The Journal of Systems and Software 107 (2015) 187–203

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

A small world based overlay network for improving dynamic load-balancing

Eman Yasser Daraghmi, Shyan-Ming Yuan*

DCS Lab, Department of Computer Science, National Chiao Tung University, Hsinchu 300, Taiwan, ROC

ARTICLE INFO

Article history:
Received 5 January 2015
Revised 25 May 2015
Accepted 1 June 2015
Available online 9 June 2015

Keywords:
Diffusion
Distributed systems
Dynamic load-balancing

ABSTRACT

Load-balancing algorithms play a key role in improving the performance of distributed-computing-systems that consist of heterogeneous nodes with different capacities. The performance of load-balancing algorithms and its convergence-rate deteriorate as the number-of-nodes in the system, the network-diameter, and the communication-overhead increase. Moreover, the load-balancing technical-factors significantly affect the performance of rebalancing the load among nodes. Therefore, we propose an approach that improves the performance of load-balancing algorithms by considering the load-balancing technical-factors and the structure of the network that executes the algorithm. We present the design of an overlay network, namely, functional small world (FSW) that facilitates efficient load-balancing in heterogeneous systems. The FSW achieves the efficiency by reducing the number-of-nodes that exchange their information, decreasing the network diameter, minimizing the communication-overhead, and decreasing the time-delay results from the tasks re-migration process. We propose an improved load-balancing algorithm that will be effectively executed within the constructed FSW, where nodes consider the capacity and calculate the average effective-load. We compared our approach with two significant diffusion methods presented in the literature. The simulation results indicate that our approach considerably outperformed the original neighborhood approach and the nearest neighbor approach in terms of response time, throughput, communication overhead, and movements cost.

1. Introduction

Load-balancing algorithms have become increasingly popular and powerful techniques in modern distributed computing systems in recent years (Chang et al., 2014). They provide opportunities for increasing the performance of large-scale computing systems and applications since they are designed to redistribute the workloads over the components of the distributed system in a way that ensures expanding resource utilization, maximizing throughput, minimizing response time, and avoiding the overload situation (Abdelmaboud et al., 2014). To achieve the goal of maximum performance, it is prerequisite to smoothly spread the load among the nodes to avoid, if possible, the situation where one node is heavily loaded with excess of workloads while another node is lightly loaded or idle (Chwa et al., 2015; Luo et al., 2014).

Load-balancing algorithms can be categorized into either static or dynamic (Cybenko, 1989; Fang and Wang, 2009). Static load-balancing necessitates complete information of the entire distributed system and workloads information, whereas dynamic load balancing requires light assumption about the system or the workloads. As in practical applications (i.e. real world networks) the workloads are generally not completely known, and each node has different capacity and runs at different speed, it is more efficient to employ the dynamic load balancing algorithms for practical applications. The diffusion approach (Hu and Blake, 1999; Luque et al., 1995) is one of the dynamic load balancing techniques that have received much attention by researchers in the past decades to solve the load-balancing problem. In standard diffusion approach, a system which has different nodes exchanges workloads via the communication links between these nodes. The workloads are distributed among the nodes, and the load balancing process works in sequential rounds. In every round, each node is allowed to balance its load with all its neighbors by exchanging the workloads to balance the total system load globally, meaning to minimize the load difference between the nodes with minimum and maximum load. The nearest-neighbor approach (Tada, 2011) is another dynamic technique that allows the nodes to communicate and migrate the excess workloads with their immediate neighbors only. Each node balances the workload among its neighbors in the hope that after a number of iterations the entire system will approach the balanced state.

* Corresponding author. Tel.: +886 3 5715900; fax: +886 3 5721490.
E-mail addresses: eman.yasser85@gmail.com (E.Y. Daraghmi), smyuan@cs.nctu.edu.tw, smyuan@gmail.com (S.-M. Yuan).

http://dx.doi.org/10.1016/j.jss.2015.06.001
0164-1212/

Since load-balancing algorithms play an important role of improving the performance of practical distributed computing systems, researchers have been motivated to propose several dynamic algorithms for balancing the workloads among nodes. However, dynamic load-balancing algorithms still present fundamental challenges when being executed at large-scale heterogeneous distributed systems. Previous research (Hui and Chanson, 1999, 1996, 1997) concluded that three **structural factors**, which refer to the structure of the network that executes the load-balancing algorithm, decrease the performance of any load-balancing algorithm as well as affect the algorithm convergence rate. The factors are: (1) increasing the number of nodes in the system (i.e. the number of the nodes that exchange their workload information); (2) increasing the network diameter which is defined as the longest shortest path between any two nodes of the network; (3) increasing the communication overheads or the communication delays among the nodes. These factors, from one hand, make it not feasible for a node to collect the load-information of all other nodes in the system. Moreover, even if a node collects the load-information of all other nodes in the system, this information will be not up to date when it is used (i.e. old information may not reflect the current load of a node) as more communication delays make this information old and thus the task of balancing the load is significantly damaged. From the other hand, it is intuitive that a network with longer diameter will take longer time to converge as the number of iterations to propagate the workloads to all nodes is proportional to the network diameter. *Therefore, the first objective of this research aims at improving the performance of load-balancing algorithms by considering the structural factors of the network that executes the algorithm.*

In addition, previous studies concluded that (Zomaya et al., 2001) **technical load-balancing factors**, which refer to the algorithm policies that should be considered when designing a load-balancing algorithm, such as the load migration rule, affect the performance of load-balancing algorithm. Therefore, these studies propose improved algorithms that consider these factors to enhance the performance of load-balancing (i.e. improvements include: the derivation of a faster algorithm that transfers less workloads to achieve a balanced state than other algorithms, or a mechanism for selecting and transferring the workloads to other nodes). However, when applying a dynamic load-balancing to practical distributed system, the functionality of the node and the migrated task must be checked to ensure that the node can process that received task. Thus, if the nodes distributed randomly, some situations that affect the performance of the load-balancing algorithm negatively may occur. For instance, n_i is a node in a practical distributed system. Since n_i is overloaded, it migrates a task to another lightly loaded node n_j. When n_j receives the migrated task, the load-balancing algorithm runs at n_j checks the scope of services of node n_j to ensure that the task can be processed by n_j. Thus, if the migrated task is out of n_j services scope, then the task will be migrated again to another node. Moreover, the task may be retuned again to n_i. Practically, re-migrating the task to another node decreases the performance of load-balancing algorithms because of the task re-migration time delay. Increasing the number of re-migrating task increases the time delay and thus decreases the performance of load-balancing. *Therefore, the second objective of this research aims at improving the performance of load-balancing algorithms by decreasing the time delay results from re-migrating tasks (i.e. re-migrating tasks results from the node out of services scope). To achieve our goal, we construct the FSW to allow nodes migrate tasks to other nodes that have similar services scope.*

In this research, we aim at improving the performance of load balancing algorithm by considering both the structural and the technical load-balancing factors. We also consider the node services scope to decrease the negative effect of tasks re-migration process. To achieve our goal, we propose a two-stage approach that, first, designs an overlay network which employs both the concept of small world network and the node services scope, and then, proposes an improving load-balancing to be applied within the overlay network.

First, practically, the nodes of practical distributed systems execute various computational-functions (each node has services scope). These computational-functions can be easily derived from the role of a node within the system and identified by k-element set (i.e. the role of the node within the system refers to the node services scope), namely the functional set (FS). Each element in the set represents a particular function that can be executed within the system. The FS of a node can be mapped to a point in a cluster and thus can be seen as a point in that cluster. In real-world distributed systems, each node plays a key role within the system. For instance, the m-cafeteria recommendation system is a practical distributed system that consists of several cafeteria nodes. Each cafeteria serves a menu, set of meal (i.e. the menu is considered as the FS of a cafeteria node, FS = {serving orange juice, serving butter waffle, etc.}). A user can via his/her mobile phone request a meal from a cafeteria node, if a cafeteria node is overloaded, then the request will be migrated to another node that has similar functionality. Similar functionality is defined as the difference between the amount of functions in-common among nodes and the amount of functions unique to nodes. It is clear that functions in common increase similarity, whereas functions that are unique to one node decrease similarity.

In fact, a small world (SW) network has a small average path length and large cluster coefficient properties. Thus, constructing an overlay network that satisfies the small world network properties and considers functional similarity minimizes the negative effects of the structural and technical factors (i.e. 1. decrease the number of nodes that exchange the workloads information, 2. minimize the network diameter, 3. deteriorate the communication overhead, and 4. decrease the impact of out of services scope and thus decrease the time delay results from re-migrating tasks). In this research, we construct an overlay network based on the small world principle, namely, the functional small world (FSW) that supports efficient load-balancing and thus increasing the performance of distributed computing systems.

Second, this research also presents an efficient load-balancing algorithm that considers the capacity of each node and the load-balancing technical factors, such as the initialization rule, the information exchange rule, the load-measurement rule and the load-migration rule.

Precisely, the advantages of creating the FSW instead of randomly distribute the nodes into clusters are: (1) simulating real world heterogeneous distributing systems which facilitate applying the load-balancing algorithm to real world distributed system; (2) decreasing the effect of time delay results from task re-migration that occurs because of the node out of services scope.

In summary, this paper presents the design of the FSW overlay network to support efficient dynamic load-balancing in heterogeneous systems. The primary contribution of this work is fourfold:

1. We adopt an effective clustering strategy that places nodes in clusters based on the nodes functional similarity and satisfies the properties of the small world principle.
2. We show a way of building a functional small world (FSW) overlay network that supports dynamic load-balancing, which is scalable to large network sizes yet adapts to dynamic membership and content changes. For simplicity, we refer to the functional small world overlay network as FSW in the rest of this paper.
3. We propose an efficient and improved dynamic diffusion load-balancing algorithm to be executed in the constructed FSW.
4. We conduct extensive experiments to evaluate the performance of proposed solution on various aspects, including throughput, response time, communication overhead and movements cost.

2. Literature review

2.1. Background on small world networks

A small-world network is a type of mathematical graph in which most of the nodes are not neighbors of one another, but these nodes can be reached from every other by a small number of hops or steps (Daraghmi and Yuan, 2014). Many empirical graphs are well-modeled by small-world networks. A certain category of small-world networks were identified as a class of random graphs by Watts and Strogatz in 1998 (Watts and Strogatz, 1998). They noted that graphs could be classified according to two independent structural features, namely the clustering coefficient, which is defined as the probability that two neighbors of a node are neighbors themselves and average node-to-node distance (also known as average shortest path length). Watts and Strogatz measured that in fact many real-world networks have a small average shortest path length, but also a clustering coefficient significantly higher than expected by random chance. A network is said to be small world if it has a small average path length and large cluster coefficient.

2.2. Related works

Previous studies have proposed numerous load-balancing algorithms targeting at static, small-scale, homogeneous and/or heterogeneous environments (Aakanksha and Bedi, 2007; Hu and Blake, 1999; Karger and Ruhl, 2004; Meyerhenke, 2009; Neelakantan, 2012; Yagoubi and Meddeber, 2010). The diffusion approach (Hu and Blake, 1999; Neelakantan, 2012) is a dynamic load-balancing technique where each node simultaneously sends the excessive workloads to its under loaded neighbors and receives workloads from its neighbors with higher workload (Boillat, 1990; Cybenko, 1989). In 1990, Boillat et al. (Boillat, 1990) presented a new approach to solve the load balancing problem for parallel programs. In 1989, Cybenko (Cybenko, 1989) studied the diffusion schemes for dynamic load balancing on a message passing multiprocessor networks. Elsasser et al. (Elsässer et al., 2002) generalized the standard diffusion schemes for homogenous networks to deal with the heterogeneous network. In Bahi et al. (2007), the first order diffusion load balancing, relaxed diffusion (RFOS) and generalized adaptive exchange (GAE) algorithms for totally dynamic networks were investigated. In Aakanksha and Bedi (2007), the authors proposed a modified version of diffusion algorithm for load balancing on dynamic networks. The authors in Adolphs et al. (2012) considered a neighborhood load balancing algorithm in the context of selfish clients. They assumed that a network of n processors is given, with m tasks assigned to the processors. The processors may have different speeds and the tasks may have different weights. Every task is controlled by a selfish user. The objective of the user is to allocate his/her task to a processor with minimum load, where the load of a processor is defined as the weight of its tasks divided by its speed. Neighborhood load balancing algorithms (Akbari et al., 2012) are diffusion algorithm that have the advantage that they are very simple and that the vertices do not need any global information to base their balancing decisions on.

