sensors

MDPI

Article

GWS—A Collaborative Load-Balancing Algorithm for Internet-of-Things

Hongyu Xiao [1,2], Zhenjiang Zhang [2,3,*] and Zhangbing Zhou [4]

1 Department of Electronic and Information Engineering, Beijing Jiaotong University, Beijing 100044, China; yuhongxiao@aliyun.com
2 Key Laboratory of Communication and Information Systems, Beijing Municipal Commission of Education, Beijing Jiaotong University, Beijing 100044, China
3 Department of Software Engineering, Beijing Jiaotong University, Beijing 100044, China
4 School of Information Engineering, China University of Geosciences at Beijing, Beijing 100083, China; zhangbing.zhou@gmail.com or zbzhou@cugb.edu.cn
* Correspondence: zhjzhang1@bjtu.edu.cn; Tel.: +86-133-0111-7395

Received: 29 June 2018; Accepted: 30 July 2018; Published: 31 July 2018

Abstract: This paper firstly replaces the first-come-first-service (FCFS) mechanism with the time-sharing (TS) mechanism in fog computing nodes (FCNs). Then a collaborative load-balancing algorithm for the TS mechanism is proposed for FCNs. The algorithm is a variant of a work-stealing scheduling algorithm, and is based on the Nash bargaining solution (NBS) for a cooperative game between FCNs. Pareto optimality is achieved through the collaborative working of FCNs to improve the performance of every FCN. Lastly the simulation results demonstrate that the game-theory based work-stealing algorithm (GWS) outperforms the classical work-stealing algorithm (CWS).

Keywords: collaborative; Internet-of-Things; fog computing; Nash bargaining solution; Pareto optimality; scheduling; time-sharing

1. Introduction

Along with the rapid development of IoT, fog computing has emerged as a promising architecture for IoT applications. As the necessary complement to cloud computing, fog computing serves IoT devices by undertaking part of their work load. IoT devices typically feature weak computing capacity and low energy. With the help of fog computing, IoT devices can deliver some tasks to a fog computing nodes (FCNs) to relieve their load and reduce the energy consumption [1]. The IoT architecture is shown in Figure 1.

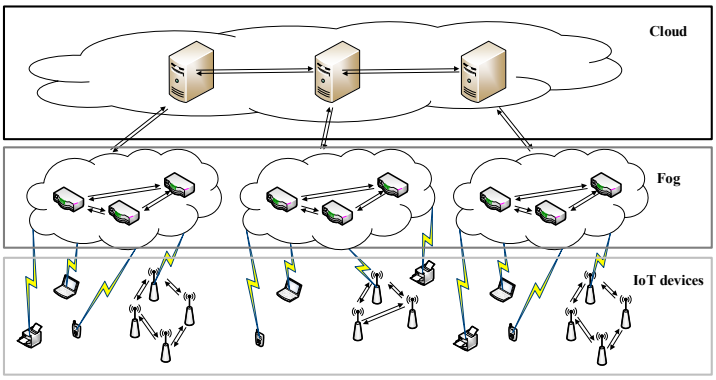

Figure 1. IoT architecture for fog computing.

The IoT architecture is planar, comprised of a cloud layer, a fog layer and a layer of IoT devices. The fog layer—the-so-called "fog computing network" helps the cloud to process IoT tasks. Because the fog layer is much closer to the edge, it achieves low latency and quick response time, both of which are necessary for IoT applications. For example, in a wireless sensor network, sensors may utilize the fog computing nodes to do some computing and make decisions. At this point, FCNs simply act as rule engines, so each FCN must respond as quickly as possible for higher quality of service (QoS). If the FCNs respond slowly, then some operations of the wireless sensor network will be delayed.

Fog computing nodes (FCNs), as the key components of fog computing, are always located at the edge in contrast to the cloud computing data centers at the center of the Internet. FCNs have low latency for IoT applications. 'Latency', here, refers to the FCN's response time for IoT tasks. The work mechanism is somewhat important factor influencing latency. The work mechanism of FCNs can be divided into three categories: concurrent, priority and FCFS [2]. When an FCN possesses only one single-core processor, the concurrent mechanism degenerates into a TS. We will discuss FCNs that possess only one single-core processor or just utilize a single-core for fog computing like a great number of routers and switches which support fog computing.

Some papers have modeled the FCN as an M/M/1 queuing system [3–5] that uses an FCFS mechanism to arrange tasks in the task deque, where they wait until the processor idle. However, since the M/M/1 queuing system is no longer appropriate for fog computing, this paper proposes a TS system perfectly suited to fog computing. Furthermore, this paper puts forward an important measure of FCN performance—the concurrency coefficient—which denotes the expected response time for one task with a specific number of instructions. The concurrency coefficient is an important measurement for multi-class tasks. In [6,7], the authors merely analyze single-class tasks, which is unrealistic for various IoT applications.

In fog computing, heterogeneous FCNs differ in processing rate and input load, the heterogeneity gives rise to a load imbalance. For example, some weak FCNs become over-loaded while some strong FCNs remain idle. The imbalance largely decreases the processing capacity of the whole fog computing network and increases the response time for IoT applications. Accordingly, we propose a load-balancing algorithm that guides FCNs to collaborate by an adjusted work-stealing scheduler, which is decentered in contrast to its counterpart. Moreover we prove that the scheduling algorithm can achieve Pareto optimality based on the Nash bargaining solution.

At the end of this paper, some simulations are summarized. The simulation results prove that our load balancing algorithm can reduce the response time more than the classical work-stealing algorithm, especially for light tasks.

This paper is organized as follows:

- Section 2 introduces related progress on collaborative load-balancing algorithms for IoT;
- Section 3 analyzes the FCN working mechanism, proposes a work-stealing algorithm for a TS system and solves the probability allocation problem by means of the Nash bargaining solution;
- Section 4 elaborates the simulation results and proves the validity and efficiency of GWS; and
- Section 5 concludes the content and proposes future work.

2. Related Work

In fog computing, FCNs are always modeled as an M/M/1 queuing system [3–5], which adopts the first-come-first-serve (FCFS) mechanism, however, this mechanism no longer satisfies the needs of IOT applications, especially for wireless sensor and actuator networks which require a quick response. Section 3.1 proves that the FCFS mechanism leads to a fixed delay for any tasks. It is poor for applications which are full of light tasks, like wireless sensor and actuator networks. Light tasks are those need little processing time and need a response as quickly as possible.

Load balancing algorithms fall into two categories: work-sharing and work-stealing, or static and dynamic. 'Work-sharing' means that all tasks are dispatched to other processors through a center,

whereas 'work-stealing' means that tasks are stolen from processors proactively. 'Static' means that the algorithm rules are predefined, and 'dynamic' means that the algorithm rules are decided in run time and always changing. For FCNs, the dynamic work-stealing algorithm is best, because a fog computing network is versatile with tasks arriving continually. A classical work-stealing scheduler was proposed in [8], which applies to multiprocessors for multithreaded applications. In a fog computing network, an FCN can also be viewed as a processor; however, there are still some vital differences between an FCN and a processor. For example, the memory swap and synchronization between processors is much faster and easier than in FCNs, which are located at different sites.

