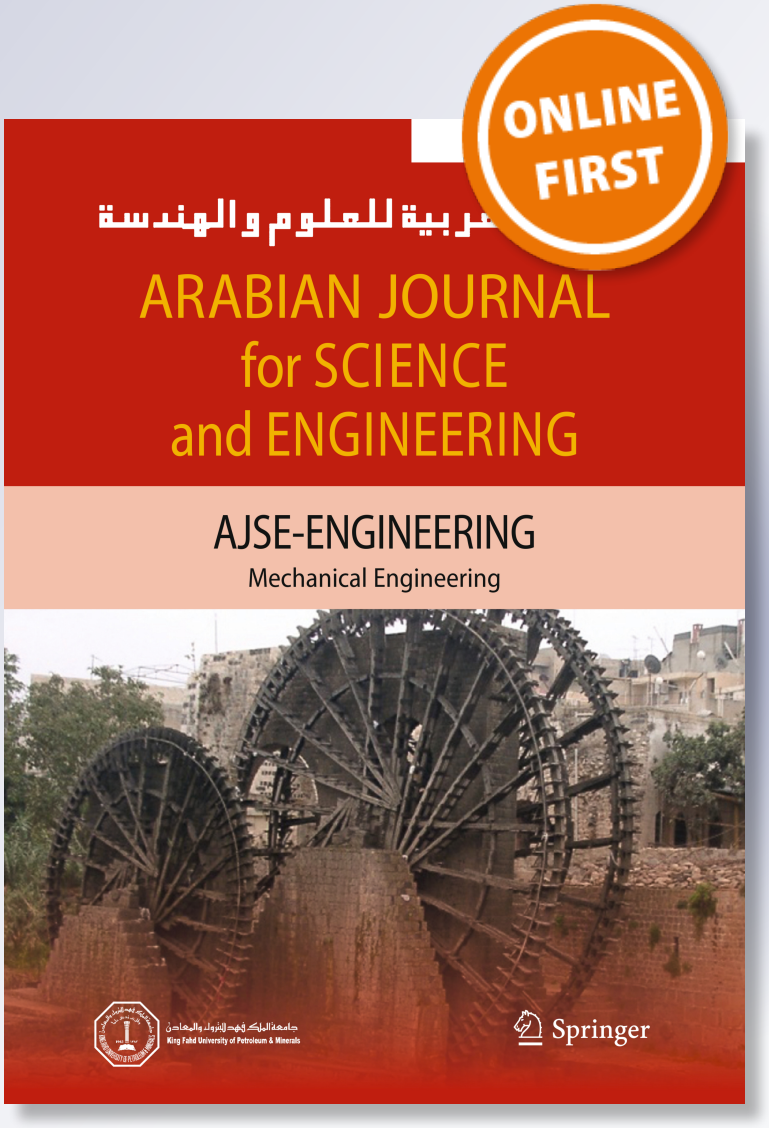
SSLB: Self-Similarity-Based Load Balancing for Large-Scale Fog Computing

Changlong Li, Hang Zhuang, Qingfeng Wang & Xuehai Zhou

Arabian Journal for Science and Engineering

ISSN 2193-567X

Arab J Sci Eng
DOI 10.1007/s13369-018-3169-3



Springer

Arabian Journal for Science and Engineering
https://doi.org/10.1007/s13369-018-3169-3

RESEARCH ARTICLE - COMPUTER ENGINEERING AND COMPUTER SCIENCE

SSLB: Self-Similarity-Based Load Balancing for Large-Scale Fog Computing

Changlong Li[1] · Hang Zhuang[1] · Qingfeng Wang[1] · Xuehai Zhou[1]

Received: 16 July 2017 / Accepted: 14 February 2018

Abstract
As a novelty approach to achieve Internet of things and an important supplement of cloud, fog computing has been widely studied in recent years. The research in this domain is still in infancy, so an efficient resource management is seriously required. Existing solutions are mostly ported from cloud domain straightforward, which performed well in many cases, but cannot keep excellent when the fog scale increased. In this paper, we examine the runtime characteristics of fog infrastructure and propose SSLB, a self-similarity-based load balancing mechanism for large-scale fog computing. As far as we know, this is the first work try to address the load balancing challenges caused by fog's 'large-scale' characteristic. Furthermore, we propose an adaptive threshold policy and corresponding scheduling algorithm, which successfully guarantees the efficiency of SSLB. Experimental results show that SSLB outperforms existing schemes in fog scenario. Specifically, the resources utilization of SSLB is 1.7× and 1.2× of traditional centralized and decentralized schemes under 1000 nodes.

Keywords Fog computing · Internet of things · Dynamic load balancing · Self-similarity

1 Introduction

Cloud computing [1] has emerged as a dominating paradigm in various domains during last few years. By organizing and managing resource-rich machines as a whole, a large number of services are provided. However, with the development of the Internet of things (IoT), unprecedented amounts of data are generated, which overwhelm cloud's storage and processing capacity: as estimated, it could have an economic impact of 11 trillion dollars per year by 2025 [2]. What is worse, most IoT devices located on the edge network. As a result, the delay caused by transferring data/applications to the cloud and then back to the end user is always unacceptable. Fortunately, the success studies on software- defined networking [3], network function virtualization [4] and 5G cellular networks [5] make people realize that resource-poor devices (which means the computing and storage capacity is seriously constrained) such as routers, switches, base stations, and end devices have potential to provide more efficient services by coordinating with each other, so-called fog computing. As presented by Bonomi, fog computing extends the cloud computing paradigm to the edge of the network [6].

Nowadays, more and more applications that do not fit well with the cloud, like geographically distributed applications, intelligent transportation applications, or applications with high demand on low latency, are benefiting from fog computing paradigm. For instance, Cao et al. [7] proposed FAST, a fog computing- assisted distributed analytics system in monitoring fall for stroke patients. In their work, authors carefully examined the special needs and constraints of stroke patients and proposed a patient-centered design. Zhu et al. [8] discussed the use of edge servers for improving Web sites performance. In Zhu's work, users connect to the Internet through fog nodes; hence, each HTTP request made by a user goes through a fog node. There are also many other services provided by fog, like [9,10]. In order to support these services with high performance, submitted applications should be efficiently managed, such that resources in fog can be fully utilized and no request has to wait for a long time. This paper focuses on digging out the potential of fog devices by maintaining the load balancing in the infrastructure.

The study of load balancing in fog computing started from 2015. In [11], Oueis et al. proposed a load balancing scheme in fog computing in order to improve users' quality of experience (QoE). In their work, a reduced com-

✉ Changlong Li
liclong@mail.ustc.edu.cn

[1] University of Science and Technology of China, Hefei, China

Published online: 23 February 2018

plexity job scheduling algorithm was introduced. Although the proposed approach outperforms earlier solutions in many cases, it suffers from a high complexity for large- scale fog computing, since the algorithms proposed are more suitable for low dense computing infrastructure. After that, Deng et al. [12] studied the trade-off between power consumption and delay in a cloud–fog computing system and then proposed a novel model to support the workload allocation in cloud–fog computing. However, the work mainly considered centralized infrastructures, which conflicts with fog's decentralized nature. In summary, existing works can be categorized as centralized [13] and decentralized scheme [14]. The centralized scheme manages the load on each server of the system by maintaining a master node in fog, and decentralized schemes are arguably more robust since all nodes in the fog bear equal importance in a distributed approach. However, the above solutions were mostly ported from cloud domain straightforward and did not fully consider the characteristics of fog computing (we will discuss in detail in Sect. 2). Our comparison experiments show that both traditional centralized and decentralized schemes suffer challenges in the fog scenario, especially considering the fact that fog scale is always very large.

