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

Yupeng Zhang and Wing Shing Wong
Department of Information Engineering
The Chinese University of Hong Kong
s0760929@cuhk.edu.hk, wswong@ie.cuhk.edu.hk

***Abstract*—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 paper. 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 paper, 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.**

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

I. INTRODUCTION

Load balancing problem has been studied in many previous works. Chow and Kohler [1] 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 [1], classified into two categories: deterministic and nondeterministic. 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, 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 [1].

Chow and Kohler [1] 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 paper, we are focus on the latter class of problems, which can be defined as "distributed load balancing".

Ni and Hwang [4] propose a recursive probabilistic routing algorithm. It uses parameters of the system such as job incoming rate and service rate to adjust probabilistic routing matrix recursively to finally obtain an optimal matrix. Some other previous work such as [6] and [7] are focused on how to exchange information among users and servers to distribute jobs evenly to all servers. It can be viewed as a trade-off between performance and information.

In this paper, 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 and defines the optimization function to make the

The work is supported by a grant entitled "Time Critical Applications over a Shared Network" of the Shun Hing Institute of Advanced Engineering, The Chinese University of Hong Kong.

system performance robust. Section III introduces the shift invariant protocol sequences studied in [2] and [3], and applies them in the distributed load balancing problems. Section IV shows the numerical results and the comparison among different strategies. Section V concludes the paper.

II. SYSTEM MODEL

As shown in figure 1, 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.

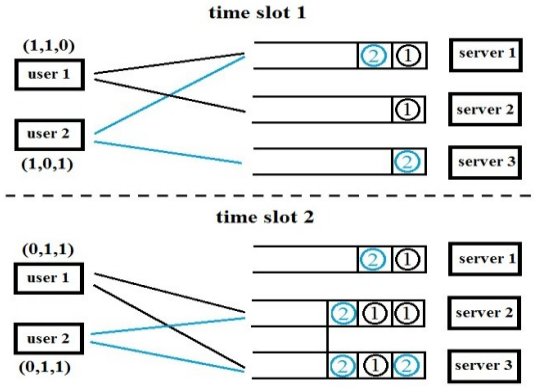

Fig.1 Illustration of multi-server system model

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 $S_2 = \{(1,0,0,1,0,0),(0,1,0,0,1,0),(0,0,1,0,0,1),.....\}$.

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.

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 from; 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 $s_i^t(l)$.

$$A_t(l) = \sum_i s_i^t(l). \quad (6)$$

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 in Appendix B.

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.

III. SHIFT INVARIANT PROTOCOL SEQUENCES

Protocol sequences are first proposed in [2] to be applied in collision channel without feedback in wireless communication. Users are arranged to send packets based on the binary sequences allocated to them. If there are more than one packet sent to a receiver simultaneously, collision occurs and these packets are dropped. For users that cannot be synchronized, the set of sequences should ensure that no matter how the sequences are shifted, the number of collisions, defined as "cross correlation" of the sequences, should be the same. This kind of protocol sequence is called shift invariant protocol sequences (SIS).

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 2 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 Sequence
(1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0)
(1,1,1,0,0,0,0,0,0,1,1,1,0,0,0,0,0,0,0,1,1,1,0,0)
(1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)

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)
(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)

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

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

Fig.2 example of shift invariant protocol sequence

TABLE I. AVERAGE RESPOND TIME FOR DIFFERENT τ

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

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 $n \times L$ (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 3 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.3 sequences allocated to users (L = 30, K = 3, P = 3)

IV. NUMERICAL RESULTS

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 defined by (12). 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. 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.

Figure 4 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.

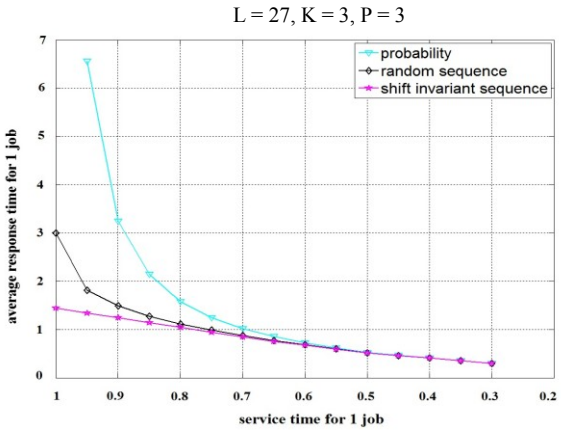

Fig.4 comparison of average response time

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. To simulate this, in previous system we further compare the maximum respond time for jobs sent in one time slot from one

user. Figure 5 shows that the shift invariant sequences routing performs much better than others in this case.

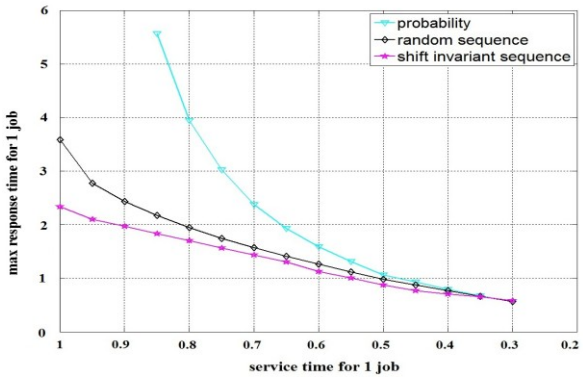

Fig.5 comparison of maximal response time in one time slot

V. CONCLUSIONS

In this paper, 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. Further investigations are required to deal with cases having arbitrary number of users in the system.

REFERENCES

[1] 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).

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

[3] 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).

[4] 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.

[5] Shubham Gupta, "Transient Analysis of D(t)/M(t)/1 Queuing System with Applications to Computing Airport Delays", Master Thesis of Massachusetts Institute of Technology, June 2010.

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

[7] P. Krueger and R. A. Finkel, 'An adaptive load balancing algorithm for a multicomputer', Computer Science Department, University of Wisconsin, Madison, Wisconsin, 1983.

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

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

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

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

……

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

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

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

APPENDIX B

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

$s_{\sigma_i^l}^i(l)=1, \sigma_i^l=1,2,...,P$

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 t s 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.

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

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

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

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 $S_1,S_2,...,S_K$.