3. Functional small world (FSW) network

In this section, we present an overview of the functional small world (FSW) design and provide the technical details of constructing the FSW overlay network. The notations used in this paper are summarized in Table 1.

3.1. Overview

FSW plays two important roles: 1) an overlay network that provides connectivity among nodes, and 2) a distributed solution that supports efficient dynamic load-balancing. In FSW, the nodes are organized in accordance with the functionality set (FS) defined by each node in the system. Nodes with similar functionality sets form one cluster.

Table 1
The symbols used in the paper.

Symbol	Description
FSW	Functional small world
FS	The functionality set
G	The system that executes the load-balancing algorithm
N	The nodes in the system
E	The connection-links among nodes
AF	All functions set
$WL(n_i)$	The set of assigned workloads for node n_i
c_i	The capacity of node n_i
ld_i	The load of node n_i
$Adj(n_i)$	The set of neighbor nodes for node n_i
$Info$	The set stored the information of neighbor nodes for node n_i
mig	The array that store the amount of migrated workloads
l_i	The effective-load of node n_i
l_{avg}	The average effective-load
N_{lower}	The set of assistant neighbors
LD	The load difference
δ_i	The excess workloads that node n_i must migrate
α_i	The amount of workloads that node n_i can accept

We based on the concept proposed by Tversky (Tversky, 1977) to define the relation of similar functionality employed in our research.

Definition 1 (Similar functionality). Generally, similar functionality is defined as the difference between the amount of functions in-common among nodes and the amount of functions unique to nodes.

Formally, given any nodes $n_i, n_j \in N$ with a functionality set of each node FS_i, FS_j, the relation of similar functionality is defined by:

$s(n_i, n_j) = |FS_{n_i} \cap FS_{n_j}| - (|FS_{n_i} - FS_{n_j}|) - (|FS_{n_j} - FS_{n_i}|)$. Therefore, nodes with $s(n_i, n_j) < 0$ are not similar, while nodes with $s(n_i, n_j) \geq 0$ are similar.

For example, if A is a node, with FS = {1, 2, 3} and B is a node with FS = {2, 4, 1, 2, 3}, then according to our definition $s(n_i, n_j) = 3 - 0 - 2 = 1$, *Thus,* A and B have similar functionality. It is clear that functions in common increase similarity, whereas functions that are unique to one node decrease similarity.

In practice, the practical distributed system is modeled as an undirected graph $G = (N, E)$, where N represents the set of heterogeneous nodes in the system, and E describes the connection-links among them. Each node $i \in N$ has its role within the system and executes several functions, such as printing, computing, etc.; thus, each node based on its role within the system defines a set, namely, the functionality set (FS). Since a small world network has two properties: (1) low average hop count between any two random chosen nodes, and (2) high clustering coefficient, our approach, in order to construct the FSW, categorizes the nodes in the system into two types: 1) *an in-domain node*, and 2) *a master node.* The *in-domain node* represents a node in which located in one cluster and only has connections via *short-links* with all *in-domain nodes* placed in the same cluster and the *master node* of that cluster. The *master node* represents a node located in one cluster and has a connection via *short-links* with all *in-domain* nodes placed in the same cluster and at the same time has connection via *long-links* with some *master nodes* located in other clusters. Fig. 1 shows an illustration example of FSW, where nodes n_1, n_4 and n_6 are *in-domain nodes*, while nodes n_2, n_3 and n_5 are *master nodes*. The *long-links* (i.e. blue lines in Fig. 1) creates connections among *master nodes* and is responsible for achieving the high clustering coefficient in the network (property 2 in small world networks). *Short-links* (i.e. black lines) create connection among *in-domain nodes*, and among *master nodes* and *in-domain nodes. Short-links* and the *long-links* aim at achieving the properties (1) and (2).

In our design, we also define the cluster-size M to be the maximum number of nodes that are allowed to form one cluster. Predefining the cluster size is important to keep small number of nodes

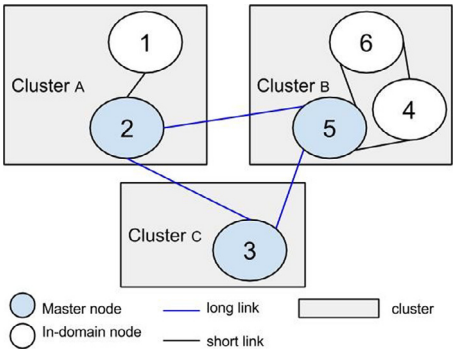

Fig. 1. An example of FSW overlay network, where an in-domain node connects with all in-domain nodes located in the same cluster, and the master node of that cluster, while a master node connects with in-domain nodes located in the same cluster and the other master nodes distributed among clusters. (For interpretation of the references to color in the text, the reader is referred to the web version of this article.)

in one cluster and to maintain good clustering effect. In this research, we adopt the guidelines proposed by Hui et al. (Hui et al., 2006) to define the cluster size *M*. The authors suggested that the cluster size ranges from 1 to 64 maintains good clustering effect. Practically, designing a FSW overlay network plays an important role in decreasing the number of nodes that will exchange the workloads information, minimizing the network diameter, deteriorating the communication overhead, and decreasing the time delay results from the task re-migration process; therefore, this approach is efficient to be applied not only for the entire system but also clustering inside the cluster to increase the performance of the load-balancing algorithms.

In summary, a FSW overlay network can be formed as follows: Each node maintains *long-links* to ensure the connectivity among the *master nodes* (i.e. the connectivity among the clusters to provide shortcuts to allow a node reach other nodes that execute similar functionality and located in other clusters quickly) and/or *short-links* to ensure the connectivity among the *in-domain nodes and* the connectivity among the *in-domain nodes* and the *master nodes* so that a balancing message issued from any node can reach any other node in the network. Via *short-links* and *long-links*, navigation and broadcasting in the network can be performed efficiently. In the following sections, we introduce our approach in details of designing and constructing a FSW.

3.2. Constructing functional small world (FSW) overlay network

Constructing a FSW overlay network depicted above involves three major tasks: 1) functional-clustering, 2) cluster-formation, and 3) overlay network construction.

3.2.1. Functional-clustering (FC)

In general, the *functional-clustering (FC)* task aims at 1) defining the clusters (i.e. the number and the name of clusters) that should be created within the overlay network based on the functional executed within the system, and 2) adding the nodes initially to the cluster(s) based on the in-common functions between the node and the cluster. In other words, if there is at least one function in-common between the node and the cluster, then the node will be added initially to that cluster. Note that: initially, in this step a node can be added to more than one cluster, but finally in the next tasks a node will only be added to one cluster.

This task is executed before or when a node joins the network. Each node n_i in the system defines its functionality set (FS), which indicates the functions that a node can perform and execute within the system, such as $FS_i = \{f_1, f_2, \ldots, f_k\}$, where FS_i is the functionality set of node n_i, f_1 is a function that can be executed by node n_i, and k is the number of functions that node n_i can execute. In our manuscript, a cluster, namely, $Cluster_{i,j,\ldots,k}$ has a functionality set $FS_{Cluster_{i,j,\ldots,k}} = \{i, j, \ldots, k\}$. Likewise, $Cluster_A$ has FS = {A}.

Table 2
The nodes and their functionality sets.

Node	FS
1	{X, C}
2	{X}
3	{A, C}
4	{X, A}
5	{C}
6	{A}
7	{C}
8	{C}
9	{X, C}
10	{A, C}
11	{X, A}
12	{A, C, X}

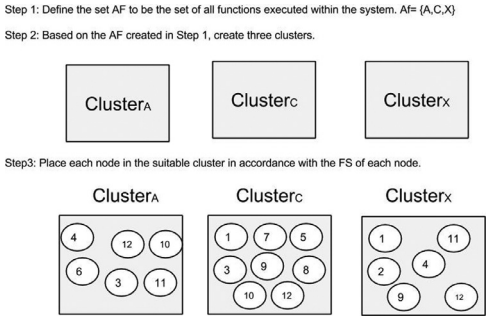

Fig. 2. The three steps of performing the functional-clustering task.

Following are the steps performed by the *functional-clustering* task:

1 Let AF (all functions) be the set of all functions executed in the system $AF = FS_1 \cup \ldots \cup FS_n = \{f_1, f_2, \ldots, f_s\}$, where s is the total number of functions executed within the system, and FS_i is the functionality set of node n_i. In other words, AF is the union of all FSs defined in the system.
2 For each function $f \in AF$, create a cluster, namely, $cluster_f$.
3 Since each node n_i has its functionality set $FS_i = \{f_1, \ldots, f_k\}$, in this step initially node n_i will be simultaneously added to $cluster_{f_1}, cluster_{f_2}, \ldots, cluster_{f_k}$. In other nodes, if a node n_i executes a function f_a, then there is an in-common function between a node n_i and $cluster_A$. Thus, the node n_i will be added to $cluster_A$.

Note that, the number of clusters that a node can be added to depends on the number of functions that a node executes within the system; a node that executes more than one function will be added initially to more than one cluster at the end of this task.

An example is shown below to illustrate in details the *functional-clustering (FC)* task. Given a system *G* that includes 12 nodes. Table 2 shows the nodes and the FS of each node. The steps executed by the *functional-clustering* task are shown in Fig. 2.

As shown in Fig. 2, in this example, after performing the *functional-clustering* task, nodes will be added initially to the created clusters (e.g. node 12 will be added initially to $cluster_A$, $cluster_C$ and $cluster_X$ since there is in-common functions between the node n_{12} and the clusters $cluster_A$, $cluster_C$, $cluster_X$).

3.2.2. Cluster-formation

As shown in the functional-clustering (FC) task, a node initially can be added to more than one cluster. Therefore, the *cluster-formation (CF) task* is a key task to ensure that a node will

be added to only one cluster regarding the functional similarity. According to Definition 1, nodes are considered as similar nodes if the amount of in-common functions among nodes is more than the amount of functions unique to nodes.

This task aims at: 1) deciding the nodes that must finally be added to the cluster, and 2) checking the cluster size; thus, if the cluster size exceeds *M*, which is a preset defined maximum cluster size, the cluster will be split into two clusters in order to maintain good clustering effect. To determine the cluster size, we adopt the guidelines proposed by Hui et al. (Hui et al., 2006). The authors suggested that the maximum cluster size is 64 in order to maintain good clustering effect. If the cluster size exceeds *M*, the steps of the functional-clustering task, and the cluster-formation task will be applied to split that cluster (i.e. Note, new clusters with new names, such as $cluster_{A1}$ instead if $cluster_A$, will be created upon re-performing the tasks to split cluster(s)).

Following pseudo code shows the steps performed by the *cluster-formation* task. Note, |FS| is the number of elements in the functional set. The steps executed by the *cluster-formation (CF)* task are shown in Fig. 3.

Pseudo code of the cluster formation task

Cluster- Formation task

Initialization

Let A[]= {< $cluster_1$, | $cluster_1$ |>,< $cluster_2$, | $cluster_2$ |,...,< $cluster_f$, | $cluster_f$ |}

where | $cluster_1$ |, | $cluster_2$ |,...,| $cluster_f$ | is the size of cluster$_1$, cluster$_2$,...,cluster$_f$

begin

1.*int* *m*[] = *A.minArray*();

/ / *Finding the clusters that have the least cluster size*

2.For each cluster "cluster$_a$" in A[]

3. For each node n_i added initially to cluster$_a$ {

3.1. if | FS_i | = 1, *then* add n_i to cluster$_a$.

/ / this means the functional similrity between a node

// and the cluster is ≥ 0 since a node can execute one function and added to one cluster

3.2. if | FS_i | > 1, *then*

// the node is initially added to more than one cluster

//thus, these steps ensure positive similarity between a node and a cluster

3.2.1. if cluster$_a$ ∈ *m*[] and | *m*[] | == 1 then add n_i to cluster$_a$.

