Computers and Electrical Engineering 81 (2020) 106517

Contents lists available at ScienceDirect

Computers and Electrical Engineering

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

# Gossip-based fault-tolerant load balancing algorithm with low communication overhead ☆

Moumita Chatterjee [a], Anirban Mitra [b], Sanjit Kumar Setua [a], Sudipta Roy [c,*]

[a] Department of Computer Science and Engineering, Calcutta University Technology Campus, JD-2, Sector-III, Salt Lake, Kolkata 700098, India
[b] Department of Computer Science and Engineering, Academy of Technology, Adisaptagram 712121, West Bengal, India
[c] PRT2L, Washington University in St. Louis, Saint Louis, MO 63110, USA

ARTICLE INFO

*Article history:*
Received 17 February 2019
Revised 13 November 2019
Accepted 14 November 2019
Available online 25 November 2019

*Keywords:*
Load balancing
Fault-tolerance
Gossip protocol
Expander graph
Stability threshold
Network optimization

ABSTRACT

In large-scale distributed-computing environments, numerous factors may affect the predictable performance of load balancing policies causing variable load distribution in the network and bottlenecks. This paper presents a two-level load balancing algorithm for solving the dynamic load balancing problem considering the loss of processors and connectivity of the network. The proposed method privileges local load balancing over global load balancing thus reduces the communication cost over the global network. To further minimize communication costs and to handle loss of connectivity, the algorithm proposes a new communication model for global communication among the processors. The performance, correctness, and scalability of the algorithm are analyzed which show that our algorithm requires O(log m) rounds and O(m log m) messages for m clusters and achieves minimum cluster and global load deviation. Simulation results also support our analytical studies.



## 1. Introduction

Load balancing is an approach by which the load of a heterogeneous network is distributed among individual processors of the network where they work collaboratively so that large loads can be distributed among them in a reasonable and efficient way. Load balancing strategies guarantee that the workload is distributed over all processors depending on their availability and processing speed such that the overall execution time is minimized. However, in some practical distributed systems, the attributes of the incoming workload are unknown and so the processors display non-deterministic runtime behavior. Thus, it is necessary to perform load balancing during run time at regular intervals, so that the run time unpredictability is reduced. This is known as dynamic load balancing. Nevertheless, load balancing at periodic intervals requires frequent communication among the processors to share knowledge about the condition of the system so that individual processors can make an informed decision regarding the assignment of loads to less busy processors according to some load balancing policy.

In a heterogeneous environment, there are a wide variety of issues that need to be considered viz. Different processors may have different capacities because of processor heterogeneity and according to loads imposed on them. Further, in a

☆ This paper is for regular issues of CAEE. Reviews processed and recommended for publication to the Editor-in-Chief by Associate Editor Dr. Fatos Xhafa.

* Corresponding author.
*E-mail address:* sudiptaroy01@yahoo.com (S. Roy).

https://doi.org/10.1016/j.compeleceng.2019.106517
0045-7906/

large scale dynamically changing system the set of processors that are available for computation may change or become unavailable due to failures. These systems consist of processors that can fail at any time or be reassigned to other tasks, despite the load allotted to them. Such settings instigate an uncertainty in the number of functional processors, where each processor may fluctuate between a "working" state and a "failed" state. Thus, it is necessary to design load balancing policies that are robust in order to endure disturbances in the computing medium. We consider cases where each processor can fail with some probability at any random time and can subsequently recover independently. To ensure continuity of operations in the system, the load of a failed processor must be transferred to other functional processors in the system. Clearly, the unpredictability in the number of functional processors may degenerate the performance of any load balancing policies that do not take into consideration the above-mentioned node failure and recovery.

Although a large number of load balancing policies [3–5] has been developed that takes into account unavailability of nodes and the delay introduced due to communication during load balancing, these methods deals with work redistribution and effectiveness of job completion in the presence of node failures and time delays. These methods usually do not deal with scalability issues in terms of the number of nodes and the necessity to acquire global knowledge by the nodes of the system to execute load balancing decisions. Nearly all of the work mentioned above considers the processors to be homogeneous. The emergence of computing systems that are heterogeneous requires load balancing policies that are dynamic, scalable, and robust against failures and considers the processing capacity and the availability of the functional nodes to make load balancing decisions. To solve the problem efficiently, we propose a fault-tolerant load balancing algorithm called GBFTLB which considers the processors to be heterogeneous and takes into account the concept of processor capacity when allocating the load to it. GBFTLB is applicable to systems with arbitrary topologies and can function in both asynchronous and synchronous networks. The goal of GBFTLB is to simultaneously optimize the number of messages and the number of rounds required in fault-tolerant load balancing.

To account for the large scale nature of distributed systems, GBFTLB uses two-level load balancing among the processors in the system. The network is considered as a collection of clusters and local load balancing is used to balance load among the processors in the clusters under the control of cluster heads. Cluster load deviation and Global load deviation are defined for the network and we prove mathematically that minimum values of these parameters correspond to the state of equilibrium in the network. For global balancing, gossiping is used as a mean of communication among the clusters. Gossip based algorithms are usually scalable, fault-tolerant and robust against failures of nodes and links in the network. These algorithms do not necessitate any error recovery mechanism. Even though message dissemination using gossip is probabilistic in nature, these algorithms can achieve high stability of the network against disruptions and scales to a large number of nodes [6,7,10]. Nevertheless, these algorithms incur high message overheads. Thus, we propose to use a highly connected sparse graph known as the expander graph to ensure connectivity and reliable communication of the network in the presence of arbitrary failures of nodes [8,11,12]. We propose a probability transition matrix for gossiping using an expander graph and show by analysis that this method achieves low message complexity and faster convergence compared to other methods of routing. The Contributions of the paper are as follows:

1) We present an algorithm to add *failure detection* to the distributed load balancing problem.
2) We resolved the *scalability* issue by considering the network as a two-level cluster and using a two-level load balancing to balance the workload between the nodes.
3) We proposed to use expander graphs to communicate among the cluster heads *to optimize the message complexity and the number of rounds and provide connectivity in the presence of failures.*
4) We defined the *stable, saturated* and *available* states of the cluster that are used by the cluster heads to make load balancing decisions.
5) We defined a *3-tuple load metric* for each processor to calculate the workload of a processor. CPU and Memory Utilization, Response time and Load level of a processor are considered as load metrics.
6) We derived *equations* to determine the proportion of jobs to be transferred from faulty to non-faulty processors.
7) We proposed distributed *workload estimation and task transfer algorithms* for intra-cluster and inter-cluster load balancing.
8) We proved by analysis that the balancing of load at the local and the global level reduces the total load deviation in the network.

The rest of the paper is organized as follows: Section 2 describes the network model and defines the fault-tolerant load balancing problem. Section 3 describes the network optimization model using an expander graph. Our proposed algorithm is described in Section 4. Section 5 presents the analysis of the algorithm. The performance of our algorithm is presented in Section 6. Finally, Section 7 concludes the paper.

## 2. Models and definitions

### 2.1. Network model

The distributed network consists of n nodes at time t denoted as $V(t) = \{1..n\}$.The nodes in the network are connected by bidirectional links for communicating with each other denoted by E(t). That is node u and v can send messages to each other at time t if and only if $(u,v) \in E(t)$. The network is dynamic in the sense that at every unit of time t, some nodes enter

and leave the system. We consider a failure-prone network where nodes fail and recover independently with a mean-time to failure of mttf and a mean-time-to-recover of mttr. The mean-time between failures is mtbf = mttr + mttf. $p_f = \frac{mttr}{mtbf}$ . We assume the failure of communication links between the nodes during the delivery of messages. We also assume network partitioning, which causes nodes and links to fail concurrently. From the above assumptions, we may consider the network to be changed from time to time. Such a network is defined as follows:

$$G(t) = (V(t), E(t)) \tag{1}$$

The first neighbors of a node $v_i$ are defined as the set of nodes that are directly connected to $v_i$ at time t. i.e.

$$\Gamma_i^1(t) = \left\{v_j \in V(t) : \left\{v_i, v_j\right\} \in E(t)\right\}. \tag{2}$$

The number of nodes in the first neighborhood of $v_i$ at time t is defined as the degree of that node [9]. i.e.

$$\beta_i(t) = \left|\Gamma_i^1(t)\right| \tag{3}$$

We define $\delta = \min_{i \in V(t)} \beta_i(t)$ as the minimum degree of the network. The average degree of the network is defined as $d = 2|E(t)|/n$ and the maximum degree is defined as $\Delta = \max_{i \in V(t)} \beta_i(t)$.

