Distributed Load Balancing in a Multiple Server System by Shift-invariant Protocol Sequences

ZHANG, Yupeng

A Thesis Submitted in Partial Fulfilment

of the Requirements for the Degree of

Master of Philosophy

in

Information Engineering

The Chinese University of Hong Kong

September 2013

Abstract of thesis entitled:

Ideally, many application systems for distributed users should be designed without requiring a centralized controller, for example cloud computing or wireless sensor networks. A fundamental challenge to developing distributed algorithms for these systems is load balancing, which is the focus of study in this thesis. A common feature of these distributed algorithms is that routing decisions should be derivable without requiring much information from the system, probabilistic routing is one example coming to mind. In this thesis, we propose a new routing strategy based on the idea of *shift-invariant protocol sequences*. We study this load balancing approach in the context of a queuing model of multi-server system. Our model and strategy can be applied to many practical systems, including wireless networks. Numerical studies were carried out to compare our strategy with other routing strategies such as probabilistic routing and random sequences routing. The results show that the proposed algorithm has better performance than these strategies. We also implement a simulation for multiple server system with practical assumptions such as Poisson job arrival time and exponential job service time. Different strategies are applied in this system and numerical results show that our strategy has better performance than others. To overcome some limitations of shift-invariant protocol sequences, new sets of protocol sequences based on *Chinese Remainder Theorem* are used and numerical results show that

its performance is suboptimal.

Keywords - Wireless sensor network; Multiple server system; Distributed load balancing; Shift- Invariant Protocol Sequences

Submitted by ZHANG, Yupeng

for the degree of Master of Philosophy

at the Chinese University of Hong Kong in June 2013

論文摘要：理論上，中央控制器不應該出現在許多擁有分散用戶的實際系統中，例如雲計算和無線傳感器網絡。為這些系統設計分佈式算法的一個重要的問題是如何做到負載平衡，而這一點正是本篇論文所要研究的。這些分佈式算法的共同點是不需要大量的系統信息，就可以完成任務分配，按概率分配正是其中的一種算法。在本篇論文中，我們根據**移位不變性協議序列**的思想提出了一個新的任務分配算法。本文在一個多服務器系統的排隊模型中去研究這個負載平衡的方法。這個模型和算法可以應用到很多包括無線網絡在內的實際系統中。本文通過數字模擬對比了我們的算法和其他算法，比如按概率分配和隨機數列分配。結果顯示這個算法比其他算法有更好的性能。同時我們也建立了一個有著跟實際系統類似的假設的多服務器系統模擬程序，這些假設包括泊松分佈的任務到達時間和指數分佈的任務需求服務時間。在這個模擬程序中，我們應用了多個算法，結果顯示我們的算法同樣有更好的性能。為了克服一些移位不變性協議序列的限制條件，我們應用了基於**中國餘數定理**而得來的新的協議序列。數字模擬結果顯示它的性能和最優方法很接近。

關鍵詞：無線傳感器網絡；多服務器系統；分佈式負載平衡算法；移位不變性協議數列

Acknowledgement

I would like to express my gratitude to my supervisor Professor Wing Shing Wong for the useful comments, remarks and engagement through the learning process of this master thesis. I can’t say thank you enough for his tremendous support and help. Without his encouragement and guidance this project would not have materialized.

Table of Contents

I. Introduction

1. Objectives

The idea of employing deterministic *protocol sequences* instead of probabilistic random accessing is first proposed in [1] for collision channel without feedback in wireless communication. A protocol sequence is a periodic binary sequence assigned to each user, which succinctly captures the information of when a user can transmit (sequence values equal to 1) and when to remain silent (sequence value equal to 0). The main property of protocol sequence is that even users are not synchronized and their relative time offsets are unknown to each other, the number of *collisions* in one period is bounded and certain level of throughput is guaranteed for each user in all possibilities of relative time offsets. Detail knowledge of protocol sequences will be introduced in the following section.

Protocol sequences can also be applied to other systems where there is no communication among users to replace probabilistic strategies. The objective of this thesis, in particular, is to apply the idea of protocol sequences into distributed load balancing problems where users try to distribute incoming jobs evenly to different servers without communication among each other. Similar to wireless communication, probabilistic routing strategies are well studied in previous works and background of load balancing problems will be presented in section I.3. We will show that using routing strategies derived from protocol sequences, the

throughput of all users can be guaranteed in all situations and the performance are improved compared to probabilistic routing strategies.

2. Background of Protocol Sequences

Protocol sequences are first proposed in [1] to be applied in collision channel without feedback in wireless communication. In wireless communication, many researches are done about random accessing due to its simplicity and effectiveness in different systems and applications [2]–[6]. In a shared channel, when users transmit without communication, collisions will occur generally. For system simplicity, it is desirable to have a simple multiple access protocol which does not require stringent time synchronization, frequent channel monitoring, and complicated processing such as backoff algorithm or random number generation [1]. The result will be particularly useful in emerging communication systems such as impulse radio [7] and wireless sensor networks [8] in which well-coordinated transmissions and time offsets may be too costly to devices with constrained resource. The idea of using protocol sequence to define random accessing in a collision channel without feedback is proposed in [9]. In this case, senders are not synchronized and their relative time offsets are unknown to each other due to a lack of feedback link. Besides, packet retransmission and backoff mechanisms are not employed. Each sender will just transmit packets at the times governed by his own protocol sequence.

A time-slotted system is considered in [1], consisting of a number of transmitters and one receiver. A user sends a packet within the boundaries of a time slot. The packet is received successfully only when exactly one user is active

at that time slot. A collision occurs when two or more users transmit simultaneously and the packets are assumed unrecoverable. If no user transmits in a time slot, that time slot is idle. Each user repeats his assigned binary protocol sequence periodically, and transmits a packet if and only if the value of the protocol sequence at that time slot equals one. The transmission schedule is independent of the data being sent and there is no cooperation among the users.

The advantages of employing deterministic protocol sequences instead of probabilistic random accessing is that theoretically, we can derive the zero-error capacity region for the collision channel without feedback when deterministic protocol sequences are employed. Provided that the protocol sequences are properly designed, it can be guaranteed that, with probability one, the throughput of each user is greater than a positive constant regardless of their relative delay offsets.

Under the model of collision channel without feedback, Massey and Mathys [9] have shown that a reliable multi-access communication is indeed achievable and a corresponding scheme with carefully designed protocol sequences was presented in [9]. More general sequence constructions using constant-weight cyclically permutable codes are reported in [10], [11] afterwards. A recent survey is also presented in [12] on coding for multiple access channel without feedback. Built on the concept of prime sequences, a family of protocol sequences, called wobbling sequences [1], is designed to support multi-rate service and a large

number of active users. They are suitable to serve as random access protocols, in particular for applications in some wireless ad hoc or sensor networks.

Optimal protocol sequence sets are proposed in [13]. The “cross correlation” of these sequences which is defined as the number of collisions of the sequences in [1], are the same no matter how they are shifted and the variance of the throughput is 0. This kind of protocol sequence is called shift invariant protocol sequences (SIS). In section IV, we will employ this SIS protocol sequence to the distribute load balancing problem.

3. Background of Load Balancing Problems

Load balancing problem has been studied in many previous works. Chow and Kohler [14] present a queuing model for a heterogeneous multiple server system. In the model, an arriving job is routed by a job dispatcher to one of parallel servers. Different routing strategies are also studied in [14], classified into two categories: deterministic and nondeterministic. As shown in figure 1, in deterministic strategies, an incoming job is sent to a particular server to minimize or maximize the expected performance of a related criterion function, such as minimizing response time, minimizing system time or maximizing throughput. In nondeterministic strategies in figure 2, an arriving job is sent to a server with certain probability, where the routing decision is usually based on independent probability distinctions. Performance of these strategies are analyzed and compared in [14].

