A Hybrid Dynamic Load Balancing Algorithm for Distributed Systems

Mayuri A. Mehta
Sarvajanik College of Engineering and Technology, Department of Computer Engineering, Surat, India
Email: mayuri.mehta@scet.ac.in

Devesh C. Jinwala
S. V. National Institute of Technology, Department of Computer Engineering, Surat, India
Email: dcjinwala@acm.org

***Abstract*—Dynamic load balancing is essential for improving the overall utilization of resources and in turn to improve the system performance. In this paper, we propose a novel hybrid dynamic load balancing algorithm. We discuss our efforts on empirical evaluation of the same and justify its effectiveness in a typical distributed setup. Addressing the key issues in the design of such an algorithm, we also propose two new algorithms for supernode selection in a cluster. Further, we analyze the performance of algorithm under different cluster configurations, different load scenarios, and different network topologies. Our experimental results show that the hybrid algorithm potentially outperforms the classical centralized and decentralized approaches for the design of a load balancing algorithm.**

***Index Terms*—dynamic load balancing, distributed system, cluster, cluster head**

I. Introduction

Dynamic Load Balancing (DLB) has emerged as one of the most important techniques in solving the problem of performance of heterogeneous distributed systems. With ever-increasing network traffic, DLB can achieve improved performance in distributed systems to cope with fluctuating workload [1]. Dynamic Load Balancing can be broadly viewed to be either following centralized or decentralized approaches [2],[3],[4]. The centralized approach is simple in terms of implementation and overhead. However, if the central load balancing unit (or the coordinator) fails, the scheduling in a system would cease [5],[6],[7]. Further, it does not scale up well as the coordinator can become a performance bottleneck [8]. On the other hand, in decentralized approach, typically all nodes participate in load balancing [1],[5],[9],[10],[11],[12]. Though this approach performs better for large sized, heterogeneous systems, it entails increased communication overhead. Consequently, the design of an effective hybrid dynamic load balancing algorithm involves critical tradeoffs that is expected to overcome the drawbacks of centralized and decentralized approaches.

Numerous DLB algorithms have been proposed in the literature. Though the majority of existing algorithms provide the enhanced performance for distributed systems; inherent by design, they are constrained to a specific targeted system environment [1],[9],[10],[13],[14],[15]. Several algorithms are designed specifically for the applications having CPU-bound jobs whereas the others are designed for applications having memory-bound OR I/O-bound jobs. In addition, these algorithms are mutually exclusive in nature [7],[12],[16],[17],[18].

Thus, in general, the spectrum of the applicability of the existing DLB algorithms turns out to be limited. We believe that a DLB algorithm must absorb in its design, all the design issues involved such as system and task heterogeneity, target applications, load measurement parameters, components (of DLB algorithm), and evaluation parameters. We discuss these issues in our earlier work [19].

The hybrid DLB algorithm that we propose here, sits between the centralized and decentralized approaches. As shown in Fig. 1, utilizing the established notion of divide-and-conquer, the hybrid DLB algorithm partitions the nodes of distributed system into virtual groups, called

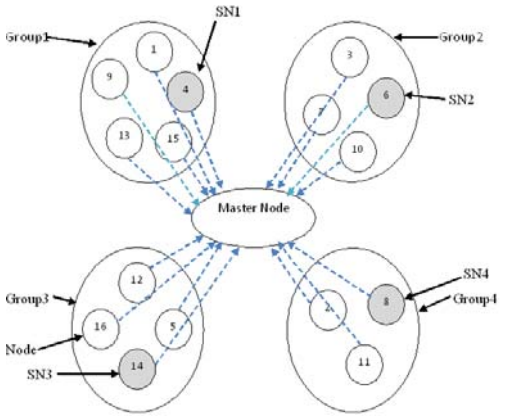


Figure 1. View of clusters in simplified distributed system [19].

clusters. Clustering we believe, is useful in improving the scalability and in reducing the communication overhead significantly. In each cluster, one node is designated as the SuperNode (SN) to define the dynamic threshold value

doi:10.4304/jcp.9.8.1825-1833

periodically. Communication between clusters is possible via a Central Master Node (CMN). When any node becomes overloaded, it first searches for a lightly loaded node in its cluster in a decentralized fashion. If it does not find a lightly loaded node in the same cluster, its overload will be transferred to a lightly loaded node in another cluster with the help of CMN. Consideration of the design issues, as in [19], makes our hybrid algorithm very promising in a wide range of environments and applications.