In this paper, we propose SSLB, a self-similarity-based load balancing mechanism, which makes full use of the advantages of both centralized and decentralized schemes. In the design, fog nodes are clustered as multiple groups, which is called cell. In each cell, nodes determine task schedule independently. Meanwhile, workload migrations across cells will happen once predefined conditions triggered. With the help of this novelty structure, the issues caused by fog's 'large-scale' characteristic are addressed. Furthermore, we propose an adaptive threshold policy and corresponding scheduling algorithm, which guarantees the efficiency of SSLB. Experimental results show that SSLB outperforms existing load balancing schemes in this scenario. Specifically, the resources utilization of SSLB is 1.7× and 1.2× of traditional centralized and decentralized schemes under 1000 fog nodes.

The rest of this paper is organized as follows. In Sect. 2, we introduce the fog computing model and discuss the challenges in this scenario. Then, we describe design details of SSLB in Sect. 3. Section 4 simulates the performance of the proposed mechanism. In Sect. 5, we show the related works, and finally, Sect. 6 presents our conclusion and future works.

2 Problem Statement

2.1 Fog Computing Infrastructure

Fog computing, also termed edge computing [15], is designed as a distributed computing infrastructure that provides elastic resources by bringing services to devices that located

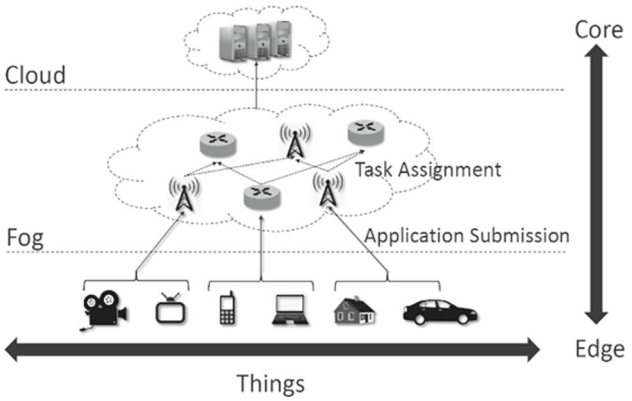


Fig. 1 Fog computing infrastructure. Devices in edge network submit application requests to fog. Most of the applications will be executed by the fog in parallel, while part of them will be further uploaded to cloud

geographically near to the end user. It is based on the idea of enabling computing directly at the edge of the network instead of being transmitted to the core data center infrastructure [16]. Figure 1 illustrates a brief structure of the system model. As shown in the figure, fog nodes communicate with users (things) through wireless connections such as WiFi or cellular networks. In addition, fog connects to the cloud infrastructure over the Internet so as to leverage the rich computing and content resources of the cloud. The connection with cloud infrastructure can be established via both wired and wireless approaches. Once applications were submitted to the fog, they will be processed by fog nodes. Part of them will be uploaded to cloud if necessary. In this paper, we focus on the applications processed on fog infrastructure.

When an application is submitted, the operation of the system model is as follows:

- First of all, end user submits an application request to fog node located at the edge of the network.
- Second, the node decomposes the application into a set of tasks.
- Next, the tasks will be assigned to multiple nodes and line in the queue.
- Then, each node sends its processing results to an administrator node.
- Finally, the result is sent back to the mobile user as a service response.

2.2 Problem Formulation

To make the task allocation problem easier to understand, we formulate the applications processing problem in fog computing environment. An application is defined as a process that corresponds to a service request established by an end user. We assume the system consists of m fog nodes, Fn_1 to Fn_m, and n applications. These applications are represented as:

$$\mathrm{AP} = \left\{\mathrm{App}_1, \mathrm{App}_2, \ldots, \mathrm{App}_n\right\} \tag{1}$$

Once received by the fog, each application among the n applications can be partitioned into a set of tasks:

$$\mathrm{TS} = \left\{\mathrm{AppTsk}_{i1}^{x}, \mathrm{AppTsk}_{i2}^{y}, \ldots, \mathrm{AppTsk}_{iq}^{z}\right\} \tag{2}$$

The tasks are assigned to various nodes for parallel processing. Meanwhile, each node maintains a queue:

$$\mathrm{QE} = \{\mathrm{Que}_1, \mathrm{Que}_2, \ldots, \mathrm{Que}_r\} \tag{3}$$

Tasks are queued in the system when nodes are busy. Here r is the length of the task queue. For instance, tasks of App_i:

$$\mathrm{TS} = \left\{\mathrm{AppTsk}_{i1}^{3}, \mathrm{AppTsk}_{i2}^{6}, \mathrm{AppTsk}_{i3}^{9}\right\} \tag{4}$$

are performed as follows: 1st task (AppTsk_{i1}^{3}) is assigned to node Fn_3, 2nd task (AppTsk_{i2}^{6}) is assigned to node Fn_6, and 3rd task (AppTsk_{i3}^{9}) is assigned to node Fn_9. Consequently, each node executes a set of disjoint subset of the decomposed applications set. For its assigned applications, node Fn_j maintains tasks in local queue as follows:

$$Que_j = \left\{\mathrm{AppTsk}_{ax}^{j}, \mathrm{AppTsk}_{by}^{j}, \mathrm{AppTsk}_{cz}^{j}\right\} \tag{5}$$

In the system, workloads are defined by the number of tasks on the node as well as their expected execution time. The total execution time of all tasks (workloads) in the queue is shown as follows:

$$\mathrm{Execution\_Time}\left(\mathrm{Que}_j\right) = \sum\nolimits_{i \in \mathrm{Que}_j} \mathrm{AppTsk}_i.\mathrm{ExeTime} \tag{6}$$

where AppTsk_i.ExeTime is the execution time of AppTsk_i in the queue. Therefore, the load balancing problem can be formulated as a recommendation of set.

$$\mathrm{TS} = \left\{\mathrm{App}_1\mathrm{Tsk}, \mathrm{App}_2\mathrm{Tsk}, \ldots, \mathrm{App}_m\mathrm{Tsk}\right\} \tag{7}$$

In order to achieve the load balancing across the nodes, the gap between the heaviest loaded and lightest loaded nodes Gap (TS) should be minimized.

$$\begin{aligned}\mathrm{Gap\ (TS)} = {} & \mathrm{Max}_{j=1}^{m}\left[\mathrm{Execution\_Time}\left(\mathrm{Que}_j\right)\right] \\ & - \mathrm{Min}_{k=1}^{m}\left[\mathrm{Execution\_Time}\left(\mathrm{Que}_k\right)\right]\end{aligned} \tag{8}$$

2.3 Challenges Discussion

Before addressing the load balancing issue of fog computing, the characteristics of fogs and challenges in this scenario are carefully considered.

In our survey, the number of fog devices (e.g., routers, switches, base stations) are much more than that of cloud data centers. For instance, the number of 4G base stations which belong to China mobile by 2016 is 1.44 million [17], and the scale of switches and routers is even larger. In this condition, many challenges are caused by the characteristics of fog computing. (1) Different from devices in cloud data center, either computing or storage capacity of the fog device is seriously limited. For example, the clock rate of the processor in a router is always no more than one GHz. As a result, centralized load balancing schemes cannot manage the system efficiently, since the performance bottleneck of the central node responsible for workload allocation can easily occur. (2) Fog nodes communicate with each other in wireless approach, so the bandwidth is always small. As estimated, the maximum theoretical network bandwidth of WiFi is 300 Mps (802.11n standard) and 1000 Mps (802.11ac standard). Generally, it operates even no faster than about 50% of the theoretical peak. As a result, traditional decentralized schemes cannot port to fog straightforward either, because each node in the system needs to broadcast its load information to the rest of the nodes for updating. The communication overhead is always too high to meet users' high demand on low latency. (3) Unlike cloud infrastructure, fog nodes tend to join in or drop out more frequent, so the proposed mechanism should also take this problem into consideration.