/ / | *m*[] | == 1 *means* the number of clusters that has the smallest cluster size is 1

3.2.2. elseif cluster$_a$ ∈ *m*[] and | *m*[] | ≠ 1 then

// here more than one cluster has the smallest cluster size

3.2.2.1. if n_i added to (one cluster cluster$_a$ ∈ *m*[] and the other clusters not in *m*[])*then*

add n_i to cluster$_a$

// this step ensures similarity and add node to cluster with smallest size

3.2.2.2. if n_i added to (more than one cluster$_a$ ∈ *m*[]) then

add a "wait " tag of n_i

// this mean a node has in-common functions with two clusters in the same size, since

// each cluster has different functionality, the similarity between a node and the cluster

//may be negative; thus, additional steps must be done to ensure positive similarity

3.2.3. elseif cluster$_a$ ∉ *m*[] then

3.2.3. chech the FS ={f_1,..,f_{id}} of n_i if the is a cluster ∈ m has the name cluster$_{f_{id}}$

then n_i *leave* cluster$_a$ otherwise add a tag "wait" to n_i }

4.For each node n_i tagged as wait

4.1. find TFS= $FS_1 \cup FS_2 \cup ....\cup FS_z$, *where* z is the nodes z has a tag "wait"

4.2. create new cluster,namely, *cluster*$_{FS_1 \cup FS_2 \cup ....\cup FS_z}$

4.3. add n_i to *cluster*$_{FS_1 \cup FS_2 \cup ....\cup FS_z}$

End

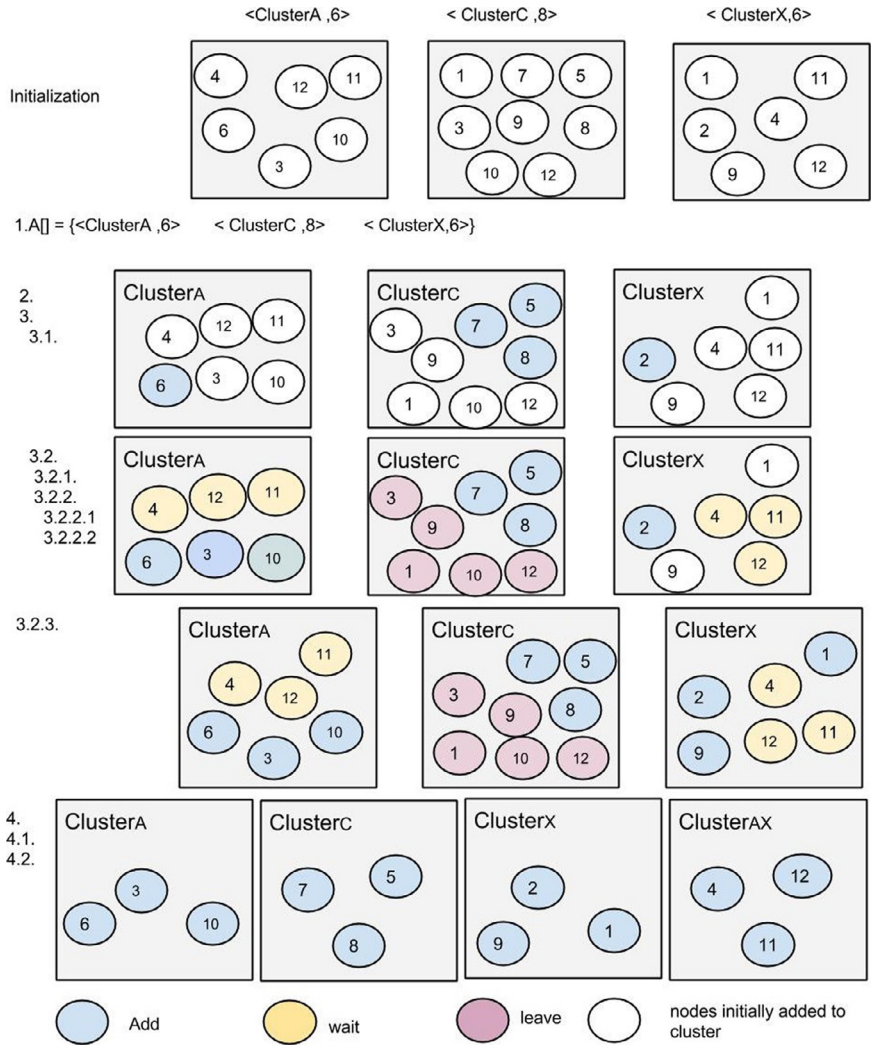

Fig. 3. The steps performed by the cluster formation task, where the numbers in the left side indicate the step defined in the pesudo code.

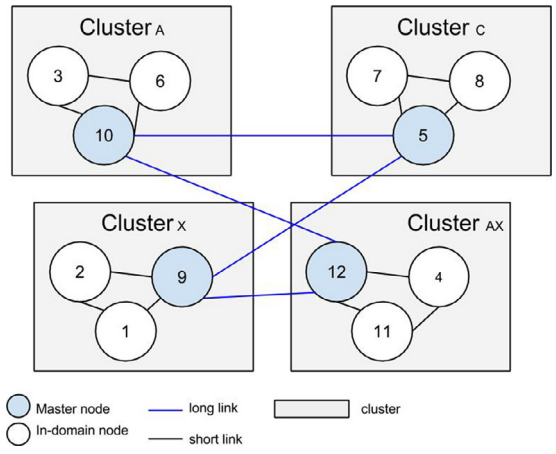

Fig. 4. The results of performing the overlay network constructing task.

3.2.3. Overlay network construction

Fig. 4 shows the results of the *overlay network-construction* task. This task constructs the FSW overlay network across the created clusters (i.e. after performing the previous two tasks) to form a functional small world network by:

(1) Defining the *in-domain nodes* and the *master nodes*.

The size of the FS of each node located in one cluster will be checked (i.e. the number of functions that a node can execute); therefore, a node that has the largest FS size in $cluster_i$ will be defined as *a master node* for $cluster_i$, and the other nodes located in $cluster_i$ will be defined as the *in-domain nodes* for that cluster. Note, when two or more nodes have the largest FS size, then only one node from these nodes will be selected randomly as a *master node for* a cluster since that each cluster has only one master node.

(2) Adding *long-links* and *short-links* among the nodes.

Long-links connect a *master node* located in one cluster with other *master nodes* located in other clusters based on the functional similarity between theses master nodes (i.e. see Definition 1). For example, n_{10} is the master node of $cluster_A$ and has a FS = {A, C} and n_{12} is the master node of $cluster_{AX}$ and has a FS = {A, C, X}; thus, the functional similarity = (2)-(0)-(1) = 1 which means a long-link will be added between them. In contrast, n_{10} is the master node of $cluster_A$ and has a FS = {A, C} and n_9 is the master node of $cluster_X$ and has a FS = {C, X}; thus, the functional similarity = (2)-(1)-(1) = −1 which means no long-link will be added between them.

Short-links connect the *in-domain nodes* located in one cluster with the other *in-domain nodes* located in the same cluster, and it also connects the *in-domain nodes* located in a cluster with the

Table 3
The connection-links before and after constructing FSW overlay network.

n_i	FS	Connection-links before constructing the FSW overlay network	B[a]	After constructing FSW	A[b]
1	X, C	{(1, 2), (1, 4), (1, 9), (1, 11), (1, 12), (1, 3), (1, 5), (1, 7), (1, 8), (1, 10)}	10	{(1, 2), (1, 9)}	2
2	X	{(2, 1), (2, 4), (2, 9), (2, 11), (2, 12)}	5	{(2, 1), (2, 9)}	2
3	A, C	{(3, 4), (3, 5), (3, 9), (3, 11), (3, 12), (3, 1), (3, 7), (3, 8), (3, 10), (3, 6)}	10	{(3, 6), (3, 10)}	2
4	X, A	{(4, 2), (4, 1), (4, 9), (4, 11), (4, 12), (4, 3), (4, 5), (4, 7), (4, 8), (4, 10)}	10	{(4, 11), (4, 12)}	2
5	C	{(5, 1), (5, 3), (5, 7), (5, 8), (5, 9), (5, 10), (5, 12)}	7	{(5, 7), (5, 8), (5, 10), (5, 9)}	4
6	A	{(6, 4), (6, 10), (6, 11), (6, 12), (6, 3)}	6	{(6, 3), (6, 10)}	2
7	C	{(7, 1), (7, 3), (7, 5), (7, 8), (7, 9), (7, 10), (7, 12)}	7	{(7, 5), (7, 8)}	2
8	C	{(8, 1), (8, 3), (8, 7), (8, 5), (8, 9), (8, 10), (8, 12)}	7	{(8, 7), (8, 5)}	2
9	X, C	{(9, 2), (9, 4), (9, 1), (9, 11), (9, 12), (9, 3), (9, 5), (9, 7), (9, 8), (9, 10)}	10	{(9, 1), (9, 2), (9, 5), (9, 12)}	4
10	A, C	{(10, 4), (10, 5), (10, 9), (10, 11), (10, 12), (10, 1), (10, 7), (10, 8), (10, 3), (10, 6)}	10	{(10, 3), (10, 6), (10, 5), (10, 12)}	4
11	X, A	{(11, 2), (11, 4), (11, 9), (11, 1), (11, 12), (11, 3), (11, 5), (11, 7), (11, 8), (11, 10)}	14	{(11, 12), (11, 4)}	2
12	A, C, X	{(12, 1), (12, 2), (12, 3), (12, 4), (12, 5), (12, 6), (12, 7), (12, 8), (12, 9), (12, 10), (12, 11)}	11	{(12, 4), (12, 11), (12, 9), (12, 10)}	4

[a] B is the number of connection links before constructing FSW.
[b] A is the number of connection links after constructing FSW.

master node of the same cluster. *In-domain nodes, master nodes, long-links* and *short-links* play a key role in reducing the effect of the structural factors and transforming the network into a small world.

In order to show how our proposed design reduces the communication overhead, we summarize the connection-links among nodes before and after constructing the FSW overlay network. Table 3 summarizes the nodes and their connection-links before and after constructing the FSW overlay network, where column 4 shows the number of connection-links before constructing FSW and column 6 shows the number of connection-links after constructing FSW. Our approach reduces the number of connection-links by creating shortcuts via using the idea of *master nodes* and *long-links.*

4. Dynamic load-balancing

In this section, we present an overview of pitfalls in designing dynamic diffusion load-balancing algorithm and propose an improved diffusion load-balancing algorithm to be executed within the constructed FSW.

4.1. Pitfalls in designing load-balancing algorithm

4.1.1. The initialization rule

This rule aims at defining the set of nodes that should be considered as the *neighbor-nodes* of node n_i that currently runs the load-balancing algorithm. Constructing the FSW overlay network plays a key role in limiting the set of *neighbor-nodes* for node n_i to those nodes that execute similar functionality and have connection-links, either *long-links* or *short-links*, with node n_i. Therefore, each node sends/receives the workload information to/from only the *neighbor-nodes* set which in turn results in reducing the number of nodes that exchange the workloads information and reducing the communication overhead.

4.1.2. Information exchange rule

This rule specifies how to collect the required information for making the load-balancing decisions. Our proposed algorithm uses the on-state information exchange (Neelakantan, 2012), a node broadcasts its information to only the set of *neighbor-nodes* whenever its status changes. In fact, the on-state information exchange strategy has the advantages of making more accurate decisions, however, the large overhead in communication makes it impractical for large systems (i.e. many previous algorithms loses the advantage of this strategy because of the bad effect of large communication overhead). Since our approach reduces the communication overhead by constructing the FSW overlay network, it becomes practical to take the advantages of the on-state information exchange strategy without increasing the communication overhead.

4.1.3. Load measurement rule

The load measurement rule aims at deciding whether a node is overloaded or not. Each node in the system has its processing capacity that will be used with the weight of the assigned workloads to calculate the effective-load and thus calculating the average effective-load among the set of *neighbor-nodes*. The average effective-load will be used to decide if the node is overloaded or not. When the effective-load of a node is higher than the average effective-load, then the node is considered as an overloaded node; otherwise, it is considered as under loaded node. The decision that a node is overloaded or not in turn decides when to begin the balancing operations to migrate the excess workloads to under the other loaded node.