A path between two nodes $v_i$ and $v_j$ is a series of nodes $\langle v_0, v_1, \ldots, v_{k-1}, v_k \rangle$ where $\{v_{i-1}, v_i\} \in E(t)$ for all $i = 1 \ldots k$. The number of edges along the path is defined as the length of the path. We let $d_{ij}(t)$ denote the shortest path between any two nodes $v_i$ and $v_j$ at time t.

Let $\Gamma_i^k(t)$ denote the set of nodes at a distance of k from node $v_i$ i.e.

$$\Gamma_i^k(t) = \left\{v_j \in V(t) \middle| d_{ij}(t) = k\right\}. \tag{4}$$

The diameter of the graph is the length of the longest shortest path denoted as

$$D(t) = \max_{v_i, v_j \in V(t)} \left\{d_{ij}(t)\right\} \tag{5}$$

We define $N_i(t)$ as the number of actual edges in the first neighborhood of node $v_i$ .i.e. $N_i(t)$ is given by

$$N_i(t) = \left|\{v_l, v_k\} : v_l, v_k \in \Gamma_i^1(t) \wedge \{v_l, v_k\} \in E(t)\right| \tag{6}$$

All the nodes in the network are partitioned into $L = \lfloor \frac{n}{\log n} \rfloor$ clusters each of size at most log n.

### 2.2. Problem formulation

We assume that each node $n_i$ in the network has different characteristics such as processing speed and memory capacity denoted by $c_i > 0$. For any node, the inter-arrival time of jobs is assumed to have an average rate of $\lambda_i$ jobs/sec. The service time of the jobs on node $n_i$ requires an average of $1/\mu_i$ seconds. The jobs are assumed to be independent and can be executed on any processor. Each node has a buffer of infinite capacity to store jobs waiting for execution, thus eliminating the dropping of jobs due to unavailable space. Each node possesses an initial number of jobs which is the amount of work to be performed denoted by $l_i \geq 0$. Since $c_i$ and $li$ vary with time, so these variables are denoted as $c_i(t)$ and $l_i(t)$.The jobs are assumed to be non-preemptive and follow the FCFS policy. A system is unbalanced at time t if at least one of the following conditions occurs:

1) Some nodes in the system have failed or are removed at time t.
2) The capacity $c_i(t)$ for some node $n_i \in V(t)$ has changed.
3) The total load of the system $(\sum_i l_i(t))$ has changed.

In practice, the network will become unbalanced from time to time or some nodes in the system may fail. The purpose of our fault-tolerant load-balancing algorithm is to identify the faulty nodes and quickly adjust to the unbalance and attain a balanced state if possible before the next unbalance occurs. The fault-tolerant load balancing problem can be summarized as follows:

**Definition 1** (Fault-tolerant Load Balancing Problem)**.** Given a set of processors with different processing capabilities, where each processor initially has a fixed amount of load, the goal is to distribute the load equally among the processors, according to their capabilities, in the presence of any number of crashes such that if the network is not unbalanced for some finite time T, then the following condition is satisfied:

The load allocated to each functional node $n_i$ is fair, i.e.

$$\forall n_i,\ n_i \in V(t)\ l_i[t+T] = \sum_j l_j . \frac{c_i}{\sum_j c_j}. \tag{7}$$

This is known as global fairness. [2]

### 2.3. Concept of load and global balanced state of the network

In this section, we derive the function for measuring the level of fairness of the network.

**Definition 2** Load level**.** The load level of a functional node $n_i$ is defined as

$$LL_i^t = \frac{l_i}{c_i} \tag{8}$$

**Definition 3** Utilization**.** The utilization of each functional node $n_i$ is determined by the processor time used and memory usage. Higher values for utilization indicate a highly busy node and the lower value indicates a free node. $n_i^{UTL(t)}$ is defined as using the formula: [1]

$$n_i^{UTL(t)} = CPU_i^{\text{busy}} \ * \ RAM_i^{used} \tag{9}$$

where

- $CPU_i^{\text{busy}}$ – Denotes the percentage of time the processor of node $n_i$ is busy.
- $RAM_i^{used}$ – Denotes the percentage of usage of RAM of node $n_i$.

As $CPU_i^{\text{busy}}$ and $RAM_i^{used}$ are percentages, their product gives the Utilization between 0 and 1.

**Definition 4** Response Time**.** Response time is the total amount of time taken by a processor to respond to a service request. The response time of any given job increases with the increase in the workload of the system. So the response time of a system gives a quantitative estimation of its workload. Response time is defined as [1]:

Let a job $J$ comprises of n components $J_1 J_2 \ldots J_n$, then the response time of each component $J_i$ is given as:

$$RT_{J_i}^t = WT_{J_i}^t + ST_{J_i}^t + IPCtime_{J_i}^t \tag{10}$$

where,

- $WT_{J_i}^t$:- $ST_{J_i}^t$- $AT_{J_i}^t$.
- $ST_{J_i}^t$ - Time to process $J_i$.
- $AT_{J_i}^t$ - Arrival Time of $J_i$.

$$IPCtime_{J_i}^t = \beta^t + T^t * msg_{delay}^t \tag{11}$$

where,

- $\beta^t$ - Internal communication among processes in a node at time t.
- $T^t$ - External communication among processes in different nodes at time t.
- $msg_{delay}^t$ - Message delay at time t.

The response time of job J is given as:

$$RT_J^t = \sum_{i=1 \ to \ n} RT_{J_i}^t \tag{12}$$

Average Response time of each node $n_i$ at time t,

$$ART_i^t = \frac{1}{N} \sum_{k=1 \ to \ N} RT_k^t \tag{13}$$

**Definition 5** Cluster Load Level**.** The cluster load level of a cluster is defined as the sum of the load levels of all the functional nodes in the cluster.

$$CLL_i^t = \sum_{n_i \in C} LL_i^t \tag{14}$$

**Definition 6** Cluster Optimum Load Level**.** The optimum load level of a cluster is defined as

$$OLL^t(\text{Ci}) = \frac{CLL_i^t}{\log n} \tag{15}$$

**Definition 7** Global Load level**.** The global load level of a system is defined as the sum of the load levels over all nodes in the network.

$$\text{GLLt} = \sum_{n_i \in V(t)} LL_i^t \tag{16}$$

**Definition 8** Global Optimum Load level**.** The global optimum load level is defined as

$$GLL^t_{opt} = \frac{\mathrm{GLL^t}}{n} \tag{17}$$

**Definition 9** Cluster Load Deviation**.** The cluster load deviation measures the average deviation between the load levels of the functional nodes in the cluster from the optimum load level. The $CLD^t_i$ is given by:

$$CLD^t_i = \sqrt{\frac{\sum_{i=1}^{\lceil \log n \rceil} \left(LL^t_i - \ OLL^t(\mathrm{C_i})\right)^2}{\lceil \log n \rceil}} \tag{18}$$

**Definition 10** Global Load Deviation**.** The global load deviation measures the average deviation between the load levels of the functional nodes in the network from the optimum global load level.

$$\mathrm{GLD^t} = \sqrt{\frac{\sum_{i}^{\lfloor \frac{n}{\log n} \rfloor} \ (CLD^t_i)^2 + \ (OLL^t(\mathrm{C_i}) \ - GLL^t_{opt})^2}{\lfloor \frac{n}{\log n} \rfloor}} \tag{19}$$

## 3. Network optimization model

This section describes the procedure used in our work for minimizing the message complexity and providing fault tolerance in our algorithm. To limit communication among the nodes during gossiping, we define a communication graph as a conceptual data structure consisting of all the nodes and edges in the graph. The nodes in the communication graph correspond to processors and the interconnections among the processors are represented by edges. A processor can send a message to any other processor in the communication graph. Nodes in the communication graph may fail at any time during the computation. When failures occur, the processors that have failed are deleted from the graph. Thus to ensure progress in communication the neighbourhood of the functional processors adjusts dynamically.

### *3.1. Expander graph*

Expander graphs belong to the group of sparse graphs having strong connectivity properties. The connectivity of expander graphs is measured using vertex or edge expansion. Expander graphs are defined as follows:

**Definition 11** (Expander graphs)**.** Let H denote the sub graph of $G(t)$ induced by the subset of V(t) ($H \subset V(t)$). Let $N_G(H)$ denote the set of all nodes in H and all their neighbours. A graph G(t) is called an expander graph if there exists a positive constant $b > 1$ such that $E(H,H^c) \geq b|H|$ whenever $|H| \leq |V(t)|/2$ where $E(H,H^c) = |\{(i,j) \in H, j \in H^c\}|$.

For detailed descriptions of properties and explicit construction of Expander graphs refer to Supplementary material (Appendix B).