3 Proposed Approach

3.1 Design Goals

As introduced in Sect. 2, the load balancing of fog computing suffers many difficulties and challenges, which motivates us to propose SSLB. The design of SSLB is driven by the following guiding principles.

- *Low latency* Different from traditional clouds which focus on improving the ability of large-scale data process, users in fog pay more attention to high speed interaction.
- *Flexibility* Fog nodes always join a system temporarily; meanwhile, they tend to leave frequently. Therefore, the mechanism should be flexible enough to reflect newly joined nodes as well as nodes revocation.
- *Scalability* There are always tens of thousands of edge devices in a fog. The cost of load balancing should not grow significantly as the number of fog nodes increasing which means SSLB should work well no matter the system scale is small or large.

The design goals of SSLB are to combine these three attractive features in large-scale fog computing, which are further

challenging due to the inherent tension between some of them.

3.2 SSLB Architecture

Load balancing approaches are classified as static and dynamic strategies. Static strategies are suitable for homogeneous and stable environments and usually work well. In the environment, applications will be assigned to nodes once submitted. However, they are not flexible enough to match the dynamic changes in fog during execution time. In our design, we choose the dynamic approach to adapt to changes in fog environments. In real-world implementation, three components are maintained in each fog node: *LBMonitor*, *LBScheduler*, and *LBMessenger*. As illustrated in Fig. 2, the components on each node are responsible for monitoring, scheduling, and communicating.

LBMonitor is responsible for obtaining both intranode and internode information, which includes queue length of tasks, expected execution time, load information of other nodes. The information collected by the monitor is acquired by the LBScheduler component. Once an unbalance is detected, a rescheduling will happen soon. The LBScheduler locates on each node and represents the distributed scheduler that executes the distributed load placement policy. It executes the distributed scheduling algorithm on every node. In our design, only tasks assigned to the node can be managed by this component, which can make decisions to reassign the tasks or not. To get information from neighbor nodes and further help LBScheduler make decisions, nodes in the system need to communicate with each other. This component is orchestrated as a classical adaptive software system that reacts to internal and external changes of the operating conditions through a feedback control loop. LBMessenger is right designed to send and receive messages across nodes. To make each node be informed of all the load-related information of the other nodes, the thread of LBMessenger is periodically executed. Furthermore, to address the churn issue, such as node crashes or joins, heartbeat information is introduced by LBMessengers.

In summary, SSLB detects local loads, communicates with other nodes for global information, and then makes decisions to balance the workload periodically. Note that these components are preferred to avoid complex in terms of implementation and operations since a high implementation complexity would lead to a complex process which could cause a number of negative performance issues.

3.3 Self-Similarity Structure

To avoid the drawbacks of traditional centralized and decentralized schemes in fog scenario, we present a novel structure to reorganize fog nodes. The idea of SSLB is motivated by a conspicuous feature in biosphere, called self-similarity [18,19]. It is a typical property of artificial fractals. In the structure, a self-similar object is exactly or approximately similar to a part of itself (i.e., the whole has the same shape as one or more of the parts). Taking Fig. 3 as an example, the system contains 64 fog nodes. They are clustered to 8 cells. Each cell consists of 8 nodes, which can be described as set C. To achieve the communication between cells, each cell elects a *seed* (nodes with white color) for outside communication. The seed nodes are not responsible for the decision making of the whole system, so it would not become the bottleneck. Assume the whole system is described as W, and the structure of W is $F(C)|C \in W$. A part of W appears to be the same when compared, so the structure of the subset C is $F(S)|S \in C$. Thus, the structure can be defined as follows.

$$W = F(C)|C \in W = F[F(S)|_{S \in C}]|_{C \in W} \tag{9}$$

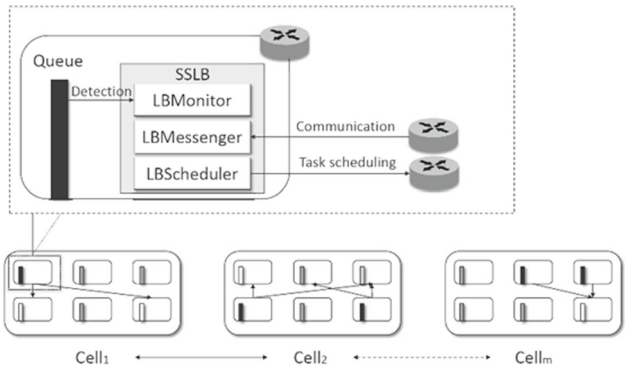


Fig. 2 SSLB architecture. SSLB consists of three important components, LBMonitor, LBScheduler, and LBMessenger. They are maintained at every fog node

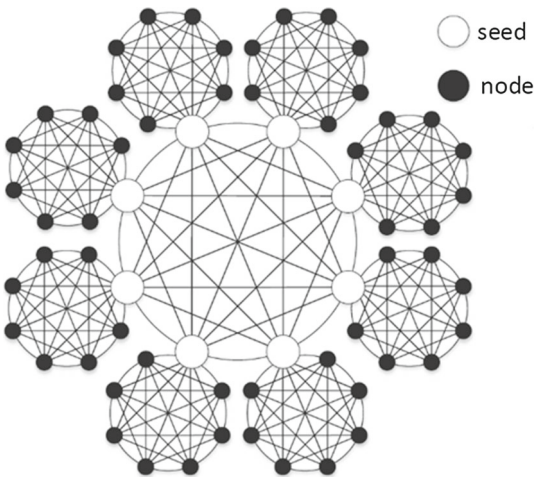


Fig. 3 Self-similarity structure. In this structure, adjacent nodes are clustered as a group, called cell. Nodes in the same cell interact with others while electing a seed for outside communication

To realize the architecture, how to divide fog nodes into cells is the first issue to address. We utilize k-means algorithm to split the system into cells. In the design, adjacent nodes in geography are clustered as a cell. The scale of a cell is configurable. We organize cells in this way because fog devices always communicate with adjacent nodes with higher bandwidth and lower energy consumption. What is more, our statistic shows that devices in the adjacent geographical region tend to present similar behavior, such as join in or revocation. To calculate the 'distance' between nodes and further help clustering cells, Euclidean distance [20] is utilized to perform distance calculations. In the design, fog nodes are described as the set V of n points in Euclidiean space. The goal is to partition V into k sets called cells $C_1, C_2, \ldots, C_k$ and choose one seed s_i for each cell C_i to minimize $\sum_{i=1}^{k} \sum_{v \in S_i} ||v - s_i||_2^2$. Here k-means algorithm is chosen because it is easy to implement. Most importantly, it has the ability to cluster nodes with large scale. The Euclidean distance can be expressed as the following matrix:

$$R = \begin{bmatrix} x_{1,1} & x_{1,2} & \ldots & x_{1,m} \\ x_{2,1} & x_{2,2} & \ldots & x_{2,m} \\ \vdots & \vdots & \ldots & \vdots \\ x_{m,1} & x_{m,2} & \ldots & x_{m,m} \end{bmatrix} \tag{10}$$

In the matrix, $x_{i,j}$ is zero when $i = j$. Besides, $x_{i,j} = x_{j,i}$. Consider a graph on m vertices denoted *Fn1, Fn2, ..., Fnm*, whose elements correspond to cluster's m nodes and distance between them represents transfer time. The whole procedure consists of following steps:

- *Step 1* Select k out of the given m patterns as the initial cluster. Assign each of the remaining $m - k$ patterns to one of the k clusters; a pattern is assigned to its closest cluster.
- *Step 2* Compute the cluster based on the current assignment of patterns.
- *Step 3* Assign each of the m patterns to its closest cluster.
- *Step 4* If there is no change in the assignment of patterns to clusters during two successive iterations, then stop; else, go to Step 2.

