DISTRIBUTED JOIN-THE-IDLE-QUEUE: NEW ALGORITHM AND ANALYSIS

by

Yonghui Lu

B. Electrical Engineering and Automation, Hohai University, 2016

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF APPLIED SCIENCE

in

THE COLLEGE OF GRADUATE STUDIES
(Electrical Engineering)

The University of British Columbia
(Okanagan)

December 2018

The following individuals certify that they have read, and recommend to the College of Graduate Studies for acceptance, the thesis entitled:

DISTRIBUTED JOIN-THE-IDLE-QUEUE: NEW ALGORITHM AND ANALYSIS

submitted by Yonghui Lu in partial fulfillment of the requirements of the degree of Master of Applied Science.

Chen Feng, School of Engineering
Supervisor
Julian Cheng, School of Engineering
Supervisory Committee Member
Zheng Liu, School of Engineering
Supervisory Committee Member
Eric Li, Faculty of Management
University Examiner

Abstract

Load balancing plays an important role in large-scale cloud systems. Power-of-d-choice (Pod) and Join-the-idle-queue (JIQ) are two popular load balancing strategies. In this thesis, two new load balancing algorithms are proposed that combine Pod and JIQ, leading to a better performance-cost trade-off. This thesis shows analysis of these two new algorithms by using mean-field approximation and evaluates their performance through extensive simulations and system implementation on Amazon EC2.

Lay Summary

In this thesis, we proposed two novel algorithms to improve cloud computing performance. Based on the proposed algorithms, we derived performance of algorithms numerically. Also, we did simulations and implementations. In conclusion, we found one of the mentioned algorithms in this thesis outperforms the others based on cost-performance trade-off.

Preface

This thesis is based on the research work conducted under the supervision of Dr. Chen Feng in the School of Engineering at The University of British Columbia, Okanagan Campus. The main content in this thesis is based on our submitted journal paper, under the supervision of Dr. Chen Feng and Dr. Julian Cheng. Chapter 3, Chapter 4 and Chapter 5 have been submitted to the IEEE/ACM Transactions on Networking.

Table of Contents

List of Figures

Acknowledgments

I was extremely excited when finishing this thesis. There are many people to whom I have to say “thank you”. First, I have to say “thank you” to my supervisor Dr. Chen Feng.

Second, I have to thank my families. Thank my girlfriend Liyan Liu for her encouragement and accompany when I was trapped into depression. The days were dark and struggling at that time, but she was always my bright sun beside me which made me fearless to face challenges. Also, I have to thank my parents for their great support on both physical and mental ways. Although they are far away from me, their encouragements were never stopped. I can feel their miss and wishes over the Pacific Ocean which is the great motivation to me.

In the end, I should say thank to my friends: Chunpu Wang, Renming Qi, Junchi Bing, Shuo Liu, Fang Shi, Huan Liu, Junyuan Leng, Jianyu Niu, Jinfeng Tong, Yuwei Guo, Yiqun Duan and Andy Zhang. They colored my life and let me believe in “A friend in need, a friend indeed”.

Chapter 1

Introduction

1.1 Cloud Computing and Motivation

With the rapid growth of cloud computing market these years, cloud computing attracts much attention from both industry and academia. The concept of cloud computing can be traced back to 1950s when the implementation of static clients to access mainframe computers appeared. After decades of development, by now, cloud computing plays an indispensable role in the industry by supplying online application services. The meaning of cloud computing refers to a service provided by a remote data center which not only offers application through the Internet but also hardware and software of the servers in the data center, and "cloud" refers to servers in a remote data center which provides hardware and software. Due to the existence of cloud computing, computing finally becomes a kind of utility, just like water and electricity in our daily life.

The development of cloud computing has lasted for years, and a vast amount of manufacturers are now developing different cloud computing services. Cloud computing has various forms, and simple cloud computing service can be found everywhere in people's daily online applications, such as Google Chrome, Google Doc and so on. At present, the main kinds of cloud computing service forms are SaaS (Software as a Service), PaaS (Platform as a Service) and IaaS (Infrastructure as a Service). Nevertheless, most of the time, when using applications through cloud computing service, it referred to SaaS service. With years of developing, there are

many companies offering cloud computing service now. The most popular cloud computing platforms, at present, are Amazon Elastic Compute Cloud (Amazon EC2) [1], Google Cloud Platform [2] and Microsoft Azure [3]. Analogously, all of these platforms supply pay-to-use computing service which is accessible "anytime and anywhere". Through this way, both service providers and users can fully leverage the preponderance of SaaS. From provider's side, they are satisfied with the centralized hardware management and convenient software setup; to users, they no longer need to worry about real servers maintenance and setup from remotely controlling cloud server where data can be stored safely. Moreover, regarding small companies which cannot afford local servers, cloud computing provides relatively cheaper service which is cost-effective. For instance, running 100 servers for one hour, unexpectedly, costs less than running one server for 100 hours, which is both financial and time efficiency.

Although cloud computing has many advantages, it is still facing imperfections and challenges, such as data privacy, data security, user habits, network latency, etc. In detail, concerning data privacy: how to ensure that private data stored in the cloud away from illegal use requires not only technical improvements but also needs to improve the law; concerning data security: Some data is the commercial secrets and the security of the data related to the survival and development; concerning user habits: how to let users adapt to the network application is a long-term challenge; concerning network latency: cloud computing relies on network and network latency badly influences the performance of service.

Cloud computing needs to supply an efficient and flexible approach to upload and download data and manage vast amounts of data transmissions. In terms of network latency, there are requirements of specific techniques and algorithms to ensure a steady and high-powered performance for users. Thus, it becomes significant to explore approaches in cloud computing which can help reduce the latency and improve the performance. One non-neglectful issue among the approaches is load balancing. Therefore, in this thesis, my work is concentrated on load balancing.

1.2 Load Balancing Problems

Nowadays, load balancing is applied widely in large-scale cloud computing systems. For example, Amazon EC2 offers a service called Amazon EC2 Container Service (Amazon ECS) [1], which provides flexible load balancing methods to assign incoming computing tasks to less-loaded servers. Load balancing is a fundamental cloud computing problem. Load balancing aims to dispatch tasks among a set of servers as evenly as possible. For instance, assume that a number of tasks should be dispatched to amount of servers which have the same computing, and each server can only process one task at a time. In order to achieve the minimum final processing time, the approach is to ensure leveraging the parallel machines efficiently, which is the purpose of load balancing. Ideally, a load balancing algorithm should minimize the *task response time* (i.e., the time between the arrival and the completion of a computing task). Join-the-Shortest-Queue (JSQ) is an idealized algorithm to achieve short response time. It tracks the queue lengths of all the servers in the system and selects the least-loaded server when a computing task arrives. Although JSQ is proven to achieve the shortest response time in heavy-traffic limit [4], it doesn't scale well because of the need of tracking the queue lengths of all the servers, which is quite resource consuming for large-scale systems.

To alleviate this issue, Power-of-d-choices (Pod) has been proposed as an "approximation" of JSQ. Rather than tracking all the servers, Pod only probes d servers (uniformly at random) upon a task arrival and then selects the least-loaded one among the d servers for the new task. Surprisingly, this simple strategy achieves low average response time even when d is as small as 2 [5]. For this reason, Pod has been widely used in various cloud computing systems (see, e.g., Sparrow [6]). Despite its excellent performance in terms of overhead and average response time, the tail response time of Pod still remains high for large-scale systems [7]. Furthermore, the probing operation in Pod incurs additional delay compared with JSQ.

Join-the-Idle-Queue (JIQ) has been recently proposed as a promising new approximation of JSQ [8]. Rather than tracking all the servers, JIQ only tracks idle servers in the system. To do so, JIQ employs a number of schedulers operating in a distributed manner. Specifically, each scheduler maintains an I-queue that stores

a list of idle servers. Whenever a server becomes idle, it reports to one of these schedulers chosen at random. In this way, each scheduler tracks a subset of idle servers using its I-queue. Once a new computing task arrives, it will contact one of the schedulers and will be assigned to an idle server if the scheduler's I-queue is non-empty. Otherwise, the task will be assigned to a random server chosen by the scheduler. Compared with JSQ, each scheduler in JIQ only maintains local information. As such, JIQ is scalable to large systems if there are sufficient schedulers. Compared with Pod, each scheduler in JIQ simply assigns a new task to an idle server without any probing operation as long as its I-queue is non-empty. Thus, JIQ works very well when the majority of I-queues are non-empty. However, the performance of JIQ deteriorates sharply if the majority of I-queues are empty since a new task is very likely to be assigned to a server chosen at random.

Can we combine Pod and JIQ to achieve better performance? Mitzenmacher has proposed one such combination in 2016 [9], where an idle server applies Pod to find the shortest I-queue (among the randomly selected d I-queues) to report. Another combination has been proposed in our previous work [10], where a scheduler with an empty I-queue randomly probes d servers and selects the least-loaded one. Although these two combinations indeed achieve better performance concerning the task response time, both of them incur extra communication cost compared with the original JIQ. In other words, there is a tradeoff between the delay performance and communication cost for variants of JIQ.

Can we achieve a better performance-cost tradeoff? In this thesis, we propose two new variants of JIQ inspired by the previous work [9, 10]. Our first variant is called JIQ-NE, which aims to assign a new task to a scheduler with *non-empty* I-queue by probing at most d schedulers in a sequential manner. Our second variant is called JIQ-E, which aims to assign an idle server to a scheduler with *empty* I-queue by probing at most d schedulers in a sequential manner. Furthermore, we apply the mean-field approximation to analyze the delay performance of these two new variants, deriving several semi-closed-form expressions. Our expressions suggest that JIQ-NE achieves a better performance-cost tradeoff than existing JIQ variants. Finally, we conduct extensive simulations to verify our theoretical analysis.

1.3 Related Work

The Po*d* algorithm is one of the most well-known load balancing strategies [5] to approximate JSQ. It is based on a supermarket queue model which can be analyzed via mean-field analysis. Nowadays. Po*d* algorithm and its variants are extensively leveraged in cloud computing systems. As one such variant, the authors in [6] proposed a stateless distributed scheduler by using batch sampling. In particular, batch sampling can dramatically reduce tail response time than Po*d*. In [7], the authors successfully made the number d close to 1 while keeping the task response time low and upper-bounded in batch arrival scenario. A recent work [11] shows a hybrid algorithm combining Po*d* with an extra "helper" in a system. With this "helper", the hybrid algorithm can effectively restrict the size of the maximum servers.

Another well-known load balancing strategy is JIQ algorithm [8]. Based on the simple idea of using idle server information, A. Stolyar [12, 13] analyzed the performance of centralized JIQ (the number of I-queue is fixed while the number of servers tends to infinity) through mean-field analysis. The main result is that the probability of centralized JIQ to route each newly arrived task to an idle server goes to 1 in the large-systems limit. Meanwhile, Mitzenmacher [9] introduced and studied distributed JIQ in which the ratio r is a fixed number (as in our system model). He derived differential equations to describe the whole system, which inspired both [10] and this thesis.

Except for Po*d* and JIQ, there are many other techniques of load balancing in cloud computing. The authors in [14] proposed a two-level task scheduling mechanism. They are the first who proposed mapping tasks to virtual machines to maintain load balancing. Through their method, it can help improve task response time and cloud computing environment. A two-phase scheduling algorithm [15] is proposed to achieve a better executing efficiency and keep maintaining system load. It combines two existing load balancing algorithms to help system in an efficient state of resources utilization. To solve the high migration cost and load imbalance problems in cloud computing, a scheduling strategy was designed in [16] on load balancing of virtual machine resources. Through leveraging a genetic algorithm, this strategy improves system load balancing and effectively reduces the

dynamic migration.

This thesis builds upon our previous work [10]. In [10], we not only obtained semi-closed form expressions of the stationary queue-length distribution but also proposed the JIQ-Pod algorithm, which outperforms both JIQ and Pod. In this thesis, we take a further step to analyze the pros and cons of different combinations of JIQ and Pod in terms of the task response time and communication cost.

1.4 Organization

This thesis contains six chapters, and the rest chapters are organized as follows.

In Chapter 2, we present the system model. Then, we introduce JIQ, JIQ-Pod and two new algorithms (namely, JIQ-NE and JIQ-E) respectively.

In Chapter 3, we first establish a Markov model for our JIQ algorithms, based on which we characterize the delay performance and communication cost of various algorithms. We then compare their performances analytically.