4.1.4. Transfer strategy rule

This rule aims at defining the set of *assistant-neighbors*, the nodes among the set of *neighbor-nodes* that are currently under loaded, and finding the amount of the excess workload to be sent in the case of overloaded. Since the goal of load-balancing algorithms is redistributing the loads among nodes, obviously heavier nodes should send the excessive workloads to the lighter nodes. Therefore, node n_i selects its *assistant-neighbors* that are currently under loaded to migrate its excess workloads. The amount of workloads to be migrated among the set of *assistant neighbor-nodes* would have direct impact on the performance and the convergence rate of the algorithm. In general, the amount of the workloads to be moved from the overloaded node to the *assistant neighbor-nodes* would be the difference of load between the overloaded node n_i and the average effective-load. To achieve the fairness state the amount of load to be sent should depend on how much the current node is overloaded with respect to its *neighbor-nodes*. Therefore, our algorithm considers the average effective-load which has an advantage of being fair and simple as all nodes receives the same load and no node is privileged. As a result, using the information received only from the *neighbor-nodes* plays a key role in: 1) giving us more accurate estimation of the average effective-load, and 2) restricting the communication between the *neighbor-nodes* only which resulted in communication delay is suppresses.

4.2. Diffusion load-balancing

In this section, we explain the proposed load-balancing algorithm that will be executed in the constructed FSW overlay network. We first formulate the problem in Section 4.2.1, then we present our proposed algorithm in Section 4.2.2, and finally, we show how our proposed algorithm guarantees converges to fairness state in Section 4.2.3.

4.2.1. Problem formulation

Generally, the entire distributed system is modeled as an undirected graph $G = (N, E)$, where N represents the set of heterogeneous

nodes, and E describes the connections among them. Each node in the system (i.e. whether *an in-domain node* or *a master node*) will be assigned some workloads *wl* during the execution of the system, where each workload assigned to a node consumes effort and time; thus, each workload has different weight *w*. The weight of the total workloads assigned to a node is referred to as the load of a node $ld_i > 0$. Each assigned workload also is associated with a function that can process the assigned workload. Each node also has a capacity $c_i > 0$ which specifies its processing capacities (i.e. the largest amount of workload that can be assigned to a node n_i), where c_i, $ld_i \in Z$. Since the capacity of each node in heterogeneous systems is not equal, our proposed algorithm considers the processing capacity of each node when deciding whether a node is overloaded or not.

Definition 2 (The effective-load)**.** Given a node $n_i \in N$ that has a capacity and assigned some workloads, the effective-load l_i of node n_i is defined as the total weight of the workloads assigned to node n_i divided by the capacity of node n_i. Formally, the effective-load of node n_i is the load of n_i divided by the capacity of n_i.

$$l_i = \frac{ld_i}{c_i} = \frac{\sum_{wl_j \in WL(n_i)} w(wl_j)}{c_i}$$

where $WL(n_i) = \{< wl_1, w_1, ctr_{id}, F_{id} >, \ldots, < wl_z, w_z, ctr_{id}, F_{id} >\}$ is the set of workloads assigned to node n_i.

4.2.2. Our proposed algorithm

Our proposed algorithm is shown in Algorithm 1, NeighborhoodLB. Each node n_i in the system G executes the same algorithm in parallel. As mentioned before, based on the role of each node n_i within the system, n_i defines its functionality set (FS). Thus, the structure of the system is simplified by constructing the FSW to decrease the graph diameter, the number of nodes that exchange the load information and communication overhead. The steps of constructing FSW overlay network are illustrated in Section 3. The nodes will be spread into clusters, and each node will have in addition to the node id n_{id}, a cluster id ctr_{id} to show the cluster in which a node is located and FS_{id} to check if the received task can be processed by a node n_{id}. Following paragraphs demonstrate with an illustration example the proposed load-balancing algorithm that will be executed within the constructed overlay network in details. Our proposed load-balancing algorithm involves six major stages: 1) the initialization stage, 2) the information broadcasting stage, 3) computing the average effective-load stage, 4) finding the set of assistant-neighbors stage, 5) the workloads transfer stage, and 6) the load-balancing mechanism stage.

- The Initialization Stage

Let $WL(n_i)$ be the set of workloads assigned to node n_i during the execution of the computing distributed system, where $WL(n_i) = \{< wl_1, w_1, ctr_{id}, F_1 >, < wl_z, w_z, ctr_{id}, F_z >\}$. Each assigned workload *wl* consumes time and efforts until being completed; thus, each assigned workload has weight *w*. Each workload *wl* assigned initially to ctr_{id} and associated with a function *F* (i.e. *F* is the function that can process the workload). Each node n_i also has, after constructing FSW, a pre-defined set of *neighbor-nodes* $Adj(n_i)$ to store the nodes that have connection-links either *long-links* or *short-links* with node n_i. Each node n_i initializes its state (initialization stage) in steps 1 through step 3.

1. **Step 1 (Line 1 in NeighborhoodLB Algorithm)**: Each node n_i defines a set $Info = \{< ctr_{id}, n_{id}, ld_{id}, c_{id}, FS_{id} >\}$ to store the information of the nodes in the *neighbor-nodes* set, where ctr_{id}: is the id of the cluster in which a node the has n_{id} is located, n_{id}: the id of a node, $ld_{id} = \sum_{wl_j \in WL(n_{id})} w(wl_j)$ the load of node n_{id} (i.e. the total weight of all workloads assigned to the node n_{id}), c_{id}: is the processing capacity of n_{id}, and FS_{id} is the functional set of n_{id}.

ALGORITHM 1, NEIGHBORHOODLB

Algorithm 1, NeighborhoodLB

n_{id} : The node where the algorithm is executed.
ctr_{id} : The id of a cluster in which n_i is located
c_i : The processing capacity of node n_i
$Adj(n_i) = \{< ctr_{id}, n_{id} >\}$The set of neighbor-nodes
$WL(n_i) = \{<wl_{id}, w, ctr_{id}, F_{id}>\}$: The set of assigned workloads for n_i
FS_{n_i}: the functionality set of n_i

Begin

1.Let $Info_i = \{< ctr_{id}, n_{id}, ld_{id}, c_{id}, FS_i >\}$

2.Let mig(n_j)=0 for all $n_j \in Adj(n_i)$

3.Compute the effective-load: $l_i = \frac{ld_i}{c_i} = \frac{\sum_{wl_j \in WL(n_i)} w(wl_j)}{c_i}$

4.For each node $n_j \in Adj(n_i)$ do

a.if n_i is master node then send message

$<ctr_{id}, n_i, ld_i, \frac{c_i}{|long\_links|+1}, FS_i, "B", [0, ""]>$

b.else send message$<ctr_{id}, n_i, ld_i, c_i, FS_i, "B", [0, ""]>$

5.Read messages from the messages queue

a. if T="B" then $Info = Info \cup \{< ctr_{id}, n_f, ld_f, c_f, FS_f >\}$

b. if T="G" then

1)$Info = Info \cup \{< ctr_{id}, n_i, ld_i + g, c_i, FS_i >, < ctr_{id}, n_f, ld_f - g, c_f, FS_j >\}$

2)$l_i = \frac{ld_i + g}{c_i}$

3)For each node $n_j \in Adj(n_i)$ do

a. if n_j is master node then send message

$<ctr_{id}, n_i, ld_i + g, \frac{c_i}{|long\_links|+1}, FS_i, "B", [0, ""]>$

b.else send message$<ctr_{id}, n_i, ld_i + g, c_i, FS_i, "B", [0, ""]>$

6.Compute the average effective-load $l_{avg} = \frac{ld_i + \sum_{j \in Info} ld_j}{c_i + \sum_{j \in Info} c_j}$

7. For each node $n_j \in Adj(n_i)$ do //Define the Assistant Neighbors

a. if $\frac{ld_j}{c_j} < l_{avg}$ and $\frac{ld_j}{c_j} \le l_i$ then $N_{lower} = N_{lower} \cup n_j$

8.Let load-difference $LD_i = (l_i - l_{avg})$

9.If $LD \le 0$ then exit; else $LB(WL(n_i), N_{lower}, LD_i)$

EndBegin

2. **Step 2 (Line 2 in NeighborhoodLB Algorithm):** Each node n_i also defines an array $mig(n_i)$ to store the amount of the migrated workload that node n_i will transfer to the under loaded nodes of the set *neighbor-nodes*. Initially, the workloads that will be transferred to other nodes is 0 for all nodes in the set of *neighbor-nodes*.
3. **Step 3 (Line 3 in NeighborhoodLB Algorithm):** Each node n_i computes its initial effective-load l_i via the equation defined in Definition 2 (i.e. *the total weight of the workloads assigned to node n_i divided by the capacity of node n_i*).

To illustrate our proposed algorithm, we will use the example shown in Section 3. Fig. 4 shows the constructed FSW overlay network via the steps illustrated in Section 3. Table 4 shows the initial status of the nodes before and after executing the initialization stage. The node information before start executing the load-balancing

Table 4
The status of the nodes.

	Before starting the algorithm				After the initialization				After the broadcasting stage
n_{id}	ctr_{id}	M[a]	c_i	$Adj(n_i)$	ld_i	$Info$[b]	mig	l_i	$Info$[c]
1	X	–	60	{<X, 2>, <X, 9>}	30	{}	(0, 0)	0.5	{<X, 2, 50, 50>, <X, 9, 15, 10>}
2	X	–	50	{<X, 1>, <X, 9>}	50	{}	(0, 0)	1	{<X, 1, 30, 60>, <X, 9, 15, 10>}
3	A	–	70	{<A, 6>, <A, 10>}	40	{}	(0, 0)	0.571	{<A, 6, 20, 45>, <A, 10, 25, 15>}
4	AX	–	30	{<AX, 11>, <AX, 12>}	10	{}	(0, 0)	0.33	{<AX, 11, 20, 55>, <AX, 12, 25, 20>}
5	C	√	60	{<C, 7>, <C, 8>, <X, 9>, <A, 10>}	70	{}	(0, 0, 0, 0)	1.167	{<C, 7, 40, 60>, <C, 8, 10, 50>, <X, 9, 15, 10>, <A, 10, 25, 15>}
6	A	–	45	{<A, 3>, <A, 10>}	20	{}	(0, 0)	0.444	{<A, 3, 40, 70>, <A, 10, 25, 15>}
7	C	–	60	{<C, 5>, <C, 8>}	40	{}	(0, 0)	0.677	{<C, 5, 70, 20>, <C, 8, 10, 50>}
8	C	–	50	{<C, 5>, <C, 7>}	10	{}	(0, 0)	0.2	{<C, 5, 70, 20>, <C, 7, 40, 60>}
9	X	√	30	{<X, 1>, <X, 2>, <C, 5>, <AX, 12>}	15	{}	(0, 0, 0, 0)	0.5	{<X, 1, 30, 60>, <X, 2, 50, 50>, <C, 5, 70, 20>, <AX, 12, 25, 20>}
10	A	√	45	{<A, 3>, <A, 6>, <C, 5>, <AX, 12>}	25	{}	(0, 0, 0, 0)	0.556	{<A, 3, 40, 70>, <A, 6, 20, 45>, <C, 5, 70, 20>, <AX, 12, 25, 20>}
11	AX	–	55	{<AX, 4>, <AX, 12>}	20	{}	(0, 0)	0.364	{<AX, 4, 10, 30>, <AX, 12, 25, 20>}
12	AX	√	60	{<AX, 4>, <AX, 11>, <A, 10>, <X, 9>}	25	{}	(0, 0, 0, 0)	0.417	{<AX, 4, 10, 30>, <AX, 11, 20, 55>, <A, 10, 25, 15>, <X, 9, 15, 10>}

[a] A node is master node or not.
[b] After the initialization stage and before the broadcasting.
[c] After the broadcasting stage.

algorithm (i.e. before executing the initialization stage), the node id, the cluster id, whether a node is *a master node* or not, the node capacity, the *neighbor-nodes* set, and the load of a node, are given in columns 1, 2, 3, 4, 5, and 6 respectively. Each node in the system executes the same proposed algorithm in parallel. In the initialization stage, each node: (1) defines *Info* set to store the information about its *neighbor-nodes*, (2) defines *mig* array to store the amount of excess workload to be transferred, and (3) computes its effective-load. Columns 7, 8, and 9 in Table 3 show the status of the nodes after executing the initialization stage. Because of size constrains, we did not add the FS of each node in the table (i.e. the FS of each node is given previously in Table 2).

