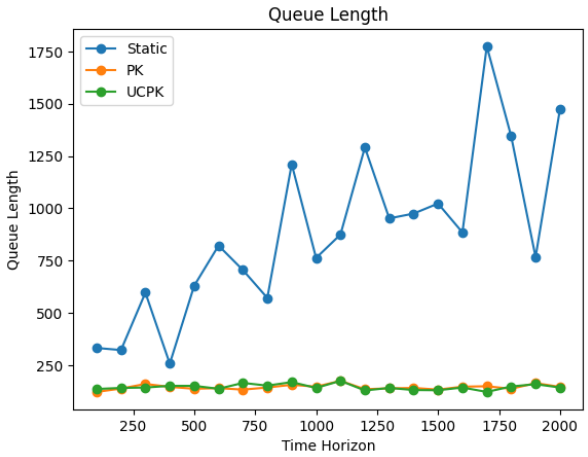

Figure 10: Queue Length of Different Policies

Chapter 6

Extension to General Bipartite Network

A. Bipartite Network Model

We now study a general system which has N job dispatchers and M parallel servers, where each dispatcher is connected to some number of servers. This connection network can be treated as a general bipartite graph G.

We identify the set of servers that a dispatcher (u_n) is connected to by S_{u_n}. The set of job dispatchers connected to a server s_m is indicated by N_{s_m}.

Let $a_n(t)$ jobs arrive at job dispatcher u_n at a time interval t with $E[a_n(t)] = \lambda_n$, where $a_n(t)$'s are independent and identically distributed random variables and the values of λ_n, i.e., arrival rates are not known.

Let the dispatcher u_n sends server s_m, $a_{mn}(t)$ jobs at a time interval t.

Now, the queue length of a server s_m can be given by:

$$Q_m(t+1) \coloneqq \left[Q_m(t) + \sum_{n=1}^{N} a_{mn}(t) - c_m(t) \right]^+ , \text{and } [.]+ = \max\{., 0\}.$$

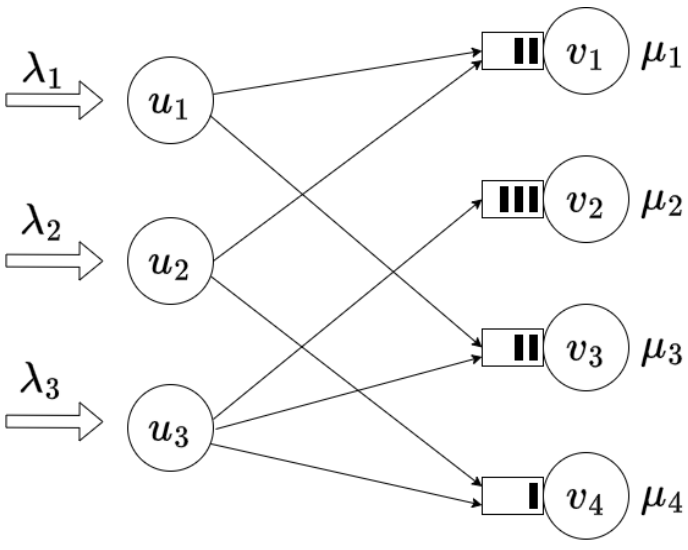

Figure 11: Bipartite Network Model

Creating a policy that decides how to route incoming jobs from each dispatcher is our main objective. This will be known as the generalized optimal routing problem from now on. Rather than trying to find a policy that minimizes logarithmic regret for every case, we investigate whether, when, and how our prior work and strategies can be expanded to general cases.

B. Generalized Routing Policy

Let us extend the idea of the Priority-K policy discussed earlier to the case of a general bipartite network problem and call it a Generalized Priority-K policy. Every job dispatcher (u_n) adheres to a localized style of the previous policy, examining the set of servers present in S_{u_n} according to the underlying utilities' decreasing order. Incoming jobs are then routed to the first such server which has a length of queue no more than K. If all servers connected to a dispatcher have length K, then we assign all the remaining tasks for that dispatcher to the last server according to their ordering.

This means that we apply Priority-K policy on each dispatcher individually considering only those servers which are connected to that particular dispatcher. It is evident that for this general problem also, it is possible to achieve logarithmic regret using this policy.

To tackle the problem of unknown utilities, we start by sending a limited number of jobs to each server, and then based on the response of utility gain received, we can reorder the servers according to the decreasing order of utilities. Whenever there is a change in the ordering, we can check on the servers' utilities using this trick. Otherwise, we just continue to dispatch jobs based on the generalized Priority-K policy.

Chapter 7

Conclusion

In this project, we first studied a simple server optimization problem. We demonstrated the impact of different approaches on system performance, particularly in minimizing processing time and maximizing server utilization. The results underscore the importance of prioritizing high-capacity servers and dynamically adjusting task assignments to minimize idle time and enhance overall system efficiency.

Then we examined the optimal routing problem, where a set of parallel servers are connected to a single job dispatcher. We leveraged the idea to decrease the idle time of best servers to design a Priority-K policy in case of the known utility of the servers. Then, we characterized the Upper Confidence Priority-K policy for unknown utility cases by managing the trade-off between exploration and exploitation. The suggested policies work on minimizing the regret value and the unfinished jobs represented as servers' queue length by the finishing of time interval T. We created routing strategies with regret value that follows a logarithmic increase over the given time interval, T.

The proposed policies have been simulated and the results have been included in the report. The results can be seen to achieve logarithmically growing regret for the single job dispatcher problem. Further, we have discussed the generalized version which considers multiple job dispatchers connected to multiple servers forming a bipartite graph. The policies are then extended to the generalized version.

Bibliography

[1] X. Fu and E. Modiano, "Optimal Routing to Parallel Servers With Unknown Utilities—Multi-Armed Bandit With Queues," in IEEE/ACM Transactions on Networking, vol. 31, no. 5, pp. 1997-2012

[2] S. Stidham, “Optimal control of admission to a queueing system,” IEEE Trans. Autom. Control, vol. AC-30, no. 8, pp. 705–713, Aug. 1985.

[3] V. Gupta, M. H. Balter, K. Sigman, and W. Whitt, “Analysis of join-theshortest-queue routing for web server farms,” Perform. Eval., vol. 64, nos. 9–12, pp. 1062–1081, Oct. 2007.

[4] Z. Rosberg and A. M. Makowski, “Optimal routing to parallel heterogeneous servers-small arrival rates,” IEEE Trans. Autom. Control, vol. 35, no. 7, pp. 789–796, Jul. 1990.

[5] X. Fu and E. Modiano, “Elastic job scheduling with unknown utility functions,” Perform. Eval., vol. 152, Dec. 2021, Art. no. 102229.

[6] A. L. Stolyar, “Optimal routing in output-queued flexible server systems,” Probab. Eng. Informational Sci., vol. 19, no. 2, pp. 141–189, Apr. 2005.