In Chapter 4, we conduct extensive simulations, focusing on the delay performance and communication cost. We also evaluate the empty I-queue rate and the influence of system parameters.

In Chapter 5, we implement the proposed JIQ algorithms using 56 Amazon EC2 instances, demonstrating their feasibility and practicality.

Finally, in Chapter 6, we conclude the entire thesis.

Chapter 2

Models of JIQ and its Variants

In this chapter, we define the models of all JIQ-based algorithms. In the beginning, we give the general model, and then we provide the particular strategies of all the algorithms in detail. In the end, we show additional discussions based on models.

2.1 General Model

Consider a system that has N servers and M schedulers (each maintaining an I-queue). We define the ratio $r \triangleq N/M$. Time is set to be continuous. In this system model, the following events may happen:

1. *Task arrivals*: The task arrival process is assumed to be a Poisson process with rate λN where $0 < \lambda < 1$ is a system parameter. When a new task arrives, it will select a scheduler according to a specific scheduler-selection strategy.

2. *Schedulers*: Each scheduler maintains an I-queue in order to store a list of idle servers. Upon a task arrival, a scheduler with non-empty I-queue will select an idle server uniformly at random from its I-queue and then remove this idle server from its I-queue. Otherwise, it will apply a specific server-selection strategy to choose a server for the new task.

3. *Servers*: Each server has a first-in first-out (FIFO) queue to store incoming tasks. The task processing time is assumed to be exponentially distributed

with mean 1, which is independent across tasks and across servers. When a server becomes idle, it will join an I-queue by applying a specific I-queue-joining strategy.

Once the three strategies (i.e., scheduler-selection, server-selection, and I-queue-joining strategies) are specified, we can obtain a particular version of the JIQ algorithm and we will present four such algorithms in the rest of this chapter. The general model is shown in Fig 2.1

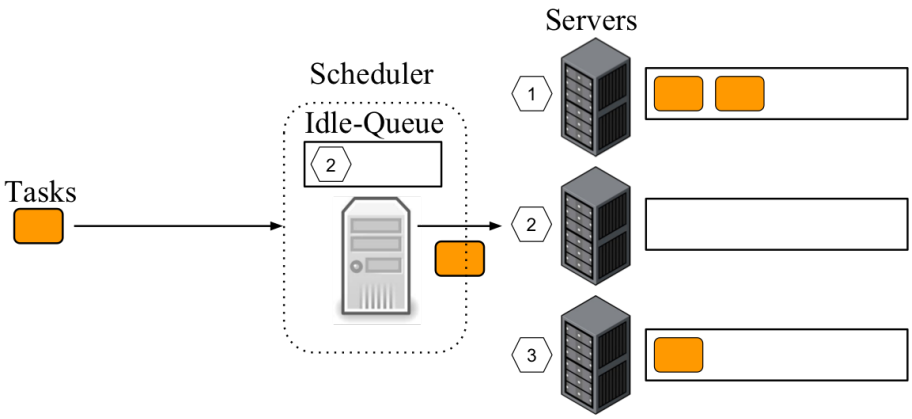


Figure 2.1: The operating process of the general model.

2.2 Join-The-Idle-Queue (JIQ)

The standard JIQ algorithm uses the following strategies:

1. *Scheduler-selection strategy*: When a new task arrives, it will select a scheduler uniformly at random from all the schedulers.

2. *Sever-selection strategy*: When a new task arrives, a scheduler with empty I-queue will select a server uniformly at random from all the servers.

3. *I-queue-joining strategy*: When a server becomes idle, it will select an I-queue uniformly at random from all the I-queues and then join the selected I-queue.

2.3 JIQ-Power-of-d-Choices (JIQ-Pod)

JIQ-Pod is a very recent algorithm proposed in our previous work [10] that aims to take the best of two worlds (i.e. JIQ and Pod). It uses the following two strategies:

1. *Scheduler-selection strategy*: The same as standard JIQ.
2. *Sever-selection strategy*: When a new task arrives, a scheduler with empty I-queue will randomly select d servers from all the servers, then choose the server with the shortest queue among the selected d servers.
3. *I-queue-joining strategy*: The same as standard JIQ.

Fig. 2.2 illustrates JIQ-Pod in which the empty I-queue selects server 2 and server 3 and then assigns the new task to server 2.

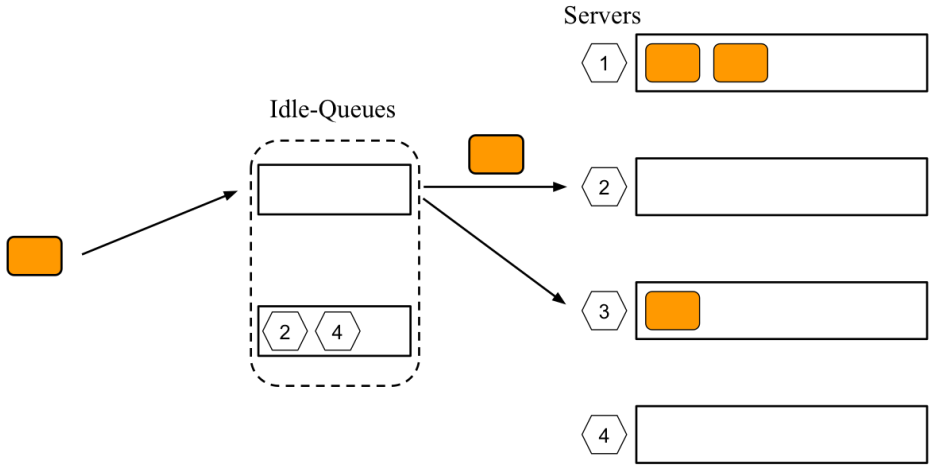


Figure 2.2: An illustration of JIQ-Pod

2.4 JIQ-Empty (JIQ-E)

JIQ-E allows a new idle server to find a scheduler with an empty I-queue through a reasonable amount of effort. It uses the following strategies:

1. *Scheduler-selection strategy*: The same as standard JIQ.

2. *Sever-selection strategy*: The same as standard JIQ.

3. *I-queue-joining strategy*: When a server becomes idle, it will probe at most d schedulers in a sequential manner until it finds a scheduler with an empty I-queue. Specifically, the server firstly probes $d-1$ schedulers uniformly at random. If the server finds a scheduler with an empty I-queue, it selects the scheduler immediately. Otherwise, it selects the scheduler (i.e., the dth scheduler) uniformly at random from all the schedulers without checking the status of its I-queue.

Fig. 2.3 illustrates JIQ-E in which new idle server 3 first probes the scheduler on the bottom and then it probes the scheduler on the top whose I-queue is empty.

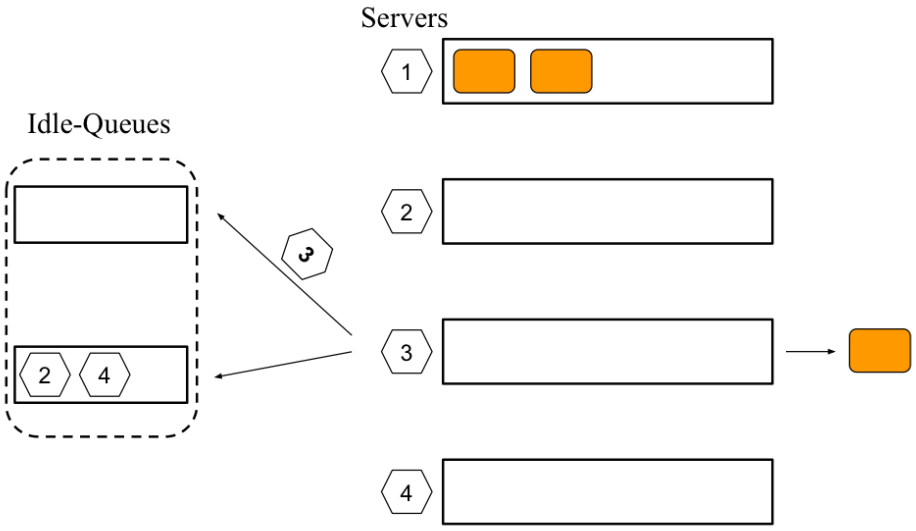


Figure 2.3: An illustration of JIQ-E

2.5 JIQ-Non-Empty (JIQ-NE)

JIQ-NE allows a new task to find a scheduler with a non-empty I-queue through a reasonable amount of effort. It uses the following strategies:

1. *Scheduler-selection strategy*: When a new task arrives, it will probe at most d schedulers in a sequential manner until it finds a scheduler with a non-empty I-queue. Specifically, the task first probes $d-1$ schedulers uniformly at random. If the task finds a scheduler with a non-empty I-queue, it selects the scheduler immediately. Otherwise, it selects the scheduler (i.e., the dth scheduler) uniformly at random from all the schedulers without checking the status of its I-queue.

2. *Sever-selection strategy*: The same as standard JIQ.

3. *I-queue-joining strategy*: The same as standard JIQ.

Fig. 2.4 illustrates JIQ-NE in which the new task first probes the scheduler on the top and then it probes the scheduler on the bottom which directs it to server 2.

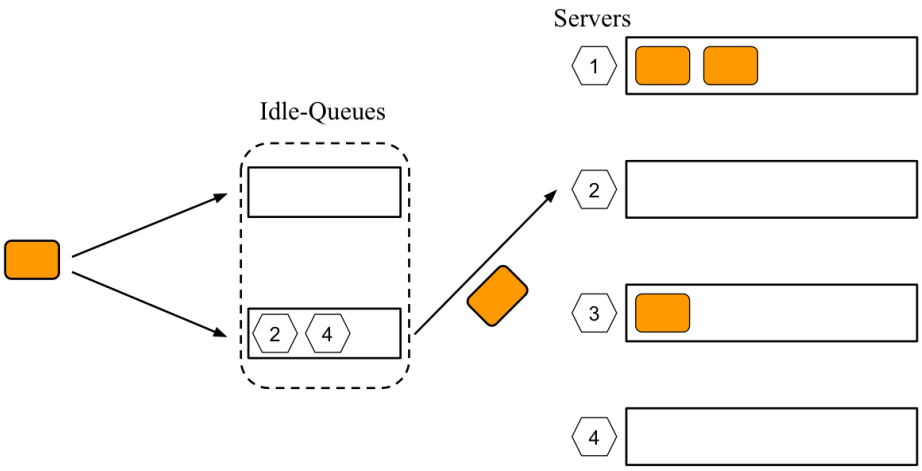


Figure 2.4: An illustration of JIQ-NE

2.6 Discussions

To summarize, there are three choices to make for any JIQ algorithm:

1. How does a new task select a scheduler?

2. How does a scheduler with an empty I-queue choose a server?

3. How does a new idle server join an I-queue?

The standard JIQ applies the "uniform-at-random" strategy to make all the choices, leading to simple implementation (without the need for tracking any status of the system) yet sub-optimal performance. In order to improve the performance, JIQ-NE makes a smarter choice to select a scheduler by probing at most d schedulers, JIQ-Pod makes a smarter choice to select a server that has the shortest queue among d servers, and JIQ-E makes a smarter choice to join an I-queue by probing at most d I-queues. Also, notice that JIQ-Pod probes d servers simultaneous whereas JIQ-NE and JIQ-E do the probing sequentially in order to reduce the communication cost.

Although these JIQ variants improve different aspects of the system, a *unified* theoretical framework will be developed based on the mean-field approximation to compare their performances in the next chapter.

Chapter 3

Analysis of JIQ-NE and JIQ-E Algorithms

In this chapter, the delay performance of new proposed two JIQ variants, namely JIQ-NE and JIQ-E, is studied, through mean-field approximation. First, a continuous-time Markov-chain model for the system evolution is established. Then, regarding JIQ-NE algorithm and JIQ-E algorithm, the detailed derivation of the stationary distribution of a single server is shown under the large-system limit, which allows me to characterize the delay performance such as the average task response time and the tail response time. Finally, the delay performance of our JIQ variants is compared with existing variants.

3.1 Markov Model

In order to build a Markov model, we first define the state of a single server. Specifically, we denote by $S_i^{M,N}(t)$ the *state* of the ith server at time t in a system of N servers and M I-queues, where $S_i^{M,N}(t)$ takes values in the set

$$\mathscr{S} = \{(0,1),(0,2),(0,3),\ldots,1,2,3,\ldots\}.$$

Here, the state $(0,j)$ means that the server is an idle server associated with an I-queue of length j, and the state j means that the queue length of the server is j. For instance, the 1st server in Fig. 2.2 is in state 2 and the 2nd server in Fig. 2.2 is in

state $(0,2)$.