- The information Broadcasting Stage

1. **Step 4 (Line 4 in NeighborhoodLB Algorithm):** Each node n_i broadcasts its initial state (i.e. initial information after executing the initialization stage) to only its *neighbor-nodes* (the nodes stored in the set *adj*). Since a *master node* has connections with some *master nodes* located in other clusters that have similar functionality via *long-links*, and it has also connections with the *in-domain nodes* located in the same cluster via *short-links* (see Fig. 4, node 10 is *a master node* that has *short-links* with nodes 3 and 6, and at the same time has *long-links* with nodes 5 and 12), the capacity of a *master node* that will be sent to other nodes is divided among the clusters $c_i/|long-links|+1$ in the broadcasting stage.

   In fact, each node maintains a FIFO message queue which holds the incoming messages. Each message has the format $\langle ctr_{id}, n_f, ld_f, c_f, FS_f, \text{“T"}, [g, \text{“F"}]\rangle$, where ctr_{id} is the cluster id where the node that sends the message is located in, n_f is the id of the sender node, ld_f the loads of the sender node, c_f is the capacity of the sender node, FS_f is the functionality set of the sender node, T is the type of the message, g is the migration information (i.e. information about the migrated task and the function F that can process the migrated task). There are two types of messages:

   Workload migration message (“G”): n_i sends a “G”-message to n_j to tell it that n_i wants to migrate g units of workload to n_j.

   Broadcast message (“B”): broadcast the status (i.e. cluster id, node id, load, capacity, and FS to all neighbor-nodes).
2. **Step 5 (Line 5 in NeighborhoodLB Algorithm):** The main part of the algorithm starts when a node takes the first message from the queue and processes the message according to its type. If the message type is “B”, then the node only updates its information stored in the *Info* set. If the message type is “G”, then it updates the information stored in the *Info* set, computes its effective load, and broadcasts its new status to its *neighbor-nodes*. Initially, the first message received by each node is “B” type messages. Table 4 holds the state of the system after broadcasting (step 4 and step 5). The set *Info* stores the information about the node and its neighbor-nodes.

- Computing the average effective-load

1. **Step 6 (Line 6 in NeighborhoodLB Algorithm):** After updating the information stored in the *Info* set (i.e. after the broadcasting stage), each node computes the average effective-load l_{avg} of a node and its *neighbor-nodes* to facilitate 1) making a decision (i.e. whether a node overloaded or not) later by a node, and 2) defining the set of *assistant neighbors* in the next stage. The average effective-load is computed by the following equation:

$$l_{avg} = \frac{ld_i + \sum_{j \in \inf o} ld_j}{c_i + \sum_{j \in \inf o} c_j}$$

Note that, in the above formula the capacity of all nodes is considered since in heterogeneous systems the capacity is varied

Table 5
The system status.

n_{id}	l_i	l_{avg}	N_{lower}	LD_i
1	0.500	30 + 50 + 15/(60 + 50 + 10) = 0.792	{}	−0.292
2	1.000	30 + 50 + 15/(60 + 50 + 10) = 0.792	{1}	+0.208
9	**0.500**	**15 + 30 + 50 + 70 + 25/(30 + 60 + 50 + 20 + 20) = 1.056**	**{1}**	**−0.556**
3	0.571	40 + 20 + 25/(70 + 45 + 15) = 0.654	{6}	−0.083
6	0.444	40 + 20 + 25/(70 + 45 + 15) = 0.654	{}	−0.210
10	**0.566**	**25 + 40 + 20 + 70 + 25/(45 + 70 + 45 + 20 + 20) = 0.900**	**{6}**	**−0.334**
5	**1.167**	**70 + 40 + 10 + 15 + 25/(60 + 60 + 50 + 10 + 15) = 0.821**	**{8}**	**−0.346**
7	0.677	40 + 70 + 10/(60 + 20 + 50) = 0.923	{8}	−0.246
8	0.200	10 + 70 + 40/(50 + 20 + 60) = 0.923	{}	−0.723
4	0.332	10 + 20 + 25/(30 + 55 + 20) = 0.524	{}	−0.191
11	0.364	20 + 10 + 25/(55 + 30 + 20) = 0.524	{4}	−0.160
12	**0.417**	**25 + 10 + 20 + 25 + 15/(60 + 30 + 55 + 15 + 10) = 0.558**	**{4, 11}**	**−0.142**

Bold text refers to master nodes, while non bold text refers to in-domain nodes.

from one node to another. Column 3 in Table 5 shows the *average effective-load* computed by each node.

- Finding the set of *assistant-neighbors* stage

1. **Step 7 (Line 7 in NeighborhoodLB Algorithm):** According to the average effective-load computed in step 6 by each node, each node defines in this stage its assistant-neighbors N_{lower}. The set of assistant-neighbors N_{lower} of node n_i are the set of nodes that have effective-load lower than the average effective-load computed by node n_i.

   Column 4 in Table 5 shows the assistant neighbors N_{lower} of each node.

- Workload transfer strategy

1. **Step 8 (Line 8 in NeighborhoodLB Algorithm):** The decision of calling a procedure LB to migrate the excess workloads or not depends on the load difference between the current *effective-load* of node n_i and the *average effective-load* computed by n_i. Therefore, the excess workload will be migrated if the load difference is positive. As seen in column 5, Table 5, n_2 and n_5 are over-loaded nodes.

- Load-balancing mechanism (Procedure LB)

The pseudo-code of the procedure LB is given in Procedure 1. In the procedure LB, the load difference LD_i, the set of *assistant-neighbors* N_{lower}, and the set of the assigned workloads $WL(n_i)$ are formed the procedure input parameters. The procedure will be called if the LD_i is positive, and it works until the load difference of the heavily loaded caller node n_i becomes less than zero $LD_i = l_i - l_{avg} < 0$. In other words, the procedure works until the heavily loaded node becomes under-loaded, which means the *effective-load* of a node is less than the *average effective-load* computed by a node. The procedure first computes the excess workload δ_i of the heavily-loaded node n_i that needs to be transferred.

Then, it sorts: 1) the set of *assistant-neighbors* N_{lower} in descending order based on their *effective-loads*, and 2) the set of submitted workloads $WL(n_i)$ in ascending order in accordance with the weight of each submitted workload. The procedure also checks each node in the set N_{lower} and computes how much a node can receive α (i.e. the workload that a node can receive is equal to the difference between the *effective-load* of a node and the *average effective-load*). The procedure migrates only the workload that has weight less than or equal to α. This step plays a key role in redistributing the excess workloads to the *assistant-neighbors* in a way that ensures that the node who receives the workload maintains the under-loaded status. The LB procedure is terminated when the load difference of the caller heavily-loaded node becomes negative. In other words, the procedure is terminated when the node becomes under-loaded. In the given example, node n_5 is an overloaded node located in$cluster_C$. Node n_5

PROCEDURE LB

Procedure LB($WL(n_i), LD_i, N_{lower}$)
Begin
While($LD_i > 0$)
1. Compute the excess workload of n_i : $\delta_i = LD_i \times c_i$
2. sort the submitted workloads in ascending order
3. sort the assistant neighbours in descending order
4. Let j=0
5. For a node n_j in N_{lower}
  a. compute the excess workload n_j can receive $\alpha = (l_{avg} - l_j) \times c_j$
  b. If $w(wl_k) \le \alpha$ and F is in FS_{n_j} then
    1) k= k+1
    2) send message to node $n_j < n_i, l_i, c_i, FS_i$, "G", $[\alpha, F]$ >
  else
    1) go to step 5
End For
End While
End Begin

Procedure 1.

calculates its excess workloads, sorts the set of *assistant-neighbors* in descending order, and sorts the assigned workloads in ascending order to migrate the excess workloads to the lightly-loaded nodes stored in the *assistant-neighbor* set. It checks the amount of workloads that each node can accept and checks if the function that can process the task is in the FS of the node in the set of *assistant-neighbors*. This step is important to ensure that the node that will receive the migrated workloads 1) will maintain the under loaded status, and 2) will not re-migrate tasks (see Fig. 5).

4.2.3. The convergence

Our proposed solution (i.e. constructing FSW and the proposed algorithm) guarantees converges to the fairness state given sufficient time.

Definition 3 (**Fairness** state)**.** For all $n_i \in N$, when the effective-load l_i of all n_i are equal $l_1 == l_n = l_{avg}$, it is said that the system G achieved fairness status.

Lemma 1. *Given $L^t = (l_1^t, l_2^t, l_n^t)$ is the array of effective-loads for the nodes in the system at time t sorted in ascending order, where l_1^t is effective-load of node 1 at time t. In time t, if there is at least one over-loaded node n_i (i.e. $LD_i > 0$) then L^{t+1} is lexicographically greater than L^t. In other words, the lightly-loaded nodes at time t will receive the excess workload at time $t + 1$.*

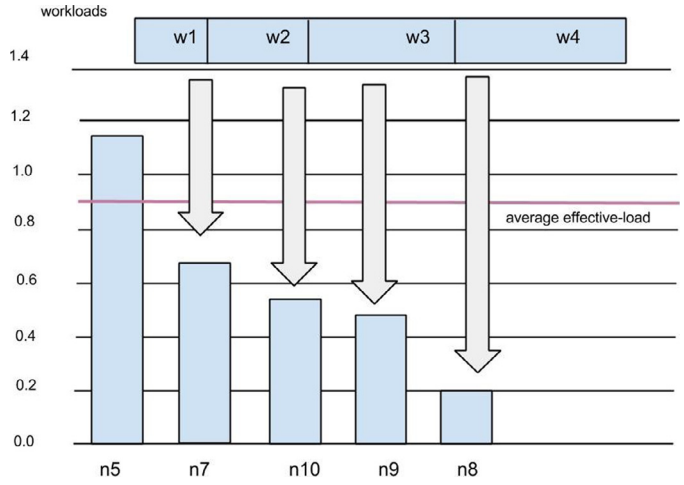

Fig. 5. Workloads migration example. Workloads will be migrated to nodes whose effective-load is less than the average effective-load.

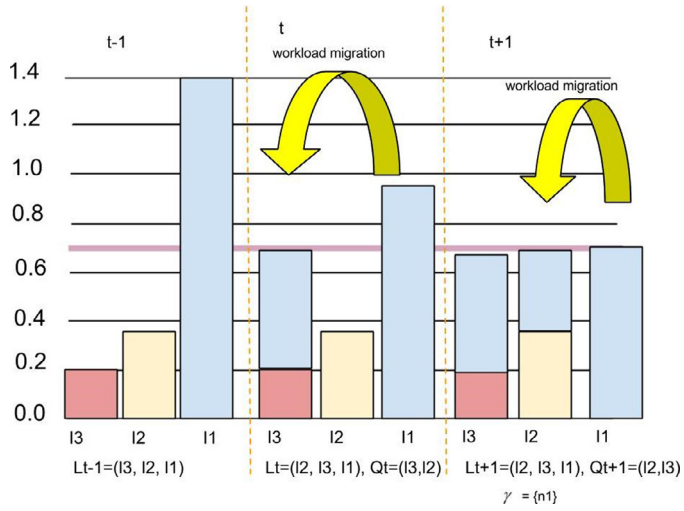

Fig. 6. An illustration example supports Theorem 1, where $t-1$ shows the state of nodes in time $t-1$, t shows the migration in time t, and $t+1$ shows the migration in time $t+1$.