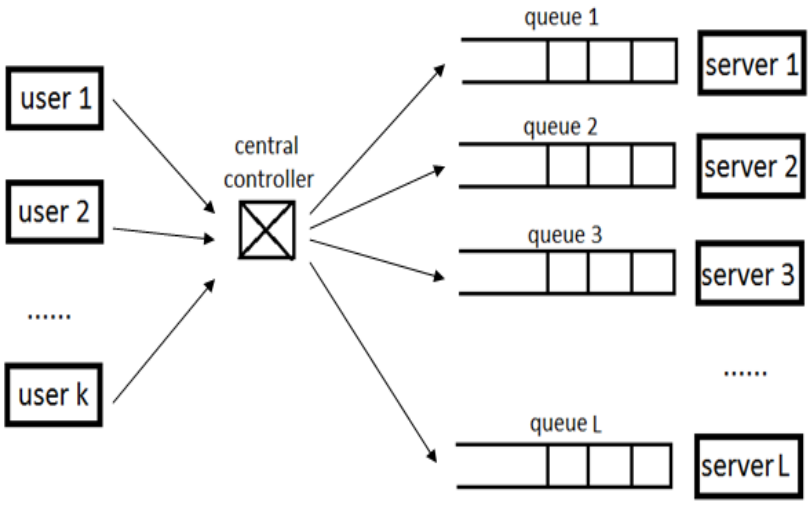

Fig.1 Deterministic routing

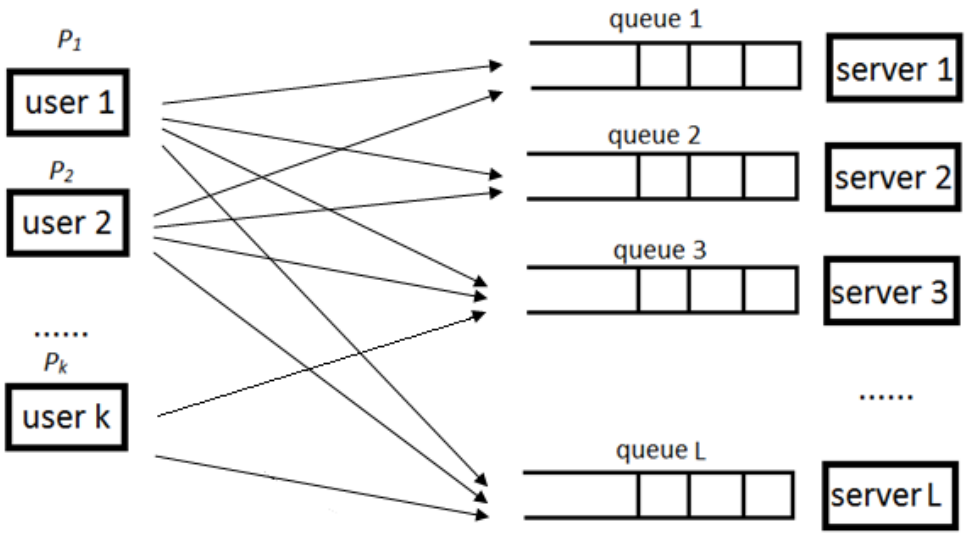

Fig.2 Nondeterministic routing

Chow and Kohler [14] show that deterministic strategies have better performance than nondeterministic strategies. However, in deterministic strategies, there must be a central controller to optimize their criterion function and the computational complexity is usually high, especially in systems with large number of servers. On the contrary, in nondeterministic strategies, the packets are routed based on a probabilistic routing matrix, which can be distributed to users who are sending jobs to the multiple server system and no central controller is needed. In this thesis, we are focus on the latter class of problems, which can be defined as "distributed load balancing".

Load balancing problems can be viewed as a trade-off between performance and the cost of information. As shown in figure 3, probabilistic routing and deterministic routing with central controller lie on the two ends of this trade-off. Probabilistic routing only requires information about the number of servers, but

the throughput of this strategy is low. On the contrary, central controller requires all information about the system, including number of users, number of servers, packets arrival time, packets service time and queuing patterns of all servers and it can reach high throughput in turn. Researches about the load balancing in recent years can also be viewed in two ways.

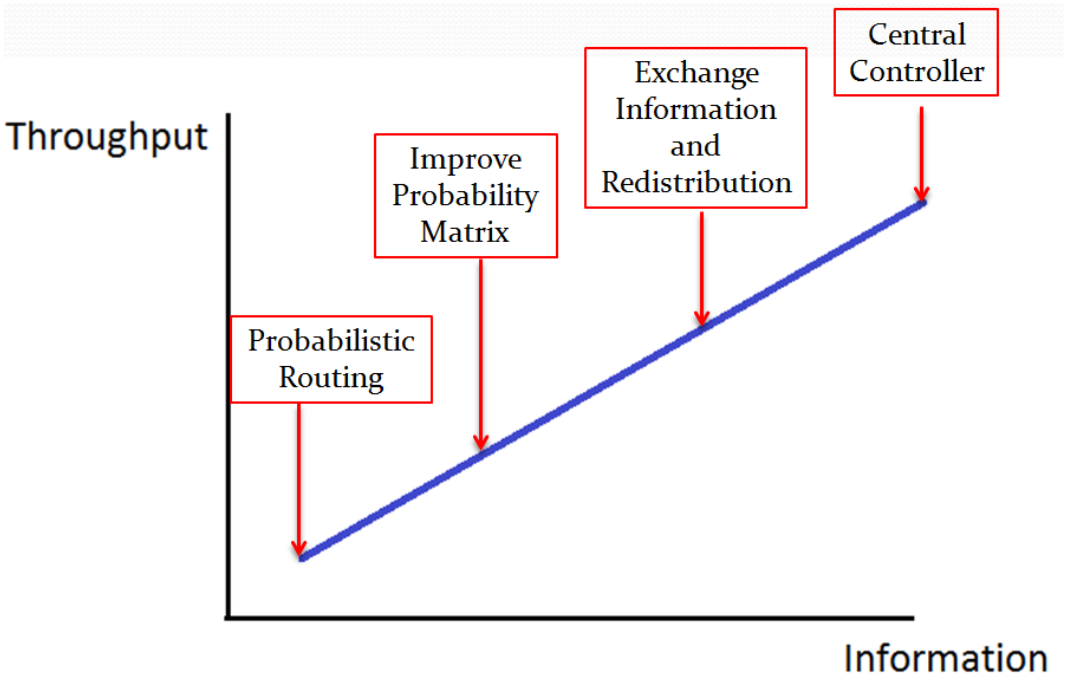

Fig.3 Trade-off between throughput and information

Ni and Hwang [15] propose a recursive probabilistic routing algorithm. It uses parameters of the system such as job incoming rate and service rate to form a nonlinear programming problem with linear constraints. An optimal probabilistic load balancing algorithm is proposed to solve this nonlinear programming problem. The proposed load balancing method is proven globally optimum in the sense that it results in a minimum overall average job response time on a

probabilistic basis.

Some other previous works such as [16] are focused on how to exchange information among users and servers to distribute jobs evenly to all servers. It can be viewed as estimating the calculation of central controller. A pre-emptive load-balancing algorithm is proposed in [16] for a multicomputer having homogeneous node architecture with a communication network that completely connects all the nodes by allowing direct communication between any pair of nodes. New jobs may arrive at any node and jobs execute equally well on any node, independent of the node where they arrive. We also assume that no information about the execution time of the process is available. The load-balancing policy consists of two algorithms. The first, called the information exchange algorithm, is responsible for the continuous exchange of the load information between the processors. The second algorithm, called the processor load, is used by each processor to monitor its own local load on a continuous basis. This load is based on the ability of the processor to provide services to user's processes; thus it can be used to estimate the response time of that processor.

There are also many previous work about distributed load balancing for cloud computing. An algorithm called honeybee foraging algorithm is proposed in [17]. In this algorithm, idle servers follow one of two behavior patterns: a server that reads the *advert board* will follow the chosen advert, and then serve the request; thus mimicking harvest behavior. A server not reading the advert board reverts to

forage behavior; servicing a random virtual server's queue request. A server successfully fulfilling a request will calculate profitability of the just-served job and post it on the advert board with a probability. The completed server influences system behavior by comparing its calculated profit with the colony profit on the advert board, and then adjusts the probability to post and colony profit accordingly. Given a robust profit calculation method, this behavior pattern provides a distributed and global communication mechanism; ensuring "profitable" jobs appear attractive to and are allocated to available servers.

Biased random sampling algorithm is proposed in [18]. In this approach, the load on a server is represented by its connectivity in a virtual graph. The initial network is constructed with virtual nodes to represent each server node, with each in-degree mapped to the server's free resources or some measure of desirability. As such, a number (consistent with its available resources) of inward edges are created, connected from randomly-selected nodes. This approach creates a network system that provides a measure of initial availability status, which as it evolves, gives job allocation and usage dynamics. Edge dynamics are then used to direct the load allocation procedures required for the balancing scheme. When a node executes a new job, it removes an incoming edge; decreasing its in-degree and indicating available resources are reduced. Conversely, when the node completes a job, it follows a process to create a new inward edge; indicating available resources are increased again. In a steady state, the rate at which jobs arrive equals the rate at which jobs are finished; the network would have a static

average number of edges.

Active Clustering is considered in [19] as a self aggregation algorithm to rewire the network. This procedure is intended to group like (i.e. similar service type) instances together. Active Clustering consists of iterative executions by each node in the network: 1. At a random time point a node becomes an *initiator* and selects a *matchmaker* node randomly from its current neighbors; the only condition being that the *matchmaker* is of a different type. 2. The *matchmaker* node then causes a link to be formed between one of the *matchmaker's* neighbors that match the type of the *initiator* node. 3. The *matchmaker* removes the link between itself and the *initiator*.