Once cells have been generated, they would communicate with each other through seed nodes. If the load balancing condition was triggered, which means the workload of one node (cell) is significantly heavier than the others and exceeds the threshold, task scheduling will happen between them. The following subsection will introduce the dynamic threshold definition policy. Meanwhile, to avoid communication overhead, we utilize and adapt the power of two choices technique [21] in the design. As M. Mitzenmacher researched, with high probability, the maximum load in any node is approximately $\log n \div \log \log n$. Suppose instead that each node is placed sequentially into the least full of d nodes chosen independently and uniform at random, the maximum load is then only $\log \log n \div \log d + O(1)$ with high probability. In implementation, d is a configuration parameter. The self-similarity structure is organized by metadata, which contains all related information such as the map between fog nodes in the same cell. As mentioned above, node crashes or joins will be detected by LBMessenger components. Once churns happened, the metadata will be updated at once. To reduce the metadata updating cost, the metadata is maintained in memory and write back to disk periodically. Meanwhile, since the node crashes or joins are not happens very frequent, the cost is acceptable.

3.4 Threshold Policy

Algorithm 1 : *Task Distributing*

Input: $System\_state == 'light'$

```
for A ∈ Schedulers[ ] do
    if A.load > αTL then
        B, C ← random in Schedulers[ ];
        B ← light_choose(B, C);
        if A.load > B.load then
            task_distribute(A, B);
        end if
    end if
end for
```

Besides the self-similarity-based architecture, an adaptive threshold policy also plays an important role in the design of this load balancing mechanism. Note that thresholds are detrimental to the fitness of the generated task mapping, they must be dynamic (static threshold values do not work well in this scenario since the load state of the system is changing all the time). In the design, workload migration happens in each cell independently. In order to determine proper thresholds, the average load of each cell must be calculated first:

$$L_{\text{ave}}^{i} = \left(\sum_{j=1}^{n} \left[\text{CTT}_j + \text{QE}_j \right] \right) \div n.$$

Algorithm 2 : *Task Grasping*

Input: $System\_state == 'heavy'$

```
for A ∈ Schedulers[ ] do
    if A.load < βTH then
        B, C ← random in Schedulers[ ];
        B ← heavy_choose(B, C);
        if A.load < B.load then
            task_grasp(A, B);
        end if
    end if
end for
```

Here n represents nodes number of Cell_i, CTT_j is the remaining execution time of the task currently being pro-

cessed by node j, and QE_j is the execution time of all the tasks waiting at task queue on node j. Since using this value as a threshold is too restrictive, we further present two dynamic thresholds based on L^i_{ave}: *light threshold* and *heavy threshold*. Nodes with loads that are less than the light threshold are referred to as *light* state, whereas those with loads higher than the heavy threshold are categorized with *heavy* state. Finally, those with loads between the heavy and light thresholds are defined as normal state. The heavy and light thresholds are derived as: $TH = H * L^i_{ave}$ and $TL = L * L^i_{ave}$. Here TH is the heavy threshold, TL is the light threshold, H and L are constant multipliers that determine the flexibility of the load balancing mechanism. H is greater than one, and it determines the amount that the average workload can be exceeded by before the node becomes heavily loaded. Conversely, L is less than one and it shows how much the processor loads can be short of the average before the node becomes lightly loaded. In our real implementation, the values are set 1.2 and 0.8 as default. These threshold values are set at initialization stage and will be recalculated periodically during the running time.

In order to reduce the communication overhead, we make efforts to avoid nonsense task scheduling requests. In the design, two independent algorithms are proposed: *Task Distributing* and *Task Grasping*. They run under different conditions. *Task Distributing* is triggered when a node is regarded as a heavy state, while most of the other nodes in the same cell are light. For example, if the state of $Cell_i$ is light, node A will choose node B and C randomly and communicate with them as long as its workload $t > \alpha * TL$. Supposing lighter one of the two receivers is also lighter than A, part of workloads will be migrated on it from A; else, the scheduling request will be refused (Algorithm 1). In the algorithm, only heavy nodes perform probing, light nodes do not. A random probe performed by a heavy node is more likely to find light nodes and avoid invalid probing when the state of a cell is light.

The ultimate goal of load balance is the performance optimization. Therefore, one case should not be ignored: there is a cluster which all nodes undertake heavy burdens, with the help of scheduling algorithm, some tasks from B migrated to D; then later before those migrated tasks executed, since new coming, D may become the hot spot and tasks will be migrated back to B for load balance. In this case, load balancing scheme just did a task exchange between two nodes, performance of the system reduced instead of improved. To avoid the challenge, we present Algorithm 2 to reduce the probability of invalid scheduling, so-called *Task Grasping*. In this algorithm, only light nodes perform probing, heavy nodes do not perform any probing at all: nodes check load regularly and make a comparison with the threshold. If the load of node A is less than β^* TH, then task grabbing request will be sent to two randomly selected nodes B and C. Supposing heavier one of the two also heavier than A, tasks scheduling will happen; else the request will be denied.

The design of SSLB should avoid conflict, given that concurrently load balancing mechanism may make conflicting task scheduling decisions among nodes. Consider a case where two different nodes A and B probe the same idle node C at the same time; since C is idle, both

A and B are likely to place their tasks on it; however, only one of the two load placed on the node will arrive in an empty queue and only one prerequisite of the two migrate replications satisfied, C will no longer idle once tasks from one node arrived. We avoid this conflict by setting lock: in the case that both nodes A and B probe C for load balancing, they will exam the state of C. If locked, the request would be refused; else the applicant will establish a connection and sustain a lock on C until migration completed and connection cut. In the real implementation, only small amount of schedulers will send requests, so the ratio of lock conflict is very low. On the other hand, the scheduling mechanism between cells is similar to that between nodes inside a cell. Of course, many other solutions such as 'late binding' [22] can be introduced to SSLB too. Besides, as introduced above, LBScheduler components make decision autonomously by analyzing load information. We achieve information sharing through multicast: nodes share information with each other by LBMessenger in the cell through multicast and make decisions after that.

3.5 Discussion

In this subsection, we will discuss the reason why self-similarity- based structure outperforms traditional load balancing schemes. The key demand of a successful load balancing scheme is that if the system starts in balanced, it will stay that way; if not, the load of the system should tend to balance with the regulation of schedulers. Meanwhile, the key evaluation criteria of load balancing effect are the idle ratio of system nodes. As a result, the idle ratio of one system is very small at the beginning of each load balancing cycle. However, as time goes on, there are many applications completed while many new applications join in, so the idle ratio of the system will increase before the load balancing scheme is triggered in the next cycle. Theoretically, the time window of each cycle should not be very large.