Next, we define the state of the system at time t as

$$\mathbf{S}^{M,N}(t) \triangleq \left\{S_1^{M,N}(t), S_2^{M,N}(t), \cdots, S_N^{M,N}(t)\right\}.$$

Clearly, $\mathbf{S}^{M,N}(t)$ forms a continuous-time Markov chain. Moreover, one can show that $\mathbf{S}^{M,N}(t)$ is irreducible, nonexplosive, and positive recurrent. Hence, $\mathbf{S}^{M,N}(t)$ has a unique stationary distribution.

However, it is often difficult to find the unique stationary distribution for $\mathbf{S}^{M,N}(t)$ due to the "curse of dimensionality." As such, we apply the mean-field approximation to "approximate" the unique stationary distribution. As we will see later, such an approximation tends to be *exact* as M and N tend to infinity with fixed ratio $r = N/M$.

Note that the state of each server is identically distributed due to the symmetry. Now, we assume that the state of each server is also independently distributed when the system reaches the steady state. This i.i.d. assumption, which is an essential element of mean-field approximation, allows us to focus on the state evolution of a *single* server in the system, thereby avoiding the curse of dimensionality.

Recall that the possible states of a server are from the set

$$\mathscr{S} = \{(0,1),(0,2),\ldots,1,2,\ldots\}.$$

Let

$$\mathbf{q} \triangleq \left\{q_{(0,1)}, q_{(0,2)}, \cdots, q_1, q_2, \cdots\right\}$$

denote the stationary distribution of a single server. By the Strong Law of Large Number, we can then use $q_{(0,j)}$ as the fraction of servers with state $(0,j)$ in the large-system limit. Similarly, q_j as the fraction of servers with state j in the large-system limit.

For convenience, we denote by p_j the fraction of I-queues of size j. Let

$$\mathbf{p} \triangleq \{p_0, p_1, p_2, \cdots\}.$$

By the Strong Law of Large Number, we have

$$p_j = \frac{rq_{(0,j)}}{j}$$

for $j \geq 1$ in the large-system limit. Here, ratio $r = M/N$ and note that p_0 is the fraction of empty I-queues, which will play an important role later. We are now ready to solve for $\mathbf{p}$ and $\mathbf{q}$ by using the mean-field approximation.

3.2 Stationary Distribution Under JIQ-NE

First, we derive the transition rates for the state evolution of a single server as follows:

- $r_{i,i-1} = 1$, for $i \geq 2$.
  The processing rate of a task in a server is exponentially distributed with mean 1.

- $r_{i-1,i} = \lambda p_0^d$, for $i \geq 2$.
  The task arrival rate is λN, the probability of continuously joining d empty I-queues is p_0^d, and the probability of selecting the target server over all servers is $\frac{1}{N}$. Hence, the transition rate is $\lambda N \cdot p_0^d \cdot \frac{1}{N}$.

- $r_{1,(0,j)} = p_{j-1}$.
  The processing rate of a task in a server is exponentially distributed with mean 1, and the probability of joining an I-queue of size $j-1$ is p_{j-1}.

- $r_{(0,j),1} = \lambda \left[p_0^d + \frac{r}{j} \sum_{l=0}^{d-1} p_0^l \right]$.
  The task arrival rate is λN and two events, resulting from a new arrival task, leads to this state change. The first event is that a new task is continuously routed to d empty I-queues and then directed to the target server. The probability of this event is $p_0^d \cdot \frac{1}{N}$. The second event is that a new task, after routed to an empty I-queue or empty I-queues continuously, is finally routed to the I-queue associated with the target server and then directed to the target server. The probability of this event is $\sum_{l=0}^{d-1} p_0^l \cdot \frac{1}{M} \cdot \frac{1}{j}$. Hence, the transition

rate is $\lambda N\left(p_0^d\cdot\frac{1}{N}+\frac{r}{N}\cdot\frac{1}{j}\cdot\sum_{l=0}^{d-1}p_0^l\right)$.

- $r_{(0,j-1),(0,j)}=rq_1$, for $j\geq 2$.
  The generating rate of idle servers is q_1N, and the probability of selecting the I-queue associated with the target server is $\frac{1}{M}$.

- $r_{(0,j),(0,j-1)}=\lambda(j-1)\left[p_0^d+\frac{r}{j}\sum_{l=0}^{d-1}p_0^l\right]$, for $j\geq 2$.
  The task arrival rate is λN. There are two events resulting in this state change. The first event is that a new task, after routed to d empty I-queues, is finally directed to an idle server having the same I-queue as the target server. The probability of this event is $p_0^d\cdot\frac{j-1}{N}$. The second event is that the new task, after routed to an empty I-queue or empty I-queues, is finally routed to the I-queue associated with the target server and then directed to another idle server. The probability of this event is $\sum_{l=0}^{d-1}p_0^l\cdot\frac{1}{M}\cdot\frac{j-1}{j}$. Hence, the transition rate is $\lambda N\left(p_0^d\cdot\frac{j-1}{N}+\frac{r}{N}\cdot\frac{j-1}{j}\cdot\sum_{l=0}^{d-1}p_0^l\right)$.

Based on the above transition rates, one can easily write down the local balance equations as

$$\begin{cases} q_ir_{i,i-1}=q_{i-1}r_{i-1,i}, \text{ for } i\geq 2,\\ q_{(0,j)}r_{(0,j),1}=q_1r_{1,(0,j)}, \text{ for } j\geq 1,\\ q_{(0,j)}r_{(0,j),(0,j-1)}=q_{(0,j-1)}r_{(0,j-1),(0,j)}, \text{ for } j\geq 2. \end{cases} \tag{3.1}$$

By solving the local balance equations, we have the following theorem:

Theorem 1. *The stationary distribution of the state of a single server under JIQ-NE in the large-system limit is*

$$\begin{cases} q_{(0,i)}=\frac{r^{i-1}(1-\lambda p_0^d)^i}{\prod_{j=1}^{i}(r\sum_{l=0}^{d-1}p_0^l+jp_0^d)}ip_0, \textit{for } i\geq 1,\\ q_i=p_0^{d(i-1)}\lambda^i(1-p_0^d\lambda), \textit{for } i\geq 1 \end{cases} \tag{3.2}$$

where p_0 is the unique solution to the following equation

$$x+\sum_{i=1}^{\infty}\frac{r^i(1-\lambda x^d)^i}{\prod_{j=1}^{i}(r\sum_{l=0}^{d-1}x^l+jx^d)}x=1 \tag{3.3}$$

over the interval $(0,1)$.

Proof. In order to prove Theorem 1, we set

$$f(x)\triangleq x+\sum_{i=1}^{\infty}\frac{r^i(1-\lambda x^d)^i}{\prod_{j=1}^{i}(r\sum_{l=0}^{d-1}x^l+jx^d)}x$$

and (3.3) to $f(x)=1$. The proof consists of two parts. First, we will show that $f(x)=1$ indeed has a unique solution over the interval $(0,1)$. Second, we will show that the stationary distribution which is given in (3.2) indeed satisfies the local balance equations.

Proof of Part 1. It suffices to show the following properties

Property 1. $f(0)=0$ and $f(1)>1$;

Property 2. $f(x)$ is differentiable over the interval $(0,1)$;

Property 3. $f(x)$ is monotonically increasing when $f(x)\geq 1$ over the interval $(0,1)$.

By Properties 1 and 2, we know that $f(x)=1$ has a solution over the interval $(0,1)$. By Property 3, we know that $f(x)=1$ has a unique solution over the interval $(0,1)$.

Property 1 is obvious by showing

$$\lim_{x\to 0}f(x)=0$$

$$\lim_{x\to 1}f(x)>1.$$

In order to prove Property 2, we need to use two Gamma functions defined

below

$$\Gamma(a) = \int_0^\infty t^{a-1}e^{-t}dt$$
$$\Gamma(a,b) = \int_b^\infty t^{a-1}e^{-t}dt.$$

Note that

$$\begin{aligned}\Gamma(a) - \Gamma(a,b) &= \int_0^\infty t^{a-1}e^{-t}dt - \int_b^\infty t^{a-1}e^{-t}dt \\ &= \int_0^b t^{a-1}e^{-t}dt.\end{aligned}$$

According to the Welerstrass's definition for gamma function and generalized Laguerre polynomials, we can have

$$\Gamma(a) - \Gamma(a,b) = b^a e^{-b} \sum_{k=0}^{\infty} \frac{b^k}{a(a+1)\cdots(a+k)}.$$

Hence, we have

$$\begin{aligned}f(x) = x + &\left[\Gamma\left(a(x)\right) - \Gamma\left(a(x), b(x)\right)\right] \\ &\times \left[r\left(-\lambda + \frac{1}{x}\right)\right]^{-r\sum_{l=0}^{d-1} x^{l-d}} \times e^{r(-\lambda+\frac{1}{x})} \times x \quad (3.4)\end{aligned}$$

where $a(x) = 1 + r\sum_{l=0}^{d-1} x^{l-d}$ and $b(x) = r\left(-\lambda + \frac{1}{x}\right)$.

Now, we define a new function $v(x)$ as

$$v(x) \triangleq \Gamma(a) - \Gamma(a,b). \quad (3.5)$$

Except for $v(x)$, the other parts in (3.4) are clearly to be differentiable. Therefore, in order to show $f(x)$ is differentiable over (0,1), we need to show that $v(x)$ is

differentiable over (0,1). We have

$$\begin{aligned} v(x) &= \int_0^b t^{a-1}e^{-t}dt \\ &= \int_0^{r\left(-\lambda+\frac{1}{x}\right)} t^{r\sum_{l=0}^{d-1}x^{l-d}} \cdot e^{-t}dt. \end{aligned}$$

According to the Leibniz's integral rule, if $v(x)$ meets the following conditions

1. $b(x) = r\left(-\lambda+\frac{1}{x}\right)$ has continuous derivative over $(0,1)$;
2. $g(x,t) = t^{r\sum_{l=0}^{d-1}x^{l-d}} \cdot e^{-t}$ and its partial derivative $\frac{\partial}{\partial x}g(x,t)$ are continuous in the region of $0 < x < 1$ and $0 \le t \le b(x)$,

we can say $v(x)$ is differentiable, and both conditions can be easily verified. Therefore, the function $f(x)$ is differentiable over $(0,1)$.

In order to prove Property 3, we introduce

$$y_i(x) \triangleq \begin{cases} x, \text{ for } i = 0; \\ \frac{r^i(1-\lambda x^d)^i}{\prod_{j=1}^{i}(r\sum_{l=0}^{d-1}x^l + jx^d)}x, \text{ for } i \ge 1. \end{cases} \tag{3.6}$$

Clearly, $f(x) = \sum_{i=0}^{\infty} y_i(x)$. Now, we need to verify that $\sum_{i=0}^{\infty} y_i'(x) > 0$ when $\sum_{i=0}^{\infty} y_i(x) \ge 1$.

According to (3.6), it is easy to see that

$$y_{i+1}(x) = \frac{r(1-\lambda x^d)}{r\sum_{l=0}^{d-1}x^l + (i+1)x^d}y_i(x), \text{ for } i \ge 0, \tag{3.7}$$

and we set

$$h(x) = \frac{r(1-\lambda x^d)}{r\sum_{l=0}^{d-1}x^l + (i+1)x^d}.$$

So, we have

$$y_{i+1}'(x) = h(x)'y_i(x) + h(x)y_i'(x), \text{ for } i \ge 0.$$

Here, we find

$$h'(x) = -\frac{r\lambda d x^{d-1} + r(1-\lambda x^d)(r\sum_{l=0}^{d-1} l x^{l-1} + d(i+1)x^{d-1})}{(r\sum_{l=0}^{d-1} x^l + (i+1)x^d)^2} < 0.$$

Because $y_i(x) > 0$ and $h(x) > 0$, we can say that if there is some $y_k'(x) < 0$, then for all $i \geq k$, $y_i' < 0$. This gives rise to two cases:

1. All $y_i'(x) \geq 0$, for all $i > 0$;
2. There exists some integer k such as for all $i < k$, $y_i'(x) \geq 0$ and for all $i \geq k$, $y_i'(x) < 0$.