In this thesis, a model is presented to study the distributed load balancing problem. There are three basic assumptions: (1) users independently distribute their tasks to servers according to pre-assigned binary sequences, which will be explained in following section; (2) no real-time synchronization is required; (3) there is no centralized controller after the initial sequences assignment. These assumptions usually hold in wireless networks without a centralized controller or base station and do not provide guarantee that users are synchronized. Based on these assumptions, a new strategy is proposed. It can be shown that the performance of the proposed strategy is better than probabilistic routing.

The strategy has many applications in practical systems. For example, in some wireless sensor network, users transfer data through different frequencies.

As long as several users are transmitting on the same frequency, packet collision may occur and additional techniques such as slotted aloha should be used to resolve the contention and the effective transmitting time will increase. The different frequencies can be viewed as different "servers" and the extra transmitting time when frequency collision occurs can be estimated by queuing model. Thus, the frequency allocation in a wireless sensor network can be analyzed in the context of load distribution in our multi-server model, and our strategy can help distribute different frequencies to users evenly in order to reduce the transmission time.

Section II presents the model of distributed load balancing problems. Section III analyzes the model and defines the optimization function to make the system performance robust. Section IV applies the shift invariant protocol sequences in the distributed load balancing problems. Section V shows the numerical results and the comparison among different strategies. Section VI compares performances of different strategies in a practical simulation. Section VII introduces a suboptimal strategy to overcome some limitations of shift invariant protocol sequences and shows its performance in the simulation. Section VIII concludes the thesis.

II. System Model

Following the model of multiple server system [14], there are L servers in the multiple server system and K users are sending jobs to the system. We assume that each user sends the same number of jobs to servers each time slot and we use N to denote it. The pattern for one user at one time slot can be represented by a binary sequence, where the positions of 1s denote the servers selected to send jobs to by this user. For example, if there are 6 servers, user i sends 2 jobs at time slot t to server 1 and 2, we can use the sequence $s_t^i = (1,1,0,0,0,0)$ to represent the case.

In order to distribute the jobs evenly, each user is required to send jobs to different servers in different time slots, until all servers are used once. Therefore, each user will follow a series of sequences to send jobs and the positions of 1s in these sequences should not overlap with each other. After using all servers once, the user will go back to follow the same pattern of sequences and thus the series is periodical. We define the period as $P = L/N$ and to simplify the model, we assume L is divisible by N. We define this series of sequences as S_i for user i

$$S_i = (s_1^i, s_2^i, ..., s_P^i, s_1^i, s_2^i, ..., s_p^i, ...). \quad (1)$$

For example, suppose there are 6 servers and 2 users and each user sends 2 jobs every time slot. User 1 selects server 1 and 2 at the first time slot, server 3 and 4 at the second time slot and server 5 and 6 at the third time slot. This scenario can be

represented as $S_1 = ((1,1,0,0,0,0),(0,0,1,1,0,0),(0,0,0,0,1,1),.....)$. Similarly, user 2 may follow $S_2 = ((1,0,0,1,0,0),(0,1,0,0,1,0),(0,0,1,0,0,1),.....)$.

Figure 4 is an illustration of the system. The two users follow sequence sets S_1 and S_2 correspondingly as mentioned above for two time slots to send jobs to 6 servers and jobs queue in front of each server.

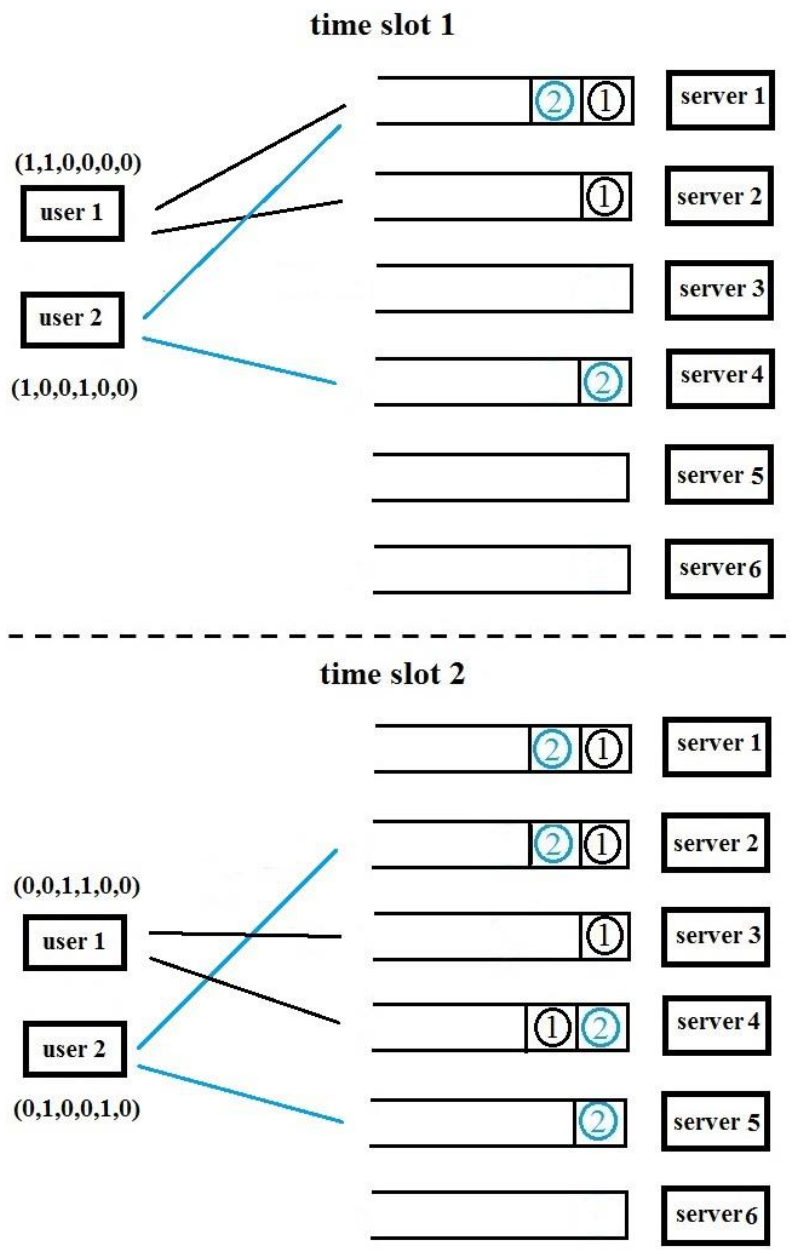

Fig.4 Illustration of multi-server system model

As there is no communication among different users, jobs from different users may arrive at the same server in one time slot. If more than one jobs are sent to the same server in a time slot, we assume the queuing order for these jobs are determined randomly. In our model, we only consider the simplest case where one server will serve exactly one job per time slot. Under this assumption, a stationary condition is that $K \times N \le L$ and $P = L / N \ge K$.

Moreover, as the users are independent, the time to start sending jobs are not synchronized and thus we need to introduce a time difference for each user when the whole system starts. We define the rotation RS_i of S_i by

$$RS_i = (s_2^i, s_3^i, ..., s_P^i, s_1^i, s_2^i, s_3^i, ...). \quad (2)$$

Then we introduce the concept of a time difference $\tau_i \in [0, P-1]$ for user i so that when the system starts, user i follows τ_i rotations of S_i

$$R^{\tau_i} S_i = (s_{\tau_i+1}^i, s_{\tau_i+2}^i, ..., s_P^i, s_1^i, s_2^i, ...). \quad (3)$$

In the previous example, if $\tau_1 = 0$ and $\tau_2 = 0$, two users will just follow the sequences in S_1 and S_2. However, if $\tau_1 = 0$ and $\tau_2 = 2$, the sequences for user 2 will become $((0,0,1,0,0,1),(1,0,0,1,0,0),(0,1,0,0,1,0),.....)$ because user 2 already goes to the third sequence in S_2 when the system starts. Therefore, different combinations of $\tau = (\tau_1, \tau_2, ..., \tau_K)$ result in different queuing patterns for servers.

Under this system model, we can derive the average waiting time for all jobs

from all users in terms of sequence sets and time difference for the users. The average waiting time is an important parameter to describe the performance of the whole system.