Having proposed earlier, a proof of concept of a hybrid DLB algorithm along with its theoretical analysis that shows the reduction in communication overhead in [19]; we present here two vital issues concerning the *clustering* and the *supernode selection*. In this paper, we propose two novel algorithms for supernode selection in each cluster based on our study on clustering algorithms [20-32]. Further, we show the effectiveness of the hybrid DLB algorithm by evaluating its performance under different *cluster configurations*, different *load scenarios*, and different *network topologies*. Our experimental results show that the hybrid algorithm performs faster than the classical centralized and decentralized algorithms. Based on the same, we argue that it is largely suitable for the heterogeneous distributed systems.

The remainder of the paper is organized as follows. In section II, we present some related work. Section III describes the system model. In section IV, we address the primary issues of hybrid algorithm that are crucial to arrive at the novel design. These issues are clustering and supernode selection. Experimental results and analysis are presented in section V. Finally, our conclusions and some future work are specified in section VI.

II. Related Work

Dynamic load balancing algorithms have been widely addressed in the literature. An adaptive decentralized sender-initiated load balancing algorithm that utilizes the load estimation approach is presented in [1]. In [9], authors have proposed an efficient mechanism for updating the state information quickly. This mechanism is further utilized to improve the average task completion time. The major drawback of these algorithms is that they restrict the job migration limit that may result in imbalanced system workload and degraded performance.

In [13], the content based DLB using multi parameters has been proposed. The DLB algorithms for multi-user jobs are presented in [14]. Authors have shown that at low communication overheads, dynamic algorithms perform better than static algorithms, however, as the overheads increase, dynamic algorithms perform similar to that of the static algorithms. A sender-initiated decentralized DLB algorithm is presented based on optimal one-shot load balancing strategy in [10]. In this algorithm, only the receiver node autonomously executes the load balancing every time an external load arrives at the node. The centralized DLB algorithms that suffer from scalability problem are proposed in [5],[15]. The two-level centralized scheduling model for dynamic load balancing in grid is proposed in [33].

A DLB algorithm with new load measurement policy is proposed in [12],[16] for efficient load balancing in homogeneous system. This algorithm provides enhanced performance for CPU-bound, network-bound, and memory-bound applications. An I/O-intensive DLB algorithm that provides higher performance for homogeneous as well as heterogeneous system is described in [17]. An application independent algorithm is presented and analyzed in [18]. The influence and quality of several load indices on the performance of a dynamic load balancing are empirically evaluated in [34],[35].

Although these existing algorithms attempt to enhance the performance, some algorithms are effective only for homogeneous distributed systems [12],[16]. Though some of these algorithms consider parameters other than CPU utilization or CPU queue length to measure the load of the node [1],[9],[12],[16],[17],[18], none of them provide equivalent performance for all kind of applications, namely CPU-bound, memory-bound, and I/O bound. Moreover, very few algorithms aim to cut down the communication overhead [10],[9],[14]. The key weakness of these existing algorithms is that none of them considers all the significant design issues, as in [19], to generate a competent DLB algorithm as we do to design the hybrid DLB algorithm. Consideration of these design issues makes our hybrid algorithm capable and applicable in heterogeneous distributed environments and to a wide range of applications. In general, the benefits from hybrid DLB algorithm are as follows. It

- overcomes the scalability issue of centralized approach
- minimizes the communication overhead that is incurred by decentralized approach
- considers heterogeneity in computing resources; hence, it is applicable to homogeneous as well as heterogeneous distributed systems
- aids highly loaded node to obtain lightly loaded node in lesser time
- utilizes effective information policy and location policy [36]
- utilizes efficient load measurement policy
- is applicable to a wide range of distributed applications such as CPU-bound, memory-bound, I/O-bound, and various combinations of these

III. System Model

Before discussing the system model, we first describe the notations which are used throughout the paper in Table I.

The distributed system consists of a large number of geographically dispersed heterogeneous computing resources, namely P_1, P_2,…, P_n, and a diverse set of users that are connected using links and routers. We assume that there are maximum n nodes in the system. Further,

TABLE I.
LIST OF NOTATIONS

Notation	Description
n	Total number of nodes in the system
P_i where $1 \le i \le n$	i^{th} node of the system
C_i	Processor load of P_i
M_i	Memory load of P_i
IO_i	I/O load of P_i
L_i	Total load of the node P_i
m	Total number of clusters
g_j	j^{th} cluster, where $1 \le j \le m$
T_j	Threshold value of the cluster g_j
k_j	Size of the cluster g_j

no assumptions are made about the underlying network topology. We consider the following properties of the distributed system.