### *3.2. Network modelling using expander graph*

We now consider the expander graph described in Section 3.1 built on clusters of the partition. Each cluster has a specific processor designated as the Cluster-Head. The Cluster-Head of each cluster are connected to form a large connected sub graph of G(t) of logarithmic diameter. The Cluster-Head of each cluster maintains a list of all neighbouring cluster heads in G (t).

The stages to construct the network are as follows:

**Stage 1:** The aim of this stage is to elect a processor from each cluster $C_1 \ldots C_{\lfloor \frac{n}{\log n} \rfloor}$ as the cluster head of that cluster. Initially, each processor sends a ClusterheadAck (CA) message to the processor that it considers tobe a cluster head. A processor that receives a majority of CA messages becomes a cluster head and sets its ClusterHead-selected = true and sends Cluster-Initialize messages to other processors in the cluster. Other processors in the network wait for a Cluster-Initialize message from the cluster-head. In all the clusters $C_1, .. C_{\lfloor \frac{n}{\log n} \rfloor}$, the cluster heads are selected in rounds (in parallel).

**Stage 2:** In this stage the cluster head of each cluster tries to build an expander graph according to Algorithm EXCONST in which the cluster heads of two clusters $C_i$ and $C_j$ are adjacent to each other if the cluster $C_i$ and $C_j$ are adjacent in the graph G. Each cluster head tries to find the cluster head of its adjoining clusters by sending messages to all the processors in that cluster until the corresponding cluster head is found. Each cluster head selects an adjacent cluster and sends a Cluster-Head-Acknowledge (CHA) message along all its edges. This is done in d steps (where d is the degree of the graph) and each of the d steps requires at most $\log n - 1$ rounds. Upon completion of this stage, the cluster heads, which are adjacent in the expander graph, know about each other.

## 4. Proposed method: gossip-based fault-tolerant load balancing algorithm (GBFTLB)

The algorithm has two levels at which it must be described: the intra-cluster level and the inter-cluster level. The cluster head responsible for the balancing process receives information about workload from the worker nodes in the cluster and L−1 clusters. A cluster head invokes intra-cluster load balancing whenever it detects an imbalance in that cluster. If the cluster cannot balance load among the nodes, it invokes inter-cluster load balancing to balance load among adjacent clusters. Thus the load balancing is achieved by using this two-level structure which decentralizes the load balancing process. This decentralization minimizes overhead of communication and so it can be scaled to large networks. The communication costs between clusters may vary due to the heterogeneity of the distributed network, however, intra-cluster communication costs are same as processor nodes in a cluster are connected by LAN which provides similar bandwidths to all the nodes.

### *4.1. Local phase: intra cluster load balancing*

At the intra-cluster level, each worker node estimates its load and sends the information to the cluster head. In addition, each processor periodically sends heart-beat messages to the cluster head. Depending on the current load and availability of the cluster, the cluster head decides whether local or global load balancing is needed.

Each worker node in the system is responsible for:

- Send heart beats to cluster head every $T_{faultdetection}$ second.
- Determining its workload.
- Sending the load information to the cluster head.
- Performing job transfers as decided by its cluster head.

A cluster head is mainly responsible for-

- Receiving load information from all the worker nodes and estimating the current load of the cluster.
- Estimating availability of the cluster.
- Decide local load balancing parameters and informing the decision of intra-cluster load balancing to each node in the cluster.
- Decide to start inter-cluster load balancing.

#### *4.1.1. Identification of faulty processors by the cluster-head*

At each periodic interval of time $T_{faultdetection}$, each worker node in the cluster sends heartbeat messages $h_1^n$, $h_2^n$, $h_3^n$,..... to the Cluster-Head. Each message $h_i^n$, is tagged with a sequence number i and the index of the worker node. Let $\sigma_i^n$ denotes the time of sending of heartbeat message $h_i^n$ of worker node n. The cluster head forwards the $\sigma_i^n$ s by $\delta$ (the propagation time of messages in the cluster) to calculate the sequence of expected receiving times $\tau_1^n$, $\tau_2^n$, $\tau_3^n$,..... where $\tau_1^n < \tau_2^n < \tau_3^n$ and $\tau_i^n = \sigma_i^n + \delta$ . The cluster head uses the $\tau_i^n$s and the time of receiving the heartbeat messages to decide if it can trust or suspect n.

Let us consider the interval $[\tau_i^n, \tau_{i+1}^n)$. At $\tau_i^n$, the cluster head checks whether it has received any messages $h_j^n$, from n where $j > i$. If it has received the message, then the cluster head starts trusting n during the whole interval $[\tau_i^n, \tau_{i+1}^n)$. Otherwise, it starts suspecting n. If it receives the heartbeat message at some time before $\tau_{i+1}^n$, then the cluster head trusts n from that time onwards until $\tau_{i+1}^n$. If the cluster head has not received any message $h_j^n$ with $j \geq i$, then it suspects n during the whole interval $[\tau_i^n, \tau_{i+1}^n)$. The cluster head updates the status of each node in the cluster in the status vector. This process is repeated for every interval. The pseudo-code is given in Algorithm 1.

**Algorithm 1** Failure detection at the cluster head.

**Worker node n:**
1. **For all** $i \geq 1$, and at time $\sigma_i^n = i$ .$T_{faultdetection}$
   1.1 Send heartbeat message $h_i^n$ to cluster head

**Cluster Head:**
2. **For all** worker node n
   2.1 **Initialize** $fault_i$ [n]=*S*
   2.2 **For each** n and for all $i \geq 1$ and at time $\tau_i^n = \sigma_i^n + \delta$
   2.3 **If** it does not receive $h_j^n$ with $j \geq i$ **Then**
      2.3.1. $fault_i$ [n]=*S*
   2.4 On receiving $h_j^n$ at time t $\in [\tau_i^n, \tau_{i+1}^n)$
   2.5 If $j \geq i$ **Then**
      2.5.1 $fault_i[n]$=*T*

#### *4.1.2. Availability estimation of the cluster*

In this step, the Cluster-Head tries to estimate the availability of the cluster from the status vector. The state of each node in the cluster as defined in the status vector describes the configuration of the cluster.

**Definition 12** Configuration of the cluster**.** A configuration of the cluster $C_i$, denoted by $CONF_{C(i)}$, specifies the state of each node in the cluster (whether the nodes are trusted or suspected to have failed). The probability of a cluster $C_i$ to be in a configuration $CONF_{C(i)}$ with k functional nodes as derived from the status vector is

$$\text{Prob}\left(CONF_{C(i)}\right) = \binom{\lfloor \log n \rfloor}{k} \left(1 - p_f\right)^k \; p_f^{(\lfloor \log n \rfloor - k)}. \tag{20}$$

The availability of the cluster $C_i$ specifies the prevailing status of the cluster for the requisite number of failed/suspected nodes for which the cluster is able to process jobs when the cluster is in a given configuration. For a specified configuration, the availability index of the cluster is defined as:

**Definition 13** Availability Index of the cluster**.** Availability index of the cluster $C_i$ in a given configuration $CONF_{C(i)}$, over a time interval of duration t is defined as the ratio of the number of functional nodes available in the cluster to the total number of nodes in the cluster in an interval of time length t when the cluster is in the specified configuration. Availability index of cluster $C_i$ ($Av_i$ (t)) is defined as:

$$Av_{\text{i}}\ (\text{t}) = \frac{FN_i^t}{TN_i^t} \tag{21}$$

where

- $FN_i^t$ =Total number of functional nodes in the cluster i at time t.
- $TN_i^t$ = Total number of nodes in clusters i at time t.

*4.1.3. Intra-cluster transfer of jobs from faulty to non-faulty nodes*

In this step, we calculate the overall arrival rate of jobs at node n. Jobs at node n arrive randomly with exponentially distributed inter-arrival time of $1/\lambda_n$. In addition, the cluster head sends jobs from the faulty nodes to each functional node. The transfer proportion of the target node w is determined by the rate at which jobs are processed at w.

$$TRANS_{PROP} = \left(LL_w^t - \left(\frac{\lambda_w}{\mu_w}\right) LL_w^t\right) \forall w \in C \tag{22}$$

Thus the average transfer proportions $p_{ij}$ can be calculated as follows:

$$p_{ij} = \sum_{i=1}^{\binom{\lfloor \log n \rfloor}{k}} \text{Prob}\left(CONF_{C(i)}\right) \times TRANS_{PROP}(i) \tag{23}$$

When a node j fails, the proportion of jobs that is moved to neighboring processors when the system is in configuration $CONF_{C(i)}$ is $p_{ij}$.