For Case 1, we have $\sum_{i=0}^{\infty} y_i'(x) > 0$, because $y_0'(x) = 1$ and $y_i'(x) \geq 0$, when $i \geq 0$.
For Case 2, according to (3.7), we have

$$\left(r\sum_{l=0}^{d-1} x^l + (i+1)x^d\right) y_{i+1}(x) = r(1-\lambda x^d) y_i(x).$$

Then

$$\sum_{i=1}^{\infty}(r\sum_{l=0}^{d-1} x^l + ix^d) y_i(x) = r(1-\lambda x^d)\sum_{i=0}^{\infty} y_i(x)$$

$$\sum_{i=0}^{\infty}(r\sum_{l=0}^{d-1} x^l + ix^d) y_i(x) - rx\sum_{l=0}^{d-1} = r\sum_{i=0}^{\infty} y_i(x) - \lambda x^d \sum_{i=0}^{\infty} y_i(x).$$

It follows that

$$x^d \sum_{i=0}^{\infty} i y_i(x) + r\sum_{l=0}^{d-1} x^l \cdot \sum_{i=0}^{\infty} y_i(x)$$
$$= r\sum_{i=0}^{\infty} y_i(x) - \lambda x^d \sum_{i=0}^{\infty} y_i(x) + rx\sum_{l=0}^{d-1} x^l.$$

Hence,

$$\sum_{i=0}^{\infty} i y_i(x) = r\sum_{l=0}^{d-1} x^{-l} - r(\lambda + \sum_{l=1}^{d-1} x^{-l})\sum_{i=0}^{\infty} y_i(x).$$

Now, by computing the derivatives we have

$$\sum_{i=0}^{\infty} iy_i'(x) + r(\lambda + \sum_{l=1}^{d-1} x^{-l}) \sum_{i=0}^{\infty} y_i'(x)$$
$$= r \sum_{l=1}^{d-1} lx^{-(l+1)} (\sum_{i=0}^{\infty} y_i(x) - 1).$$

Recall that k is the smallest integer such that $y_k'(x) < 0$. Thus, we have

$$\sum_{i=0}^{\infty} ky_i'(x) > \sum_{i=0}^{\infty} iy_i'(x).$$

Therefore,

$$z \sum_{i=0}^{\infty} y_i'(x) + r(\lambda + \sum_{l=1}^{d-1} x^{-l}) \sum_{i=0}^{\infty} y_i'(x) > r \sum_{l=1}^{d-1} lx^{-(l+1)} (\sum_{i=0}^{\infty} y_i(x) - 1)$$

$$(k + r\lambda + r \sum_{l=1}^{d-1} x^{-l}) \sum_{i=0}^{\infty} y_i'(x) > r \sum_{l=1}^{d-1} lx^{-(l+1)} (\sum_{i=0}^{\infty} y_i(x) - 1)$$

$$\sum_{i=0}^{\infty} y_i'(x) > \frac{r \sum_{l=1}^{d-1} lx^{-(l+1)}}{(k + r\lambda + r \sum_{l=1}^{d-1} x^{-l})} (\sum_{i=0}^{\infty} y_i(x) - 1).$$

Note that

$$k + r\lambda + r \sum_{l=1}^{d-1} x^{-l} > 0$$

$$r \sum_{l=1}^{d-1} lx^{-(l+1)} > 0,$$

this shows that when $f(x) = \sum_{i=0}^{\infty} y_i(x) \geq 1$, $f'(x) = \sum_{i=0}^{\infty} y_i'(x) > 0$.

Proof of Part 2. We show that (3.2) satisfies (3.1). The proof of this part is

simple and straightforward and we have

$$\begin{aligned} q_i r_{i,i-1} &= p_0^{d(i-1)} \lambda^i (1 - p_0^d \lambda) \times 1 \\ q_{i-1} r_{i-1,i} &= p_0^{d(i-2)} \lambda^{i-1} (1 - p_0^d \lambda)(\lambda p_0^d) \\ &= p_0^{d(i-1)} \lambda^i (1 - p_0^d \lambda), \end{aligned}$$

$$\begin{aligned} q_{(0,j)} r_{(0,j),1} &= \frac{r^{j-1}(1-\lambda p_0^d)^j}{\prod_{i=1}^{j}(r\sum_{l=0}^{d-1} p_0^l + i p_0^d)} j p_0 \cdot \lambda \left[p_0^d + \frac{r}{j} \sum_{l=0}^{d-1} p_0^l \right] \\ &= \frac{r^{j-1}(1-\lambda p_0^d)^j}{\prod_{i=1}^{j-1}(r\sum_{l=0}^{d-1} p_0^l + i p_0^d)} \lambda p_0 \\ q_1 r_{1,(0,j)} &= \lambda (1 - p_0^d \lambda) \cdot \frac{r q_{(0,j-1)}}{j-1} \\ &= \lambda (1 - p_0^d \lambda) \cdot \frac{r^{j-1}(1-\lambda p_0^d)^{j-1}}{\prod_{i=1}^{j-1}(r\sum_{l=0}^{d-1} p_0^l + i p_0^d)} p_0 \\ &= \frac{r^{j-1}(1-\lambda p_0^d)^j}{\prod_{i=1}^{j-1}(r\sum_{l=0}^{d-1} p_0^l + i p_0^d)} \lambda p_0 \end{aligned}$$

and

$$
\begin{aligned}
q_{(0,j)}r_{(0,j),(0,j-1)} &= \frac{r^{j-1}(1-\lambda p_0^d)^j}{\prod_{i=1}^{j}(r\sum_{l=0}^{d-1}p_0^l + ip_0^d)} jp_0\cdot\lambda(j-1)\left[p_0^d + \frac{r}{j}\sum_{l=0}^{d-1}p_0^l\right] \\
&= \frac{r^{j-1}(1-\lambda p_0^d)^j}{\prod_{i=1}^{j-1}(r\sum_{l=0}^{d-1}p_0^l + ip_0^d)}\lambda p_0(j-1) \\
q_{(0,j-1)}r_{(0,j-1),(0,j)} &= r\lambda(1-p_0^d\lambda)\cdot\frac{r^{j-2}(1-\lambda p_0^d)^{j-1}}{\prod_{i=1}^{j-1}(r\sum_{l=0}^{d-1}p_0^l + ip_0^d)}(j-1)p_0 \\
&= \frac{r^{j-1}(1-\lambda p_0^d)^j}{\prod_{i=1}^{j-1}(r\sum_{l=0}^{d-1}p_0^l + ip_0^d)}\lambda p_0(j-1).
\end{aligned}
$$

□

Based on Theorem 1, we can figure out the value of p_0 numerically. In particular, we consider a truncated version of $f(x)$ as

$$
f_n(x) \triangleq x + \sum_{i=1}^{n}\frac{r^i(1-\lambda x^d)^i}{\prod_{j=1}^{i}(r\sum_{l=0}^{d-1}x^l + jx^d)}x
$$

and define the error term as $f(x) - f_n(x)$. We have

$$
\begin{aligned}
f(x) - f_n(x) &= \sum_{i=n+1}^{\infty}\frac{r^i(1-\lambda x^d)^i}{\prod_{j=1}^{i}(r\sum_{l=0}^{d-1}x^l + jx^d)}x \\
&= \frac{r^n(1-\lambda x^d)^n}{\prod_{j=1}^{n}(r\sum_{l=0}^{d-1}x^l + jx^d)}x\cdot\sum_{i=1}^{\infty}\frac{r^i(1-\lambda x^d)^i}{\prod_{j=n+1}^{n+i}(r\sum_{l=0}^{d-1}x^l + jx^d)}.
\end{aligned}
$$

Note that the term $\frac{r^n(1-\lambda x^d)^n}{\prod_{j=1}^{n}(r\sum_{l=0}^{d-1}x^l + jx^d)}x$ in the above product is precisely the last item

of $f_n(x)$. Note also that

$$\sum_{i=1}^{\infty} \frac{r^i(1-\lambda x^d)^i}{\prod_{j=n+1}^{n+i}(r\sum_{l=0}^{d-1}x^l+jx^d)}x < \sum_{i=1}^{\infty} \frac{r^i(1-\lambda x^d)^i}{(r\sum_{l=0}^{d-1}x^l+nx^d)}$$

and

$$\sum_{i=1}^{\infty} \frac{r^i(1-\lambda x^d)^i}{(r\sum_{l=0}^{d-1}x^l+nx^d)^i} = \frac{r(1-\lambda x^d)}{r\sum_{l=0}^{d-1}x^l+(n+\lambda r)x^d},$$

which tends to 0 as n increases. Therefore, for large n, the error term $f(x)-f_n(x)$ is negligible compared to $f_n(x)$. In fact, according to our numerical calculations, $f_n(x)$ is very close to $f(x)$ as long as $n > 20$.

After we solve for p_0, we can calculate the stationary tail distribution and the expected task response time. Here, stationary tail distribution is denoted by s_i which represents the probability for a server to have no less than i tasks in process.

Corollary 1. *In the large-system limit, the stationary tail distribution under JIQ-NE is*

$$s_i = \begin{cases} 1, \textit{ for } i = 0, \\ p_0^{(i-1)d}\lambda^i, \textit{ for } i \geq 1 \end{cases}$$

and the expected task response time under JIQ-NE is $\frac{1}{1-p_0^d\lambda}$.

Proof. First, we derive the stationary tail distribution s_i as follows:

$$s_i = \sum_{j=i}^{\infty} q_j = \sum_{j=i}^{\infty} p_0^{d(j-1)}\lambda^j(1-p_0^d\lambda) = p_0^{d(i-1)}\lambda^i.$$

Second, we derive the expected task response time. When a new task arrives, two events could happen

1. The new task is finally routed to an empty I-queue. This event happens with probability p_0^d. Under this event, the expected task response time is $\sum_{i=0}^{\infty}(i+1)q_i$;
2. The new task is finally routed to a non-empty I-queue. This event happens

with probability $1-p_0^d$. Under this event, the expected task response time is 1.

Therefore, the total expected response time is

$$p_0^d \sum_{i=0}^{\infty} (i+1)q_i + (1-p_0^d) = \frac{1}{1-p_0^d\lambda}.$$

□

3.3 Stationary Distribution Under JIQ-E

Similarly, we first derive the transition rates for the state evolution of a single server in JIQ-E system as follows:

- $r_{i,i-1} = 1$, for $i \geq 2$.
  The processing rate of a task in a server is exponentially distributed with mean 1.

- $r_{i-1,i} = \lambda p_0$, for $i \geq 2$.
  The task arrival rate is λN, the probability of joining an empty I-queue is p_0, and the probability of selecting the target server over all servers is $\frac{1}{N}$. Hence, the transition rate is $\lambda N \cdot p_0 \cdot \frac{1}{N}$.

- $r_{1,(0,j)} = (1-p_0)^{d-1} p_{j-1}$, for $i \geq 2$.
  The processing rate of a task is exponentially distributed with mean 1, and the probability of joining an I-queue of size $j-1$ is $(1-p_0)^{d-1} p_{j-1}$

- $r_{1,(0,1)} = 1-(1-p_0)^d$.
  The processing rate of a task is exponentially distributed with mean 1, and the probability of continuously joining d non-empty I-queues is $(1-p_0)^d$. Hence, the transition rate is $1-(1-p_0)^d$.

- $r_{(0,j),1} = \lambda(p_0 + \frac{r}{j})$.
  The task arrival rate is λN and two events, resulting from a newly arrival task, lead to this state change. The first event is that the new task is routed to an empty I-queue and then directed to the target server. The probability

of this event is $p_0 \cdot \frac{1}{N}$. The second event is that the new task is routed to the I-queue associated with the target server and then directed to the target server. The probability of this event is $\frac{1}{M} \cdot \frac{1}{j} = \frac{r}{N} \cdot \frac{1}{j}$. Hence, the transition rate is $\lambda N \left(p_0 \cdot \frac{1}{N} + \frac{r}{N} \cdot \frac{1}{j} \right)$.

- $r_{(0,j-1),(0,j)} = rq_1(1-p_0)^{d-1}$, for $j \geq 2$.
  The generating rate of idle servers is $q_1 N$, and the probability of selecting the I-queue associated with the target server is $\frac{1}{M} \cdot (1-p_0)^{d-1}$.