Moreover, the work-stealing for TS systems is rather difficult and expensive to implement practically [9]. Thus, Section 3.2 proposes GWS to steal raw tasks from the residual queue. Moreover, [7] explores how to share tasks in a TS system; however there is only one task input source, and all tasks must be allocated through it. This is unrealistic for fog computing, where there are numerous dispersed task input sources.

Another work-stealing strategy was proposed by routing the stealing and response based on the trajectories of moving users [10]. The author predicts future users' addresses according to their historical trajectory; however the trajectory prediction may be imprecise, and a large number of users would cause too much overhead to calculate trajectories and manage routes. In fog computing, the number of different users and applications is too large for a prediction algorithm. So we insist that the work-stealing strategy be succinct to avoid heavy computing load and network congestion.

A load-balancing algorithm for single-class tasks was proposed in [6], which treated the load balancing problem as a cooperative game between processors based on game theory. We propose a scheduling algorithm for multi-class tasks.

Another hierarchical work-stealing algorithm was proposed for reducing the stealing frequency [11] by clustering FCNs and selecting one FCN as the leader; however central management may give rise to a single point of failure, as FCNs are not very stable. This paper avoids selecting one FCN as the leader and instead takes full advantage of the cloud, which is powerful and stable. With the help of the cloud, we can efficiently manage load balancing between FCNs.

Considering the modern trend of Big Data, an overview for Big IoT Data Analytics was proposed by [12], and a survey of service migration in edge computing was conducted by [13]. These all are promising aspects of fog computing in IoT applications.

3. Collaborative Work-Stealing Algorithm

3.1. FCN Working Mechanism

Fog computing is a virtual service for IoT applications. FCNs can be routers, switchers and other network devices that support fog computing service. FCNs serve other IoT devices nearby by helping to process the tasks that arrive from them as Figure 2 shows.

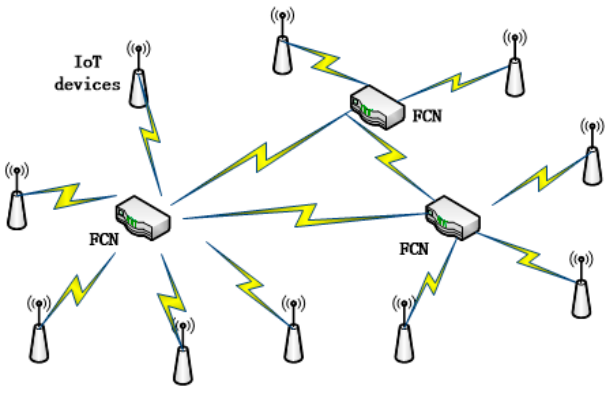

Figure 2. Fog computing nodes and IoT devices nearby.

Since the tasks' sources are various and arrival intervals are random, the arrival process can be modeled as a Possion process. We assume the arrival rate as λ. For task processing, we do not assume the service time as an exponential distribution as [3–5], as that is not reasonable. We discuss the general distribution below. The system is shown in Figure 3. The task is stored in one deque, which is a variety of queues that permit in-out operations from both ends [14].

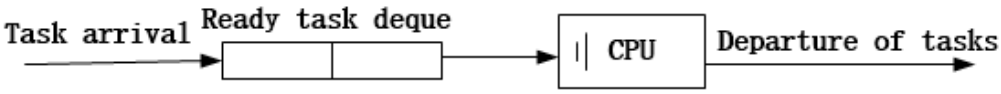

Figure 3. Fog computing node of M/G/1 model.

Thus the FCN is a M/G/1 queuing system with the following properties: the arrival rate is λ, the process time is $T_{service}$, $E(T_{service})$ denotes the average service time per task, and $E\left({T_{service}}^2\right)$ denotes the second moment of $T_{service}$. According to [15], the M/G/1 queuing system holds the following two equations:

$$T_{response} = T_{wait} + T_{service}, \tag{1}$$

$$\rho = \lambda E(T_{service}), \tag{2}$$

$$E(T_{wait}) = \frac{\lambda E\left({T_{service}}^2\right)}{2(1-\rho)}, \tag{3}$$

In the above equations ρ denotes load intensity, T_{wait} denotes the waiting time in the queue and $T_{response}$ denotes response time which equals a task's entire time in the FCN. As the FCN abides by the FCFS mechanism, arriving tasks first wait in the ready queue then receive service until the processor is idle, so the $T_{response}$ consists of T_{wait} and $T_{service}$. Here we propose a new important measure factor $E(T_{response}|T_{service} = x)$ which denotes the conditional expectation of $T_{response}$ while $T_{service}$ is set to x. Since how long one task waits is independent of its service time, we obtain the following equation:

$$E\left(T_{response}\middle|T_{service} = x\right) = E(T_{wait}) + x. \tag{4}$$

Equations (3) and (4) can be simplified as:

$$E\left(T_{response}\middle|T_{service} = x\right) = \frac{\lambda E\left({T_{service}}^2\right)}{2(1-\rho)} + x. \tag{5}$$

According to Equation (5), we find that no matter how small a task is, the response time can be no shorter than $\frac{\lambda E\left({T_{service}}^2\right)}{2(1-\rho)}$, which is the lower bound limit of time; however, some sense-and-actuate-loop IoT applications like a wireless sensor and actuator network, which generate a large number of light tasks and demand quick response time, will obtain terrible QoS due to the unavoidable lower-bound time limit, so the FCFS system is not appropriate for the FCN. We propose a classical mechanism that can cut off the lower-bound time limit–time-sharing (TS) mechanism. A TS mechanism is efficient for concurrency, which has been applied to computer systems successfully. As Figure 4 shows, the wireless sensor shunts its tasks to the FCN for service, and the time-sharing FCN can respond quickly and provide high QoS.

Next, we elaborate the TS system. As Figure 5 shows, once a task arrives, it is pushed into the task deque from the back. The CPU obtains a task from the front of the task deque and gives it a little quantum of service. When the task is completed, it leaves the FCN; otherwise, it is pushed into the task deque from the back, which is called 'cycled arrival'. Through time-sharing, each task receives service in turn. The service quantum is so little that every task seems to be served at the same time.

The scheduling mechanism is also called Round-Robin (RR) time-sharing. According to [16], we obtain a key equation:

$$E\left(T_{response}\middle|T_{service}=x\right)=\frac{x}{1-\rho}. \tag{6}$$

so if FCN adopts the RR algorithm, the expectation of T_{stay} is proportional to $T_{service}$. Some IoT applications, like wireless sensor and actuator networks which are comprised of small tasks, can obtain a quick response in contrast to waiting for a fixed time in the FCFS system. in Section 4, we compare the two scheduling algorithms through simulations.