Thus the mean overall arrival rate of jobs, including jobs transferred from failed nodes at node w is:

$$Mean\ Arrival\ Rate_m = \lambda_{\text{m}} + \sum_{j \neq m} p_{ij} \tag{24}$$

So the average loading of a functional node w is

$$LOAD_w^{AVG} = \frac{Mean\ Arrival\ Rate_w}{\mu_w} \tag{25}$$

*4.1.4. Workload estimation of the cluster*

At each periodic interval of time, each functional/trusted node $n_i$ in the cluster computes its workload information parameters and sends the load information to the cluster head. Memory Utilization, CPU Utilization, Response Time and Load level are used to determine the workload of $n_i$. Each worker node $n_i$ then sends this workload information to its cluster head in the form of a 3 tuple $< n_i^{UTL(t)}, ART_i^t, LOAD_i^{AVG} >$.

After receiving the 3-tuple workload information from each worker node $n_i$ in the cluster, the cluster head computes the average of $LOAD_i^{AVG}$ and $ART_i^t$ . It then arranges the $n_i^{UTL(t)}$ values collected from each node to classify the nodes into different groups based on their memory and CPU utilization [1].

**Definition 14** Stable state of a cluster**.** A cluster is in a stable state at time t if loads are uniformly balanced within the cluster and intra-cluster load balancing is not required at time t. To check stability, both the following two conditions are to be satisfied:

1. The distance between a defined stability threshold $\delta$(t) and average response time $L_{avg}$(ART) is computed by using the following formula:

$$\text{D}(\delta(\text{t}),\ \text{L}_{\text{avg}}\ (\text{ART})) = \frac{\delta(t) - L_{avg}(ART)}{\delta(t)} \tag{26}$$

**Algorithm 2** Workload estimation.

1. **For** each worker node of Cluster C **do**
   1.1 Calculate Node Utilization- $n_i^{UTL(t)} = CPU_i^{\text{busy}} * RAM_i^{used}$
   1.2 Calculate Average Response Time $ART_i^t = \frac{1}{N} \sum_{k=1\ to\ N} RT_k^t$
   1.3 Calculate Average Load of node $LOAD_i^{AVG} = \frac{Mean\ Arrival\ Rate_i}{\mu_i}$
2. $n_i$ forwards its load estimation in the form of 3 tuple $<n_i^{UTL(t)}, ART_i^t, LOAD_i^{AVG}>$ to C.
3. Upon receiving all workloads C performs
   3.1 Computes average of ART $L_{avg}(ART) = \frac{1}{\log n} \sum_{i=1\ to\ \log n} ART_i^t$
   3.2 Sort $n_i^{UTL(t)}$ accepted from each node $n_i$ and categorize each node according to the following condition:

$$\text{Load} = \begin{cases} \text{Idle} & \text{if } n_i^{UTL} < 10\% \\ \text{Low} & \text{if } n_i^{UTL} < 50\% \\ \text{Normal} & \text{if } n_i^{UTL} > 50\% \text{ and } n_i^{UTL} < 90\% \\ \text{High} & \text{if } n_i^{UTL} > 90\% \end{cases}$$

   3.3. Compute Stability Criteria $D(L_{avg}(ART), \delta(t)) = \frac{\delta(t) - L_{avg}(ART)}{\delta(t)}$
   3.4. Compute Saturation Criteria $ST = \prod_{\forall i \in C} n_i^{UTL}$
   3.5. Compute Cluster Load Deviation $CLD_i^t = \sqrt{\frac{\sum_{i=1}^{\lceil \log n \rceil} (LL_i^t - OLL^t(C_i))}{\lceil \log n \rceil}}$
4. **If** $D > 0$ and $CLD_i^t \to 0$ **Then**
   4.1. C is stable.
5. **End If**
6. **If** ST < 1% or $\forall k\ LL_k^t > OLL^t(C_i)$ **Then**
   6.1. C is saturated.
7. **End If**
8. Divide C into Over loaded (O), Under-loaded (U), and Balanced (OK) nodes.
9. Initially $O = \emptyset$, $U = \emptyset$, $OK = \emptyset$
10. Evaluate the load on each node $n_i$ and compare it with values calculated in Step 3(3.1 and 3.2)

**Switch**

**Case 1**: **If** $LOAD_i^{AVG} \le OLL^t(C_i)$ **AND** Load($n_i$) = Idle **AND** $ART_i^t \le L_{avg}(ART)$ **then** $U = U\ n_i$
**Case 2**: **If** $LOAD_i^{AVG} \le OLL^t(C_i)$ **AND** Load($n_i$) = Low **AND** $ART_i^t \le L_{avg}(ART)$ **then** $U = U\ n_i$
**Case 3**: **If** $LOAD_i^{AVG} \le OLL^t(C_i)$ **AND** Load($n_i$) = Normal **AND** $ART_i^t \le L_{avg}(ART)$ **then** $U = U\ n_i$
**Case 4**: **If** $LOAD_i^{AVG} \le OLL^t(C_i)$ **AND** Load($n_i$) = High **AND** $ART_i^t \le L_{avg}(ART)$ **then** $OK = OK\ U\ n_i$
**Case 5**: **If** $LOAD_i^{AVG} > OLL^t(C_i)$ **AND** Load($n_i$) = Normal **AND** $ART_i^t > L_{avg}(ART)$ **then** $O = O\ U\ n_i$
**Case 6**: **If** $LOAD_i^{AVG} > OLL^t(C_i)$ **AND** Load($n_i$) = High **AND** $ART_i^t > L_{avg}(ART)$ **then** $O = O\ U\ n_i$

**End Switch**

11. Balance each node by Intra Cluster Task Transfer.

If D is less than 0, the cluster is unstable. The value of $\delta(t)$ is different for different clusters. $\delta(t)$ is defined as follows:

$$\delta(t) = \alpha \times \delta(t-1) + (1-\alpha) \times L_{avg}(ART)(t-1) \tag{27}$$

where $0 \le \alpha \le 1$ is called the decision factor and depends on the availability of the cluster and number of jobs in the cluster at that load balancing interval. As the number of jobs increases and the availability decreases, $\alpha$ increases from 0 towards 1.

2. $CLD_i^t \to 0$.

**Definition 15** Saturation state of the cluster**.** A cluster may be in a saturated state while being stable. In a saturated state, all the working nodes of the cluster will be overloaded and thus it is useless to perform intra-cluster load balancing. We use the following two conditions to measure saturation:

1. A saturation index ST is used. ST is defined as

$$ST = \prod_{\forall i \in C} n_i^{UTL} \tag{28}$$

**If** ST< 1%, **then** the cluster is saturated.

2. For node k in the cluster

$$LL_k^t > OLL^t(C_i) \tag{29}$$

A cluster is saturated if either condition 1 or 2 are satisfied. The Cluster head ascertains the saturation state of the cluster by verifying the conditions. It then partitions the nodes of the cluster into under loaded, overloaded and balanced nodes based on the decision criteria. The pseudo-code is given in Algorithm 2.

*4.1.5. Intra cluster task transfer*

The Cluster head maintains two priority queues for overloaded and under-loaded nodes. The overloaded set's queue is sorted in descending order of $ART_i^t$ and $LOAD_i^{AVG}$. The under-loaded set's queue is sorted in ascending order of $ART_i^t$ and

**Algorithm 3** Intra-cluster task transfer.

1. Add nodes of O to priority queue in descending order of $LOAD_i^{AVG}$ and $ART_i^t$.
2. Add nodes of U to priority queue in ascending order of $LOAD_i^{AVG}$ and $ART_i^t$.
3. **For** each $n_i$ in O **Do**
   3.1. Select a node $n_j$ from U with lowest $ART_i^t$ and $LOAD_i^{AVG}$.
   3.2. Calculate workload to be transferred from $n_i$ to $n_j$ as $S_k = \gamma \frac{c_i c_j}{c_i + c_j} (LOAD_i^{AVG} - LOAD_j^{AVG})$
   3.3. Node $n_j$ accepts the workload if $S_k < \frac{1}{2} LOAD_j^{AVG}$ .
   3.4. Transfer $S_k$ amount of tasks to node $n_j$ from node $n_i$.
4. Update workloads of source $n_i$ and receiver $n_j$.
5. Update sets Overloaded **O**, Under-loaded **U** and Balanced **OK**.
6. **If** ($U = \varphi$ OR $O = \varphi$) **Then // Balancing Condition**
   **Exit**
7. **End If**
8. **End for**

$LOAD_i^{AVG}$. For each node $n_i$ in the overloaded set, a node $n_j$ is selected from the under-loaded set and the excess workload $S_k$ to be transferred is calculated using the following formula:

$$S_k = \gamma \frac{c_i c_j}{c_i + c_j} \left( LOAD_i^{AVG} - LOAD_j^{AVG} \right) \tag{30}$$

where $\gamma$ is the local balancing factor and varies from 0 to 1.