- $r_{(0,j),(0,j-1)} = \lambda(j-1)(p_0 + \frac{r}{j})$ for $j \geq 2$.
  The task arrival rate is λN. There are two events resulting in this state change. The first event is that the new task is routed to an empty I-queue and then directed to an idle server having the same I-queue as the target server. The probability of this event is $p_0 \cdot \frac{j-1}{N}$. The second event is that the new task is routed to the I-queue associated with the target server and then directed to another idle server. The probability of this event is $\frac{1}{M} \cdot \frac{j-1}{j}$. Hence, the transition rate is $\lambda N \left(p_0 \cdot \frac{j-1}{N} + \frac{r}{N} \cdot \frac{j-1}{j} \right)$.

By solving the same local balance equations (3.1), we have the following theorem:

Theorem 2. *The stationary distribution of the state of a single server under JIQ-E in the large-system limit is*

$$\begin{cases} q_{(0,i)} = \frac{r^{i-1}(1-\lambda p_0)^i (1-p_0)^{(d-1)i}}{\prod_{j=1}^{i}(r+jp_0)} i p_0, \textit{ for } i \geq 1, \\ q_i = p_0^{i-1} \lambda^i (1-p_0\lambda), \textit{ for } i \geq 1 \end{cases} \tag{3.8}$$

where p_0 is the unique solution to the following equation

$$x + \sum_{i=1}^{\infty} \frac{r^i (1-\lambda x)^i (1-x)^{(d-1)(i-1)}}{\prod_{j=1}^{i}(r+jx)} [1-(1-x)^d] = 1 \tag{3.9}$$

over the interval $(0,1)$.

Proof. The proof of Theorem 2 is similar to the proof of Theorem 1. In order to prove Theorem 2, we set

$$f(x) \triangleq x + \sum_{i=1}^{\infty} \frac{r^i (1-\lambda x)^i (1-x)^{(d-1)(i-1)}}{\prod_{j=1}^{i} (r+jx)} [1-(1-x)^d]$$

and (3.9) to $f(x) = 1$. The proof consists of two parts. First, we will show that $f(x) = 1$ indeed has an unique solution over the interval $(0,1)$. Second, we will show that the stationary distribution given in (3.8) indeed satisfies the local balance equations.

Proof of Part 1. It suffices to show the following properties

Property 1. $f(0) = 0$ and $f(1) > 1$;

Property 2. $f(x)$ is differentiable over the interval $(0,1)$;

Property 3. $f(x)$ is monotonically increasing over $(0,1)$.

By Properties 1 and 2, we know that $f(x) = 1$ has a solution over the interval $(0,1)$. By Property 3, we know that $f(x) = 1$ has a unique solution over the interval $(0,1)$.

Property 1 is obvious by showing

$$\lim_{x \to 0} f(x) = 0$$

$$\lim_{x \to 1} f(x) > 1.$$

In order to prove Property 2, we also use the above-mentioned Gamma functions. Based on (3.8) and Gamma functions, we have

$$\begin{aligned} f(x) = x + [\Gamma(a(x)) - \Gamma(a(x), b(x))] \\ \times \left[r\left(-\lambda + \frac{1}{x}\right) \right]^{-\frac{r}{x}} \times e^{r(-\lambda+\frac{1}{x})(1-x)^{d-1}} \times [1-(1-x)^d] \end{aligned} \quad (3.10)$$

where $a(x) = 1 + \frac{r}{x}$ and $b(x) = r\left(-\lambda + \frac{1}{x}\right)(1-x)^{d-1}$.

We also use (3.5) here. Except for $v(x)$, the other parts in (3.10) are clearly to be differentiable. Therefore, in order to show $f(x)$ is differentiable over (0,1), we need

to show that $v(x)$ is differentiable over (0,1). From above-mentioned conclusions, we can have

$$\begin{aligned} v(x) &= \int_0^b t^{a-1}e^{-t}dt \\ &= \int_0^{r\left(-\lambda+\frac{1}{x}\right)(1-x)^{d-1}} t^{\frac{r}{x}}\cdot e^{-t}dt. \end{aligned}$$

According to the Leibniz's integral rule, if $v(x)$ meets the following conditions

1. $b(x) = r\left(-\lambda+\frac{1}{x}\right)(1-x)^{d-1}$ has continuous derivative over $(0,1)$;
2. $g(x,t) = t^{\frac{r}{x}}\cdot e^{-t}$ and its partial derivative $\frac{\partial}{\partial x}g(x,t)$ are continuous in the region of $0<x<1$ and $0\le t\le b(x)$,

we can say $v(x)$ is differentiable, and both conditions can be easily verified. Therefore, the function $f(x)$ is differentiable over $(0,1)$.

In order to prove Property 3, we introduce

$$y_i(x) \triangleq \begin{cases} x, \text{ for } i=0; \\ \sum\limits_{i=1}^{\infty} \frac{r^i(1-\lambda x)^i(1-x)^{(d-1)(i-1)}}{\prod\limits_{j=1}^{i}(r+jx)}[1-(1-x)^d], \text{ for } i\ge 1. \end{cases} \tag{3.11}$$

Clearly, $f(x) = \sum\limits_{i=0}^{\infty} y_i(x)$. Now, we need to verify that $\sum\limits_{i=0}^{\infty} y_i'(x) > 0$.

According to (3.11), it is easy to have

$$y_{i+1}(x) = \frac{r(1-\lambda x^d)}{r\sum\limits_{l=0}^{d-1} x^l + (i+1)x^d} y_i(x), \text{ for } i\ge 1, \tag{3.12}$$

We set

$$h(x) = \frac{r(1-\lambda x)(1-x)}{r+(i+1)x}.$$

So, we have

$$y_{i+1}'(x) = h(x)'y_i(x) + h(x)y_i'(x), \text{ for } i\ge 1.$$

Here, we find

$$h'(x) = -\left[\frac{r(r\lambda+(i+1))(1-x)}{(r+(i+1)p_0)^2} + \frac{r(1-\lambda x)}{r+(i+1)x}\right] < 0.$$

Because $y_i(x) > 0$ and $h(x) > 0$, we can say that if there is some $y'_k(x) < 0$, then for all $i \geq k$, $y'_i < 0$. This gives rise to two cases:

1. All $y'_i(x) \geq 0$, for all $i > 0$;

2. There exists some integer k such as for all $i < k$, $y'_i(x) \geq 0$ and for all $i \geq k$, $y'_i(x) < 0$.

For Case 1, we have $\sum_{i=0}^{\infty} y'_i(x) > 0$, because $y'_0(x) = 1$ and $y'_i(x) \geq 0$, when $i \geq 0$.
For Case 2, according to (3.12), we have

$$(r+(i+1)x)\, y_{i+1}(x) = r(1-\lambda x)(1-x)y_i(x).$$

It follows that

$$\sum_{i=0}^{\infty}[r+(i+1)x]y_{i+1}(x) = \sum_{i=0}^{\infty} r(1-\lambda x)(1-x)y_i(x)$$

$$\sum_{i=0}^{\infty}(r+ix)y_i(x) - rx = \sum_{i=0}^{\infty} r(1-\lambda x)y_i(x) - \sum_{i=0}^{\infty} r(1-\lambda x)xy_i(x)$$

$$\sum_{i=0}^{\infty} ixy_i(x) - rx = -r\lambda x\sum_{i=0}^{\infty} y_i(x) - rx\sum_{i=0}^{\infty} y_i(x) + r\lambda x^2\sum_{i=0}^{\infty} y_i(x)$$

$$\sum_{i=0}^{\infty} iy_i(x) + r\lambda\sum_{i=0}^{\infty} y_i(x) = r - r\sum_{i=0}^{\infty} y_i(x) + r\lambda\sum_{i=0}^{\infty} xy_i(x).$$

Now, by computing the derivatives we have

$$\sum_{i=0}^{\infty} iy'_i(x) + r\lambda\sum_{i=0}^{\infty} y'_i(x) = -r\sum_{i=0}^{\infty} y'_i(x) + r\lambda\sum_{i=0}^{\infty} y_i(x) + r\lambda\sum_{i=0}^{\infty} xy'_i(x).$$

Recall that k is the smallest integer such that $y_k'(x) < 0$. Thus, we have

$$\sum_{i=0}^{\infty} k y_i'(x) > \sum_{i=0}^{\infty} i y_i'(x).$$

Therefore,

$$k\sum_{i=0}^{\infty} y_i'(x) + r\lambda \sum_{i=0}^{\infty} y_i'(x) > -r\sum_{i=0}^{\infty} y_i'(x) + r\lambda \sum_{i=0}^{\infty} y_i(x) + r\lambda \sum_{i=0}^{\infty} x y_i'(x)$$

$$(k + r\lambda(1-x) + r)\sum_{i=0}^{\infty} y_i'(x) > r\lambda \sum_{i=0}^{\infty} y_i(x)$$

$$\sum_{i=0}^{\infty} y_i'(x) > \frac{r\lambda}{k + r\lambda(1-x) + r} \sum_{i=0}^{\infty} y_i(x).$$

We can find that $\frac{r\lambda}{k+r\lambda(1-x)+r} > 0$ and $\sum_{i=0}^{\infty} y_i(x) > 0$. Hence, we have

$$\sum_{i=0}^{\infty} y_i'(x) > 0.$$

This shows that $f'(x) > 0$ over the interval (0,1).

Proof of Part 2. we show that (3.8) meets (3.1). Similarly, the proof of this part is simple and straightforward and we have

$$\begin{aligned} q_i r_{i,i-1} &= p_0^{i-1}\lambda^i(1 - p_0\lambda) \times 1 \\ q_{i-1} r_{i-1,i} &= p_0^{i-2}\lambda^{i-1}(1 - p_0\lambda)(\lambda p_0) \\ &= p_0^{i-1}\lambda^i(1 - p_0\lambda), \end{aligned}$$

$$
\begin{aligned}
q_{(0,j)}r_{(0,j),1} &= \frac{r^{j-1}(1-\lambda p_0)^j(1-p_0)^{(d-1)j}}{\prod_{i=1}^{j}(r+ip_0)} jp_0\cdot\lambda(p_0+\frac{r}{j}) \\
&= \frac{r^{j-1}(1-\lambda p_0)^j(1-p_0)^{(d-1)j}}{\prod_{i=1}^{j-1}(r+ip_0)}\lambda p_0 \\
q_1 r_{1,(0,j)} &= \lambda(1-p_0\lambda)\cdot\frac{rq_{(0,j-1)}}{j-1} \\
&= \lambda(1-p_0\lambda)\cdot\frac{r^{j-1}(1-\lambda p_0)^{j-1}(1-p_0)^{(d-1)(j-1)}}{\prod_{i=1}^{j-1}(r+ip_0)}p_0 \\
&= \frac{r^{j-1}(1-\lambda p_0)^j(1-p_0)^{(d-1)j}}{\prod_{i=1}^{j-1}(r+ip_0)}\lambda p_0
\end{aligned}
$$

and

$$
\begin{aligned}
q_{(0,j)}r_{(0,j),(0,j-1)} &= \frac{r^{j-1}(1-\lambda p_0)^j(1-p_0)^{(d-1)j}}{\prod_{i=1}^{j}(r+ip_0)} jp_0\cdot\lambda(j-1)(p_0+\frac{r}{j}) \\
&= \frac{r^{j-1}(1-\lambda p_0)^j(1-p_0)^{(d-1)j}}{\prod_{i=1}^{j-1}(r+ip_0)}\lambda p_0(j-1) \\
q_{(0,j-1)}r_{(0,j-1),(0,j)} &= \frac{r^{j-2}(1-\lambda p_0)^{j-1}(1-p_0)^{(d-1)(j-1)}}{\prod_{i=1}^{j-1}(r+ip_0)}(j-1)p_0\cdot r\lambda(1-p_0\lambda)(1-p_0)^{d-1} \\
&= \frac{r^{j-1}(1-\lambda p_0)^j(1-p_0)^{(d-1)j}}{\prod_{i=1}^{j-1}(r+ip_0)}\lambda p_0(j-1).
\end{aligned}
$$

□

The method to compute p_0 numerically is similar to that of JIQ-NE, we set

$$
f_n(x) \triangleq x+\sum_{i=1}^{n}\frac{r^i(1-\lambda x)^i(1-x)^{(d-1)(i-1)}}{\prod_{j=1}^{i}(r+jx)}[1-(1-x)^d]
$$

and we find when $n > 20$, the solution of $f_n(x) = 1$ is very close to the solution of (3.9).