As shown in Fig. 4, $f(t)$ means the maximum idle ratio of fog machines when time window is t. To help to understand, we assume applications are submitted to the system in random: the probability of 'busy' to 'idle' for each node per unit time t_0 is constant value p, and 'idle' to 'busy' is q. If the scale of the cluster is F and the idle ratio at time 0 is idle state $f(t_0)$, it can be proved that ($t_n = n * t_0$):

$$f(t_n) = \frac{Fp}{p+q} + \left[\frac{Fp}{p+q} - f(0)\right] * [1-(p+q)]^n \quad (11)$$

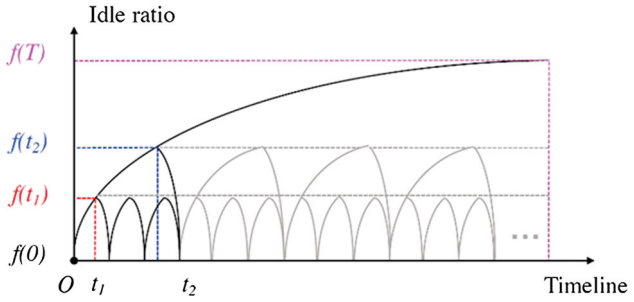


Fig. 4 Idle ratio of fog systems with multiple time windows (t_1, t_2) of load balancing cycle

If time is long enough, which means $n \to \infty$, the idle ratio will fluctuate around $Fp \div (p+q)$. Denoted by $S(t_n) = \sum_{i=0}^{n} f(t_n)$ the total idle rate within time T.

$$S(t_n) = \frac{Fp - f(0) * (p+q)}{(p+q)^2} * [1-(p+q)]^n - \frac{Fp - f(0) * (p+q)}{(p+q)^2} + \frac{Fp}{p+q} * n \quad (12)$$

From the equation, we know that the total idle ratio of the system will increase with the increase in n (time window). In other words, the time window between two load balancing cycles should not too large. Unfortunately, the time window of traditional load balancing schemes can hardly be maintained in a small level when the scale of the system increased to very large. It is because the computing or communication overhead will become the bottleneck. Based on the self-similarity structure, seed node only needs to compute the load information inside the cell, and network communication mainly happens inside the cell, so it will not take too much time to make load balancing decisions even if the scale of the system is very large.

4 Implementation and Experiment

In this section, we first introduce a simulation platform for fog computing, iFogSim, and show how SSLB is implemented based on it. Then, we evaluate the performance of SSLB and make a comparison with traditional centralized as well as decentralized schemes.

4.1 Implementation Overview

4.1.1 Simulation Environment

To help to evaluate the performance of SSLB, we need a platform that enables the quantification of the performance of resource management policies on IoT and fog computing. After carefully survey and comparison as shown in Table 1, we choose iFogSim [23] as the simulation platform. IFogSim is proposed by Gupta in 2016. It is designed in a way that makes it capable of evaluation of resource management policies applicable to fog environments with respect to their impact on latency, energy consumption, network congestion, and operational costs. The basic event simulation functionalities of iFogSim, like data centers or communicate with each other by message passing operations, are same with CloudSim [24]. Mainly classes which are utilized in our evaluation are as follows:

FogDevice This class specifies hardware characteristics of fog devices (nodes) and their connections to others. The major attributes of this class are accessible memory, processor, storage size, and network bandwidth. In the following experiments, all fog nodes are initialized based on *FogDevice*.

Tuple The other important class is *Tuple*. It forms the fundamental unit of communication between entities in the fog. Tuples are represented as instances of this class in iFogSim, which is inherited from the *Cloudlet* class of CloudSim. A tuple is characterized by its type and the destination application modules.

Application *Application* is further classified to three classes: *AppModule*, *AppEdge*, and *AppLoop*. In this paper, we mainly utilize *AppModule*. The instance of *AppModule* represents processing elements of fog applications. It is implemented by extending the class *PowerVm* in CloudSim. For each incoming tuple, an *AppModule* instance processes it and generates output tuples.

Note that the definition of class names in iFogSim is not straightforward. In our implementation, application is generated based on Tuple and will be submitted to AppModule for execution. FogDevice is extended from DataCenter, and it is responsible to manage the fog cluster.

4.1.2 SSLB on iFogSim

To implement SSLB on iFogSim, there are many instances to simulate first, including fog nodes and applications. Fog device instance extends from FogDevice class, which includes many AppModule objects. Specifically, 1000 AppModules stand for 1000 virtualized fog nodes. In the class, AppModule.mips() determines the computing capacity of each node, while AppModule.bw() defines node's bandwidth. Application instances extend from Tuple. In the class, Tuple.length() determines how long the application is, while cloudletFileSize() describes the data size the application needed. So the execution time of one application is AppModule.mips/Tuple.length. Meanwhile, to simulate the submission time of applications as well as the load

Table 1 Comparison of common used simulation platforms in recent years. Language, open-source (OS) issue, and publication year are considered

Simulation platform	Language	OS	Publication
CloudSim	Java	Yes	2009
EMUSim	Java	Yes	2012
CloudNetSim++	C++	Yes	2014
CloudSimSDN	Java	Yes	2015
WorkflowSim	Java	Yes	2015
CMCloudSimulator	Java	Yes	2016
iFogSim	Java	Yes	2016

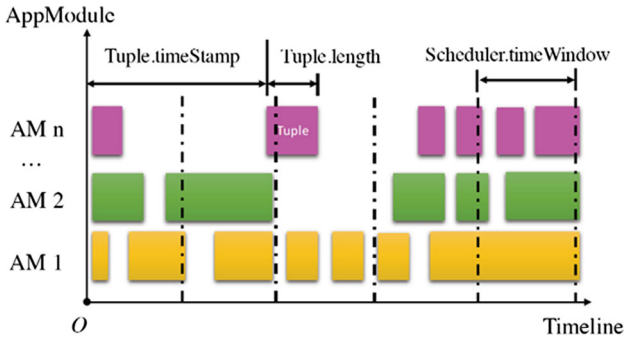


Fig. 5 Variables introduction. In the simulation, apps are submitted to AppModule after Tuple.timeStamp, and the execution time is determined by its Tuple.length. Schedulers will detect the system and start task scheduling per time window (load balancing cycle)

balancing cycle of schedulers, variables timeStamp and timeWindow are set in Tuple and SslbScheduler class.

As shown in Fig. 5, all instances are defined carefully for the simulation. In iFogSim, variables such as time window and application length can be defined by users directly. However, to let the variables closer to real-world situation, we rewrite PageRank as a demo to evaluate the execution latency and make a comparison with scheduler's overhead. Note that the PageRank algorithm runs on JVM instead of real-world fog devices (there are many works have discussed how to run the algorithm on JVM [25]. We also implemented this kind of application on other embedded environments instead of JVM, which is shown in [26]). In summary, the simulation includes two steps: in the first step, the application demo and scheduler algorithm is executed to determine the value of time window and application length; in step two, all values are configured and virtualized instances are generated for simulation.

4.2 Experimental Results

In the experiment, there are three important metrics are evaluated: system performance, resource utilization, and overhead. The system performance is used to check the efficiency of the system. Resource utilization means to check the utilization of resources in the fog. Overhead determines the amount of overhead involved while implementing SSLB.