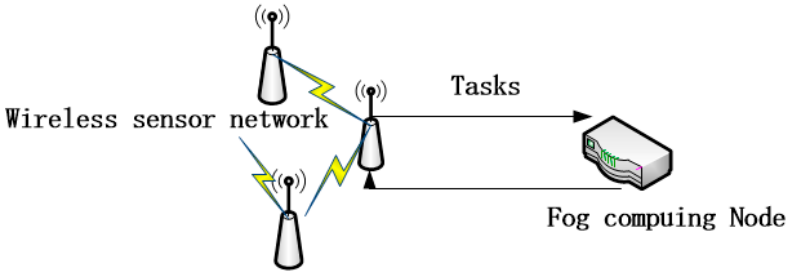

Figure 4. Wireless sensor network applications.

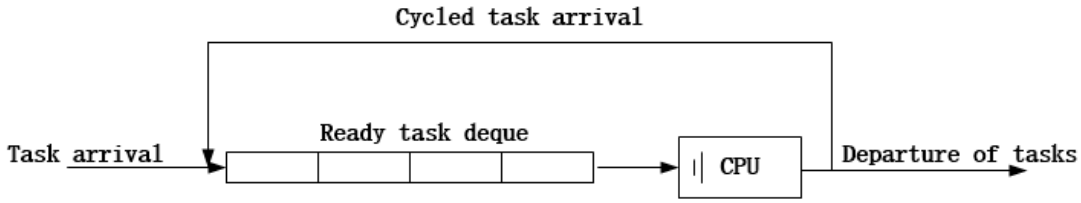

Figure 5. Time-sharing system model.

We compare Equations (5) and (6), and we find that although the TS system removes the lower-bound time limit, its coefficient for service time is larger than that of FCFS system: $\frac{1}{1-\rho}>1$. So the TS system decreases the response time for light tasks at the expense of increasing the response time for heavy tasks. But to be reasonable, the light tasks always demand more than heavy tasks. So we believe that the expense is reasonable.

We also found that the TS system has a potential advantage, as Equation (7) shows:

$$\frac{E\left(T_{response}\middle|T_{service}=x\right)}{x}=\frac{1}{1-\rho}. \tag{7}$$

The ratio between the expected $T_{response}$ and $T_{service}$ is constant, which just relates to ρ; however ρ is decided by λ and $E(T_{service})$, so the TS scheduler is completely fair to all tasks, whether small or large. This feature is necessary and helpful. As if the ratio is smaller for small tasks, users and developers tend to split a large task into smaller tasks. Or if the ratio is larger for small tasks, then users and developers tend to merge small tasks into larger ones for a quicker response. Such unfairness will cause malicious competition and add the burden of users and developers. So we insist that fairness is necessary in fog computing, which means that $\frac{E\left(T_{response}\middle|T_{service}=x\right)}{x}$ is the same for any task processed in any FCN.

In a FCN, one task may gain different $T_{service}$ in different FCNs. The service time of one task depends on its programming architecture and the processors of the serving FCN. For convenience we propose on absolute value π that denotes the number of instructions of one task to represent the

working load of tasks and an absolute value s that denotes the number of instructions processed by the FCN per unit time. We obtained modified equations as follows:

$$E(T_{service}) = \frac{E(\pi)}{s}, \tag{8}$$

$$\mu = \frac{1}{E(T_{service})} = \frac{v}{E(\pi)}, \tag{9}$$

$$\rho = \frac{\lambda}{\mu} = \frac{\lambda * E(\pi)}{s}, \tag{10}$$

$$\frac{E(T_{response}|\pi = x)}{x} = \frac{E(T_{stay}\,|T_{service} = \frac{x}{v})}{\frac{x}{s}} * \frac{1}{s} = \frac{1}{s * (1-\rho)}, \tag{11}$$

$$\mathrm{C} = \frac{E(T_{response}|\pi = x)}{x} = \frac{1}{\mathrm{s} * (1-\rho)} = \frac{1}{\mathrm{s} - \lambda * E(\pi)}, \tag{12}$$

Here, C denotes the FCN's performance which is called the 'concurrency coefficient'. As we can see, the smaller C means quicker response and better performance of FCNs. So we aim to reduce C. And C is dependent on the processing rate of the FCN, the arrival rate, and the average service time of tasks. The next section will elaborate a new scheduler that aims to minimize concurrency coefficients of FCNs based on game theory.

3.2. Game-Theory Based Work-Stealing Scheduler

Work-stealing as a scheduling algorithm is imposed on a multi-processor system to balance the load between processors. Compared to a multi-processor system, a fog computing network features a much more complex topology, more significant communication delay and dispersive memory. Stealing between FCNs is much more expensive than between processors. A normal work-stealing scheduler adopts a strategy where an idle processor steals from a randomly chosen processor. If the victim processor has more than one task, it transfers the extra task to the stealer; otherwise, it responds with a refusal command and then the stealer attempts another randomly chosen processor. In conclusion, if the scheduler is applied to a fog computing network, it suffers from the following defects:

1. Idle FCNs must wait until they successfully steal a task, which wastes time and energy.
2. A TS system is very hard and costly for the dispersive memory distribution [9]. So we ought to adjust the normal work-stealing algorithm for FCNS, which adopts the TS mechanism.

We have to adjust the normal work-stealing scheduler for fog computing. Fog computing is the complement to cloud computing (as Figure 1 shows), as every FCN is connected to the cloud. So we can utilize the cloud to help with work-stealing. A cloud manages a cluster of FCNs and orchestrates their cooperation. The parameters of the cluster of FCNs are listed in Table 1.

Table 1. Parameter names and paraphrases.

Parameter Name	Parameter Paraphrase
N	Number of FCNs in this cluster
F_i	The i_{th} FCN
λ_i	Average task arrival rate of F_i
$\overline{\pi_i}$	Average instruction number per task of F_i
s_i	Processed instruction number per unit time of F_i
μ_i	Number of processed tasks per unit time of F_i
C_i	Concurrency coefficient of F_i

In the above list, some parameters are almost fixed, like s_i, which denotes the processing capacity of the FCN. The other parameters should be bookkept by the FCN itself and reported to the cloud periodically.

In the fog computing network above, every FCN has its own concurrency coefficient based on Equation (12). This factor is an important measure of performance which denotes the average response time of the specified task. For F_i the concurrency coefficient is in Equation (13):

$$C_i = \frac{1}{s_i - \lambda_i * \overline{\pi_i}}, \quad i = 1, 2, \ldots, N. \tag{13}$$

Obviously, a different FCN may possess a different processing rate ∂_i, different task arrival rate λ_i and different average instruction number per task $\overline{\pi_i}$, which may lead to different concurrency coefficient C_i based on the Equation (13). The goal of this paper is to fairly achieve a larger C_i for every FCN.

From the Equation (13), for an FCN like F_i, the concurrency coefficient C_i only depends on s_i, λ_i and $\overline{\pi_i}$. Among these factors, s_i, which denotes the process rate of F_i, is almost fixed; λ_i, which denotes the task arrival rate of F_i, just relies upon the IoT devices in this area; and $\overline{\pi_i}$ denotes the average number of instructions per task of F_i. So the only factor that can be modified is the task arrival rate λ_i. We can modify λ_i so that C_i approximates the average value of the concurrency coefficient $\overline{C}$. The algorithm is elaborated as follows.