Then, we can calculate the stationary tail distribution and the expected task response time.

Corollary 2. *In the large-system limit, the stationary tail distribution under JIQ-E is*

$$s_i = \begin{cases} 1, \textit{ for } i = 0, \\ p_0^{i-1}\lambda^i, \textit{ for } i \geq 1 \end{cases}$$

and the expected task response time under JIQ-E is $\frac{1}{1-p_0\lambda}$.

Proof. First, we derive the stationary tail distribution s_i, as follows:

$$s_i = \sum_{j=i}^{\infty} q_j = \sum_{j=i}^{\infty} p_0^{j-1}\lambda^j(1-p_0\lambda) = p_0^{i-1}\lambda^i.$$

Second, we derive the expected task response time. When a new task arrives, two events could happen

1. The new task is routed to an empty I-queue. This event happens with probability p_0. Under this event, the expected task response time is $\sum_{i=0}^{\infty}(i+1)q_i$;
2. The new task is routed to a non-empty I-queue. This event happens with probability $(1-p_0)$. Under this event, the expected task response time is he task is 1.

Therefore, the total expected response time is

$$p_0\sum_{i=0}^{\infty}(i+1)q_i + (1-p_0) = \frac{1}{1-p_0\lambda}.$$

□

3.4 Comparisons with Existing JIQ Algorithms

We now compare JIQ-NE and JIQ-E with the existing JIQ algorithms, including JIQ and JIQ-Po*d*. We have the following results from our previous work [10].

Theorem 3. *The stationary distribution of the state of a single server under JIQ in the large-system limit is*

$$\begin{cases} q_{(0,i)} = \frac{r^{i-1}(1-\lambda p_0)^i}{\prod_{j=1}^{i}(r+jp_0)} ip_0, \text{ for } i \geq 1, \\ q_i = p_0^{i-1}\lambda^i(1-p_0\lambda), \text{ for } i \geq 1 \end{cases} \tag{3.13}$$

where p_0 is the unique solution to the following equation

$$x + \sum_{i=1}^{\infty} \frac{r^i(1-\lambda x)^i}{\prod_{j=1}^{i}(r+jx)} x = 1 \tag{3.14}$$

over the interval $(0,1)$.

Corollary 3. *In the large-system limit, the stationary tail distribution under JIQ is*

$$s_i = \begin{cases} 1, \text{ for } i = 0, \\ p_0^{i-1}\lambda^i, \text{ for } i \geq 1 \end{cases}$$

and the expected task response time under JIQ is $\frac{1}{1-p_0\lambda}$.

Theorem 4. *The stationary distribution of the state of a single server under JIQ-Pod in the large-system limit is*

$$\begin{cases} q_{(0,i)} = \frac{r^{i-1}(1-\lambda^d p_0)^i}{\prod_{j=1}^{i}(r+jp_0\frac{1-\lambda^d}{1-\lambda})} ip_0, \text{ for } i \geq 1, \\ q_i = \lambda^{\frac{d^i-1}{d-1}} p_0^{\frac{d^{i-1}-1}{d-1}}(1-\lambda^{d^i}p_0^{d^{i-1}}), \text{ for } i \geq 1 \end{cases} \tag{3.15}$$

where p_0 is the unique solution to the following equation

$$x + \sum_{i=1}^{\infty} \frac{r^i(1-\lambda^d x)^i}{\prod_{j=1}^{i}(r+jx\frac{1-\lambda^d}{1-\lambda})} x = 1 \tag{3.16}$$

over the interval $(0,1)$.

Corollary 4. *In the large-system limit, the stationary tail distribution under JIQ-*

Pod is

$$s_i = \begin{cases} 1, \textit{for } i = 0, \\ \lambda^{\frac{d^i-1}{d-1}} p_0^{\frac{d^{i-1}-1}{d-1}}, \textit{for } i \geq 1 \end{cases}$$

and the expected task response time under the JIQ-Pod algorithm is $1 + p_0 \sum_{i=1}^{\infty} (s_i)^d$.

We are now ready to compare JIQ-NE with JIQ. By Corollary 1 and Corollary 3, the mean response times of JIQ-NE and JIQ are $\frac{1}{1-p_0^d\lambda}$ and $\frac{1}{1-p_0\lambda}$, respectively. Hence, if the values of p_0 are almost the same (which can be verified from Fig. 4.5), JIQ-NE has a shorter mean response time. For the same reason, JIQ-NE enjoys a better tail distribution.

Next, we compare JIQ-E and JIQ. By Corollary 2 and Corollary 3, JIQ-E and JIQ have the same expression of the mean response time. Hence, if the value of p_0 for JIQ-E is smaller than that for JIQ (which can be shown in Fig. 4.5), JIQ-E has a shorter mean response time. Similarly, JIQ-E has a better tail distribution.

Finally, the expected task response time expression of JIQ-Po*d* is different from the expressions of the other three algorithm and it appears difficult to compare JIQ-NE, JIQ-E, and JIQ-Po*d* analytically. As such, we will compare them through simulations in the next chapter.

Chapter 4

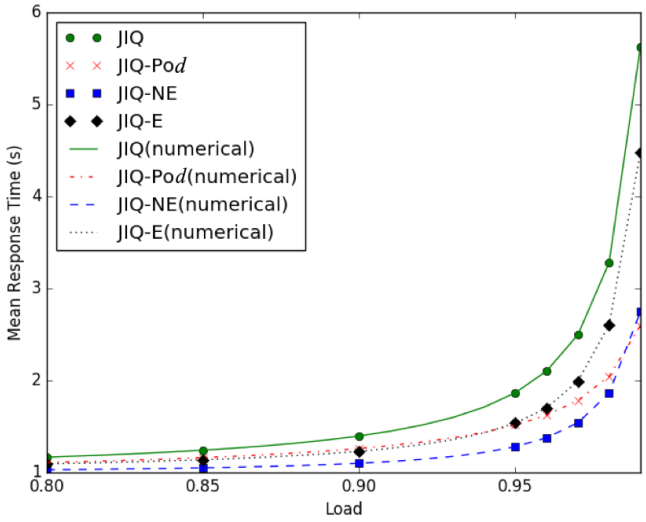
Simulation and Comparisons

In this chapter, we compare JIQ and three JIQ variants (namely, JIQ-Pod, JIQ-NE and JIQ-E) with respect to the mean response time, communication cost (which will be defined shortly), empty I-queue rate (i.e., the fraction of empty I-queues) through numerical and simulation results. We also evaluate the influence of d and r. In particular, the communication cost is measured as the total number of interactions for a certain amount of tasks. (The details can be found in Sec. 4.2.) Here, an interaction means a probing operation between a new task and a scheduler, between a scheduler and a server, or between a server and an I-queue.

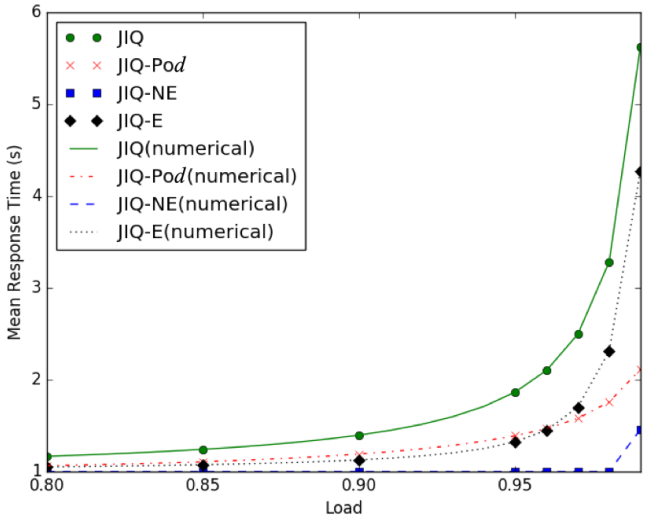
In our simulations, the task arrival process follows a Poisson process with mean λN and the task processing time is exponentially distributed with mean 1 (as described in our system model). The number of servers is set to be $10,000$ (i.e., $N = 10,000$). The parameters r and d are varied with $r = 20$ or 100 and $d = 2$ or 4. When $r = 20$, we have 500 schedulers (i.e., $M = 500$). When $r = 100$, we have 100 schedulers (i.e., $M = 100$).

4.1 Delay Performance

The delay performance of JIQ and three JIQ variants is evaluated and compared by using the mean response time as shown in Fig. 4.1 and Fig. 4.2. The numerical results are based on our theoretical analysis, which is continuous by the load (i.e. λ from 0.80 to 0.99), and it is clear that simulation results match the theoret-

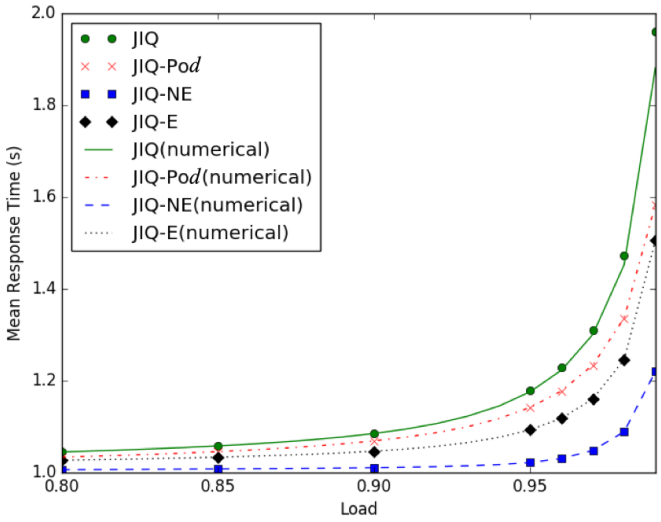


(a) $r = 20$, $d = 2$

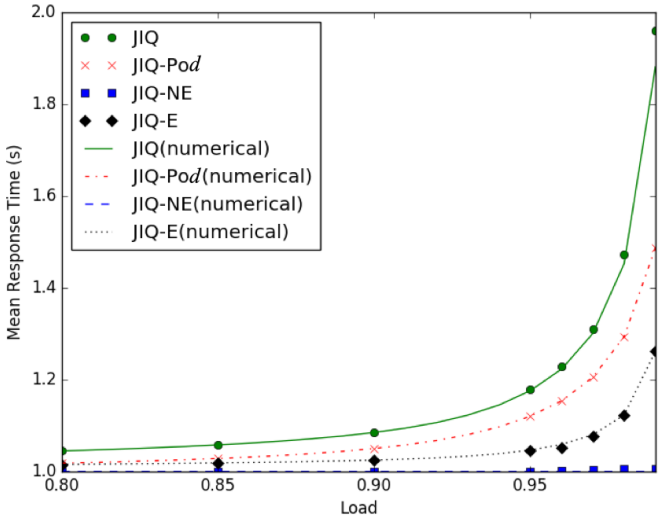


(b) $r = 20$, $d = 4$

Figure 4.1: (a): Mean response time with $r = 20$, $d = 2$ and λ changes from 0.80 to 0.99. (b):Mean response time with $r = 20$, $d = 4$ and λ changes from 0.80 to 0.99.



(a) $r = 100, d = 2$



(b) $r = 100, d = 4$

Figure 4.2: (a):Mean response time with $r = 100$, $d = 2$ and λ changes from 0.80 to 0.99. (b):Mean response time with $r = 100$, $d = 4$ and λ changes from 0.80 to 0.99.

ical analysis. From Fig. 4.1 and Fig. 4.2, it is shown that our three variants have better performance than conventional JIQ, while JIQ-NE constantly has the best delay performance with the increasing of the load from 0.80 to 0.99. Within three variants, when $r = 20$ and the load is low (i.e., $\lambda < 0.9$), JIQ-Pod has the worst performance, and JIQ-E's performance lies in the middle. When the load grows up, the performance of JIQ-Pod gets closer to JIQ-E's. When the load is over 0.9, the performance of JIQ-Pod catches up with and surpasses JIQ-E and, in Fig. 4.1(a), when $r = 20, d = 2$ and $\lambda = 0.99$, JIQ-Pod almost catches up with and tends to surpass the performance of JIQ-NE.