Node $n_j$ accepts the workload if $S_k < \frac{1}{2} LOAD_j^{AVG}$. Algorithm 3 describes intra-cluster task transfer.

### 4.2. Global phase: inter-cluster load balancing

Inter-cluster load balancing is required if the Cluster head is not able to balance the load among the worker nodes of the cluster. A cluster is considered to be overloaded if the availability is less than 10% or the saturation threshold level ST becomes less than 1%. In such situations, the Cluster head communicates with other Cluster heads of its possible load transfer decision. In inter-cluster load balancing, communication cost among clusters has to be considered.

Each cluster head maintains the availability index of its own cluster and its workload information as a 2 tuple form $< CLL_i^t, Av_i\ (t) >$. During inter-cluster load balancing, the cluster head selects a set of cluster heads from its d neighbours that have high value of P and sends the gossip message along the edge with the highest probability. When a cluster head $C_k$ receives a request, it compares its load tuple with the workload information received from the cluster head. If the availability of cluster $C_k$ is more than $C_i$ or/and workload is less, then $C_k$ sends a reply to $C_i$. In addition, it also forwards the request message to another cluster heads according to its probability P. When the cluster head $C_i$ receives responses from other cluster heads, it calculates the excess load of the cluster $C_i$ compared to each cluster $C_k$ as:

$$\forall_k \ ExcessLoad_{C_i}^{C_k} = CLL_i^t - CLL_k^t \tag{31}$$

$C_i$ then checks the availability of each cluster $C_k$ and computes the portion of load to be transferred to each cluster $C_k$ as

$$TransferedLoad_{C_i}^{C_k} = \rho \left( \frac{ExcessLoad_{C_i}^{C_k}}{CLL_i^t} \times \frac{Av_k}{Av_i} \right) \tag{32}$$

where $\rho$ is called the global balancing factor and is in the range of (0,1). The clusters are sorted according to their $TransferedLoad_{C_i}^{C_k}$ . The cluster head $C_i$ arranges the overloaded nodes in descending order according to their workload and response times. Now the Cluster-Head of the overloaded cluster $C_i$ transfers the load $TransferedLoad_{C_i}^{C_k}$ to each cluster $C_k$. The pseudo-code is given in Algorithm 4.

## 5. Analysis of the algorithm

### 5.1. Proofs on load balancing and failure detection

**Theorem 1.** *Let $Conf_{C(i)}$ is a configuration of a cluster with f non-faulty nodes and m-f faulty nodes, then the expected duration that $Conf_{C(i)}$ exists is* $\geq \frac{1}{\frac{f}{mttf} + \frac{m-f}{mttr}}$ .

**Proof.** Let us consider a small duration of time dt. The probability that a particular node that has currently failed, recovers during an interval dt is $\frac{dt}{mttr}$. Equivalently, the probability that a non-faulty node crashes in the interval dt is $\frac{dt}{mttf}$. In configuration $Conf_{C(i)}$, there are *f* functional(non-faulty) nodes and *m-f* faulty nodes. Consequently, the probability of the cluster

**Algorithm 4** Inter cluster load balancing algorithm.

**Local Phase: When overloaded $C_i$ issues a global load balancing request**
1. Make a gossip message and initiate it.
   1.1. Gossip.requesterid=$C_i$
   1.2. Gossip.timestamp=$C_i$.timestamp
2. Calculate P for each of its neighboring cluster head where P is defined as in (21).
3. Select $C_k$ with the highest probability P.
4. Send Gossip to $C_k$ .

**Remote Phase:** When $C_k$ receives Gossip from $C_i$
5. If $CLL_i^t > CLL_k^t$ and $AV_i$ (t) < $AV_k$ (t)
   5.1. Send $<CLL_k^t, AV_k(t)>$ to $C_i$

**Local Phase:** When $C_i$ receives load messages from other cluster heads
6. For each cluster $C_k$
   6.1. Compute $ExcessLoad_{C_i}^{C_k}$ .
   6.2. Compute $TransferedLoad_{C_i}^{C_k}$ .
   6.3. Sort the clusters in ascending order according to $TransferedLoad_{C_i}^{C_k}$.
   6.4. Sort the functional overloaded processors of cluster $C_i$ in descending order according to their workload and response times.
   6.5. Transfer the highest priority task from the first overloaded node of $C_i$ to the cluster $C_k$
   6.6. Update locally the workload of $C_i$.

**Remote Phase:** When the $C_j$ receives workload from $C_i$
7. Sort the under loaded processors in ascending order of their workload and response times.
8. Allocate jobs to the under loaded processors and update locally the under loaded processor list.

$C_i$ changing to some other configuration from $Conf_{C(i)}$ during an interval dt is $\leq \frac{f}{mttf} + \frac{m-f}{mttr}$. So, the expected duration for which $Conf_{C(i)}$ lasts is $\geq \frac{1}{\frac{f}{mttf} + \frac{m-f}{mttr}}$.

**Theorem 2.** *When a cluster head of cluster $C_i$ balances an overloaded worker node, then there is a positive decrease in $CLD_i^t$.*

**Proof.** The proof is based on finding the difference between CLD and modified value of CLD after the load is transferred. The details of the proof are given in Supplementary material (Appendix A).

**Lemma 1.** *The saturation state of the cluster is achieved when $\forall i\ LL_i^t > OLL^t(C_i)$ .*

**Proof.** The proof is based on determining the saturation state of the cluster. The details of the proof are given in Supplementary material (Appendix A).

**Theorem 3.** *For any network, the reduction of $CLD_i^t$ results in the reduction of $GLD^t$.*

**Proof.** The proof is based on finding the difference between $GLD^t$ and modified $GLD^t$ of the cluster. The details of the proof are given in Supplementary material (Appendix A).

**Lemma 2.** *If a cluster is balanced, then $CLD_i^t \rightarrow 0$.*

**Proof.** If a cluster $C_i$ is balanced, then

$$LL_i^t = OLL^t(C_i)$$

So, $CLD_i^t$ is calculated as

$$CLD_i^t = \sqrt{\frac{\sum_{i=1}^{\lceil \log n \rceil} (LL_i - OLL^t(C_i))^2}{\lceil \log n \rceil}} = \sqrt{\frac{\sum_{i=1}^{\lceil \log n \rceil} (OLL^t(C_i) - OLL^t(C_i))^2}{\lceil \log n \rceil}} = 0$$

**Theorem 4.** *Given a network, $GLD^t$ is minimized and tends to approach zero if and only if the state of global fairness is achieved i.e.*

$$LL_i^t = LL_{opt}^t \ \forall i$$

where $LL_{opt}^t = \frac{\sum_j l_j}{\sum_j c_j}$

**Proof.** To prove our theorem, the $GLD^t$ of the globally fair system is compared with the $GLD^t$ of all other configurations where the load levels of the nodes are not equal. For systems that are unfair, we divide the node-set V(t) into 3 subsets and load are transferred from overloaded to under-loaded nodes and difference in $GLD^t$ is calculated. For details of proof refer to Supplementary material (Appendix A).

### 5.2. Proofs on network model

**Theorem 5.** *The graph constructed by Algorithm EXCONST is an expander graph.*

**Proof.** The proof is based on calculating conductance of the probability transition matrix P of the graph given by

$$\Phi(\mathrm{P}) = \min_{H \subset V(t):|H| \leq \frac{n}{2}} \frac{\sum_{i \in H; j \in H^c} P_{ij}}{|H|} \text{ where } \mathrm{H} \subset \mathrm{V(t)} \tag{33}$$

We need to show that P has a conductance greater than 0 and independent of n. For details of proof refer to Supplementary material (Appendix B).

**Theorem 6.** *Let us consider an arbitrary initiator cluster head 'u' of the expander graph constructed using Algorithm EXCONST. The gossip-based information dissemination process starting from 'u' informs all other cluster heads within O (log m) rounds where m is the number of clusters in the network.*

**Proof.** The proof is based on the concept of informed and uninformed nodes. We consider $I_t$ as the number of informed cluster heads at the beginning of each round t and $\mathrm{H(t)} \subseteq I_t$ is the set of cluster heads that have been sent the message in the $t^{th}$ step but has not spread the information yet. Our proof has three phases and each of the phases is responsible for information dissemination from vertices in previously formed set H(t-1). The initiator is the only informed node at the beginning of the information dissemination process. At each step of the process, the neighborhood of the initiator node increases exponentially according to Property1 of the expander graph. (Supplementary material Appendix B). For details of proof refer to Supplementary material (Appendix B).