Proof. Let $X \neq \phi$ be the set of overloaded nodes in the system (i.e. nodes with $LD > 0$) who needs to migrate some workloads to other nodes at time t. In reality, a node $i \in X$ will also assign additional workloads in time t. Thus, the nodes that migrate workloads in time t will reduce their effective-load at time $t+1$. Let γ be the node that has lowest effective-load at time $t+1$. Assume that γ occupies the kth position of the array L^{t+1}, where L^{t+1} similarly is the effective-load array of nodes at time $t+1$ sorting in ascending order. Let $Q_t = (l_1^t, l_2^t, \ldots, l_{k-1}^t)$ be the array of the effective-loads in first $k-1$ positions of L^t. In order to prove this lemma we have to consider two cases: 1) A set Q_{t+1} contains a node i that received workloads in time t (node 3 in Fig. 6). Thus, node i belongs to both Q_t and Q_{t+1}, and its effective-load value is increased in time $t+1$ since it will receive some migrated workload. Therefore, there will be at least one workload value in set Q_{t+1} strictly greater than one value in Q_t. Accordingly, L^{t+1} is lexicographically greater than L^t. 2) There are nodes in which located in Q_{t+1} and did not migrate or receive workloads at time t (node 2 in Fig. 6). In this case, the effective-load value at kth position at time $t+1$ is strictly greater than the effective-load value in the same position at time t which has received workload from X^t and therefore, L^{t+1} is lexicographically greater than L^t. □

Table 6
Parameters used in the simulations.

	Description	Values
1.	The assigned workloads	1000–10,000
2.	The number of nodes in the system	100–1000
3.	The cluster size	1–64
4.	The number of functions in the FS per node	1–20

Theorem 1 (Convergence). *In heterogeneous system, if nodes in the system execute the NeighborhoodLB algorithm, then the system converges to a balanced state.*

Proof. Given a heterogeneous unbalanced system. Each node in the system separately executes the proposed algorithm in order to achieve the convergence state. Let n_i be the most heavily loaded node, i.e. $l_i - l_{avg} > 0$, and all other nodes $j \in N$ who have $l_j \leq l_i$ and $l_j < l_{avg}$ form the set of assistant neighbors N_{lower} of node n_i. When $l_i \geq l_{avg}$ (i.e. $l_i - l_{avg} > 0$). Thus, the result of Lemma 1 shall be used, which guarantees that the array of effective-loads sorted in ascending order, in the next time moment, is lexicographically greater than the array of the current step.

Given that the NeighborhoodLB algorithm is executed in a given time t. Let $S \subseteq N$ be the nodes executed the algorithm in time t. Let L^t be the array of effective-loads sorted in ascending order in time t. It has been proven in Lemma 1 that L^{t+1} is lexicographically greater than L^t.

Let $S_{\min}$ be the lightly loaded nodes in time t. There exists at least one node $v \in S_{\min}$ which is assistant neighbor to node k, such that $l_k^t > l_v^t$. Now by using the proposed algorithm, node k migrates a portion of its excess workload to node v, but v does not migrate any workloads in time t because v is under loaded. In time $t+1$ the effective-load of node k decreases; however, its effective-load value never becomes less than the effective-load value of node v which is given by $l_k^{t+1} \geq l_{avg} \geq l_v^t$. Thus, L^{t+1} is lexicographically greater than L^t meaning that the sorted array of effective-load values of nodes in time $t+1$ are lexicographically greater than the sorted array of the effective-load values of nodes at time t. □

5. Simulations

5.1. Experimental setting

We have implemented a discrete-event simulator using the SimJava (Howell and Mcnab, 1998) to compare the performance of our proposed approach with two of the most popular dynamic diffusion approaches, the nearest neighbor algorithm (Tada, 2011) and the original neighborhood algorithm (Neelakantan, 2012).

As the usefulness of any load-balancing algorithm is directly dependent on the quality of its load measurement and the efficiency of being applied to solve practical problems, each approach (our approach, the nearest neighbor algorithm, and the original neighborhood algorithm) was separately applied to a personalized m-cafeteria system (Daraghmi and Yuan, 2013), and the run-time behavior of each algorithm was investigated. The three approaches were run on a set of default values: number of assigned workloads, number of nodes, maximum cluster size, and the average number of the functions executed per node. The simulation parameters, and their values are given in Table 6.

For fairness of comparison, we have tested the three approaches on random graphs (random scenario) generated via random generator (Peixoto, 2014). In the random scenario, the generator will randomly distribute nodes with a functional set associated with each node in the graph. As shown in Table 6, maximum number of functions that each node can execute is 20. Since, in this research, we

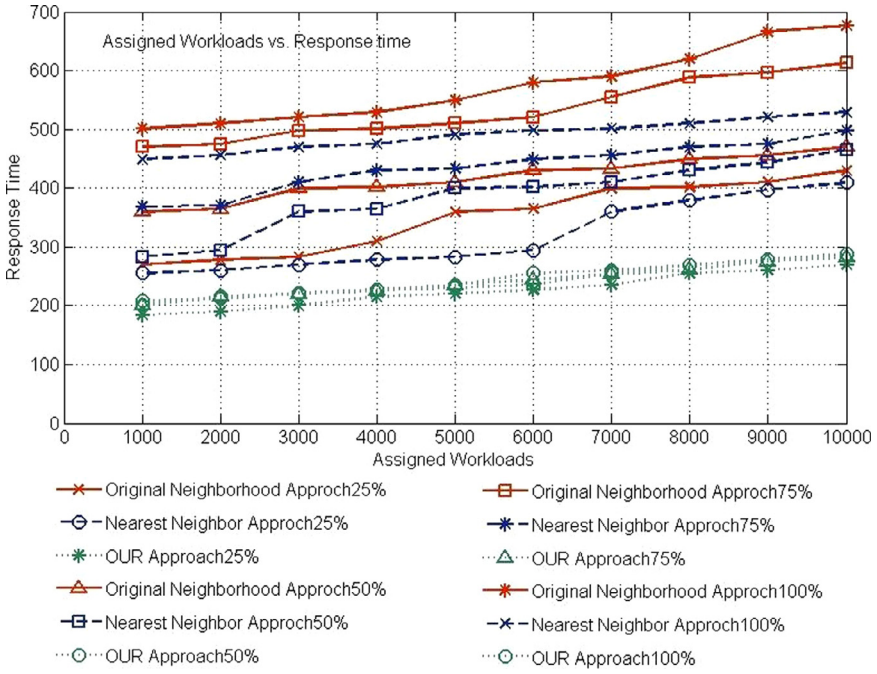

Fig. 7. The response time of original neighborhood approch, nearest neighbor approach, and our approach for various assigned workloads.

propose a two-stage approach (creating a functional small world overall network and then run the NeighborhoodLB on the created FSW) to improve the performance of load-balance algorithm, the random graph, generated previously, will be converted to FSW before executing our proposed NeighborhoodLB algorithm. On the other hand, the other two algorithms, the nearest neighbor algorithm and the original neighborhood algorithm were executed directly on the generated random graph since they do not employ the first stage of creating FSW.

The comparison tests were based on two parameters: the assigned-workloads and the number of nodes, and the measurement of the performance of the algorithm was based on four metrics: the throughput, the response time or the completion time, the communication overhead, and the movement cost. The response time measures the total time that the system takes to serve a submitted request (task). In this experiment, to simulate real world distributes systems, we randomly submitted tasks to nodes. Initially, the request state will be "submitted to node" and will be changed to "complete" upon serving that request. To measure the response time, we count the time needs to change the node response time from "submitted to node" to "complete". The throughput is the rate at which a node in the system sends or receives data (i.e. throughput = 1/response time). In other words, the throughput is defined as the number of nodes that change its status to "complete" in a time unit. As we can see from the proposed algorithm, the load balancing algorithm needs to migrate request from one node to another one in order to achieve a balanced state. We use a simple linear cost model (Ganeasan et al., 2004), where moving one request from any node to any other node costs one unit. Such a model reasonably captures both the network communication cost of transferring data, as well as the cost of modifying local data structure at the node.

Only one parameter was changed each time so that any changes in the performance would be based solely on this parameter. In fact, results achieved from these tests were used to study: (1) the behavior of the different load-balancing algorithms under the same condition; (2) the behavior of the algorithms for random systems with different number of nodes; (3) the behavior of the algorithms for different workloads distribution.

To study the effects of changing the assigned workloads on the average response time, the throughput, the communication overhead, and the movements cost, the assigned workloads were varied from 1000 to 10,000 workloads unit, and the workloads distribution among the nodes were carried in the following manner.

- The initial workload distributions varying 25% from the average effective-load to represent a situation where all nodes have similar workloads at the beginning and those workloads are close to the average effective-load; in other words, the initial situation is quite balanced.
- The initial workload distributions varying 50% from the average effective-load to constitute the intermediate situations.
- The initial workload distributions varying 75% from the average effective-load to constitute the advanced intermediate situations.
- The initial workload distributions varying 100% from the average effective-load to form the situation where the difference of workloads between nodes at the beginning is considerable.

To study the effects of changing the number of nodes on the average response time, the throughput, the communication overhead, and the movements cost, the number of nodes were varied from 100 to 1000 nodes and the distribution of the overloaded nodes were carried in the following manner.

- 25% of nodes are idle, 75% of nodes are overloaded.
- 50% of nodes are idle, 50% of nodes are overloaded.
- 75% of nodes are idle, 25% of nodes are overloaded.

5.2. Comparative study

5.2.1. Average response time

The total time taken for the three algorithms, our proposed algorithm, the original neighborhood algorithm, and the nearest neighbor algorithm, to complete the assigned workloads increased as the assigned workloads was increased as shown in Fig. 7. This situation is expected as the more workloads to be assigned, the longer it takes to complete all the assigned workloads. However, it was observed that our proposed method (i.e. the green line) performed better than

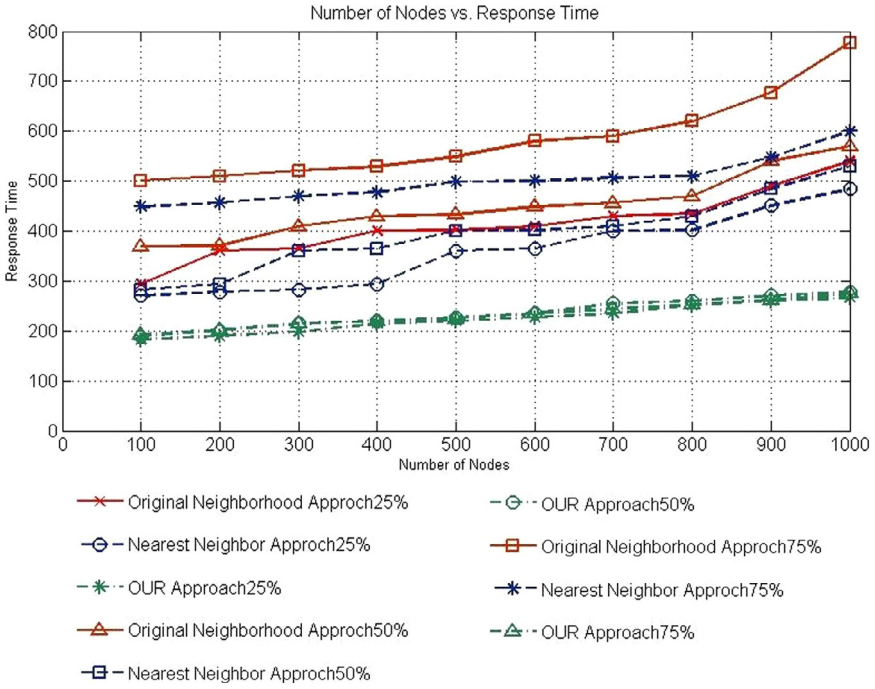

Fig. 8. The response time of original neighborhood approch, nearest neighbor approach, and our approach for various number of nodes. (For interpretation of the references to color in the text, the reader is referred to the web version of this article.)

both the nearest neighbor scheme and the original neighborhood algorithm in all cases. We can see that when comparing the results of our proposed method and the original neighborhood algorithm (i.e. the red line) and the nearest neighbor algorithm (i.e. the blue line), it is observed that the gap between these three curves was widening as the assigned workloads was increased. This shows that the method actually reduced the response time or the total completion time by a considerable amount (greater speedup) in comparison to the original neighborhood algorithm and the nearest neighbor algorithm as amount of workloads increased.