This adjusted work-stealing algorithm aims to adjust the task input intensity. We classify FCNs into two varieties: over-loaded FCNs with a large concurrency coefficient and under-loaded FCNs with a small concurrency coefficient. The work-stealing algorithm reduces the task arrival rate of over-loaded FCNs and raises the task arrival of under-loaded FCNs by shunting and stealing. The two varieties of FCNs are modeled in Figures 6 and 7.

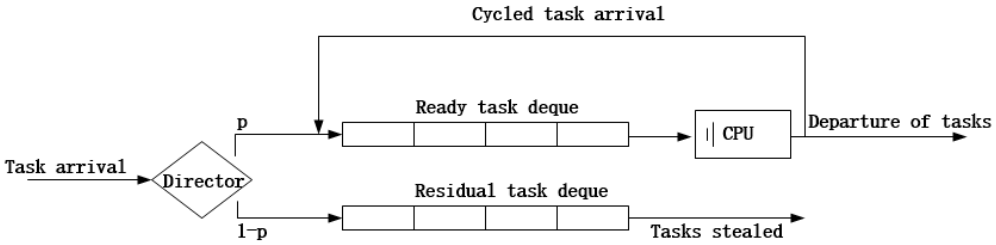

Figure 6. Over-loaded FCN model ($p < 1$).

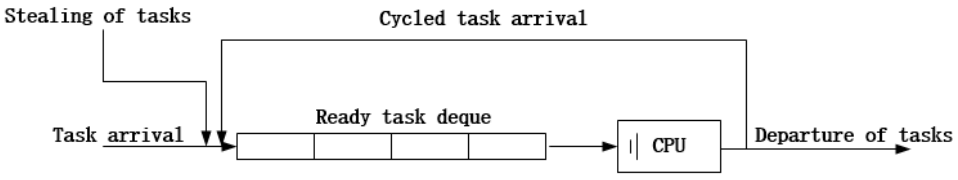

Figure 7. Under-loaded FCN model ($p > 1$).

As Figure 6 shows, an FCN contains two task deques, the ready task deque, which works with the CPU, and the residual task deque, which stores raw tasks that are ready to be stolen. Another big difference is the director, which decides whether a task goes to the ready or residual task deque on the probability of p. So what is p? This will be discussed in Section 3.3. For over-loaded FCNs like F_i, all of the arriving tasks go to the ready task deque at the probability of p_i, so the task arrival rate becomes $\lambda_i * p_i$. The updated concurrency coefficient is expressed as follows:

$$C_i' = \frac{1}{s_i - \lambda_i * p_{i*} \overline{\pi_i}}. \tag{14}$$

When p_i is less than 1, the director of this FCN will shunt arriving tasks to the ready task deque at the probability of p_i and to the residual task deque at the probability of $1 - p_i$. Tasks in the ready task deque will receive service one by one, and tasks in the residual task deque wait for a stealing request. Once a task enters the ready deque, it cannot be shared; tasks in in the residual dequee are raw and suitable for sharing. When a stealing request comes, and the residual task deque is not empty, the FCN delivers a task from the back of the residual task deque to the stealing FCN.

When p_i is greater than 1, as shown in Figure 7, the FCN is under-loaded. There is no need to maintain the residual deque as the director. The FCN has to steal another FCN so that the overall task arrival rate can increase. We set the successful stealing interval as an exponential distribution with the average value of $\lambda_i * (p_i - 1)$. A successful stealing interval means the interval between two stealing from FCS which contains extra tasks. If the FCN fails to steal, it continues without waiting.

An exceptional situation occurs when p_i is equal to 1, then FCN is just the same as an isolated TS system that does not steal or shunt any tasks to the residual task deque. This special case rarely happens, so we don't discuss this in the following sections.

As the cloud periodically updates the FCN cluster, the role of each FCN may change. Some over-loaded FCNs may become under-loaded and vice versa, so the algorithm is dynamic to the real IoT environment and evolves periodically. In next section, we will discuss how to calculate the probability set $\mathrm{p} = \{p_1, p_2, \ldots, p_N\}$. This is the key factor for our algorithm.

3.3. Nash Bargaining Solution for the Probability Set

Section 3.2 proposes an efficient scheduling algorithm of work-stealing for a TS system, but how to set the important factor p_i has not be solved. The main goal of the paper is to minimize the concurrency coefficient $C_i'(p_i)$ of each FCN. The problem can be modeled as a NBS rather than a Nash equilibrium for cooperative FCNs. This is a cooperative game, which is different from its non-cooperative counterpart [17]. Through cooperation of players (FCNs), a better profit can be achieved. The game is depicted in Figure 8.

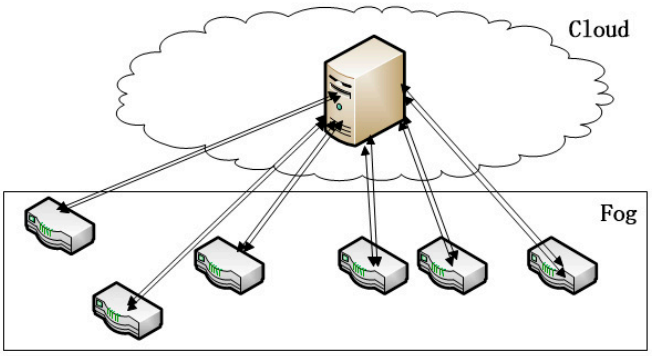

Figure 8. Cooperative game between FCNs.

As Figure 8 shows, FCNs gain common knowledge through the cloud. FCNs can communicate with each other to gain common knowledge, but this communication process is of time complexity $\mathrm{O(N * N)}$. So why not draw support from the cloud? As all FCNs are linked to the cloud, so FCNs can gain all information needed for the game by means of the central cloud with the time complexity $\mathrm{O(N)}$. Although the cloud may be far from edge, I believe a decrease in time complexity by one order of magnitude can offset it even more. Let's analyze the game to find the optimal balancing points. The mathematical problem is as follows:

$$\min C_i'(p_i), \quad i = 1, 2, \ldots, N, \tag{15}$$

$$p_i > 0, \tag{16}$$

$$p_i < \frac{s_i}{\lambda_i * \overline{\pi_i}}, \tag{17}$$

$$\sum_{i=1}^{N} \lambda_i * p_i = \lambda, \tag{18}$$

Inequation (16) guarantees the probability p_i is not negative and Inequation (17) is the stability condition of the M/G/1 queuing system. We replace $C_i(p_i)$ according to Equation (14), so objective (14) can be simplified as:

$$\max\ (\Phi p_i) = -p_i, \quad i = 1, 2, \ldots, N. \tag{19}$$

By maximizing $\Phi(p_i)$, F_i minimizes $C_i(p_i)$ at the same time. Every FCN cooperates by means of the cloud center to gain better performance. This problem can be viewed as a Nash bargaining game for cooperative players. According to [18], the NBS can realize Pareto optimal operation point; that is, NBS guarantees optimality and fairness for every FCN. According to [17], the above objective (19) is equivalent to the following objective:

$$\max \prod_{i=1}^{N} \left(\Phi(p_i) - \eta_i^0 \right), \tag{20}$$

where η_i^0 indicates the initial agreement point which denotes that $\Phi(p_i)$ must not be less than η_i^0. We set $\eta_i^0 = -\frac{s_i}{\lambda_i * \overline{\pi_i}}$ based on Inequation (17) and we set

$$\psi_i = -\eta_i^0 = \frac{s_i}{\lambda_i * \overline{\pi_i}} \tag{21}$$

In conclusion, the above problem can be elaborated as follows:

$$\max \prod_{i=1}^{N} \left(\Phi(p_i) - \eta_i^0 \right) = \prod_{i=1}^{N} (\psi_i - p_i), p_i > 0. \tag{22}$$