III. Analysis and Results

For a particular combination of τ, we can determine the queuing pattern for all servers by the following notations. We use $s_t^i(l)$ to represent the value of position l in s_t^i. It is just the number of job user i send to server l at time t (either 1 or 0). Note that $s_t^i(l)$ is dependent of τ, and we use $s_t^i(l)$ instead of $s_t^i(l,\tau_1,\tau_2,...,\tau_K)$ here for short form; notations of $A_t(l)$ and $Q_t(l)$ which are defined later should be interpreted similarly. As one user only sends one job to a particular server in one period, we have

$$s_t^i(l)+s_{t+1}^i(l)+...+s_{t+P-1}^i(l)=1. \quad (4)$$

One user sends N jobs each time slot in total to all servers in one period, so

$$\sum_l s_t^i(l)=N. \quad (5)$$

We further use $A_t(l)$ to denote the number of jobs server l receives from all users at time t, thus $A_t(l)$ can be computed directly from $s_t^i(l)$.

$$A_t(l)=\sum_i s_t^i(l). \quad (6)$$

Similarly, we have

$$A_t(l)+A_{t+1}(l)+...+A_{t+P-1}(l)=K; \quad (7)$$

$$\sum_l A_t(l)=N\times K. \quad (8)$$

As $s_t^i(l)$ is periodical,

$$A_t(l) = A_{t+P}(l). \quad (9)$$

$Q_t(l)$ is defined as the number of jobs queuing in server l at time t, it can be expressed by the following formula, provided that we let new jobs come and get served immediately, then compute $Q_t(l)$ by

$$Q_{t+1}(l) = (Q_t(l) + A_t(l) - 1)^+. \quad (10)$$

$Q_t(l)$ is also periodical except for the first P time slots. The proof of this is given in Appendix A. That is,

$$Q_t(l) = Q_{t+P}(l). \text{ when } t \geq P \quad (11)$$

Finally, we can define an average waiting time for a random job by

$$\overline{T_w}(\tau_1, \tau_2, ..., \tau_K) = \frac{\sum_l \sum_{t=P+1}^{2P} Q_t(l, \tau_1, \tau_2, ..., \tau_K)}{P \times N \times K}. \quad (12)$$

As $Q_t(l)$ is computed under one particular combination of shifts as defined by τ, $\overline{T_w}(\tau_1, \tau_2, ..., \tau_K)$ can be represented as a function of τ.

We can use $\overline{T_w}(\tau_1, \tau_2, ..., \tau_K)$ to evaluate the performance of the whole system and a desirable goal is to minimize $\overline{T_w}(\tau_1, \tau_2, ..., \tau_K)$. Obviously, $\overline{T_w}(\tau_1, \tau_2, ..., \tau_K)$ can assume different values for different combinations of τ, for a fixed set of sequences allocated to users. Moreover, we cannot predict the time differences τ for all users in practice. Therefore, we try to find a set of sequences to minimize the possible maximum value of $\overline{T_w}(\tau_1, \tau_2, ..., \tau_K)$ among all combinations of time

differences τ. That is, our goal is to determine the

$$\arg\min_{S_1,S_2,...,S_K}(\max_{\tau}(\overline{T_w}(\tau_1,\tau_2,...,\tau_K))). \quad (13)$$

For example, suppose $S_1=((1,1,0,0,0,0),(0,0,1,1,0,0),(0,0,0,0,1,1),.....)$ and $S_2=((0,0,1,1,0,0),(0,0,0,0,1,1),(1,1,0,0,0,0),.....)$, when $\tau_1=0$ and $\tau_2=0$, no jobs will go to the same server at any time slot and thus the waiting time $\overline{T_w}(1,1)=0$. However, when $\tau_1=2$ and $\tau_2=4$, the sequences that two users follow are exactly the same and the waiting time becomes much worse $\overline{T_w}(1,3)=0.5$. To determine the optimal strategy, we only need to consider the worst case for each sequence set. For example, in this example, the worst case over all shifts is $\overline{T_w}(1,3)=0.5$ and this set of S_1 and S_2 is not the optimum.

To find the solution for the optimization problem (13), we rely on the following theorem:

Theorem: $\sum_{\tau_1,\tau_2,...,\tau_K}\overline{T_w}(\tau_1,\tau_2,...,\tau_K)=C$ where C is a constant for any sequences $S_1,S_2,...,S_K$.

The proof is as following.

We can compute the average waiting time $\overline{T_w}(\tau_1,\tau_2,...,\tau_K)$ from another direction. We define σ_i^l as the time slot at which server l receives a job from user i in one period P, which means

$$s^i_{\sigma_i^l}(l)=1,\sigma_i^l=1,2,...,P$$

The jobs sent to server l in one period can be represented by the sequence $(\sigma_1^l, \sigma_2^l, ..., \sigma_K^l)$ and $A_t(l)$ can be computed by counting the number of ts that appear in $(\sigma_1^l, \sigma_2^l, ..., \sigma_K^l)$. $Q_t(l)$ can further be computed from $A_t(l)$ by (8). Therefore, the total waiting time for jobs on server l is a function of σ_i^l.

$$T_w(l, \tau_1, \tau_2, ..., \tau_K) = g(\sigma_1^l, \sigma_2^l, ..., \sigma_K^l) \tag{16}$$

$\overline{T_w}(\tau_1, \tau_2, ..., \tau_K)$ can be further computed by

$$\overline{T_w}(\tau_1, \tau_2, ..., \tau_K) = \frac{\sum_l T_w(l, \tau_1, \tau_2, ..., \tau_K)}{L \times K} \tag{17}$$

Therefore,

$$\sum_{\tau_1, \tau_2, ..., \tau_K} \overline{T_w}(\tau_1, \tau_2, ..., \tau_K) = \sum_{\tau_1, \tau_2, ..., \tau_K} \frac{\sum_l T_w(l, \tau_1, \tau_2, ..., \tau_K)}{L \times K}$$
$$= \frac{\sum_{\tau_1, \tau_2, ..., \tau_K} \sum_l g(\sigma_1^l, \sigma_2^l, ..., \sigma_K^l)}{L \times K}$$

For example, if $S_1 = ((1,1,0,0,0,0),(0,0,1,1,0,0),(0,0,0,0,1,1),.....)$ and $S_2 = ((0,1,0,0,1,0),(0,0,1,0,0,1),(1,0,0,1,0,0),.....)$, when $\tau_1 = 0$ and $\tau_2 = 0$, server 1 will receive one job from user 1 at the first time slot and one job from user 2 at the third time slot, thus $\sigma_1^1 = 1$ and $\sigma_2^1 = 3$.

As user i will send exactly 1 job to server l in one period P, we can always find a $\tau_i = \tau$ where user i sends the job to server l at the first time slot, which means $\sigma_i^l = 1$. Then if $\tau_i = \overline{\tau - 1}$, $\sigma_i^l = 2$. In this way, as the domain size of τ_i and σ_i^l are both P, we can always find exactly one τ_i correspond to one value

of σ_i^l. Therefore τ_i and σ_i^l are one to one mapping. Thus, summing up τ_i from 1 to P is the same as summing up σ_i^l from 1 to P. Meanwhile, time differences for different users are independent and σ_i^l is only affected by τ_i, we have

$$\frac{\sum_{\tau_1,\tau_2,...,\tau_K}\sum_{l} g(\sigma_1^l,\sigma_2^l,.....,\sigma_K^l)}{L\times K} = \frac{\sum_{l}\sum_{\tau_1}\sum_{\tau_2}...\sum_{\tau_K} g(\sigma_1^l,\sigma_2^l,.....,\sigma_K^l)}{L\times K} = \frac{\sum_{l}\sum_{\sigma_1^l}\sum_{\sigma_2^l}...\sum_{\sigma_K^l} g(\sigma_1^l,\sigma_2^l,.....,\sigma_K^l)}{L\times K}$$

Which is the same no matter what sequences are used.

For example, if $S_1=((1,1,0,0,0,0),(0,0,1,1,0,0),(0,0,0,0,1,1),.....)$ and $S_2=((0,1,0,0,1,0),(0,0,1,0,0,1),(1,0,0,1,0,0),.....)$, when we fix τ_2 and only consider server 1, we can find that $\sigma_1^1=1$ when $\tau_1=0$, $\sigma_1^1=2$ when $\tau_1=2$ and $\sigma_1^1=3$ when $\tau_1=1$. Therefore, when summing up all possibilities for τ_1, it is the same as summing up all possible values for σ_1^1. Similarly, all values of σ_2^1 are included once when summing up τ_2. Patterns on other servers can also be computed in this way and thus

$$\sum_{\tau_1,\tau_2}\overline{T_w}(\tau_1,\tau_2)=\sum_{\tau_1=0}^{2}\sum_{\tau_2=0}^{2}\overline{T_w}(\tau_1,\tau_2)=\sum_{l=1}^{6}\sum_{\sigma_1^l=1}^{3}\sum_{\sigma_2^l=1}^{3} g(\sigma_1^l,\sigma_2^l)$$

$$=\sum_{l=1}^{6}[g(1,1)+g(1,2)+g(1,3)+g(2,1)+g(2,2) +g(2,3)+g(3,1)+g(3,2)+g(3,3)] \quad (18)$$

When we change sequences to $S_1 = ((0,1,0,0,0,0),(0,0,1,0,0,1),(1,0,0,1,0,0),.....)$ and $S_2 = ((1,0,1,0,0,0),(0,1,0,1,0,0),(0,0,0,0,1,1),.....)$, we can find that the relationship changes to $\sigma_1^1 = 1$ when $\tau_1 = 2$, $\sigma_1^1 = 2$ when $\tau_1 = 1$ and $\sigma_1^1 = 3$ when $\tau_1 = 0$. However, when we sum τ_1 and τ_2 up, we still get the same expression as (18) and the result is irrelevant to what sequences used. Therefore, $\sum_{\tau_1,\tau_2,...,\tau_K} \overline{T_w}(\tau_1,\tau_2,...,\tau_K) = C$ where C is a constant for any sequences $S_1, S_2, ..., S_K$.