4.2.1 Resource Utilization

Figure 6 shows the resources utilization of the system. Here resources utilization RU means the percentage between utilized nodes number and total nodes number. In the evaluation, we detect the RU of the system under centralized scheme (CS), decentralized scheme (DS), and SSLB conditions. The CS algorithm is ported from the task scheduling module of Apache Hadoop [27], while DS is ported from Karger's work based on peer-to-peer systems [14]. To achieve the CS and DS algorithm, the metadata structure is reorganized. Specifically, self-similarity-related structures such as cell and seed are deleted. Besides, many other function components are reusable, including LBMessenger and AppQueue. In the experiment, parameters under various data size and various nodes scale are evaluated. Besides, our benchmark consists of a large number of applications. They provide a good basis for testing the performance of fog computing. The default input data set contains one million pages which are generated by our distributed crawler. The figure proves that the resources utilization of SSLB is higher than DS and more stable than CS. It is mainly because the decision reaction time is shorter. The nodes number in the evaluation is 100. Furthermore, we detect the average resources utilization with different fog

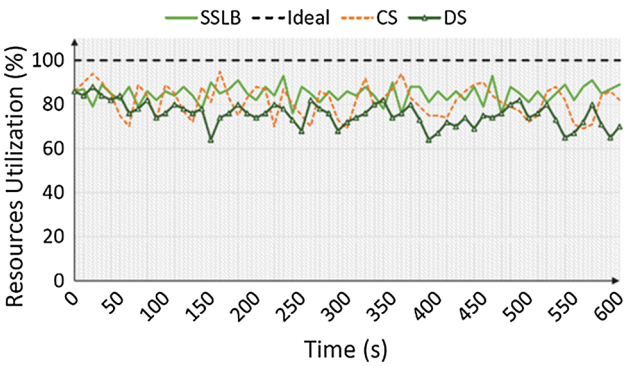


Fig. 6 Resource utilization of SSLB, CS and DS

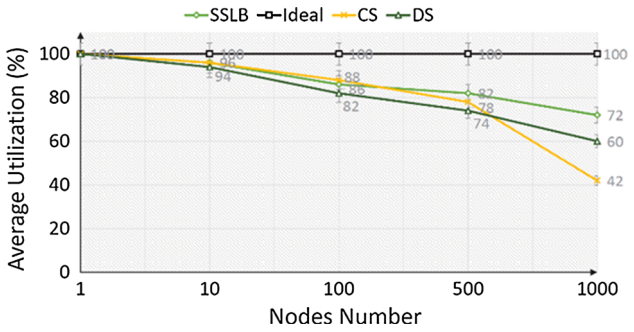


Fig. 7 Average resources utilization of SSLB, CS and DS with different fog scale

scales (the number of cloudlet object). Here $\text{Ave}(\text{RU}) = \sum_{i=1}^{t} \text{RU}_i/t$, where t means detection times during time interval T. As shown in Fig. 7, with the increase in nodes number, Ave(RU) of each scheme decreased. However, SSLB outperforms other schemes in the large-scale condition. Specifically, when the number rises to 1000, Ave(RU) equals to 72% while that of CS and DS are 42 and 66%.

4.2.2 System Performance

To understand the performance of SSLB, we submit applications on fog nodes and compare the performance of our proposed mechanism with CS and DS. Figure 8 shows the execution time of the system with different data sizes. The dataset contains a large amount of web pages as well as their properties, which will be accessed by the applications. From the figure, we find that the efficiency of SSLB is improved significantly with the increase in nodes number. Specifically, as shown in Fig. 8a, the execution time of SSLB is 30.36% of the 'no schedule' case, while the centralized and decentralized schemes are around 33.93 and 34.12% when the number of nodes is 10. However, when the scale increases to 1000, SSLB outperforms another scheme, and the execution time is 39.29% of the centralized scheme. The performance of centralized case cannot be improved significantly when the number of nodes increases to 1000 because the master node has already become a bottleneck. And for the decentralized case, communication overhead also substantially increased with the growing up of cluster scale. Similarly, we control the applications number and network bandwidth, in order to check how these factors affect SSLB's performance. The total execution speed of applications in our evaluation is not as fast as tradition cloud because the resource of fog nodes is seriously constrained.

In summary, after careful comparison between Figs. 8, 9 and 10, we find that: (1) all scheduling schemes have the ability to improve the performance of the fog system. (2) When the fog scale is small, centralized scheme presents a good performance. However, with the increase in fog nodes (from 100 to 1000), SSLB and decentralized scheme quickly exceed the performance of the centralized scheme. (3) When the scale of fog maintained at a big level, SSLB's performance is better than decentralized scheme, especially when the network bandwidth is small (Fig. 10). From the tendency, we note that SSLB will perform better when cluster scale keeps on increasing.

4.2.3 SSLB Overhead

In this experiment, we assess SSLB's overhead caused by clustering strategy, LBScheduler's decision algorithm, LBMonitor and LBMessenger's communication latency. Figure 11 shows the overheads of these factors. From the figure, we realize that cell clustering is the biggest overhead in SSLB. It is significantly affected by fog scale; when the number of fog nodes is 1000, the overhead is about 8 s. Fortunately, since cell cluster only happens at initialization step, it would not affect SSLB's normal operation directly. On the other hand, with the increase in nodes number (Fig. 11b, c), the growth rate of components' overhead will decrease.

Meanwhile, there are many configuration parameters affecting the overhead of SSLB and further the performance of the system. Cell scale k and probe ratio p are the most important two. Cell scale means the number of fog nodes in each cell, and probe ratio means the number of nodes to probe each time. Figure 12 depicts systems network load as a function of cell scale and probe ratio in a 1000-node cluster. The figures demonstrate that although the increase in p and k may improve the accuracy of decisions, excessive probe and communication eventually will hurt performance due to increased network load. In this experiment, we set the number of nodes in each cell is 125 when evaluating parameter p, so the maximum value of p in the ideal is 124. We use a probe ratio of 4 throughout our evaluation to facilitate comparison with the power of two choices and set $k = 125$. Users can adjust these parameters dynamically in consideration of the size of the cluster and network environment.

5 Related Work

The study of load balancing in fog computing environment has also started in recent years on the basis of various factors, such as bandwidth, execution time, scalability, and priority. Ningning et al. [28] proposed a fog computing load balancing mechanism of task allocation based on graph partitioning. In the design, tasks are assigned to a single or multiple virtual machines nodes according to the level of resources required by the task. However, the performance is not optimal for dynamic fog load balancing since the frequent graph repartitioning needed to cope with fog changes. In [29], the authors evaluated the distributed quality of service

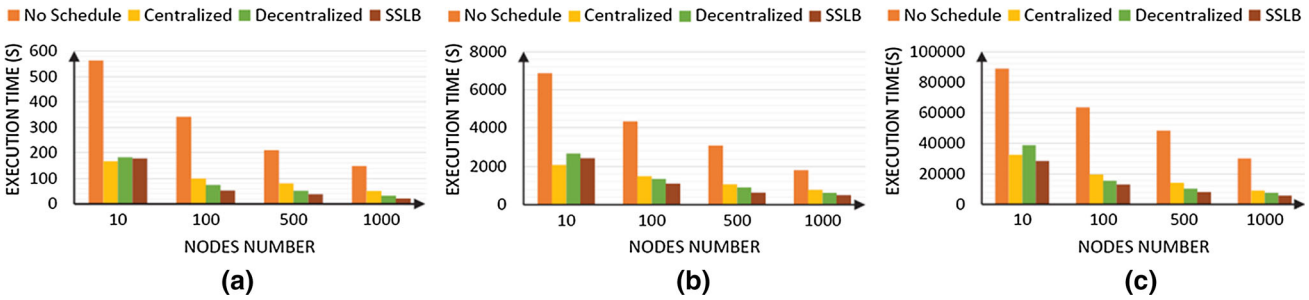


Fig. 8 Performance evaluation of proposed mechanism with traditional load balancing schemes when data size changed. **a** Data size = 1 GB, **b** data size = 10 GB, **c** data size = 100 GB