**Theorem 7.** *The global phase of GBFTLB protocol generates O (m log log m) messages where m is the number of clusters in the network.*

**Proof.** Let $d = logm - 1$. The initiator node forwards the gossip message only to those neighbours having energy greater than the energy of the initiator node. The expected number of nodes that the initiator node contacts is calculated as follows (in the worst case the messages are sent to all d nodes): The probability of sending a gossip message to exactly i nodes will be $\theta(\frac{1}{i+1}.i)$. Thus, the expected numbers of nodes receiving gossip messages from the initiator is $\sum_{i=1}^{d} \theta(\frac{1}{i+1}.i) = O(\log d)$ and expected number of gossip messages is$O(\log d)$. So by linearity of expectation, it can be written that all the nodes in the network sends an expected number of $O(m \log d) = (m \log \log m)$ gossip messages.

## 6. Performance evaluation

In this section, we discuss the results of our simulation that is used to evaluate our proposed method. We developed a discrete event simulator in C language to perform the simulation of our proposed method. The simulations are done by varying system parameters: number of clusters and number of tasks. We assume an arbitrary topology for the communication channels connecting the clusters. The environment for our simulation is as follows:

- Intel(R) Core TM i5-3210 M CPU (2.5-GHz clock) and 4GB RAM.
- Ubuntu 14.04
- C compiler: GCC version 4.8.2

In simulation, each node performs peer sampling to select its neighbors. Some assumptions are made to simplify the simulations. The assumptions are:

- The links between the nodes do not fail.
- A node can fail at any time.
- Peer samplings are performed successfully so the nodes know about their neighbours.

Table 1 shows the parameter values used for simulation. These values are implicitly used for all cases unless stated otherwise. The job dataset is downloaded from the Google Cluster dataset [15].

### 6.1. Performance analysis of proposed expander routing method in comparison to other methods of routing

In this section, we focus on the performance comparison of our proposed expander routing protocol with other stochastic methods of routing [13,14]. Appendix C, Section 1 illustrates the performance of gossip protocols in large scale systems. Fig. 1 of Appendix C shows the theoretical results and simulation results showing a fraction of informed nodes for each round and the standard deviation between theoretical and simulation results for the fraction of informed nodes for each round. The considered algorithms are *random walking method* and *receiver-degree routing method*. The gossiping protocols on these algorithms also perform peer sampling so that nodes have the knowledge of their neighbours. The random walking method is an unbiased routing method wherever each node uniformly allocates transition probabilities to its neighbours. The receiver-degree routing method for any connected node pair assigns transition probabilities to a node according to the

**Table 1**
Parameter values.

| Settings | Parameters | Values |
|---|---|---|
| Static | # nodes | (10,000),1000,100,10 |
| Static | Mode of Gossip | Push-Pull |
| Static | # nodes contacted in each round | (1),4,10,20,40 |
| Static | [peer list] | 10 |
| Dynamic | # nodes | 10,000 |
| Dynamic | Mode of Gossip | Push-Pull |
| Dynamic | # nodes contacted in each round | 1 |
| Dynamic | [peer list] | 20 |
| – | # clusters | 10,20,30,40 |
| – | Cluster availability | 100,80,50,30 |

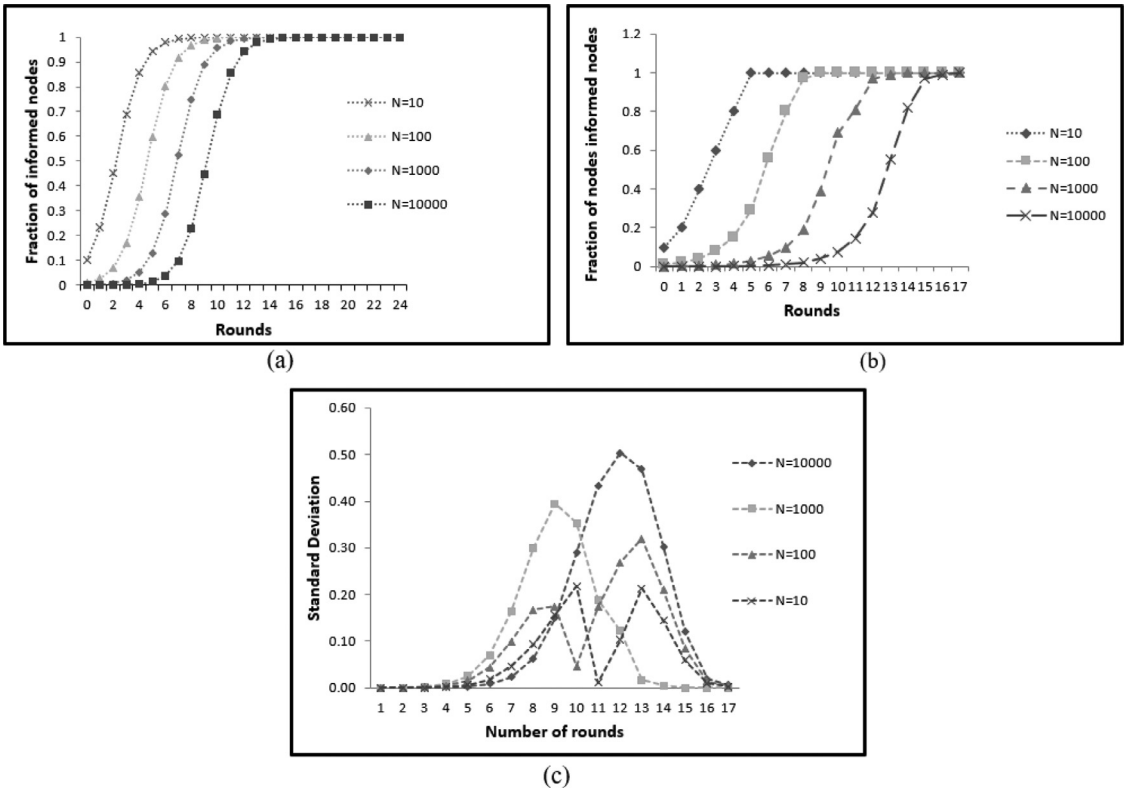


**Fig. 1.** Number of rounds required for each routing methods for different number of nodes in (a) Random geometric (b) Grid (c) Scale –free Network.

degree of the receiver node. The transition probability of a neighbouring node is higher if it has a higher degree than all other neighbours. Fig. 2 of Appendix C shows the variation in spectral gap with respect to the variation in the number of nodes.

In Fig. 1 we illustrate the communication overhead of the proposed expander routing method with random routing and receiver-initiated routing for three different types of network topologies. The final results are obtained from 100 experiments for each size of the network for each network topology and taking the average of the results. From the figures, we see that for networks with good expansion property like a random geometric graph and scale-free graph, using our proposed expander routing method requires less number of rounds compared to other routing methods.

We now compare the message complexity of our proposed method with the other two routing methods. From Fig. 2 we can see that the number of messages required for our proposed method is less than the other two routing methods. This is in accordance with Theorem 7 which states that the message complexity of our proposed method is O (m log m).

### *6.2. Performance of proposed GBFTLB method in intra cluster and inter cluster load balancing*

We measure the performance of our algorithm under different load scenarios of the network [13,14]. The main goal of the load balancing algorithm is to reduce the fluctuations in load of the network as much as possible. The figures Fig. 3(a)–