Fig. 8 shows that the response time of the proposed method (i.e. green line) slightly increased when the number of nodes was increased. In contrast, the response time of the original neighborhood method (i.e. red line) and the nearest neighbor method (i.e. blue line) sharply increased when the number of nodes was increased.

The reasons behind achieving better results (i.e. achieving better response time when increasing the assigned workloads or when increasing the number of nodes): 1) our proposed approach constructs a FSW overlay network and then executes the proposed neighborhood load-balancing within the constructed network. Specifically, constructing the overlay network reduces the number of nodes that exchange the workload information, decreases the network diameter, and the communication overhead. As a result, all the stages of the proposed algorithm, such as updating the information of the neighbor nodes, calculating the average effective-load, choosing the assistant neighbors, and migrating tasks to the assistant neighbor that can process the task, will be performed in less time. Our approach also plays a significant role in reducing the time delay results from the task re-migration process as nodes with similar functionality can communicate with each other. As illustrated before, re-migrating tasks occur because of out of the node service scope situation; 2) our proposed approach utilizes the on-state information exchange strategy to broadcast its information to only its *neighbor-nodes,* which has the advantages of achieving more accurate calculation to the effective-load and the average effective-load without increasing the communication overhead (i.e. each node collects the information from less nodes, only *neighbor nodes,* as compared with the original neighborhood approach and the nearest neighbor approach); 3) our approach utilizes the concepts of *assistant-neighbors* and thus heavily loaded nodes will send only (i.e. without accepting any workloads from other nodes since the node is currently overloaded) the excess workloads to the lightly loaded nodes "*assistant-neighbors*", whereas the lightly loaded nodes will only receive the migrated workloads without sending any workloads. In contrast, in the original neighborhood approach and the nearest neighbor approach, all nodes will send and receive workloads at the same time which in turn increase the communication overhead and thus increasing the response time; 4) it is intuitive that a system with longer diameter will take longer time to converge as the number of iterations to propagate the workloads to lightly loaded nodes is proportional to the network diameter; thus, reducing the network diameter via constructing FSW plays a key role in reducing the response time of our proposed approach. In contrast, in the original neighborhood approach and the nearest neighbor approach, each node has to collect the workloads information from larger number of nodes which in turn leads to increase response time.

5.2.2. Throughput

As shown in Fig. 9, our method outperformed the original neighborhood algorithm and the nearest neighbor method in terms of the system throughput in all assigned workloads distribution cases. We can notice that the throughput of the system that executes our proposed approach steadily increased even the assigned workloads increased, whereas the throughput of the system that execute the original neighborhood approach or the nearest neighbor approach drops quickly when the assigned workloads increased.

Fig. 10 shows that the throughput achieved by the original neighborhood algorithm as well as the nearest neighbor approach decreased sharply as the number of nodes in the system increased, while the throughput achieved by our proposed method remains stable even when increasing the number of nodes.

This is because our proposed approach reduces the task completion time which in turn increases the number of tasks completed in

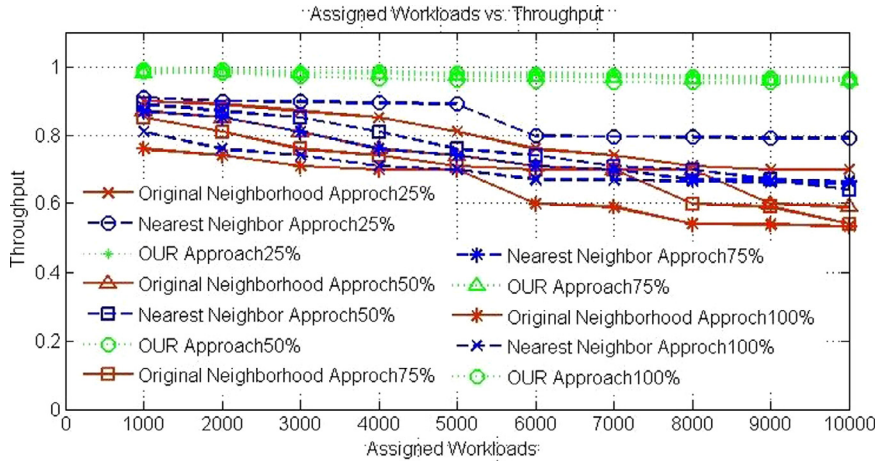

Fig. 9. The throughput of original neighborhood approch, nearest neighbor approach, and our approach for various assigned workloads.

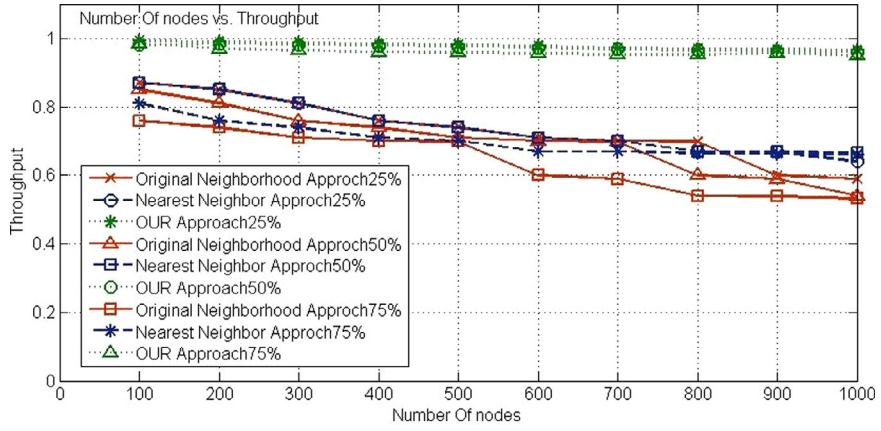

Fig. 10. The throughput of original neighborhood approch, nearest neighbor approach, and our approach for various number of nodes.

a time unit. The reasons behind this are: 1) constructing the FSW that allows nodes with similar functionality to communicate with each other, reduces the possibility of re-migrating tasks (re-migrating tasks consumes time); 2) checking the function that can process the task with the FS before migrating the task, eliminate the possibility of re-migrating tasks. Note that, the first point reduces the time of performing the second point; thus, better results are achieved; 3) reducing the number of nodes that exchange the workload information, decreasing the network diameter, and decreasing the communication overhead reduces the time of performing the proposed algorithm, such as updating the information of the neighbor nodes, calculating the average effective-load, choosing the assistant neighbors, and migrating tasks to the assistant neighbor. As a results, the number of tasks completed in a time unit will be increased; 3) utilizing the concepts of *assistant-neighbors* allowing only heavily loaded nodes to send only (i.e. without accepting any workloads from other nodes since the node is currently overloaded) the excess workloads to the lightly loaded nodes "*assistant-neighbors*". Also, the lightly loaded nodes will only receive the migrated workloads without sending any workloads. In contrast, in the original neighborhood approach and the nearest neighbor approach, all nodes will send and receive workloads at the same time which in turn increase the communication overhead and thus decreasing the task completion time. Moreover, the importance of the *average effective-load* also appears when deciding the amount of workloads to be migrated; if the migrated workloads to one node is too small, then the workload distribution will take longer (i.e. which in turn decreasing the system throughput). In contrast, if the migrated workloads to one node are too large, then the overloaded node may transfer too much workloads to its *neighbor-nodes* and thus this overloaded node will not have sufficient workload to transfer to the remaining lightly loaded nodes. Therefore, by using the *average effective-load*, each node obtains an amount of workload proportional to its capacity and thus no node is privileged which results in increasing the system throughput (i.e. the number of workloads completed in unit time).

5.2.3. Communication overhead

Fig. 11 shows that the average number of messages sent per node increased when the assigned workloads increased. This is because when the assigned workloads increased, the number of messages sent per a node to broadcast its new status increased. We can see that our proposed approach produces less communication overhead than both the original neighborhood approach and the nearest neighbor approach even when increasing the assigned workloads.

Fig. 12 shows that the average number of messages sent per node increased when the number of nodes increased. This is because when the number of nodes increased, each node will send more messages to broadcast its information to the other nodes. We can see that our proposed approach produces less communication overhead than the both the original neighborhood approach and the nearest neighbor approach because: 1) constructing a FSW decreases the number of nodes that exchange the workloads information which in turn decreases the communication overhead; 2) constructing a FSW also decreases the network diameter which directly has the impact of

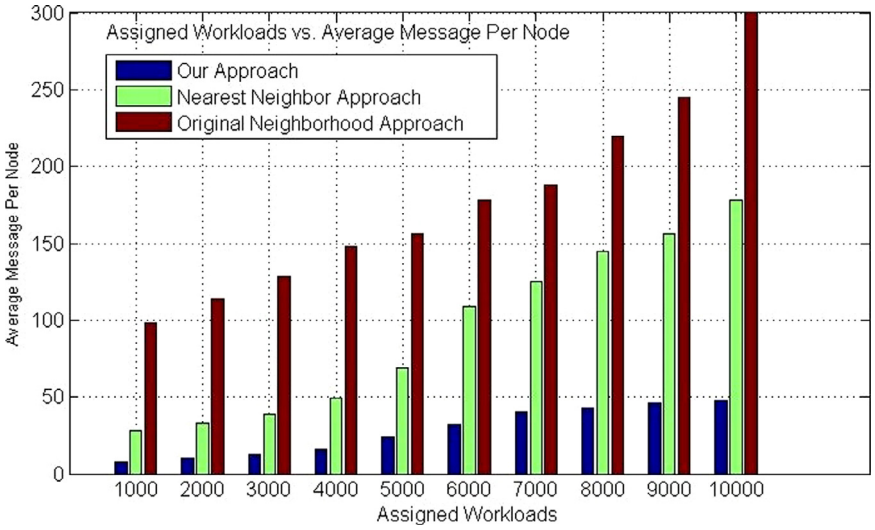

Fig. 11. The average number of messages sent per node of original neighborhood approach, nearest neighbor approach, and our approach for various assigned workloads.

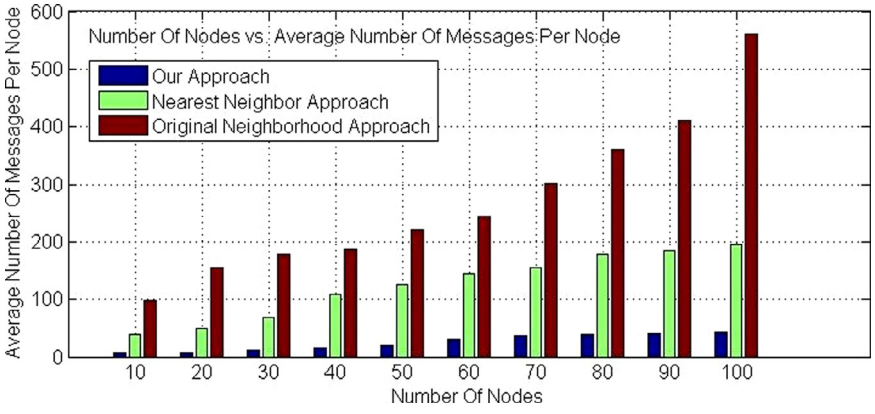

Fig. 12. The average number of messages sent per node of original neighborhood approach, nearest neighbor approach, and our approach for various number of nodes.

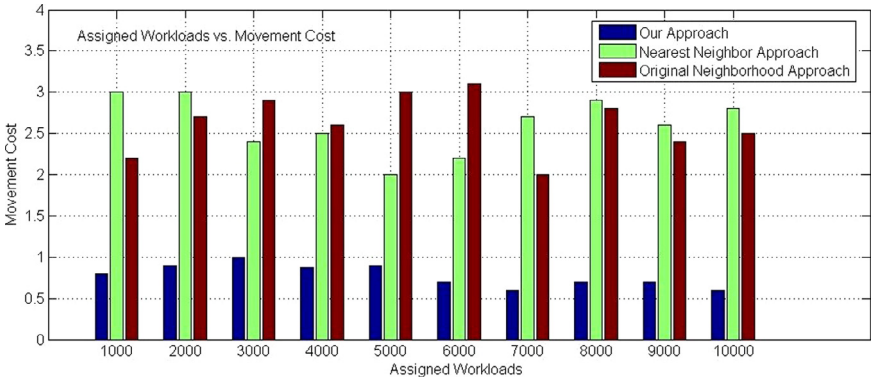

Fig. 13. The movements cost of original neighborhood approach, nearest neighbor approach, and our approach for various assigned workloads.

decreasing the communication overhead; 3) each node that executes the proposed NeighborhoodLB algorithm sends/receives messages to/from only its neighbor nodes which plays a key role in reducing the communication overhead; 4) our approach utilizes the on-state information exchange strategy which reduces the communication overhead; 5) our approach (constructing the FSW, and the proposed load-balancing algorithm) eliminates the possibility of re-migrating tasks which in turn decreases the communication overhead.