- We assume that each computing resource (or node) has an infinite capacity buffer, just to eliminate the possibility of discarding the job due to unavailability of buffer space [1].
- Nodes are heterogeneous with respect to baud rate, propagation delay, number of machines, cost of processing, processing speed (in terms of the ratings of standard benchmark Million Instructions Per Seconds-MIPS), operating system, memory capacity, and I/O capacity. Thus, the computation and communication capabilities are different for all nodes in the system.
- The load L_i of each node P_i is computed using the formula $L_i = C_i + M_i + IO_i$. The status of each node is defined as follows.
  $L_i = 0 \rightarrow$ node is idle
  $L_i \le T_j \rightarrow$ node is referred as lightly loaded node
  $L_i > T_j \rightarrow$ node is referred as highly loaded node
- Threshold T_j is defined as the average load of the cluster and is computed using the following equation. For cluster g_j,
  $$T_j = (\sum_{i=1}^{k_j} L_i) / k_j, \text{ where } 1 \le j \le m$$
- Initially, all nodes are assigned some amount of work and no node in the system is idle. During load balancing operation, the workload of the node may increase or decrease depending on its current status.
- Users generate different numbers of gridlets (or tasks) that are submitted to nodes for execution. A gridlet is a package that contains all the information related to the job and its execution management details [37].
- Users differ from each other with respect to number and sizes of gridlets, baud rate, propagation delay, and MTU (Maximum Transmission Unit).
- We consider merely nonpreemptive task transfers.

IV. THE CRITICAL DESIGN ISSUES OF THE HYBRID DLB ALGORITHM

The hybrid DLB algorithm utilizes the established notion of clustering because the clustering is an effective technique for decreasing the message complexity and increasing the scalability in large scale distributed system. In this approach, the geographically dispersed nodes of the system are divided into m disjoint virtual clusters. Once the clusters are created, one node in each cluster is selected as the supernode. The supernode is assigned a special responsibility of defining the dynamic threshold value periodically. In the design of hybrid algorithm, we consider a genuine dynamic threshold policy to facilitate the practical situations of the distributed systems. The preliminary version of hybrid algorithm is presented in [19]. At this juncture, we address the following two imperative issues related to the design of hybrid DLB algorithm: cluster formation and supernode selection in each cluster. Specifically, we describe the logical criteria to form the clusters based on the classical theory of integer partition. We also propose the two novel algorithms for supernode selection. The proposed algorithms help us to arrive at the design of an effective DLB algorithm.

A. Cluster Formation

In this section, we describe the issue of clustering formally. In favor of hybrid DLB algorithm, the key objectives of clustering are improved scalability and reduced communication overhead.

Let $N = \{P_1, P_2, P_3, \ldots, P_n\}$ is a distributed system. We assume that N is a nonempty finite set and its order is not large. The clustering procedure, as described in Fig. 2, creates the m groups, namely $g_1, g_2, \ldots, g_m$, such that

$g_j \neq \Phi$ for $j = 1, 2, \ldots, m$ where $1 < m < n/2$
$g_j \cap g_l = \Phi$ for $j, l = 1, 2, \ldots, m$ where $j \neq l$
$g_1 \cup g_2 \cup \ldots\ldots \cup g_m = N$ that is $\bigcup_{j=1}^{m} g_j = N$

In each cluster, there must be minimum 2 nodes to validate the significance of group. A cluster can have maximum n/2 nodes within it.

To form clusters, first all partitions of n are generated. From amongst the all partitions, the partitions that have 1

```
1.  Generate all partitions of integer n
2.  Remove the partitions that have 1 or n as a part
3.  From the remaining partitions, choose one
    partition pt randomly
4.  Set m = number of parts in pt
5.  FOR j = 1 to m
        Set k_j = j^th part in pt
// Following are the steps to insert nodes in clusters
6.  Create sorted list SL by sorting the system nodes
    based on their load value
7.  FOR j = 1 to m
        Set g_j = Φ
8.  start = 0, end = n, flag = 1
9.  FOR j = 1 to m
        WHILE (k_j > 0)
            IF (flag = 1)
                pos = start; start = start + 1; flag = 0
            ELSE
                pos = end; end = end – 1; flag = 1
            g_j = g_j ∪ SL[pos]
            k_j = k_j - 1
```