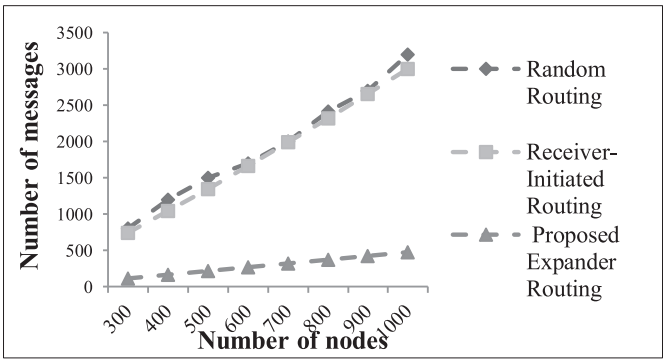


**Fig. 2.** Number of messages required for each routing protocol for different number of nodes.

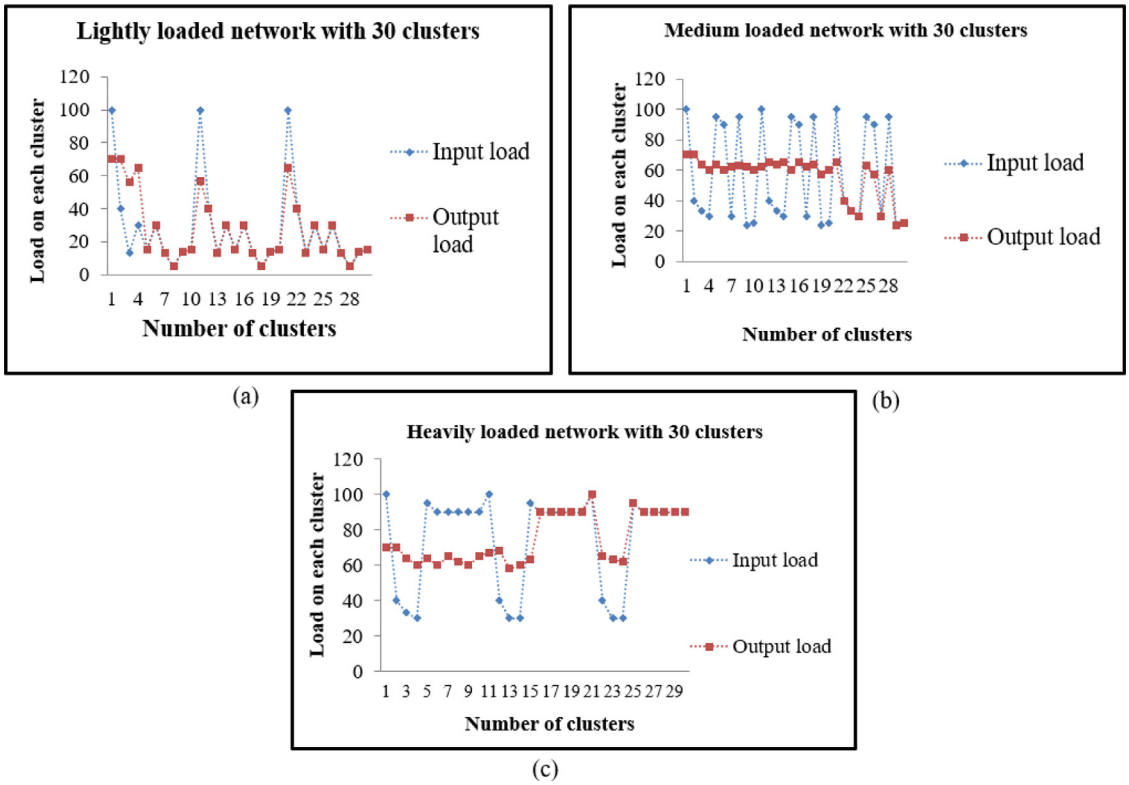


**Fig. 3.** Performance of GBFTLB in a network of 30 clusters in 3 scenarios in (a) Lightly overloaded network (b) Medium Loaded (c) Heavily loaded network.

(c) show the load information of the clusters in the initial state before applying the load balancing and in the final state after applying load balancing. The load information is represented using integer values.

In the lowly loaded system, all the highly loaded clusters give their excessive loads to other lowly loaded clusters and the system is properly balanced. In the medium loaded systems, all of the highly loaded clusters become medium loaded. In a highly loaded system more than half of the highly loaded clusters become medium loaded. A few clusters remain highly loaded as there are no low or medium loaded clusters in the network.

In Fig. 4, we consider the number of clusters that are overloaded before applying the algorithm and the number of clusters that remain overloaded after applying the algorithm. In the light and medium loaded systems, all the clusters get balanced after applying the algorithm. In the highly loaded system, some of the clusters remain overloaded after applying load balancing as there are no lightly loaded clusters left. We then perform simulations for different values of the workload of the cluster. Simulation results also follow our theoretical results.

We now consider the stability of a cluster with respect to the average response time. We calculate the stability threshold $\delta(t)$ at $t = 1, 2, 3$ taking $\alpha = 0.5$ for three different states of the cluster i.e.-overloaded, under-loaded and stable. We then calculate the distance D from Eq. (26). From Fig. 5 we can observe that for three different values of the average response

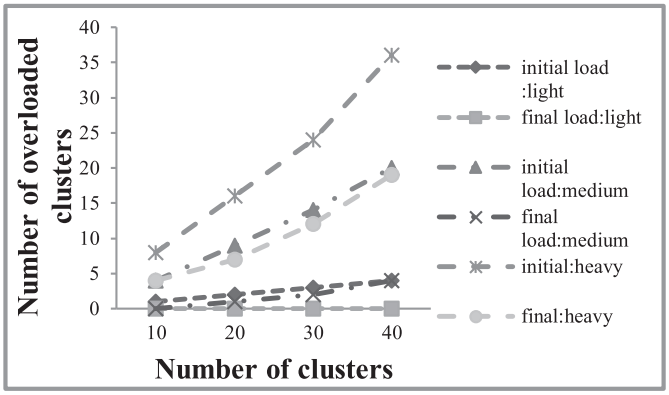


**Fig. 4.** Figure showing load variations before and after applying algorithm under different load conditions considering different number of clusters.

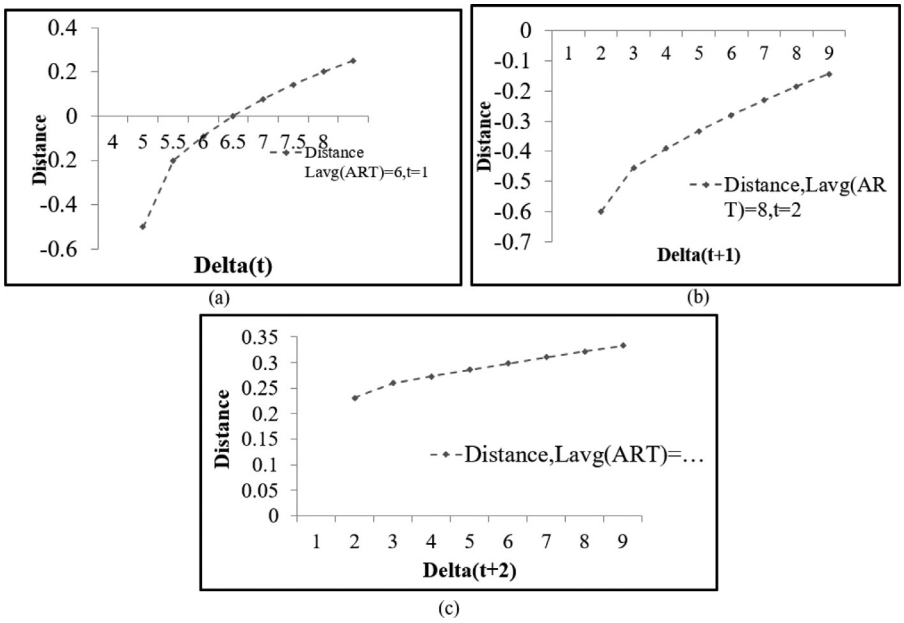


**Fig. 5.** Distance calculated for different values of stability threshold and average response time.

time of the cluster the distance varies according to $\delta(t)$. Fig. 5(a) shows that the distance is less than 0 for all values of $\delta(t)$ less than 6 which means the cluster becomes unstable when $\delta(t)$is less than the average response time is. Similarly in Fig. 5(b), the cluster is unstable as all values of $\delta(t)$ are less than average response time. In Fig. 5(c), the cluster is stable as all the distances are positive.

We now consider the change in Global Load deviation for changes in Cluster load deviation. For this simulation, we consider that all clusters are optimally balanced except one cluster, which has a load greater than the optimal load of the cluster. For different values of workload for each worker node, we obtain different values of CLD by applying intra-cluster load balancing. Using Eq. (19) $GLD^t$ is calculated for each value of CLD. This is shown as theoretical results in Fig. 6. We then perform simulations for different values of the workload of the cluster. Simulation results also follow our theoretical results.

Reducing the load of a system improves the response time of the jobs in the system. Fig. 7 considers the gain percentage of response times that are obtained by applying the load balancing algorithm on the clusters having varying number of functional nodes depending on the availability percentage. The gain percent is the highest for the system when the clusters have 100% functional nodes. For under load clusters the gain percent is zero as the clusters are already balanced and no load balancing is required. As the availability decreases, the gain in the response time also decreases, but the gain percent is always positive, which show that our algorithm performs well under network conditions having a large number of failed nodes.