Based on this theorem, as the sum of $\overline{T_w}(\tau_1,\tau_2,...,\tau_K)$ is constant, a shift invariant sequence set that yields the same value for all τ, if it exists, is optimal. In the next section, we will show that such solution exists for some system parameters.

IV. Applications of Shift Invariant Protocol Sequences

The conditions of the collision channel is quite similar to the distributed load balancing model, so we try to apply the protocol sequences to solve the optimization problem in our model.

As the cross correlation of shift invariant protocol sequences are the same among different shifts, and the response time is related to the cross correlation, we find that shift invariant protocol sequence is one of the sequence sets that can make the response time equal among all shift patterns. In other words, shift invariant protocol sequence is one of the optimal solutions to (13).

Figure 5 shows one set of shift invariant protocol sequences and the resulting routing patterns for different users. Table I shows the response time for all τ s, which is constant. We further studied the case when all users use the sequences for user 3 in previous case. Although the respond times for some τ s are smaller than that of shift invariant sequences, the maximum one is much larger. Therefore, shift invariant sequences set is one of the optimal solutions to (13) and thus has a robust performance.

Shift Invariant Protocol Sequences

(1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0)
(1,1,1,0,0,0,0,0,0,1,1,1,0,0,0,0,0,0,1,1,1,0,0,0,0,0,0)
(1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)

User 1:

(1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0)
(0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0)
(0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1)

User 2:

(1,1,1,0,0,0,0,0,0,1,1,1,0,0,0,0,0,0,1,1,1,0,0,0,0,0,0)
(0,0,0,1,1,1,0,0,0,0,0,0,1,1,1,0,0,0,0,0,0,1,1,1,0,0,0)
(0,0,0,0,0,0,1,1,1,0,0,0,0,0,0,1,1,1,0,0,0,0,0,0,1,1,1)

User 3:

(1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)
(0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0)
(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1)

Fig.5 Example of shift invariant protocol sequence. Adapted from [20]

Time Differences	(0,0,0)	(0,0,1)	(0,0,2)	(0,1,0)	(0,1,1)	(0,1,2)	…	(2,2,2)
Average waiting time for shift invariant sequences	0.444	0.444	0.444	0.444	0.444	0.444	…	0.444
Average waiting time for same sequences	1	0.333	0.666	0.333	0.666	0	…	1

Table 1 Average Response Time for Different τ. Adapted from [20]

However, the existence of shift invariant protocol sequences depends on the value of server number L and period P. For P users, the minimum value of L is P^P.

When the number of servers does not satisfy the requirements of shift invariant sequences, we propose a simple way to generate a suboptimal sequence set. Note that if there are two sequence sets S_1 and S_2 with same user number K and period P, suppose their server numbers are L_1 and L_2, we can append S_2 at the end of S_1 and let them shift in their own range with period P. In this way, we construct a new sequence set for $L_1 + L_2$ servers with the same user number K and period P.

Therefore, if there is a shift invariant sequence set S_L for server number L, we can generate optimal sequence set for n L (where n is an integer) by adding n S_L together. Based on it, for any server number $L \geq P^P$, we can firstly find the shift invariant sequence set S_P for P^P servers, and then define a minimal shift invariant sequence set for L by adding n_L S_P together where n_L is the maximal integer that satisfies $n_L \times P^P \leq L$. For the remaining part, we currently append a sequence set with same sequences for all users.

Figure 6 shows an example. For L = 30, K = 3 and P = 3, we find the shift invariant sequence set for L = 27 and append the same sequences for L = 3 at the end of sequences allocated to each user.

L = 30

User 1:

(1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,**1,0,0**)
(0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,**0,1,0**)
(0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,**0,0,1**)

User 2:

(1,1,1,0,0,0,0,0,0,1,1,1,0,0,0,0,0,0,1,1,1,0,0,0,0,0,0,**1,0,0**)
(0,0,0,1,1,1,0,0,0,0,0,0,1,1,1,0,0,0,0,0,0,1,1,1,0,0,0,**0,1,0**)
(0,0,0,0,0,0,1,1,1,0,0,0,0,0,0,1,1,1,0,0,0,0,0,0,1,1,1,**0,0,1**)

User 3:

(1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**1,0,0**)
(0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,**0,1,0**)
(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,**0,0,1**)

Fig.6 sequences allocated to users (L = 30, K = 3, P = 3). Adapted from [20]

V. Numerical Results

1. Simulation Settings

The settings of our simulation are the same as mentioned in section II. There are 27 servers and 3 users in the system and each user sends 9 jobs to different servers every time slot. We further introduce a parameter T_s to denote how long a single job needs to be served. It is assumed to be one time slot in previous part and it can vary from 1 to 1/3 in our simulations. (Jobs will not cumulate if T_s is less than 1/3, because even if a server receives 3 jobs in one time slot, it can serve them all in that time slot). We compute average response time for one job which equals average waiting time defined by (12) plus the service time of one job. Three routing strategies are compared in our simulation: probabilistic routing, random sequence routing and shift invariant sequence routing.

In probabilistic routing, each user selects 9 different servers out of 27 with equal probabilities among all servers in each time slot to send jobs. Users behave independently among different time slots and their routing strategies are not affected by previous selections.

In random sequence routing, sequence sets defined by (1) are generated randomly and allocated to users at the beginning. Users send jobs following the sequence sets in one period P = 3 time slots. Then new sequence sets will be generated randomly in next period. The effect of this strategy is equivalent to probabilistic routing with memory, which means that users send jobs randomly to

all servers at the beginning, memorize the selected servers and send jobs randomly to remaining servers in next time slots.

In shift invariant routing, we allocate the shift invariant sequences for L = 27, K = 3 and P = 3, as shown in figure 2, to users instead of random sequences in previous case to do routing.

2. Results

Figure 7 shows the comparison result. The triangle line is probabilistic routing and the performance is the worst. The system is even not robust as T_s goes to 1 and the average response time tends to infinity. Diamond line represents the performance of random sequence routing. We can see its average response time is larger than that of shift invariant sequences routing, which is denoted by star line.

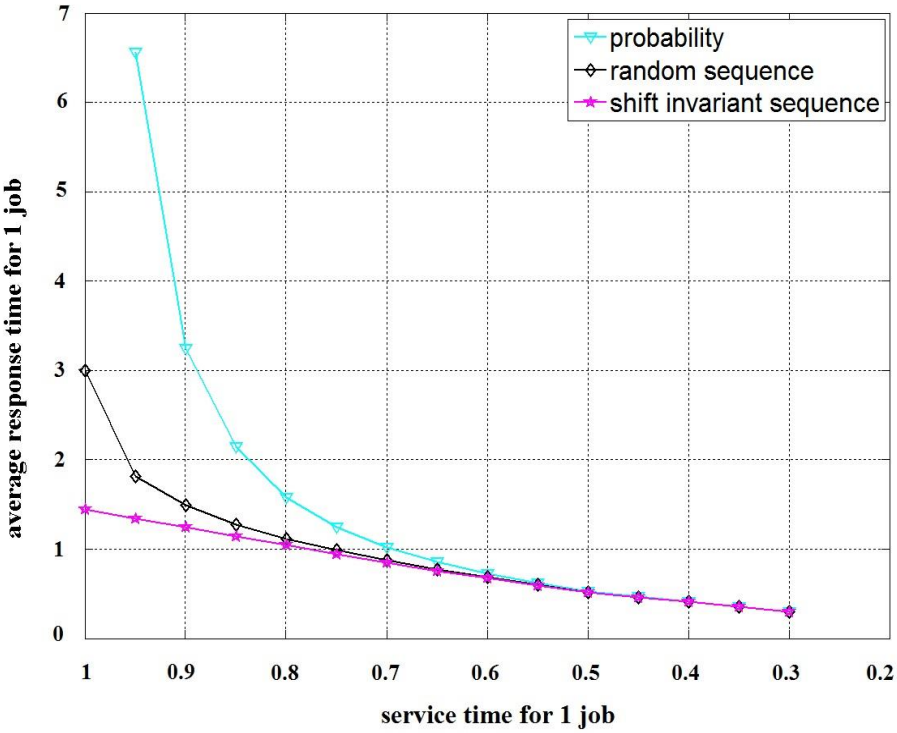

Fig.7 comparison of average response time. Adapted from [20]

In practice, users usually divide one piece of job into several parts and distribute them to servers and the total waiting time is determined by the part that returns latest. It is exactly the property of protocol sequences to guarantee the

throughput of the worst case. To simulate this, in previous system we further compare the maximum respond time for jobs sent in one time slot from one user. Figure 8 shows that the shift invariant sequences routing performs much better than others in this case.