4.2 Communication Cost

The communication cost of three JIQ variants is compared by counting the number of interactions. Because JIQ algorithm has no interaction, we only show the other 3 algorithms' results. In particular, an interaction refers to an event that a task checks the state of an I-queue, a scheduler checks the state of a server, or a server checks the state of an I-queue. We count the total number of interactions for $5,000,000$ tasks. Since the number of servers is $N = 10,000$, each server receives 500 tasks on average. We select the load λ from 0.5 to 0.99 for comparison, shown in Fig. 4.3 and Fig. 4.4.

From Fig. 4.3 and Fig. 4.4, it is shown that when the load grows from low to high, communication cost of both JIQ-Pod and JIQ-NE grows, while the cost of JIQ-E decreases. Although both JIQ-Pod and JIQ-NE have a rising trend, JIQ-NE costs less than JIQ-Pod and the gap goes up with the load increases as shown in all the four conditions in Fig. 4.3 and Fig. 4.4. Because when the load grows, the number of idle servers decreases which means the average length of I-queues decreases. In this situation, the probability of selecting an empty I-queue increases so that there should be more interactions. However, comparing JIQ-NE and JIQ-Pod, when selecting an empty I-queue, JIQ-Pod needs to cost 2 interactions, while JIQ-NE only needs to cost one interaction. That is why the cost of JIQ-Pod is twice as much as JIQ-NE.

On the contrary, the communication cost of JIQ-E decreases as the load increases. The cost of JIQ-E is high when the load is very low, but it then decreases

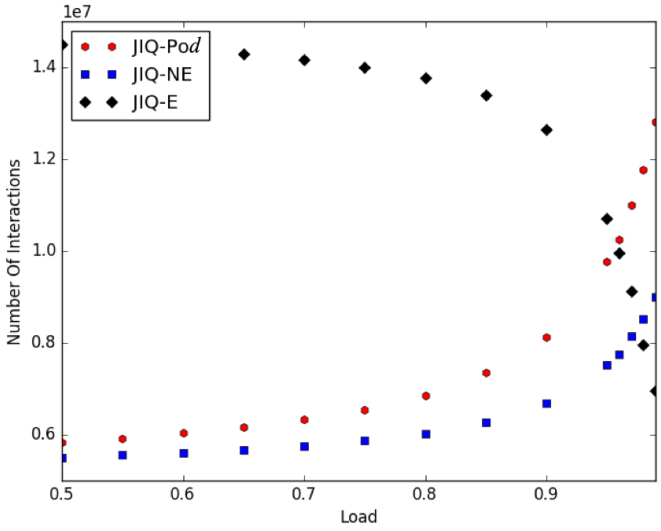
as the load increases. When the load is over 0.95, JIQ-E has the lowest cost among all the variants when $r = 20$. Because when the load grows, the number of idle servers decreases which means the average length of I-queues decreases. In this situation, the probability for a server to be idle decreases so that the interactions decreases. On the other hand, when $r = 100$, JIQ-NE constantly has the lowest cost as the load varies.

4.3 Idle Queue Rate

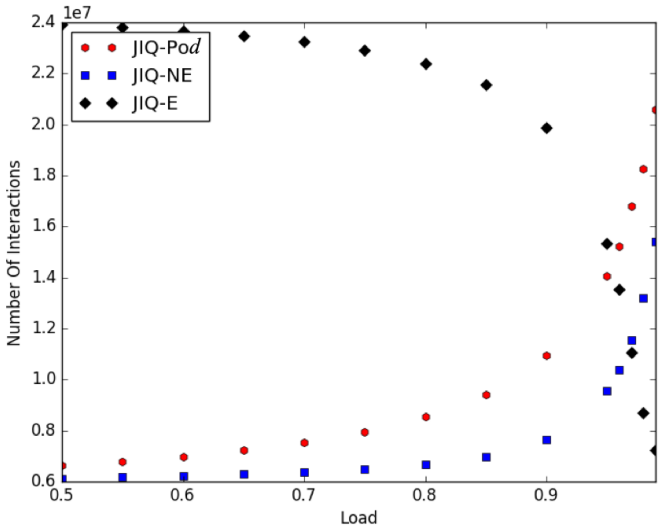
We evaluate the empty I-queue rate. As shown in Fig. 4.5, when the load increases, all the variants' rates follow a growing trend. It is shown that JIQ-NE keeps the highest rate at the beginning and JIQ-Pod is in the middle. However, when the load keeps growing up, rates of JIQ-NE and JIQ-Pod are getting close. When the load is over 0.95, JIQ-Pod catches up with JIQ-NE and surpasses JIQ-NE. JIQ-E maintains the lowest rate, but its gap with the other two variants decreases when the load grows. We also find that when the load is equal to 0.99, the rates of the three variants are close to each other. Hence, the performance of JIQ-E becomes almost the same as JIQ at high arrival rate. Moreover, with the same p_0, JIQ-NE can outperform JIQ-E with its advanced I-queue selection method.

4.4 Influence of d and r

Finally, we evaluate the role of the parameters d and r on the system performance. Since the system performance varies as we change the load λ (see, e.g., Fig. 4.1, Fig. 4.2, Fig. 4.3 and Fig. 4.4), we "average" the effect of λ by computing the average performance when λ is between 0.5 and 0.99. From Fig. 4.6, when r changes from 20 to 100, the averages of the mean response time of three strategies improve, and JIQ-E has the most improvements. When d grows from 2 to 4, the averages of the mean response time improves, and JIQ-NE has the most improvements. Overall, JIQ-NE has the best average mean response time, since the average time of JIQ-NE keeps the least among all the algorithms across different d and r. Similarly, we observe from Fig. 4.7 that JIQ-NE has the best average communication cost since the average cost of JIQ-NE keeps the least across different d and r.

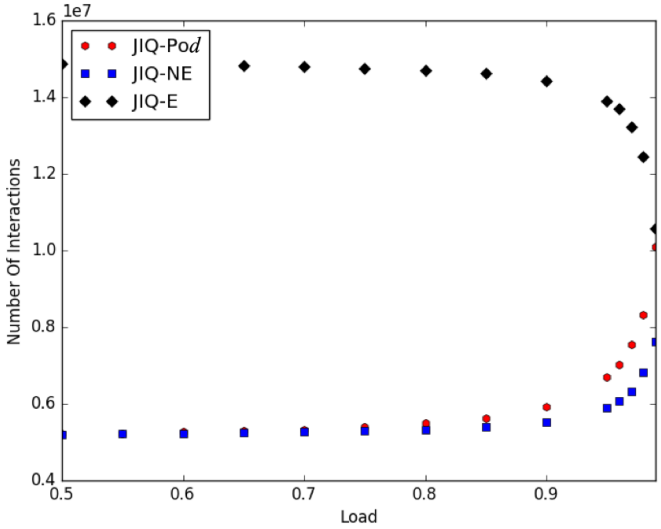


(a) $r = 20$, $d = 2$

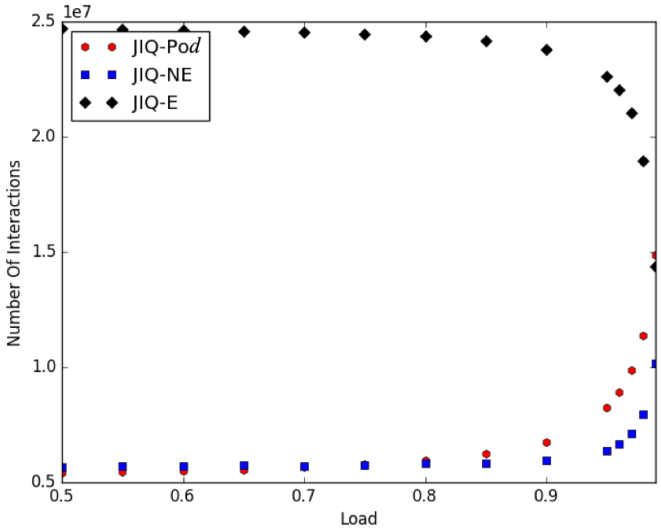


(b) $r = 20$, $d = 4$

Figure 4.3: (a): Communication cost with $r = 20$, $d = 2$. (b): Communication cost with $r = 20$, $d = 4$.



(a) $r = 100$, $d = 2$



(b) $r = 100$, $d = 4$

Figure 4.4: (a): Communication cost with $r = 100$, $d = 2$. (b): Communication cost with $r = 100$, $d = 4$.

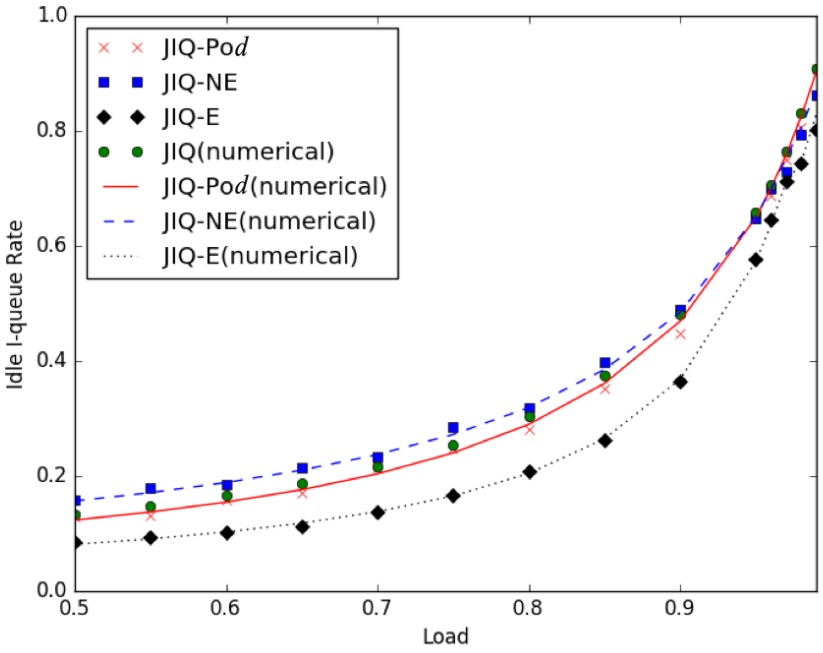


Figure 4.5: Idle I-queue rate of JIQ, JIQ-Pod, JIQ-NE and JIQ-E with $r = 10$, $d = 2$ and λ changes from 0.5 to 0.99.

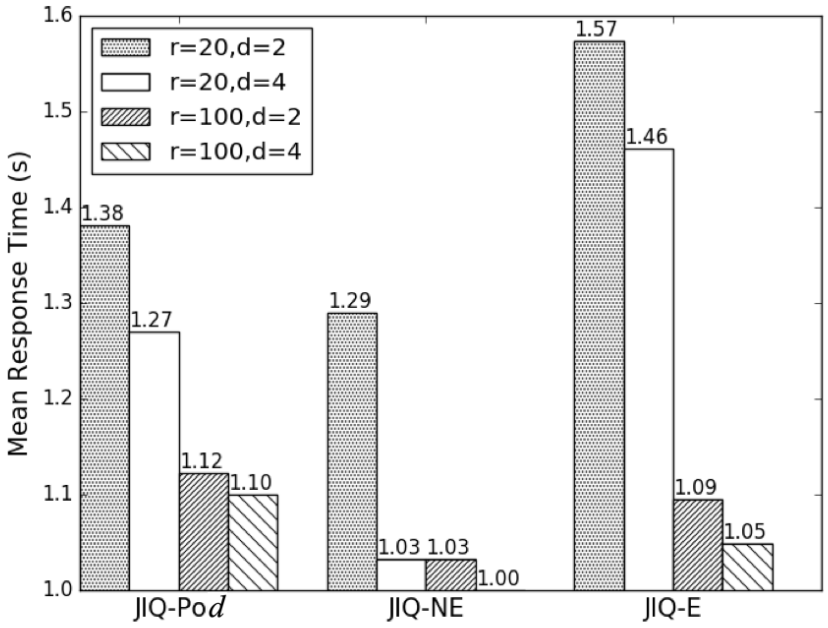


Figure 4.6: Average of mean response time with λ from 0.5 to 0.9 for JIQ, JIQ-NE and JIQ-E

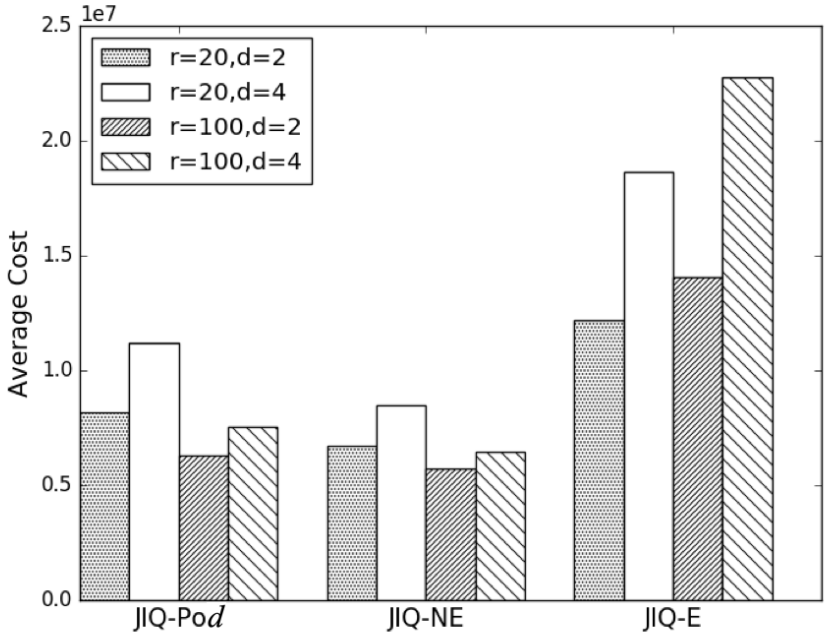


Figure 4.7: Average cost with λ from 0.5 to 0.9 for JIQ, JIQ-NE and JIQ-E

Chapter 5

Implementation on AWS EC2 Platform

To evaluate the practicability of our new algorithms, we implement JIQ and its three variants in a cloud environment by using Amazon EC2 as our platform and socket application interface as the interacting method.