Then we use the Lagrange multiplier method to find the set $\mathrm{p} = \{p_1, p_2, \ldots, p_N\}$ for the maximum objective. But first we ignore the condition $p_i > 0$ and apply it later. The Lagrange function is as follows, and u and v_i are multipliers for Equation (18) and Inequation (17).

$$\mathrm{L}(p_i, u, v_i) = \sum_{i=1}^{N} In(\psi_i - p_i) + u * (\sum_{i=1}^{N} p_i * \lambda_i - \lambda) + \sum_{i=1}^{N} v_i * (p_i - \psi_i). \tag{23}$$

Then we apply the Karush Kuhn Tucker (KKT) constraints as follows:

$$\frac{\partial L}{\partial p_i} = \frac{1}{p_i - \psi_i} + \lambda_i * u + v_i, \tag{24}$$

$$\frac{\partial L}{\partial u} = \sum_{i=1}^{N} p_i * \lambda_i - \lambda = 0, \tag{25}$$

$$v_i * (p_i - \psi_i) = 0. \tag{26}$$

According to (17) and (19), we know $p_i < \psi_i$, so we deduce $v_i = 0$, and Equations (24)–(26) can be concluded as:

$$\frac{1}{p_i - \psi_i} + \lambda_i * u = 0, \tag{27}$$

$$\sum_{i=1}^{N} p_i * \lambda_i - \lambda = 0, \tag{28}$$

The result set $\mathrm{p} = \{p_1, p_2, \ldots, p_N\}$ can be resolved from Equations (25) and (26).

$$p_i = \psi_i - \frac{\sum_{i=1}^{N} \lambda_i * \psi_i - \lambda}{\lambda_i * N}. \tag{29}$$

Until now the result set $\mathrm{p} = \{p_1, p_2, \ldots, p_N\}$ has not been solved because the constraint (16) $p_i > 0$ has not been applied. If $p_i < 0$, Equation (16) infers that ψ_i is too little. But based on (21), if ψ_i is too small, then weak processing capacity is too weak and task arrival too frequent—both of which lead to FCN failure. So we just abandon FCNs with negative p_i. According to [7], we can remove the F_i for which $p_i < 0$ by using the following algorithm in the time complexity of $\mathrm{O}(\mathrm{n} * \log(\mathrm{n}))$.

In the above algorithm, the sorting accounts for the time complexity of $\mathrm{O}(\mathrm{n} * \log(\mathrm{n}))$. The Pareto optimal point is calculated out as $\mathrm{p} = \{p_1, p_2, \ldots, p_N\}$. In the fog computing network, we adopt the result set $\mathrm{p} = \{p_1, p_2, \ldots, p_N\}$ to implement the work-stealing scheduler, and the Pareto optimal maximum for the concurrency coefficient will be achieved. 'Pareto optimality' means there is no way to improve performance of one FCN without decreasing the performance of others.

The next section elaborates the simulations that prove the efficiency of the Algorithm 1 below.

Algorithm 1: post-processing algorithm for eliminating the negative p_i

Input: task arrival rate λ_i, the overall task arrival rate λ, the parameter ψ_i and the FCN number N.
Output: probability set $\mathrm{p} = \{p_1, p_2, \ldots, p_N\}$.

1. Sort all FCNs in decreasing order of ψ_i,
2. $\Phi = \frac{\sum_{i=1}^{N} \lambda_i * \psi_i - \lambda}{N}$,
3. **While** $\left(\psi_i < \frac{\Phi}{\lambda_i}\right)$
4. $p_i = 0$
5. $\mathrm{n} = \mathrm{n} - 1$,
6. $\Phi = \left(\Phi - \frac{\lambda_{\mathrm{n}+1} * \psi_{n+1}}{\mathrm{n}+1}\right) * \frac{\mathrm{n}+1}{\mathrm{n}}$
7. **end while**
8. **for** $i = 1, 2, \ldots, \mathrm{n}$
9. $p_i = \psi_i - \frac{\Phi}{\lambda_i}$,
10. **end for.**

4. Simulations

Lastly some simulations were completed to prove the efficiency of GWS. The simulations were programmed in C++ language, and the figures were drawn using OriginPro 2016.

Simulation I is a comparison between the FCFS mechanism and TS mechanism on an FCN. In [3–5], the FCN adopts the FCFS mechanism, and we suggest the TS mechanism, which was proved in Section 2. We perform a simulation where one FCN adopts the FCFS mechanism while another adopts the time-sharing, and other parameters like task input and processing rate are kept equal. The specific parameters are as in Table 2.

Table 2. Parameters of Simulation I.

FCN Name	λ	s	$\overline{\pi}$
FCFS-200M	1.2	2.5×10^7	1×10^7
TS-200M	1.2	2.5×10^7	1×10^7
FCFS-400M	1.2	5×10^7	1×10^7
TS-400M	1.2	5×10^7	1×10^7

The four nodes possess the same task input and just differ in work mechanism and processing rate of 200 M and 400 M. The 200 M and 400 M here mean the CPU dominant frequency. A 200 M CPU can perform work of 200M clock periods per second. As an instruction needs 8 clock periods, so this CPU can complete 25 M instructions per second, and a 400 M CPU can complete 50 M instructions per second.

We can obtain the relation between service time and stay time as shown in Figure 9. The FCFS system contains a lower-bound limit of response time around 4000 ms, which is not suitable for some IoT applications especially wireless sensor networks. By contrast, TS is just appropriate for wireless sensor network applications because it avoids the lower bound. So for light tasks, the TS mechanism can guarantee very good performance. But when tasks are heavy, the TS mechanism needs more response time, in contrast to FCFS system. This is just the expense of a TS mechanism, as Section 3.1 proves. By comparing nodes of different process rates, we can find that a higher processing rate means a shorter response time. Meanwhile, when the processing rate increases, the line of FCFS and TS become closer. The reason is that when process rate increase, the work strength decreases, which means that the work deque always only contains one or fewer tasks. Then the TS and FCFS mechanism are the same.