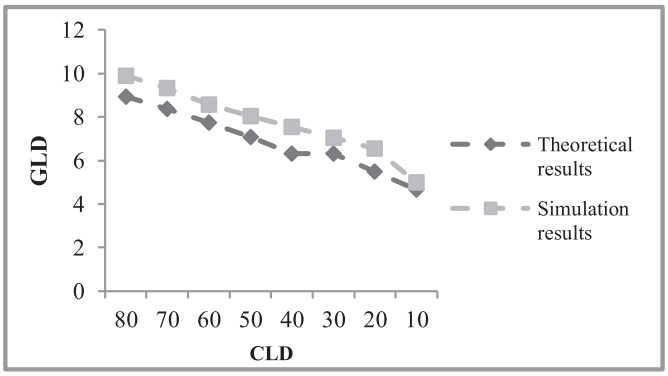


**Fig. 6.** Change in GLD with change in CLD of a single cluster.

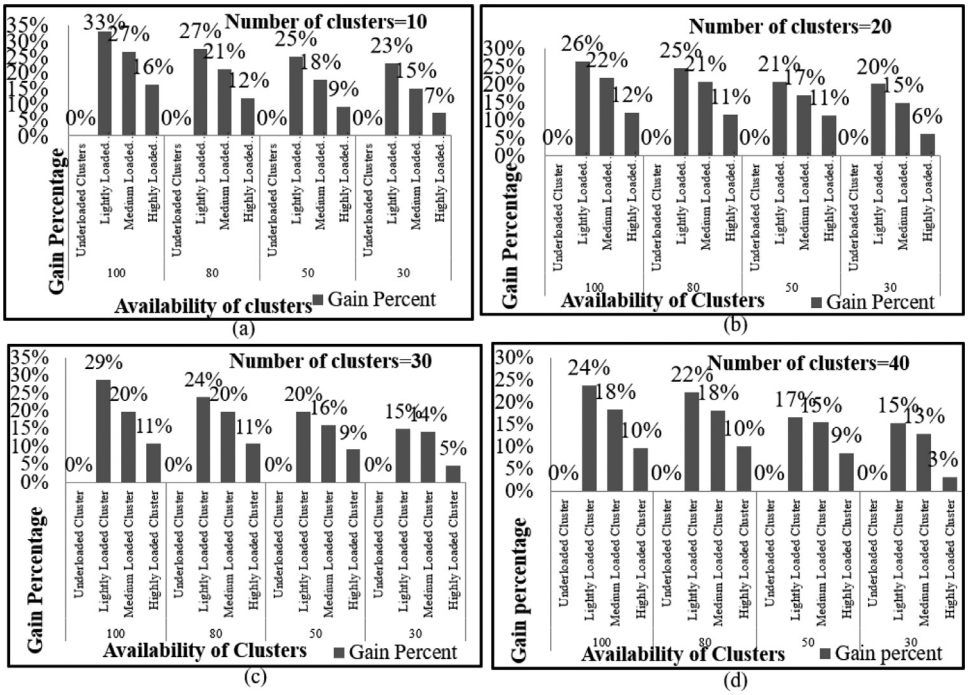


**Fig. 7.** Gain percent of the clusters according to their availability.

## 7. Conclusions

In this paper, we have presented a load balancing algorithm to solve the dynamic load balancing problem involving heterogeneous processors in a failure-prone network. The objective is to distribute the load among the processors, according to their processing speeds and availability. The network is partitioned into clusters and each cluster has a cluster head performing intra-cluster task transfers and also to communicate with other cluster heads across the network to perform global load balancing when needed. Global load balancing is performed using gossiping among the clusters where the clusters are connected in the form of an expander graph. The performance metrics that are considered in evaluating the algorithm are the cluster load deviation, global load deviation, average response time and the number of rounds required for inter-cluster load balancing. Our results prove that the proposed load balancing method using expander routing among the clusters is scalable and has low time and message complexities compared to other methods of routing. The relative performance of different methods of routing on different network topologies has been compared through simulations. Simulation results show that the expander based routing performs better than all other methods of routing for different network topologies. In addition, the performance of each cluster in the presence of varying degrees of failures is also considered and the results show that the cluster achieves a gain in response time even in the presence of failures.

## Declaration of Competing Interest

Authors have no conflict of interest to declare.

## Supplementary materials

Supplementary material associated with this article can be found, in the online version, at doi:10.1016/j.compeleceng.2019.106517.

## References

[1] Chatterjee M, Setua SK. A new clustered load balancing algorithm for distributed systems. In: IEEE Proceedings of the 2015 third international conference on computer. Communication, Control and Information Technology(C3IT); 2015.
[2] Hui C-C, Chanson ST. Hydrodynamic load balancing. IEEE Trans Parallel Distrib Syst 1999;10(11):1118–37.
[3] Dhakal S, Hayat MM, Pezoa JE, Yang C, Bader DA. Dynamic load balancing in distributed systems in the presence of delays:a regeneration-theory approach. IEEE Trans Parallel Distrib Syst 2007;18(4):485–97 Pg NoAPRIL.
[4] Dhakal S, Hayat MM, Pezoa JE, Abdallah CT, Doug Birdwell J, Chiasson J. Load balancing in the presence of random node failure and recovery. In: IEEE Proceedings 20th international parallel and distributed processing symposium; 2006.
[5] Ghanem J, Abdallah CT, Hayat MM, Chiasson J, Birdwell JD. Distributed load balancing in presence of node failure and network delays. IFAC Proc 2005;38(1):124–9 Pg No..
[6] Shah D. Gossip algorithms foundations and trends in networking. Found Trends® Netw 2008;3(1):1125 2009 D. Shah. doi:10.1561/1300000014.
[7] Han K, Ravindran B, Jensen ED. Rtg-l: dependably scheduling real-time distributable threads in large-scale, unreliable networks. In: Proceedings of IEEE pacific rim international symposium on dependable computing (PRDC); 2007.
[8] Wijetunge U, Perreau S, Pollok A. Distributed stochastic routing optimization using expander graph theory. IEEE Australian communication theory workshop Aus CTW; 2011.
[9] Kasprzyk R. Diffusion in networks. J Telecommun Inf Technol 2012 2/.
[10] Verma S, Ooi WT. Controlling gossip protocol infection pattern using adaptive fanout. In: Proceedings of the 25th IEEE international conference on distributed computing systems (ICSCS05); 2005.
[11] Doerr B, Friedrich T, Sauerwald T. Quasirandom rumor spreading: expanders, push vs pull, and robustness. Automata, languages and programming ICALP 2009. Lecture notes in computer science, 5555. Berlin, Heidelberg: Springer; 2009.
[12] Reiter, M.K., Samar, A., Wang, C., "Distributed construction of a fault-tolerant network from a tree", Proceedings of the 2005 24th IEEE symposium on reliable distributed systems (SRDS05).
[13] Chatterjee M, Mitra A, Roy S, Setua SK. Gossip based fault tolerant protocol in distributed transactional memory using quorum based replication system. Cluster Comput 2019. https://doi.org/10.1007/s10586-019-02973-7.
[14] Patel B, Roy S, Bhattacharyya D, Kim T-H. Necessity of big data and analytics for good e-governance. Int J Grid Distrib Comput 2017;10(8):11–20.
[15] Clusterdata: Wilkes 2011, clusterdata: Reiss 2011.

**Moumita Chatterjee** received her M.Sc. and M.Tech. from University of Calcutta Computer Science and Engineering. Currently she is a Ph.D. scholar in the Department of Computer Science and Engineering, University of Calcutta. Her research interests include Distributed Computing, Ad hoc Network, Network Security Routing, Clustering, Graph Theory etc. She has published many research papers in international conferences and journals.

**Anirban Mitra** received the Bachelor of Science degree in Computer Science, the M.Sc. degree in Computer science, and the M.Tech. degree in Computer Science Engineering from the University of Calcutta, India. He is pursuing his Ph.D. in the domain of Medical Image Processing from the same university. Since 2009, he has held Assistant Professor Positions at Academy of Technology, under MAKAUT, India, at the department of Computer Science and Engineering.

**Sanjit Kumar Setua** received his B.Tech. and M.Tech. from University of Calcutta Computer Science and Engineering. He is currently Associate Professor in the University of Calcutta. His research interests include Network Security, Sensor Network, Ad hoc Network, Routing, Clustering, Image Processing, Graph Theory, Distributed Computing, Cloud Computing, Database Security, DNA Computing, Big Data, Cliford Algebra etc. He has published many research papers in international journals and conferences.

**Sudipta Roy** is working at Washington University in St. Louis, MO, USA. He has received his Ph.D. from the Department of Computer Science and Engineering, University of Calcutta. He is serving as an Associate Editor of IEEE Access, IEEE and International Journal of Computer Vision and Image Processing (IJCVIP), IGI Global Journal. His fields of research interests are biomedical image analysis, image processing, steganography, artificial intelligence, big data analysis, machine learning and big data technologies.