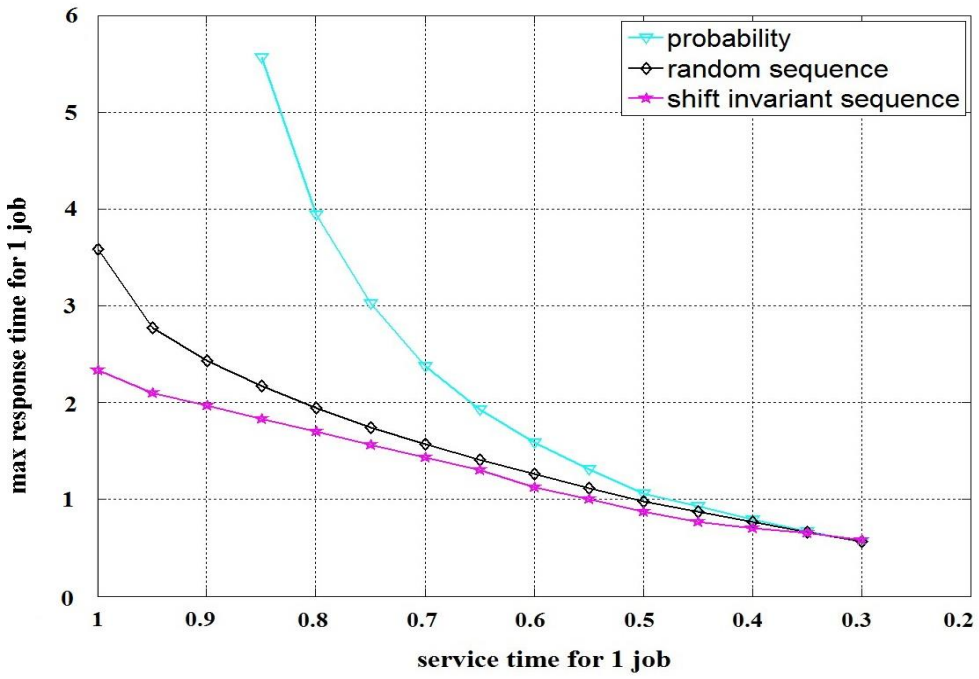

Fig.8 comparison of maximal response time in one time slot. Adapted from [20]

VI. Practical Simulation

1. Simulation Settings

In section II, a model is proposed to study the distributed load balancing problems and optimization functions are provided to find the best routing strategy. However, the assumptions are quite strong in order to do mathematical derivations. Therefore, a simulation with more practical conditions is presented in this section.

The assumptions are as following:

1. The job arrival time for each user follows Poisson Process with parameter λ_i for user i.

2. The job service time follows exponential distribution with parameter μ.

3. There are 27 servers and 3 users. When a packet arrives at one user, it will divide the job into 9 pieces and send them to the servers based on different routing strategy. The response time is defined to be the time when all the pieces of this packet are served, which is the same scenario as the second simulation in section IV.

4. The arrival time and service time for all jobs are pre-generated and all routing strategies use this same set of data in one run to guarantee a fair comparison. The average response time for 10000 runs of simulation with different arrival and service time sets are taken and compared.

5. The ratio between the overall job incoming rate and the overall server capacity $\frac{\sum \lambda_i}{\mu}$ varies from 1 to 0.1.

Four routing strategies are compared in the simulation: probabilistic routing, random sequence routing, shift invariant protocol sequence routing and same server routing. The first three routing strategies are basically the same as the settings in section IV. The only difference is that in this simulation, the time is continuous instead of slotted, thus sequences are used only when new jobs arrive.

In same server routing, each user is just assigned 9 servers and it will always send jobs to them without shift. The reason to show its performance is that in the model in section II, the waiting time is 0 and the performance is the best if we use the same server routing. However, we argue that users should shift to different servers in practice. The reason is that in the previous model, the service time is equal to the job arrival interval and thus if one user always send jobs to the same set of servers, jobs do not cumulate. On the contrary, in this simulation, as well as in practice, job arrival time and service time are randomly distributed and it is highly possible that the former jobs are not completed when new jobs come. Therefore, if we use same server routing and other servers are idle in such case, the waiting time must be longer comparing to those routing strategies that shift the servers.

Another important issue is that in practice, the number of users is much larger

and we may assign the same set of sequences to a group of users. With same server routing, if this group of users are active simultaneously, the server is highly congested. However, if these users are assigned the same set of sequences, they still can distribute jobs to different servers and the response time is smaller.

2. Results

Figure 9 shows the result with the same job arrival rate for all users. The circle line is same server routing and the triangle line is probabilistic routing. Their response times are both much larger than other strategies. Diamond line represents the performance of random sequence routing. We can see its average response time is larger than that of shift invariant sequences routing, which is denoted by star line. The performance of all routing strategies tends to be the same as the ratio between job arrival rate and job service time $\frac{\sum \lambda_i}{\mu}$ goes to 0, because the system is not congested and collision seldom occurs. Moreover, the same server routing performs better when $\frac{\sum \lambda_i}{\mu}$ is small, because there is few collision from the same user as mentioned in section V.1.

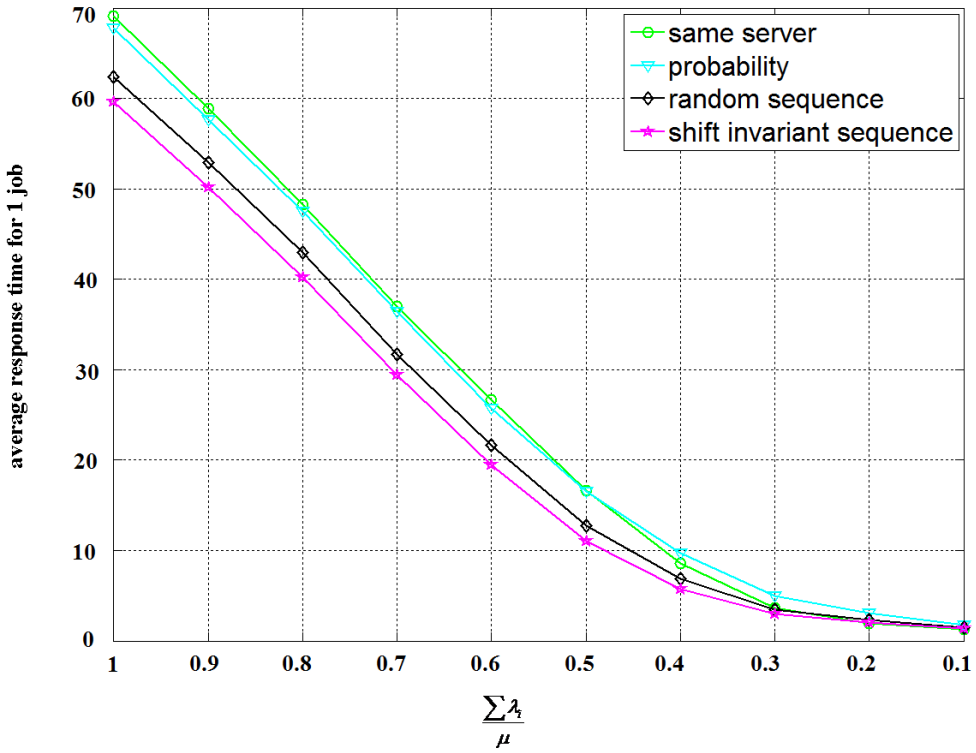

Fig.9 comparison of average response time with same λ

Figure 10 shows the result with different job arrival rate for different users. The ratio of arrival rate for 3 users is 1:2:3 and the overall arrival rate remains the same as in figure 9. We can see that in this case, same server routing performs even worse, because with different λ_i for different users, there are many jobs arrives at user 3 with largest arrival rate when the other two users are not active and same server routing cannot distribute jobs evenly to all users in this case. It is also shown that the shift invariant protocol sequence routing still performs much better than others in this case.