Figure 2. Algorithm for cluster formation.

or n as a part are removed because there can not be a cluster of 1 node and the partition having n as a part represents the distributed system itself. From the remaining partitions, one partition *pt* is chosen randomly to set the number and sizes of the clusters. For instance, if *pt* is 5+4+6 for the integer number n=15, then the number of clusters m=3 and the sizes of clusters g_1, g_2, and g_3 are k_1=5, k_2=4 and k_3=6 respectively. Once the number and sizes of clusters are set, the obvious query that arises is which nodes should be inserted in which cluster? To cope with this query, we create the sorted list SL by arranging the nodes of distributed system into increasing order of their load value. Subsequently, the nodes are inserted in each cluster one by one from top and bottom of the SL alternately, while the next cluster will continue the same series. The key objective of inserting the nodes from top and bottom of the SL is to create the averagely load clusters initially. This is so because load balancing can be achieved efficiently in averagely loaded clusters which comprise of several highly loaded nodes and several lightly loaded nodes.

B. The Proposed Algorithms for Supernode Selection

Each cluster has a supernode that periodically collects the load information of the other nodes in the cluster to compute the average cluster load. This average load is set as the new threshold value and is broadcast to other nodes in the cluster.

Selecting a supernode is a significant design issue for the hybrid algorithm. Earlier we proposed two Supernode Selection Algorithms (SSA), as in [38], based on the classical Leader Election Algorithms (LEA). In SSA1, highest id node is selected as the supernode. Though this is the traditional approach, SSA1 significantly reduces the communication overhead compared to existing LEA. It takes O(k) messages if the highest id node is alive. SSA1 may not perform successfully if the highest id node chosen as the supernode is already highly loaded. This is so because if the highest id node is assigned an additional responsibility of defining the threshold value periodically, its performance may degrade. To deal with this issue, we proposed SSA2. SSA2 selects the *average-valued-node* as the supernode. Average-valued-node is the node that has the mean load value to other nodes in the cluster. Unlike SSA1, SSA2 attempts to choose the supernode that is neither highly loaded nor lightly loaded. Additionally, it achieves this task utilizing O(k) messages.

SSA1 and SSA2 select the supernode considering a single objective that can lead to poor performance. Therefore, we propose two new supernode selection algorithms considering multi-criterion optimization [39]. Design of these algorithms is motivated by the existing clustering algorithms [20-32]. In the existing clustering algorithms, local network structure is used as the basis for cluster head selection. Hence, the nodes that are geographically close to each other, typically, form a cluster. An apparent limitation of such a scheme, that is oblivious of the eventual load after cluster formation, is probably unevenly loaded clusters. Hence, the existing algorithms for cluster formation are not applicable in our hybrid DLB algorithm. Contrary to fundamental concept of existing algorithms, in our hybrid approach, global network structure is used to form each cluster. Subsequently, a node is chosen as the supernode in each cluster based on certain criterion. In some of the existing algorithms, cluster heads are selected either randomly with a certain probability [26],[27] or based on node id [30],[31]. It is likely that randomly selected or node id based cluster head has higher load. If so, re-election will occur in order to select the new cluster head. As a consequence, the communication overhead will increase and the system performance will decrease. Thus, these algorithms are also not useful for supernode selection in hybrid algorithm. Rather than selecting the supernode randomly or based on node id, factors like resources' utilization, distance from other nodes in the cluster, number of neighbors, etc. may be of importance when selecting the supernode [20],[21],[24],[28].

1) Supernode Selection Algorithm_3 (SSA3)

SSA3 is derived from the algorithm presented in [20]. Unlike algorithm presented in [20], it considers the CPU utilization, memory utilization, and I/O utilization to show the node's ability to be elected as the supernode. As shown in Fig. 3, first the Static Threshold Value (STV) is computed. STV is an average load of the cluster considering the current CPU utilization of the nodes. Then, the NodeImportanceIndex NIv() of each node in the cluster is calculated. For simplicity we have considered merely CPU utilization as the node importance value. Then after, all the nodes are sorted in ascending order of their NIv(). The nodes of sorted list are examined one by one and if the currently examined node has CPU utilization below STV, the node is designated as the candidate supernode. Subsequently, all the candidate supernodes are arranged in increasing order of their MIOload which is a linear combination of memory load and I/O load. From this sorted list, the node with minimum MIOload that has CPU utilization below

Input: load of all nodes in the cluster
Output: supernode
//procedure utilized by the node P_i that does not find the current threshold value

Supernode Selection Algorithm_3()
1. Calculate STV
2. Calculate NodeImportanceIndex NIv() of each node in the cluster
   NIv (P_i) = C_i, where $1 \le i \le k$
3. Sort nodes of cluster in increasing order of their NIv()
4. Take out the nodes whose NIv() < STV and designate them as the candidate supernodes
5. Arrange candidate supernodes in increasing order of their MIOload
6. Supernode = candidate supernode with minimum MIOload

Figure 3. Pseudo code for supernode selection algorithm_3.