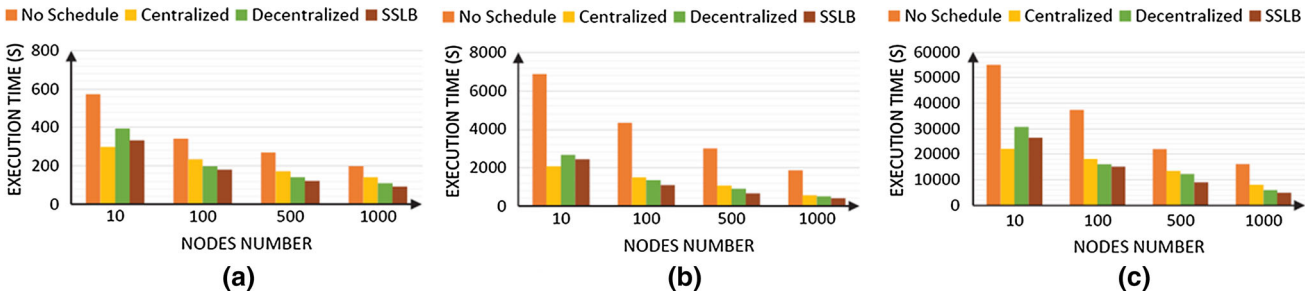


Fig. 9 Performance evaluation of proposed mechanism with traditional load balancing schemes when applications number changed. **a** App Number = 1000, **b** App Number = 10,000, **c** App Number = 100,000

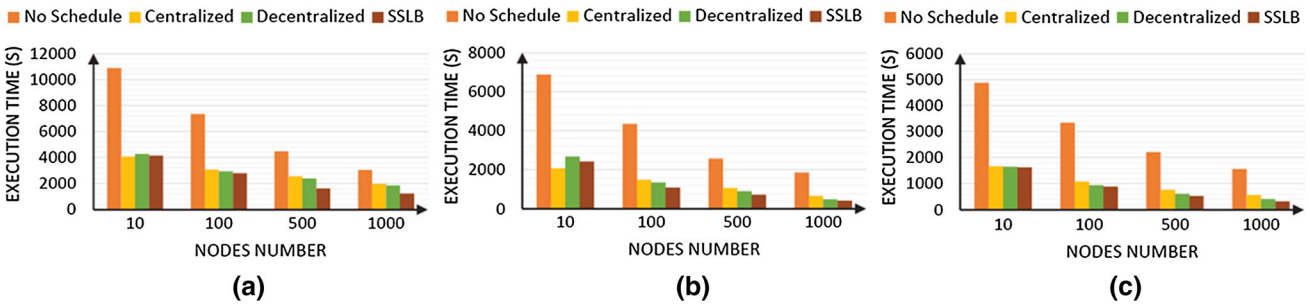


Fig. 10 Performance evaluation of proposed mechanism with traditional load balancing schemes when network bandwidth changed. **a** Bandwidth = 64 Mbps, **a** bandwidth = 128 Mbps, **a** bandwidth = 256 Mbps

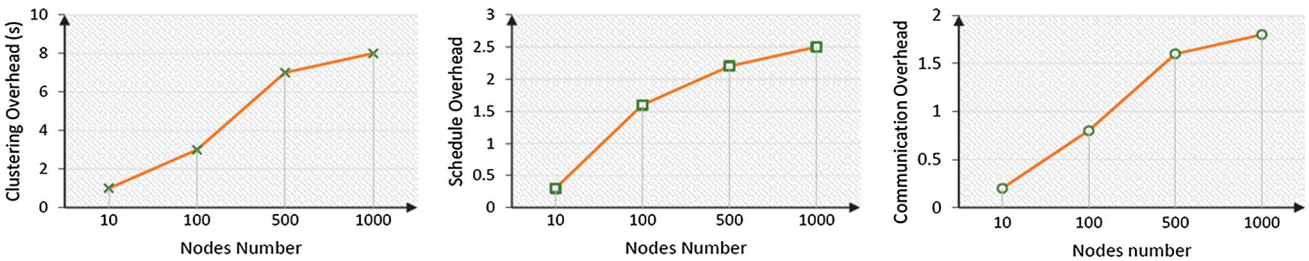


Fig. 11 Overhead evaluations of SSLB. Experimental results show that clustering strategy, load balancing decision algorithm, and communication latency will affect the efficiency of our proposed mechanism

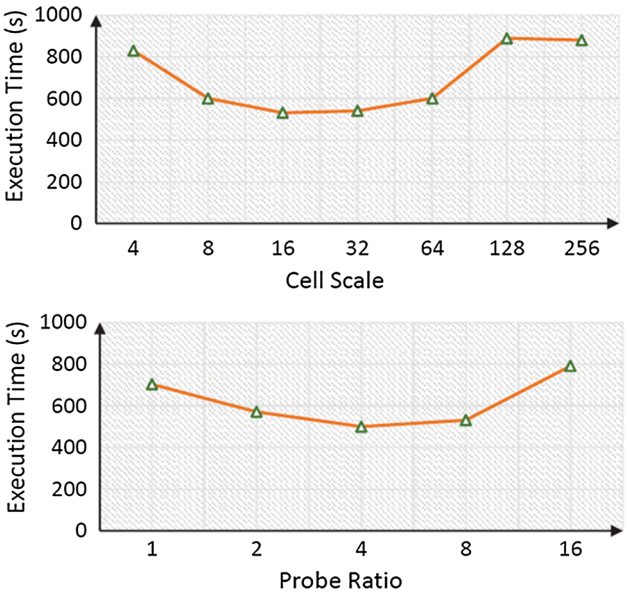


Fig. 12 Performance comparison while k changed. We perform tasks on 1000-machine cluster and test performance under different k, here $p = 4$

(QoS) aware scheduler for data stream processing operating in fog computing scenario. In the work, new components were introduced, namely a worker monitor, a QoS monitor, and an adaptive scheduler. The paper showed that the distributed QoS aware scheduler outperforms the centralized default one, thereby improving the application performance and enhancing the system with runtime adaptation capabilities. Nevertheless, complex fog topologies that involve many operators may cause some instability that can decrease the DSP application's availability. Meanwhile, in [11], the authors addressed the issue of load balancing in order to improve user's quality of experience (QoE). In the proposal, a reduced complex task scheduling algorithm fog computing was introduced. Despite the fact that this approach yields high users' satisfaction in terms of high latency gain, it can suffer from a high complexity for large-scale fog computing infrastructure because the algorithms used often give good results for low dense computing infrastructures. Intharawijitr et al. [30] proposed a fog computing architecture to help a computing system to achieve maximum efficiency by ensuring an optimal job scheduling. Three scheduling policies were considered in the study that is random policy, lowest latency policy, and maximum available capacity policy. In random policy, one fog node is randomly selected from a uniform distribution to execute an application. For the second policy, the fog node provides the lowest total latency based on the current state of the system. Finally, the maximum available capacity policy selects the fog node with the maximum remaining resources among the candidate nodes. The simulation results of this work showed that the lowest latency policy provides significantly better performance because of the availability of the resource. However, the use of one particular policy might not be the optimal solution for the whole system. There are also many other excellent efforts to achieve load balancing in fog, for example, Verma's team proposed an efficient scheduling algorithm for load balancing [31], while Bitam et al. optimized fog load balancing based on bees swarm [32]. Unfortunately, they cannot keep efficiency when the fog scale increases. Based on all these strategies and take the characteristics of fog environment into consideration, SSLB can be more suitable in this scenario.