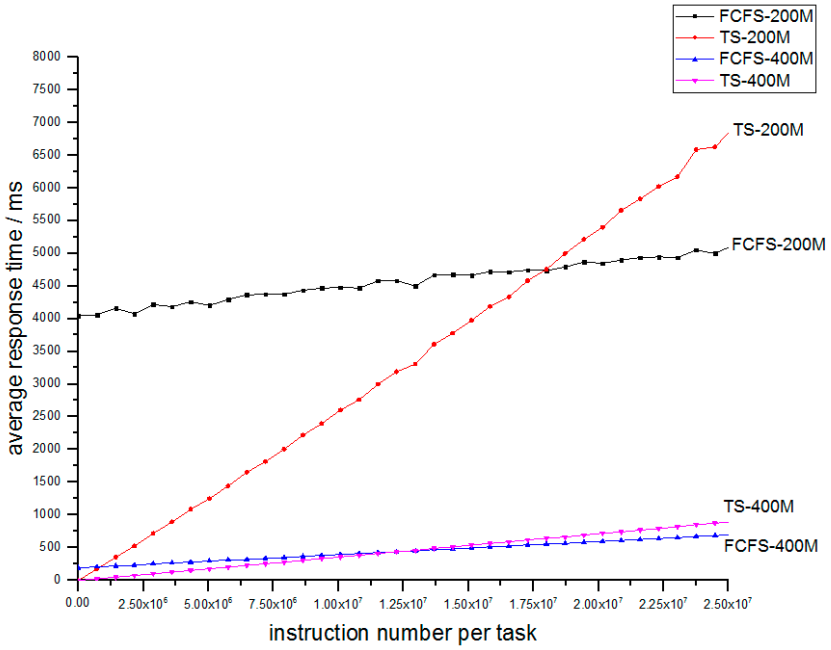

Figure 9. Relation of average response time and number of instructions for FCFS-200M and TS-200M, FCFS-400M and TS-400M.

Simulation II focuses on the performance of the whole cluster of 100 FCNs. The parameters are shown in Table 3; $\overline{\lambda}$, $\overline{s}$, $\overline{\pi}$ and N separately denotes the average task arrival rate, the average processing rate, the average number of instructions per task and the number of FCNs in the cluster of FCNs. A fog computing network, which separately adopts CWS or GWS receives the same task input over a long period. Let's see the performance according to the relation between the number of instructions and the response time in Figure 10. We find that GWS outperforms CWS. By means of GWS, the IoT task can achieve much faster response, especially for light tasks, but as for heavy tasks, the GWS needs more time. It is worthwhile because heavy tasks always hold loose time limits compared to light ones.

Table 3. Parameters of Simulation II.

Network Name	$\overline{\lambda}$	$\overline{s}$	$\overline{\pi}$	N
CWS-I	1.2	2.5×10^7	1×10^7	100
GWS-I	1.2	2.5×10^7	1×10^7	100

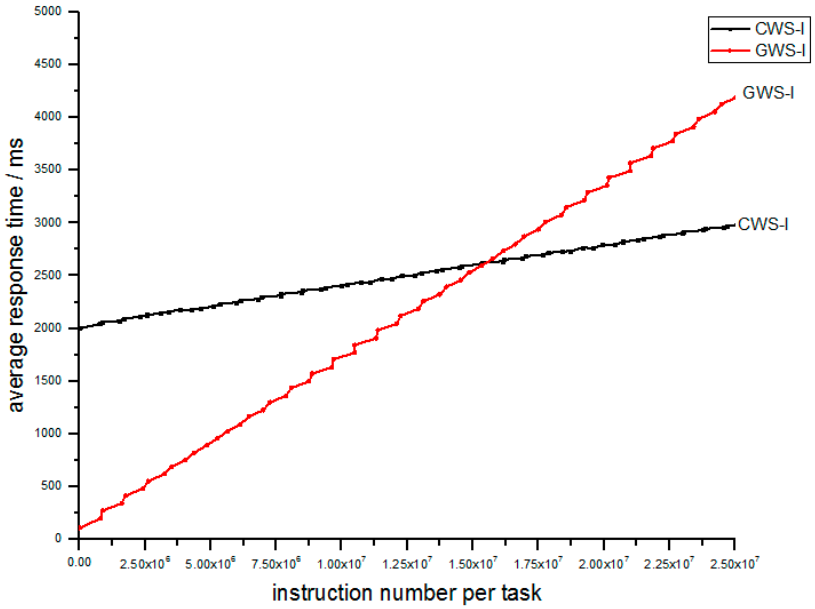

Figure 10. Relation of average response time and number of instructions per task between CWS-I and GWS-I.

Then we explore how the working load influences the fog computing network by changing the work load of the whole system. Three simulations will be carried out to explore the influence of average arrival rate, average processing rate and average number of instructions per task.

Simulation III explores the influence of arrival rate. In Table 4, we just change the average arrival rate of tasks and other parameters maintain the same. The experiment result of simulation III is depicted in Figure 11.

Table 4. Parameters of Simulation III.

Network Name	$\overline{\lambda}$	$\overline{s}$	$\overline{\pi}$	N
CWS-I	1.2	2.5×10^7	1×10^7	100
GWS-I	1.2	2.5×10^7	1×10^7	100
CWS-II	0.8	2.5×10^7	1×10^7	100
GWS-II	0.8	2.5×10^7	1×10^7	100
CWS-III	1.4	2.5×10^7	1×10^7	100
GWS-III	1.4	2.5×10^7	1×10^7	100

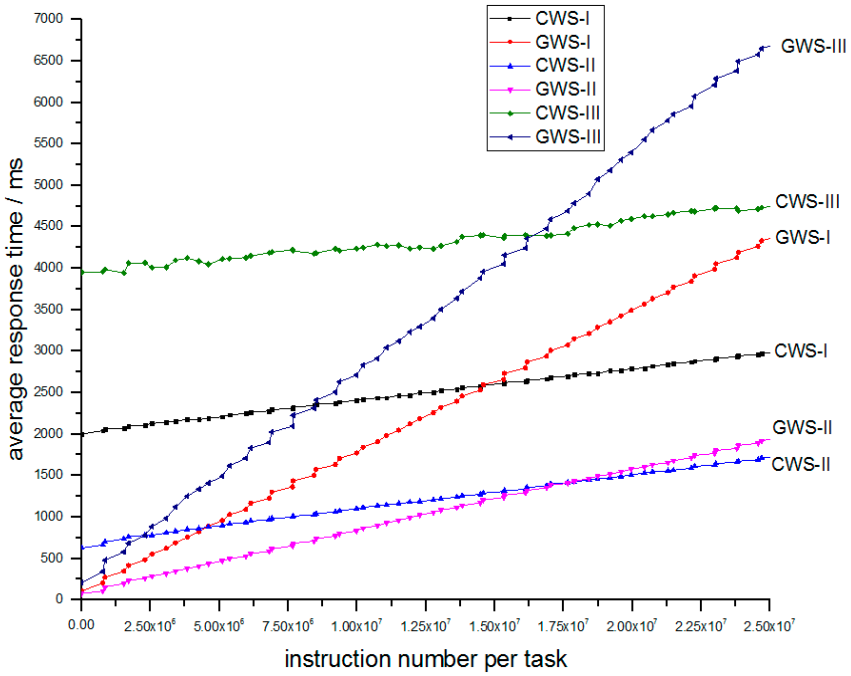

Figure 11. Relation of average response time and number of instructions per task between different arrival rates.