We use Amazon EC2 instances as our routers and run all our experiments with 56 EC2 instances, and each of the instances has 1 CPU, 1GB of memory and 8 GB of storage. Specifically, the EC2 instances are classified into three groups: Generator, Schedulers and Servers, with 1 instance as Generator, 5 instances as Schedulers and 50 instances as Servers. The experimental environment is set as follows:

1. *Generator*: There is only one generator in each experiment. The generator generates tasks and sends tasks to schedulers. The load is fixed as $\lambda = 0.9$ and $N = 50000$. That is, the task generation follows a Poisson process with rate $1/\lambda N$.

2. *Schedulers*: Each scheduler contains an I-queue. Schedulers receive tasks from the generator and assign tasks to servers according to the JIQ algorithms.

3. *Servers*: Servers receive tasks from schedulers and process tasks. The rate

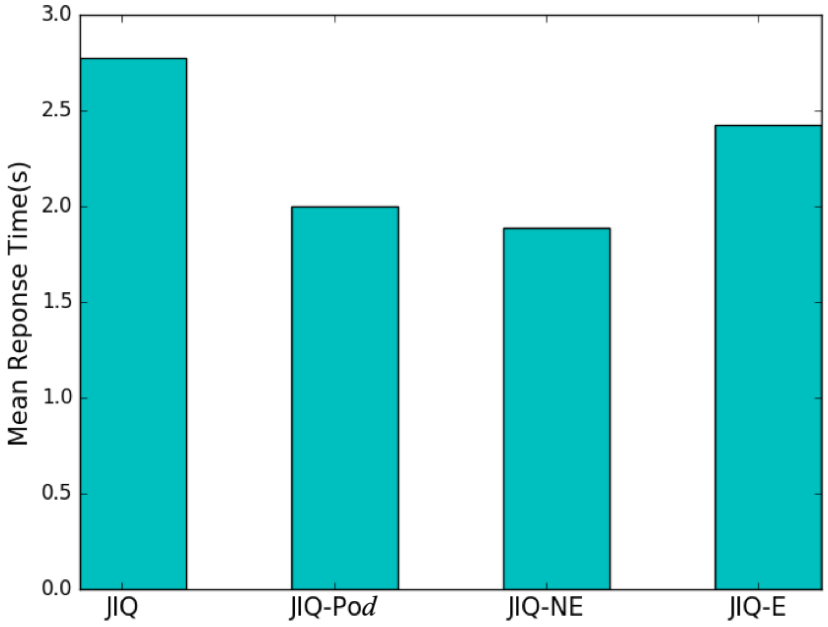


Figure 5.1: Mean response time bar with $\lambda = 0.9$

task-processing time follows exponential distribution with mean 1. Whenever a server becomes idle, it sends a request to a scheduler to join an I-queue.

4. *Tasks*: The number of tasks in each experiment is fixed to be $50,000$. For each task, we track the task generation time and the task completion time.

We also show the pseudo-code of all parts in our system. Algorithm 1 shows the pseudo-code of the generator operating mode in all experiments. Algorithm 2 shows the pseudo-code of schedulers operating mode in all experiments. Algorithm 3 shows the pseudo-code of servers operating mode in all experiments.

Experimental results are shown in Fig. 5.1. We observe that JIQ-NE still has the best performance among all the JIQ algorithms deployed on Amazon EC2. Moreover, the experimental results match our simulation results, further demonstrating the feasibility and practicability of our proposed algorithms.

Algorithm 1 Pseudo-code of generator operating mode

Ensure: $d = 2$ and all parts of the system operate at the same time. The *IntervalTime* here follows exponential distribution with mean $1/\lambda N$. The *ProcessingTime* follows exponential distribution with mean 1.

```
1:  procedure GENERATOR(TaskNum, SchedulerAddress)
2:      for n = 1 to TaskNum do
3:          GeneratedTime ← time.now
4:          ProcessingTime ← Exponential(1)
5:          IntervalTime ← Exponential(1/λN)
6:          Task.message ← GeneratedTime + ProcessingTime
7:          i ← random(SchedulerAddress)
8:          if JIQ-NE and len(Scheduler(i)) = 0 then
9:              i ← random(SchedulerAddress)
10:         end if
11:         Task ⇒ Scheduler(i)
12:         time.sleep(IntervalTime)
13:     end for
14: end procedure
```

Algorithm 2 Pseudo-code of schedulers operating mode

Ensure: $d = 2$ and all parts of the system operate at the same time.

```
1:  procedure SCHEDULER(ServerAddress)
2:      while True do
3:          if Receive Task from Generator then
4:              if len(I − queue) > 0 then
5:                  i ← I − queue[0]
6:              else
7:                  if JIQ-Pod then
8:                      i1 ← Random(SeverAddress)
9:                      i2 ← Random(SeverAddress)
10:                     i ← Min(TasksInServer(i1, i2))
11:                 else
12:                     i ← Random(SeverAddress)
13:                 end if
14:             end if
15:             Task ⇒ Server(i)
16:         end if
17:         if Receive Request from Server then
18:             Iqueue.append(Request.Address)
19:         end if
20:     end while
21: end procedure
```

Algorithm 3 Pseudo-code of servers operating mode

Ensure: $d = 2$ and all parts of the system operate at the same time.

```
1:  procedure SERVER(SchedulerAddress)
2:      while True do
3:          Keep listening and receiving
4:          if No Tasks then
5:              Request ←' JoinIqueue' + Address
6:              i ← random(SchedulerAddress)
7:              if JIQ-E and len(Scheduler(i)) > 0 then
8:                  i ← random(SchedulerAddress)
9:              end if
10:             Request ⇒ Scheduler(i)
11:         else
12:             Process Task
13:             Task.message ← Task.message +' CompletedTime'
14:             Record Task.mess
15:         end if
16:     end while
17: end procedure
```

Chapter 6

Conclusions and Future Work

6.1 Contribution

The contributions of my thesis are:

1. We have proposed two new JIQ algorithms, namely, JIQ-NE and JIQ-E. These two new algorithms are based on the combination of JIQ algorithm and Pod algorithm.

2. We have shown the detailed models based on their specific algorithms. We have defined the models one by one and made figures to illustrate how they work.

3. We have analyzed the model numerically by deriving the stationary distribution, mean response time, and tail distribution with mean-field approximation.

4. We have conducted large-scaled simulations, and compared the numerical and simulation results. The simulations are large-scaled with 50,000 servers and we have shown the delay performance, cost performance, I-queue rate and influence of r and d.

5. We have implemented our algorithms in the real industry environment. We realized four algorithms in Amazon EC2 platform, and the result is coincident with our numerical and simulation results.

Through simulations and simple implementation, we have demonstrated the unique advantage of JIQ-NE regarding the performance-cost trade-off.

6.2 Future Work

In this thesis, the three algorithms are based on JIQ algorithm, and each of these algorithms leverages Po*d* algorithm in one part. In the future, more parts of JIQ algorithm can leverage Po*d* to help make better selections. In addition, in my thesis, we used interactions as the cost metric. In future, more factors should be taken into consideration as the cost metrics and make a better cost trade-off.

Bibliography

[1] Amazon.com Inc. Amazon elastic compute cloud (amazon ec2). http://aws.amazon.com/ec2/. August 25, 2006. → pages 2, 3

[2] Google Inc. Google cloud platform. https://cloud.google.com/. April 7, 2008. → page 2

[3] Microsoft. Microsoft azure cloud computing platform & services. https://azure.microsoft.com/. February 1, 2010. → page 2

[4] Atilla Eryilmaz and R Srikant. Asymptotically tight steady-state queue length bounds implied by drift conditions. *Queueing Systems: Theory and Applications*, 72(3–4):311–359, December 2012. → page 3

[5] Michael Mitzenmacher. *The Power of Two Choices in Randomized Load Balancing*. PhD thesis, U. C. Berkeley, 1996. → pages 3, 5

[6] Kay Ousterhout, Patrick Wendell, Matei Zaharia, and Ion Stoica. Sparrow: Distributed, low latency scheduling. In *Proc. of SOSP*, pages 69–84, Farminton, Pennsylvania, November 3–6, 2013. → pages 3, 5

[7] Lei Ying, R Srikant, and Xiaohan Kang. The power of slightly more than one sample in randomized load balancing. In *Proc. of INFOCOM*, pages 1131–1139, Hong Kong, April 26 – May 1, 2015. → pages 3, 5

[8] Yi Lu, Qiaomin Xie, Gabriel Kliot, Alan Geller, James R. Larus, and Albert Greenberg. Join-idle-queue: A novel load balancing algorithm for dynamically scalable web services. *Performance Evaluation*, 68(11):1056–1071, November 2011. → pages 3, 5

[9] Michael Mitzenmacher. Analyzing distributed join-idle-queue: A fluid limit approach. In *Proc. of Allerton*, pages 312–318, Monticello, IL, September 27–30, 2016. → pages 4, 5

[10] Chunpu Wang, Chen Feng, and Julian Cheng. Distributed join-the-idle-queue for low latency cloud services. *IEEE/ACM Transactions on Networking*, 26(5):2309–2319, October 2018. → pages 4, 5, 6, 9, 32

[11] Chunpu Wang, Chen Feng, and Julian Cheng. Randomized load balancing with a helper. In *Computing, Networking and Communications (ICNC), 2017 International Conference on*, pages 518–524, Silicon Valley, USA, 2017. IEEE. → page 5

[12] Alexander L. Stolyar. Pull-based load distribution in large-scale heterogeneous service systems. *Queueing Systems*, 80(4):341–361, 2015. → page 5

[13] Alexander L. Stolyar. Pull-based load distribution among heterogeneous parrallel servers: The case of multiple routers. Technical report, Industrial and Systems Engineering, Lehigh University, 2015. → page 5

[14] Yiqiu Fang, Fei Wang, and Junwei Ge. A task scheduling algorithm based on load balancing in cloud computing. In *International Conference on Web Information Systems and Mining*, pages 271–277. Springer, 2010. → page 5

[15] Shu-Ching Wang, Kuo-Qin Yan, Wen-Pin Liao, and Shun-Sheng Wang. Towards a load balancing in a three-level cloud computing network. In *Computer Science and information technology (ICCSIT), 2010 3rd IEEE International Conference on*, volume 1, pages 108–113, Chengdu, China, 2010. IEEE. → page 5

[16] Jinhua Hu, Jianhua Gu, Guofei Sun, and Tianhai Zhao. A scheduling strategy on load balancing of virtual machine resources in cloud computing environment. In *Parallel Architectures, Algorithms and Programming (PAAP), 2010 Third International Symposium on*, pages 89–96. IEEE, 2010. → page 5