6 Conclusions

In this work, we propose SSLB, a self-similarity-based load balancing mechanism for large-scale fog computing. It makes full use of the advantages of both centralized and decentralized schemes. With the help of the novelty structure, load balancing overhead can be very low even when the scale of the fog increase to very large. To guarantee SSLB's efficiency, we propose an adaptive threshold policy, which dynamically and accurately defines the load threshold on each node. Furthermore, two scheduling algorithms, task distributing and task grasping, are proposed. Experimental results show that SSLB outperforms traditional schemes especially when the fog scale is very large.

References

1. Inayat, Z.; Gani, A.; Anuar, N.B.; Anwar, S.; Khan, M.K.: Cloud-based intrusion detection and response system: open research issues, and solutions. Arab. J. Sci. Eng. **42**(2), 399–423 (2017)
2. Dastjerdi, A.V.; Buyya, R.: Fog computing: helping the Internet of Things realize its potential. Computer **49**(8), 112–116 (2016)
3. Kim, H.; Feamster, N.: Improving network management with software defined networking. IEEE Commun. Mag. **51**(2), 114–119 (2013)
4. Hawilo, H.; Shami, A.; Mirahmadi, M.; Asal, R.: NFV: state of the art, challenges, and implementation in next generation mobile networks (vEPC). IEEE Netw. **28**(6), 18–26 (2014)
5. Demestichas, P.; Georgakopoulos, A.; Karvounas, D.; Tsagkaris, K.; Stavroulaki, V.; Lu, J.; Xiong, C.; Yao, J.: 5G on the horizon: key challenges for the radio-access network. IEEE Veh. Technol. Mag. **8**(3), 47–53 (2013)
6. Bonomi, F.; Milito, R.; Natarajan, P.; Zhu, J.: Fog computing: a platform for internet of things and analytics. In: Big Data and Internet of Things: A Roadmap for Smart Environments, pp. 169–186 (2014)
7. Cao, Y.; Chen, S.; Hou, P.; Brown, D.: FAST: a fog computing assisted distributed analytics system to monitor fall for stroke mitigation. In: NAS, pp. 2–11 (2015)
8. Zhu, J.; Chan, D.S.; Prabhu, M.S.; Natarajan, P.; Hu, H.; Bonomi, F.: Improving web sites performance using edge servers in fog computing architecture. In: SOSE, pp. 320–323 (2013)
9. Stantchev, V.; Barnawi, A.; Ghulam, S.; Schubert, J.; Tamm, G.: Smart items, fog and cloud computing as enablers of servitization in healthcare. Sens. Transducers **185**(2), 121 (2015)

10. Aazam, M.; Huh, E.N.: Fog computing and smart gateway based communication for cloud of things. In: FiCloud, pp. 464–470 (2014)
11. Oueis, J.; Strinati, E.C.; Barbarossa, S.: The fog balancing: load distribution for small cell cloud computing. In: VTC Spring, pp. 1–6 (2015)
12. Deng, R.; Lu, R.; Lai, C.; Luan, T.H.: Towards power consumption-delay tradeoff by workload allocation in cloud-fog computing. In: ICC, pp. 3909–3914 (2015)
13. Ye, Q.; Rong, B.; Chen, Y.; Al-Shalash, M.; Caramanis, C.; Andrews, J.G.: User association for load balancing in heterogeneous cellular networks. IEEE Trans. Wirel. Commun. **12**(6), 2706–2716 (2013)
14. Karger, D.R.; Ruhl, M.: Simple efficient load balancing algorithms for peer-to-peer systems. In: Proceedings of the Sixteenth Annual ACM Symposium on Parallelism in Algorithms and Architectures, pp. 36–43 (2004)
15. Shi, W.; Cao, J.; Zhang, Q.; Li, Y.; Xu, L.: Edge computing: vision and challenges. IEEE Internet Things J. **3**(5), 637–646 (2016)
16. Luan, T.H.; Gao, L.; Li, Z.; Xiang, Y.; Wei, G.; Sun, L.: Fog computing: focusing on mobile users at the edge. arXiv preprint arXiv:1502.01815 (2015)
17. A survey on the scale of base station in the China Mobile. https://www.mobileworldlive.com/asia/asianews/china-mobile-has-a-third-of-global-4g-basestations/ (2016). Accessed Dec 2016
18. Xu, J.; Zhang, L.; Zuo, W.; Zhang, D.; Feng, X.: Patch group based nonlocal self-similarity prior learning for image denoising. In: The IEEE International Conference on Computer Vision, pp. 244–252 (2015)
19. Rivaz, H.; Karimaghaloo, Z.; Collins, D.L.: Self-similarity weighted mutual information: a new nonrigid image registration metric. Med. Image Anal. **18**(2), 343–358 (2014)
20. Dokmanic, I.; Parhizkar, R.; Ranieri, J.; Vetterli, M.: Euclidean distance matrices: essential theory, algorithms, and applications. IEEE Signal Process. Mag. **32**(6), 12–30 (2015)
21. Mitzenmacher, M.: The power of two choices in randomized load balancing. IEEE Trans. Parallel Distrib. Syst. **12**(10), 1094–1104 (2001)
22. Ousterhout, K.; Panda, A.; Rosen, J.; Venkataraman, S.; Xin, R.; Ratnasamy, S.; Shenker, S.; Stoica, I.: The case for tiny tasks in compute clusters. In: HotOS, vol. 13, pp. 14–14 (2013)
23. Gupta, H.; Dastjerdi, A.V.; Ghosh, S.K.; Buyya, R.: IFogSim: a toolkit for modeling and simulation of resource management techniques in internet of things, edge and fog computing environments. arXiv preprint arXiv:1606.02007 (2016)
24. Calheiros, R.N.; Ranjan, R.; Beloglazov, A.; De Rose, C.A.; Buyya, R.: CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Softw. Pract. Exp. **41**(1), 23–50 (2011)
25. Lu, K.; Sun, M.; Li, C.; Zhuang, H.; Zhou, J.; Zhou, X.: Wave: trigger based synchronous data process system. In: Cluster, Cloud and Grid Computing (CCGrid), pp. 540–541 (2014)
26. Li, C.; Zhuang, H.; Xu, B.; Wang, J.; Wang, C.; Zhou, X.: Light weight key-value store for efficient services on local distributed mobile devices. In: The 24th International Conference on Web Services, ICWS Research Track (2017)
27. The official website of Apache Hadoop. http://hadoop.apache.org/. Accessed Apr 2017
28. Ningning, S.; Chao, G.; Xingshuo, A.; Qiang, Z.: Fog computing dynamic load balancing mechanism based on graph repartitioning. China Commun. **13**(3), 156–164 (2016)
29. Cardellini, V.; Grassi, V.; Presti, F.L.; Nardelli, M.: On QoS-aware scheduling of data stream applications over fog computing infrastructures. In: ISCC, pp. 271–276 (2015)
30. Intharawijitr, K.; Iida, K.; Koga, H.: Analysis of fog model considering computing and communication latency in 5G cellular networks. In: PerCom Workshops, pp. 1–4 (2016)
31. Verma, M.; Bhardwaj, N.; Yadav, A.K.: Real time efficient scheduling algorithm for load balancing in fog computing environment. Int. J. Inf. Technol. Comput. Sci. **8**(4), 1–10 (2016)
32. Bitam, S.; Zeadally, S.; Mellouk, A.: Fog computing job scheduling optimization based on bees swarm. Enterp. Inf. Syst. (2017). https://doi.org/10.1080/17517575.2017.1304579.