As we can see from Figure 11, no matter how arrival rate changes, the corresponding relation of CWS and GWS never changes. When arrival rate go bigger, the lines of CWS rise and the lines of GWS steepen. The CWS-III obtains bigger time lower bound than CWS-I and CWS-II, meanwhile GWS-III obtains bigger response time than GWS-I and GWS-II. But the GWS ones can still maintain less response time compared to CWS ones for light tasks.

Simulation IV explores how average processing rate influences performance. The parameters are shown in Table 5.

Table 5. Parameters of Simulation IV.

Network Name	$\overline{\lambda}$	$\overline{s}$	$\overline{\pi}$	N
CWS-I	1.2	2.5×10^7	1×10^7	100
GWS-I	1.2	2.5×10^7	1×10^7	100
CWS-IV	1.2	2×10^7	1×10^7	100
GWS-IV	1.2	2×10^7	1×10^7	100
CWS-V	1.2	3×10^7	1×10^7	100
GWS-V	1.2	3×10^7	1×10^7	100

As we can see from Table 5 that only processing rate is different. The experimental result is depicted in Figure 12.

In the Figure 12, the relative relation of CWS-IV and GWS-IV, or CWS-V and GWS-V remains as CWS-I and GWS-I, because GWS-IV and GWS-V still obtain less response time than their counterparts for light tasks. And while process rate increases, the response time also decreases.

Simulation V studies the influence of average number of instructions per task. The related parameters are shown in Table 6.

In Table 6, the networks only differ in average instruction number. The experimental result is shown in Figure 13.

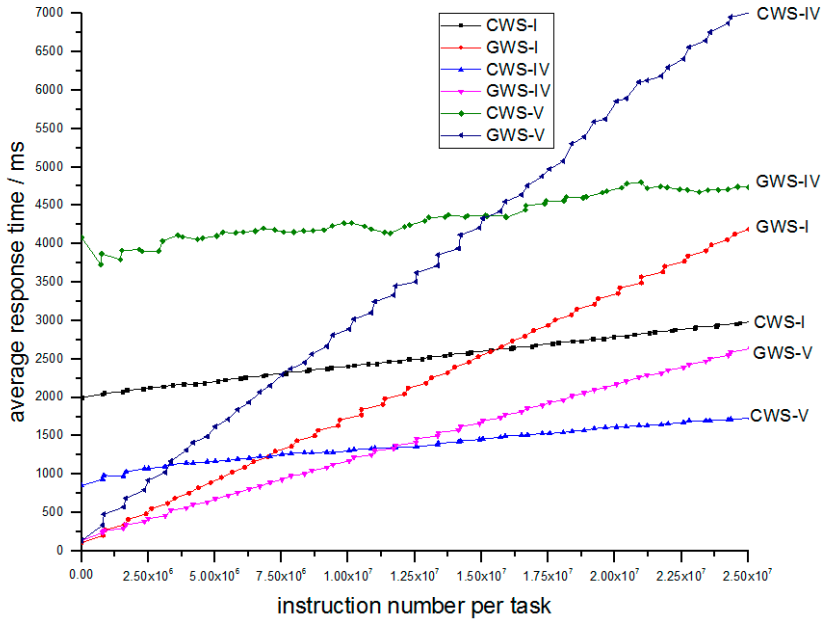

Figure 12. Relation of average response time and number of instructions per task between different process rates.

Table 6. Parameters of Simulation V.

Network Name	$\overline{\lambda}$	$\overline{s}$	$\overline{\pi}$	N
CWS-I	1.2	2.5×10^7	1×10^7	100
GWS-I	1.2	2.5×10^7	1×10^7	100
CWS-VI	1.2	2.5×10^7	0.8×10^7	100
GWS-VI	1.2	2.5×10^7	0.8×10^7	100
CWS-VII	1.2	2.5×10^7	1.2×10^7	100
GWS-VII	1.2	2.5×10^7	1.2×10^7	100

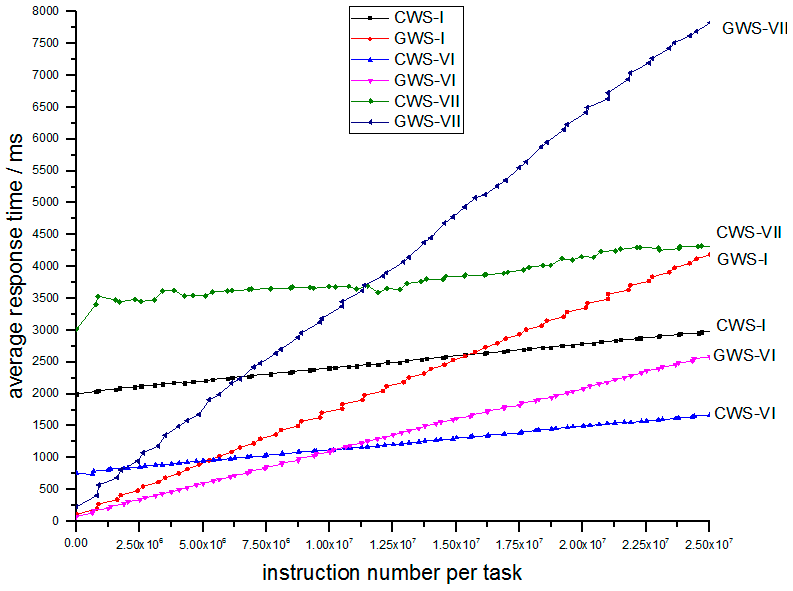

Figure 13. The relation of average response time and number of instructions per task between different average instruction number per task.

In Figure 13 above, we can find that the GWS-VI and GWS-VII are still better for light tasks, and while average number of instructions per task increases, the response time also increases.

Finally in Simulation IV, we explore the scalability of GWS by changing the FCN number N from 100 to 20 and 100 to 500. Parameters are shown in Table 7. The results are depicted in Figure 14.

Table 7. Parameters of Simulation VI.

Network Name	$\overline{\lambda}$	$\overline{s}$	$\overline{\pi}$	N
CWS-I	1.2	2.5×10^7	1×10^7	100
GWS-I	1.2	2.5×10^7	1×10^7	100
CWS-VIII	1.2	2.5×10^7	1×10^7	20
GWS-VIII	1.2	2.5×10^7	1×10^7	20
CWS-VIIII	1.2	2.5×10^7	1×10^7	500
GWS-VIIII	1.2	2.5×10^7	1×10^7	500

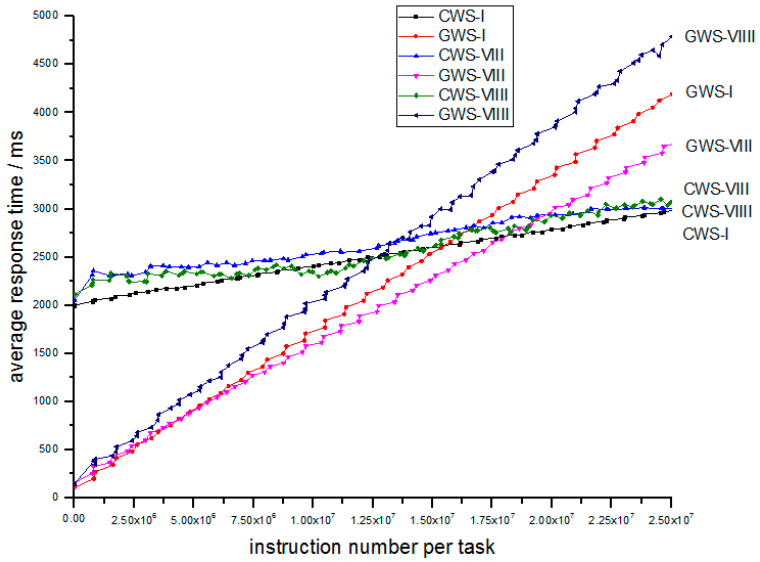

Figure 14. Relation of average response time and number of instructions per task between different FCN number.