5.2.4. Movement cost

Fig. 13 shows the movement cost of original neighborhood approach, the nearest neighbor approach, and our proposed approach vs. the assigned workloads, where the movements cost is defined as the total migrated workloads divided by the total assigned workloads in the system. Clearly, the movements cost of our proposed approach is only 0.32 times the cost of the original neighborhood approach, while the movements cost of our proposed approach is only 0.34 times the cost of the nearest neighbor approach.

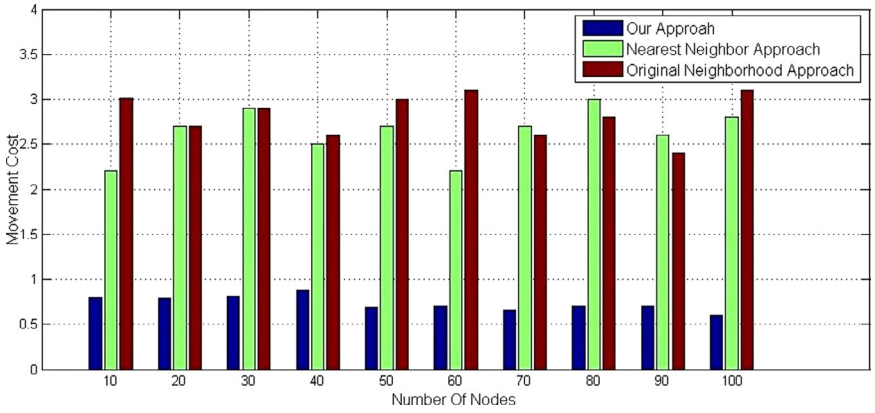

Fig. 14. The movements cost of original neighborhood approach, nearest neighbor approach, and our approach for various number of nodes.

Fig. 14 shows the movement cost of original neighborhood approach, the nearest neighbor approach, and our proposed approach. We can see that the movements cost of our proposed approach is only 0.33 times the cost of the original neighborhood approach, while the movements cost of our proposed approach is only 0.30 times the cost of the nearest neighbor approach.

This is because each node in our proposed algorithm calculates the *average effective-load* to decide whether a node itself is overloaded or not. Specifically, the importance of the *average effective-load* appears when deciding the amount of workloads to be migrated; if the migrated workloads to one node is too small, then the number of workloads that will be migrated will be high (i.e. which in turn increasing the movement costs). In contrast, if the migrated workloads to one node is too large, then the overloaded node may transfer too much workloads to one neighbor node and thus increasing the movements cost. Therefore, by using the average effective-load, each node obtains an amount of workload proportional to its capacity and no node is privileged which leads in decreasing the movements cost. Moreover, our approach utilizes the concepts of assistant-neighbors and thus heavily loaded nodes will send only (i.e. without accepting any workloads from other nodes since the node is currently overloaded) the excess workloads to the lightly loaded nodes "assistant-neighbors", whereas the lightly loaded nodes will only receive the migrated workloads without sending any workloads. In contrast, in the original neighborhood approach and the nearest neighbor approach, all nodes will send and receive workloads at the same time which in turn increase the number of workloads that will be migrated and thus increasing the movements cost. Finally, our approach (constructing the FSW, and the proposed load-balancing algorithm) eliminates the possibility of re-migrating tasks which in turn decreases the movements cost.

6. Conclusion

A novel load-balancing approach to deal with load rebalancing problem in large scale, dynamic and heterogeneous systems has been presented in this paper. Previous research concluded that the technical, and the structural load-balancing factors: (1) increasing the number of nodes in the system (i.e. the number of the nodes exchange the workload information); (2) increasing the network diameter which is defined as the longest shortest path between any two nodes of the network; (3) increasing the communication overheads or the communication delays among the nodes decrease the performance of any load-balancing algorithm as well as affect the algorithm convergence rate. Moreover, additional delay may occur because of the task re-migration process. Therefore, we propose a two-stage approach that first constructs the FSW based on the properties of the small world network and the functionality of each node. Constructing the FSW reduces the number of nodes that exchange the workloads information in the system, decreases the diameter of the network and reduces the communication overhead, and decreases the delay resulted from re-migrating tasks. We also propose a load-balancing algorithm that considers the capacity of each node in order to execute the algorithm within the constructed FSW overlay network. Our proposed approach strives to balance the loads of nodes, increase the system throughput, decrease the response time, reduce the communication overhead, deteriorate the demanded movements cost as much as possible, while taking the advantages of the nodes functionality and the nodes heterogeneity. In the absence of representative real workloads, we have investigated the performance of our proposed approach and compared it against competing algorithms, i.e. the original neighborhood approach, and the nearest neighbor approach. The simulation results are encouraging, indicating that our proposed algorithm performs very well. Our proposed approach dramatically outperforms the original neighborhood approach, and the nearest neighbor approach in terms of response time, throughput, communication overhead, and movements cost. Finally, we have proved that the proposed approach converges to the state of fairness where the effective-load in all nodes is the same since each node receives an amount of workload proportional to its processing capacity. Therefore, we conclude that this approach has the advantage of being fair, simple and no node is privileged.

References

Aakanksha, Bedi, P., 2007. Load balancing on dynamic network using mobile process groups. In: 15th International Conference on Advanced Computing and Communications (ADCOM 2007), pp. 553–558.

Abdelmaboud, A., Jawawi, D.N.A., Ghani, I., Elsafi, A., Kitchenham, B., 2014. Quality of service approaches in cloud computing: a systematic mapping study. J. Syst. Softw. 101, 159–179.

Adolphs, C.P.J., Berenbrink, P., Canada, V.A., 2012. Distributed selfish load balancing with weights and speeds categories and subject descriptors. ACM Symposium on Principles of Distributed Computing. Maderia, Portugal, pp. 135–144.

Akbari, H., Berenbrink, P., Sauerwald, T., 2012. In: PODC A simple approach for adapting continuous load balancing processes to discrete settings, pp. 271–279.

Bahi, J.M., Vernier, F., Cedex, B., 2007. Synchronous distributed load balancing on totally dynamic networks. In: IEEE International Parallel and Distributed Processing Symposium, pp. 1–8.

Boillat, J.E., 1990. Load balancing and Poisson equation in a graph. Concurr. Pract. Exp. 2, 289–313.

Chang, H.-T., Chang, Y.-M., Hsiao, S.-Y., 2014. Scalable network file systems with load balancing and fault tolerance for web services. J. Syst. Softw. 93, 102–109.

Chwa, H.S., Back, H., Lee, J., Phan, K.-M., Shin, I., 2015. Capturing urgency and parallelism using quasi-deadlines for real-time multiprocessor scheduling. J. Syst. Softw. 101, 15–29.

Cybenko, G., 1989. Dynamic load balancing for distributed memory multiprocessors. J. Parallel Distrib. Comput. 7, 279–301.

Daraghmi, E.Y., Yuan, S.-M., 2014. We are so close, less than 4 degrees separating you and me. Comput. Human Behav. 30, 273–285.

Daraghmi, E.Y., Yuan, S.-M., 2013. PMR: personalized mobile restaurant system. 5th Int. Conf. Comput. Sci. Inf. Technol. 275–282.

Elsässer, R., Monien, B., Schamberger, S., Rote, G., 2002. Toward optimal diffusion matrices £. In: International Parallel and Distributed Processing Symposium, pp. 1530–2075.
Fang, Y., Wang, L., 2009. An algorithm of static load balance based on topology for MPLS traffic engineering. In: International Conference on Information Engineering. IEEE, Taiyuan, Shanxi, pp. 26–28.
Ganeasan, P., Bawa, M., Garcia-Molina, H., 2004. Online balancing of range-partitioned data with applications to peer-to-peer systems. In: 30th Annual International Conference on Very Large Data Bases. Morgan Kaufmann, Toronto, Canada, pp. 444–455.
Howell, F., Mcnab, R., 1998. SimJava: a discrete event simulation library for Java. In: International Conference on Web-Based Modeling and Simulation, pp. 51–56.
Hu, Y.F., Blake, R.J., 1999. An improved diffusion algorithm for dynamic load balancing. Parallel Comput. 25, 417–444.
Hui, C., Chanson, S.T., 1999. Hydrodynamic load balancing. IEEE Trans. Parallel Distrib. Syst. 10, 1118–1137.
Hui, C., Chanson, S.T., 1996. A hydro-dynamic approach to heterogeneous dynamic load balancing. In: International Conference on Parallel Processing, pp. 140–147.
Hui, C.-C., Chanson, S.T., 1997. Theoretical analysis of the heterogeneous dynamic load-balancing problem using a hydrodynamic approach. J. Parallel Distrib. Comput. 43, 139–146.
Hui, K.Y.K., Lui, J.C.S., Yau, D.K.Y., 2006. Small-world overlay P2P networks: construction, management and handling of dynamic flash crowds. Comput. Netw. 50, 2727–2746.
Karger, D.R., Ruhl, M., 2004. Simple efficient load balancing algorithms for peer-to-peer systems. In: The Sixteenth Annual ACM Symposium on Parallelism in Algorithms and Architectures, pp. 36–43.
Luo, Y., Zhou, S., Guan, J., 2014. LAYER: a cost-efficient mechanism to support multi-tenant database as a service in cloud. J. Syst. Softw. 101, 86–96.
Luque, E., Ripol, A., Cortes, A., Margalef, T., 1995. A distributed diffusion method for dynamic load balancing on parallel computers. In: The Euromicro Workshop on Parallel and Distributed Processing, pp. 43–50.
Meyerhenke, H., 2009. Dynamic load balancing for parallel numerical simulations based on repartitioning with disturbed diffusion. In: 15th International Conference on Parallel and Distributed Systems. IEEE, pp. 150–157.
Neelakantan, P., 2012. Decentralized load balancing in heterogeneous systems using diffusion approach. Int. J. Distrib. Parallel Syst. 3, 229–239.
Peixoto, T.P., 2014. The graph-tool python library [WWW Document]. figshare. URL https://graph-tool.skewed.de/.
Tada, H., 2011. Nearest neighbor task allocation for large-scale distributed systems. In: 10th International Symposium on Autonomous Decentralized Systems. IEEE, pp. 227–232.
Tversky, A., 1977. Features of similarity. Psychol. Rev. 84, 327–352.
Watts, D.J., Strogatz, S.H., 1998. Collective dynamics of "small-world" networks. Nature 393, 440–442.
Yagoubi, B., Meddeber, M., 2010. Distributed load balancing model for grid computing. ARIMA J. 12, 43–60.
Zomaya, A.Y., Member, S., Teh, Y., 2001. Observations on using genetic algorithms for dynamic load-balancing. IEEE Trans. Parallel Distrib. Syst. 12, 899–911.

Eman Yasser Daraghmi is currently a PhD candidate in the Department of Computer Science and Engineering at National Chiao Tung University, Taiwan. She received her BS degree in communication and information technology from Al Quds Open University in 2008, and her MS degree in Computer Science from National Chiao Tung University, Taiwan in 2011. Her current research interests include cloud computing, distributed systems, and algorithms design.

Shyan-Ming Yuan received his BSEE degree from National Taiwan University in 1981, his MS degree in Computer Science from University of Maryland, Baltimore County in 1985, and his PhD degree in Computer Science from the University of Maryland College Park in 1989. Dr. Yuan joined the Electronics Research and Service Organization, Industrial Technology Research Institute as a Research Member in October 1989. Since September 1990, he has been an Associate Professor at the Department of Computer and Information Science, National Chiao Tung University, Hsinchu, Taiwan. He became the Professor in June 1995. His current research interests include cloud computing, Internet technologies, and distance learning.