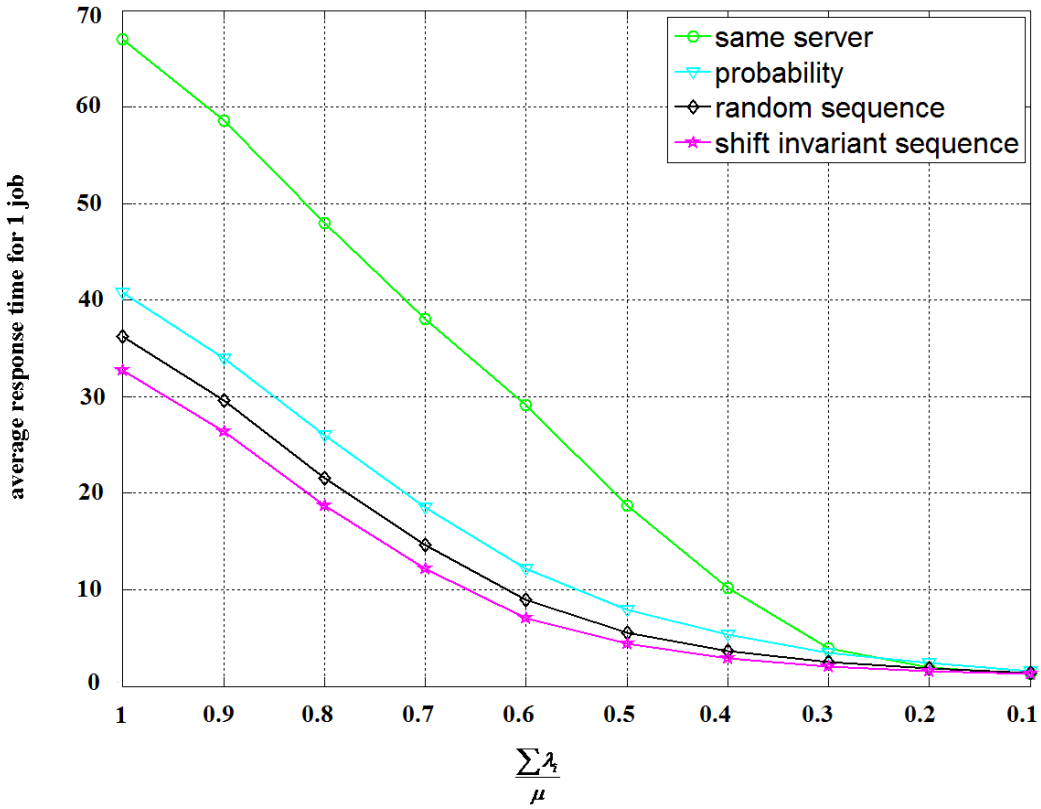

Fig.10 comparison of average response time with different λ

VII. Suboptimal Strategy

1. CRT Protocol sequences

As mentioned in section III, there is a limitation on the number of servers L and period P. To relax the limitation and to apply the routing strategy to arbitrary number of servers in practice, a new set of protocol sequences called CRT protocol sequences is presented in this section.

CRT Protocol Sequences

User 1: (1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0)

User 2: (1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0)

User 3: (1,0,0,1,0,0,0,1,0,0,0,1,0,0,1,1,0,0,1,0,0,0,1,0,0,0,1,0,0,1,0,0,1)

Fig.11 An example of CRT protocol sequences

CRT protocol sequence is proposed in [21], which is based on the bijective mapping between one-dimensional sequence and two-dimensional array by the Chinese Remainder Theorem (CRT). The cross-correlation, as defined in section I.2, between these sequences are highly concentrated around the mean value under different shifts of the sequences. It is equivalent to say that the differences among $\overline{T_w}(\tau_1, \tau_2, ..., \tau_K)$, defined by function (12), under all possible τ s are limited. Therefore, the CRT sequence is a suboptimal solution to the optimization function (13).

Another important property is that the relationship between number of servers

L and period P is $L = P \times Q$ where Q is a positive integer that is relative prime to P. Therefore, the limitation for the number of servers L is relaxed to $L = O(P^2)$, thus it is much easier to find this kind of sequences for given number of servers.

CRT protocol sequence can be generate by computing linear operations in modular arithmetic and the detail algorithm to generate such sequences can be found in [21]. An example of CRT protocol sequences for 15 servers and 3 users is shown in figure 11 and performance comparison among CRT protocol sequence and previous routing strategy will be shown in next section.

2. Numerical Results

The simulation with same assumptions as section V is done to compare the performance of the suboptimal strategy by CRT protocol sequences with other strategies. In this simulation, there are 30 servers and 3 users and we use the CRT sequence as shown in figure 11 to do routing. Other strategies including same server routing, probability routing and random sequence routing are the same as mentioned in section V and the job arrival rate for all users are the same.

Moreover, we compare the performance of a suboptimal routing strategy derived from shift invariant protocol sequences as shown in figure 6. The reason is that as mentioned above, the relationship between the number of servers L and period P for CRT sequences is $L = P \times Q$ where Q is a positive integer that is relative prime to P, while for SIS sequences, $L = P^P$. Therefore, we cannot find an L to satisfy both condition and there is no L that has both CRT sequences and SIS sequences. Thus we compare the performance of the suboptimal routing strategy derived from SIS sequences to give a rough idea of the differences between CRT and SIS sequences.

We can see in figure 12, the performances of CRT protocol sequence routing is slightly worse than the suboptimal SIS sequence, both better than that of all other strategies.

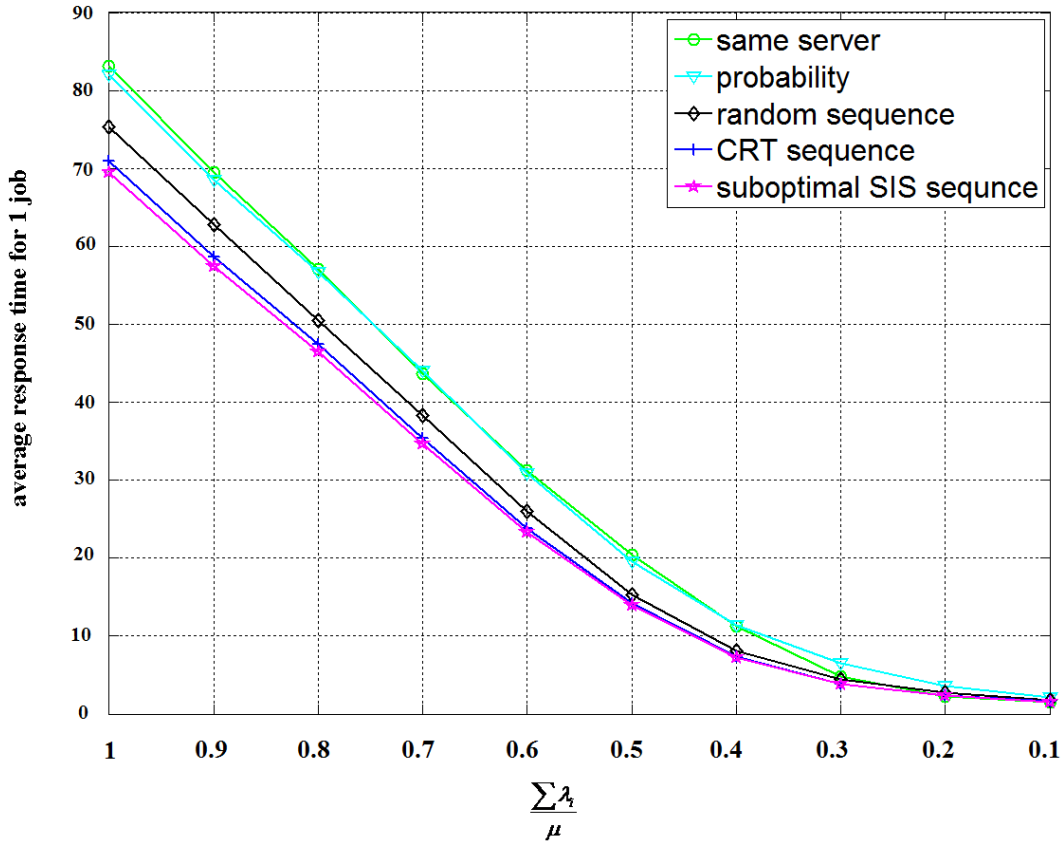

Fig.12 comparison of average response time

VIII. Conclusions

In this thesis, we present a model of distributed load balancing problem, which can be applied to many applications including frequency channel allocation in wireless sensor networks. We define an optimization problem in order to minimize the maximal response time for all combinations of time differences among the users. Under suitable technical conditions, we derive optimal solutions to the problem, which are based on shift invariant protocol sequences. Numerical results show that our algorithm performs better than other strategies such as probabilistic routing and random sequences. However, there are some limitations on the number of users. A simulation with practical conditions proves that our algorithm can improve the performance, as predicted in our mathematical model. A suboptimal routing strategy is also proposed based on CRT protocol sequences to relax the limitation on the number of users.

Further investigation about other optimal solutions for (13) is required to see whether there are better sequence sets that are easier to generate and have fewer limitations. Moreover, there is not always an optimal solution for every number of servers L and a practical routing algorithm for arbitrary number of servers need to be developed. A simplest version of routing algorithm is proposed in section III.2, but obviously we can achieve better performance by merging existing optimal and suboptimal sequences and it worth a closer look in the future.

Besides, it is common that users have different job arrival rates in practice

and employing protocol sequences with different duty factors may give better throughput comparing to the result in section V. Such sequences have yet not been studied in previous researches.