According to Figure 14, we find that the GWS is well scalable for network size as the slopes of the two lines are almost the same, so when the FCN cluster grows bigger, the GWS is still stable and robust which is necessary for fog computing. This feature is proved in NVS which is size-irrelevant.

From the simulations above, we conclude that by increasing average task arrival rate, decreasing average process rate and increasing average number of instructions per task, the response time of CWS and GWS will increase but the relative relation never changes. The GWS network always gives better performance for light tasks than the CWS network and the GWS is scalable as it is size-irrelevant.

5. Conclusions

Firstly, this paper clarifies the task processing mechanism of FCNs and proposes to replace the FCFS mechanism with the TS mechanism in FCNs. Then the validity and necessity of the TS mechanism is proved in both theory and simulation. A measurement of FCN performance for multi-class tasks is also put forward as the concurrency coefficient.

Secondly, the paper adjusts the work-stealing algorithm for the TS system by setting up residual deques for raw tasks and stealing raw tasks from other residual deques. A variant of the work-stealing algorithm is modeled as a cooperative game between FCNs.

Finally, the paper proposes a collaborative algorithm GWS to balance the load between FCNs. The collaborative algorithm is a variant of work-stealing, which is based on game theory. According to the Nash bargaining solution, we can obtain Pareto optimality. Through simulations, we prove that GWS can obtain better performance than CWS scheduler, especially for light tasks.

The paper introduces GWS, which places FCNs in collaboration with each other to achieve better performance; however, cooperation between FCNs like task-stealing causes an information swap, so our future studies will examine privacy security for IoT applications. We will aim to study safe ways of cooperating without leaking any user information.

Author Contributions: Z.Z. (Zhenjiang Zhang) and Z.Z. (Zhangbing Zhou) conceived and designed the experiments; H.X. performed the experiments; H.X. analyzed the data; Z.Z. (Zhenjiang Zhang) contributed analysis tools; H.X. wrote the paper.

Funding: This research was supported by the National Natural Science Foundation of China under Grant No. 61772064 and the Fundamental Research Funds for the Central Universities 2017YJS005.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Dastjerdi, A.V.; Buyya, R. Fog Computing: Helping the Internet of Things Realize Its Potential. *Computer* **2016**, *49*, 112–116. [CrossRef]
2. Bittencourt, L.F.; Diazmontes, J.; Buyya, R.; Rana, O.F.; Parashar, M. Mobility-Aware Application Scheduling in Fog Computing. *IEEE Cloud Comput.* **2017**, *4*, 26–35. [CrossRef]
3. Chang, Z.; Zhou, Z.; Ristaniemi, T.; Niu, Z. Energy Efficient Optimization for Computation Offloading in Fog Computing System. In Proceedings of the 2017 IEEE Global Communications Conference (GLOBECOM 2017), Singapore, 4–8 December 2017; pp. 1–6.
4. Liu, L.; Chang, Z.; Guo, X.; Ristaniemi, T. Multi-objective Optimization for Computation Offloading in Fog Computing. *IEEE Internet Things J.* **2018**, *5*, 283–294. [CrossRef]
5. Zhang, W.; Zhang, Z.; Chao, H.C. Cooperative Fog Computing for Dealing with Big Data in the Internet of Vehicles: Architecture and Hierarchical Resource Management. *IEEE Commun. Mag.* **2017**, *55*, 60–67. [CrossRef]
6. Member, N.Y.; Shimojo, S.; Members, H.M. A load balancing algorithm on multiprocessor time-sharing systems. *Syst. Comput. Jpn.* **1990**, *21*, 1–10. [CrossRef]
7. Grosu, D.; Chronopoulos, A.T.; Leung, M.Y. Load Balancing in Distributed Systems: An Approach Using Cooperative Games. In Proceedings of the 6th International Parallel and Distributed Processing Symposium, Ft. Lauderdale, FL, USA, 15–19 April 2016; p. 10.
8. Blumofe, R.D.; Leiserson, C.E. Scheduling multithreaded computations by work stealing. *J. ACM* **1999**, *46*, 720–748. [CrossRef]
9. Yadav, R.; Kumar, P. Distributed Operating System. *ACM Comput. Surv.* **1996**, *28*, 225–227.
10. Soo, S.; Chang, C.; Loke, S.W.; Srirama, S.N. Proactive Mobile Fog Computing using Work Stealing: Data Processing at the Edge. *Int. J. Mob. Comput. Multimed. Commun.* **2017**, *8*, 1–19. [CrossRef]
11. Quintin, J.N.; Wagner, F. Hierarchical Work-Stealing. In Proceedings of the 16th International Euro-Par Conference on Parallel Processing, Ischia, Italy, 31 August–3 September 2010; pp. 217–229.
12. Anawar, M.R.; Wang, S.; Zia, M.A.; Jadoon, A.K.; Akram, U.; Raza, S. Fog Computing: An Overview of Big IoT Data Analytics. *Wirel. Commun. Mob. Comput.* **2018**, *2018*, 7157192. [CrossRef]
13. Wang, S.; Xu, J.; Zhang, N.; Liu, Y. A Survey on Service Migration in Mobile Edge Computing. *IEEE Access* **2018**, *6*, 23511–23528. [CrossRef]
14. Dijk, T.V.; Pol, J.C.V.D. Lace: Non-blocking Split Deque for Work-Stealing. In *Lecture Notes in Computer Science*; Springer: Berlin/Heidelberg, Germany, 2017; Volume 8806, pp. 206–217.
15. Kleinrock, L. *Queueing Systems, Volume I: Theory—Leonard Kleinrock*; Wiley: New York, NY, USA, 1975; p. 417.
16. Kleinrock, L. *Queueing Systems: Volume II: Computer Application*; Wiley Interscience: New York, NY, USA, 1976; p. 548.

17. Aase, K.K. The Nash bargaining solution vs. equilibrium in a reinsurance syndicate. *Scand. Actuar. J.* **2009**, *2009*, 219–238. [CrossRef]
18. Stefănescu, A.; Stefănescu, M.V. The arbitrated solution for multi-objective convex programming. *Rev. Roum. Math. Pures Appl.* **1984**, *29*, 593–598.