Another direction of future work is to study the performance of our load balancing algorithm in real applications such as frequency channel allocation in wireless sensor networks and cloud computing. Practical implementations suitable for those systems are required and routing algorithms can be developed accordingly.

APPENDIX A

Considering one particular server, I use $Q(t)$ to represent the queue length at time t and $A(t)$ to represent number of jobs arrive at time t.

Define $D(t)$ as the number of jobs served at time t. (At time t, new jobs come first, then get served immediately. $Q(t)$ is computed after that.)

$$Q(t) = Q(t-1) + A(t) - D(t); \tag{14}$$

$$D(t) = \begin{cases} 1 & Q(t-1) + A(t) > 0 \\ 0 & Q(t-1) = A(t) = 0 \end{cases}. \tag{15}$$

from (13), $Q(P) = \sum_{t=1}^{P} A(t) - \sum_{t=1}^{P} D(t) + Q(0)$

$\sum_{t=1}^{P} A(t) = K$, if $Q(0) = 0$ and $Q(K) = b$, then $\sum_{t=1}^{P} D(t) = K - b$. As $D(t)$ is either 1 or 0, $P - (K - b)$ of $D(t)$ are 0s and others are 1s.

Let $D(t_1) = D(t_2) = ... = D(t_{P-(K-b)}) = 0$

($t_1 < t_2 < ... < t_{P-(K-b)}$), we can divide the scenario into several parts.

from (14), $D(t_1) = 0 \Rightarrow Q(t_1 - 1) = A(t_1) = 0$

$Q(t_1 - 1) = \sum_{t=1}^{t_1-1} A(t) - \sum_{t=1}^{t_1-1} D(t) + Q(0) = 0$ and

Appendix A and have been published in [20].

$Q(t_1) = A(t_1) = 0$

$\sum_{t=1}^{t_1-1} D(t) = t_1 - 1$ because only

$D(t_1) = D(t_2) = ... = D(t_{P-(K-b)}) = 0$

similarly,

$D(t_2) = 0 \Rightarrow Q(t_2 - 1) = A(t_2) = 0$

$Q(t_2 - 1) = \sum_{t=t_1+1}^{t_2-1} A(t) - \sum_{t=t_1+1}^{t_2-1} D(t) + Q(t_1) = 0$ and

$Q(t_2) = A(t_2) = 0$

$\sum_{t=t_1+1}^{t_2-1} D(t) = t_2 - t_1 - 1$ because only

$D(t_1) = D(t_2) = ... = D(t_{P-(K-b)}) = 0$

Now, I change the original condition $Q(0) = 0$ to $Q'(0) = b$ and consider the same points as before.

$Q'(t_1 - 1) = \sum_{t=1}^{t_1-1} A(t) - \sum_{t=1}^{t_1-1} D(t) + Q'(0) = b$, $D'(t_1) = 1$ and $Q'(t_1) = b - 1$

(because $A(t)$ does not change, $\sum_{t=1}^{t_1-1} D(t)$ cannot be larger.)

similarly,

$$Q'(t_2-1)=\sum_{t=t_1+1}^{t_2-1}A(t)-\sum_{t=t_1+1}^{t_2-1}D(t)+Q'(t_1)=b-1,\ \ D'(t_2)=1 \text{ and } Q'(t_2)=b-2$$

……

As $P \geq K$, $b \leq P-(K-b)$,

$$Q'(t_b-1)=\sum_{t=t_{b-1}+1}^{t_b-1}A(t)-\sum_{t=t_{b-1}+1}^{t_b-1}D(t)+Q'(t_{b-1})=1,\ \ D'(t_b)=1 \text{ and } Q'(t_b)=0$$

$$Q'(t_{b+1})=Q'(t_{b+2})=...=Q'(t_{P-(K-b)})=0$$

exactly b time points when $D(t)=0$ become $D'(t)=1$ for $t \leq t_{P-(K-b)}$ and

$$\sum_{t=1}^{P}D'(t)=K$$

therefore,

$$Q'(K)-Q(K)=\sum_{t=1}^{K}D(t)-\sum_{t=1}^{K}D'(t)+Q'(0)-Q(0)$$
$$=-b+b=0$$

$$Q'(K)=Q(K)=b$$

REFERENCES

[1] W. S. Wong, “New Protocol Sequences for Random Access Channels without Feedback,” *IEEE Transactions on Information Theory*, 53(6), (June 2007), pp. 2060-2070.

[2] N. Abramson, “Packet switching with satellites,” in AFIPS Conf. Proc., Nat. Comp. Conf., June 1973, vol. 42, pp. 695–702.

[3] J. L. Massey and P. Mathys, “The collision channel without feedback,” IEEE Trans. Inf. Theory, vol. 31, no. 2, pp. 192–204, Mar. 1985.

[4] P. R. Prucnal, M. A. Santoro, and T. R. Fan, “Spread spectrum fiberoptic local area network using optical processing,” J. Lightw. Technol., vol. 4, no. 5, pp. 547–554, 1986.

[5] V. Anantharam, “The stability region of the finite-user slotted ALOHA protocol,” IEEE Trans. Inf. Theory, vol. 37, no. 3, pp. 535–540, May 1991.

[6] A. A. Shaar, M. Gharib, and P. A. Davies, “Collision resolution in contention access local area networks using concatenated prime sequences,” in IEE Proc. Commun., Oct. 2002, vol. 149, no. 5, pp. 249–256.

[7] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, “A survey on sensor networks,” IEEE Commun. Mag., vol. 40, no. 8, pp. 102–114, Aug. 2002.

[8][Randles, Martin, David Lamb, and A. Taleb-Bendiab. "A comparative study

into distributed load balancing algorithms for cloud computing." *Advanced Information Networking and Applications Workshops (WAINA), 2010 IEEE 24th International Conference on*. IEEE, 2010.

[9] J. L. Massey, "The capacity of the collision channel without feedback," in Abstracts of Papers, IEEE Int. Symp. Information Theory, 1982, p.101.

[10] Q. A. Nguyen, L. Györfi, and J. L. Massey, "Constructions of binary constant-weight cyclic codes and cyclically permutable codes," *IEEE Trans. Inform. Theory*, vol. 38, no. 3, pp. 940–949, May 1992.

[11] O. Moreno, Z. Zhang, P. V. Kumar, and V. A. Zinoviev, "New constructions of optimal cyclically permutable constant weight codes," *IEEE Trans. Inform. Theory*, vol. 41, no. 2, pp. 448–455, Mar. 1995.

[12] L. Györfi and S. Gy˝ori, "Coding for multiple-access collision channel without feedback," in *Multiple Access Channels—Theory and Practice*, ser. NATO Security Through Science, E. Biglieri and L. Györfi, Eds. Amsterdam, The Netherlands: IOS Press, 2007, pp. 299–326.

[13] K. W. Shum, C. S. Chen, C. W. Sung, and W. S. Wong, "Shift-Invariant Protocol Sequences for Collision Channels without Feedback," *IEEE Transactions on Information Theory,* vol 55(7), pp. 3312-3322 , (July 2009).

[14] Y. C. Chow, and W. H. Kohler, “Models for Dynamic Load Balancing in a Heterogeneous Multiple Processor System,” *Transactions on Computers*, 28(5), (May 1979).

[15] L.M. Ni and K. Hwang, “Adaptive Load Balancing in a Multiple Processor System with Many Job Classes,” IEEE Trans. Software Eng., vol. 11, no. 5, pp. 491-496, May 1985.

[16] A. Barak and A. Shiloh, “A Distributed Load-balancing Policy for a Multicomputer,” Software-practice and Experience, VOL. 15(9), 901-913 (September 1985).

[17] S. Nakrani and C. Tovey, “On Honey Bees and Dynamic Server Allocation in Internet Hosting Centers”, Adaptive Behavior 12, pp: 223-240 (2004).

[18] O. Abu- Rahmeh, P. Johnson and A. Taleb-Bendiab, “A Dynamic Biased Random Sampling Scheme for Scalable and Reliable Grid Networks,” INFOCOMP - Journal of Computer Science, ISSN 1807-4545, 2008, VOL.7, N.4, December, 2008, pp. 01-10.

[19] F. Saffre, R. Tateson, J. Halloy, M. Shackleton and J.L. Deneubourg, “Aggregation Dynamics in Overlay Networks and Their Implications for Self-Organized Distributed Applications,” The Computer Journal, March 31st, 2008.

[20] Y. Zhang and W.S. Wong, “Distributed Load Balancing in a Multiple Server System by Shift-Invariant Protocol Sequences,” IEEE Wireless Comunications and Networking Conference, April 7th, 2013.

[21] K. W. Shum and W. S. Wong, “Construction and Applications of CRT Sequences,” *IEEE Transactions on Information Theory*, vol 56(11), 5780-5795.