STV is selected as the supernode. In case of tie, the node with higher id prevails.

It is noticeable that in this algorithm, the node with lesser resource utilization has greater opportunity to be selected as the supernode. Unlike SSA1 and SSA2, SSA3 prevents the performance degradation of a specific node by rotating the responsibility of supernode amongst the cluster nodes. It also takes O(k) messages to select the supernode.

2) Supernode Selection Algorithm_4 (SSA4)

The design of SSA4 is closely related to the multi criterion optimization technique presented in [21]. SSA4 attempts to find out the supernode whose resource utilization is optimal. Fig. 4 shows the major steps of SSA4.

The detailed explanation of the SSA4 is as follows. The option matrix OM represents the resource utilization of all the nodes of the cluster. Specifically, each row of OM represents the utilization of various resources of a node. There are three columns in OM because we consider three parameters, namely CPU utilization, memory utilization, and I/O utilization, as the decision making parameters. Thus, each element $x_{i,j}$ represents the j^{th} parameter for the i^{th} node. Subsequently, the OM is transformed into decision matrix DM utilizing the formula given for $y_{i,j}$. In this formula, MAX_j represents the worst value of j^{th} parameter and MIN_j represents the best value of j^{th} parameter. These best and worst values are unique to each decision making parameter. For instance, the best value for CPU utilization is represented by the minimum value and the worst value for it is represented by the maximum value in the column of OM. Next, the preference vector is computed using the following operation that is inspired by the preference function modeling [40].

$$S(P_i) = \frac{\sum_{j=1}^{3} x_{i,j}}{MaxL}$$

For each node P_i, this function accepts the values for

1. Build Option Matrix $(OM)_{kX3}$, where k is the size of cluster.
$$OM = \begin{bmatrix} x_{1,1} & x_{1,2} & x_{1,3} \\ x_{2,1} & x_{2,2} & x_{2,3} \\ \vdots & \vdots & \vdots \\ x_{k,1} & x_{k,2} & x_{k,3} \end{bmatrix}$$
2. Convert option matrix into Decision Matrix $(DM)_{kX3}$.
$$DM = \begin{bmatrix} y_{1,1} & y_{1,2} & y_{1,3} \\ y_{2,1} & y_{2,2} & y_{2,3} \\ \vdots & \vdots & \vdots \\ y_{k,1} & y_{k,2} & y_{k,3} \end{bmatrix}$$
where $y_{i,j} = 2\,\frac{(x_{i,j} - MAX_j)}{(MIN_j - MAX_j)} - 1$
3. Compute Weight vector $(W)_{kX1}$ by multiplying the decision matrix with the preference vector.
$$\begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_k \end{bmatrix} = \begin{bmatrix} y_{1,1} & y_{1,2} & y_{1,3} \\ y_{2,1} & y_{2,2} & y_{2,3} \\ \vdots & \vdots & \vdots \\ y_{k,1} & y_{k,2} & y_{k,3} \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_k \end{bmatrix}$$

Figure 4. Pseudo code for supernode selection algorithm_4.

each of the parameters involved in the decision process and returns a scaled value. The returned value is scaled between 0 and 1, where 0 represents the best value and 1 represents the worst value. MaxL represents the maximum possible load of the node. As we have considered CPU utilization, memory utilization, and I/O utilization as the decision making parameters, the value of MaxL is 300 which is summation of maximum possible value for each of these parameters. After computation of preference vector, the DM is multiplied by the preference vector to obtain the weight vector W. The weight vector represents the weight of the each of the available choices. In particular, it represents the weight of each node in the cluster. Finally, the node corresponding to maximum weight is selected as the supernode because it indicates the best choice, that is, it indicates the node with optimal resource utilization. Although, we have considered CPU utilization, memory utilization, and I/O utilization as the decision making parameters, more metrics can be utilized to make the finest choice. The key benefit from SSA4 is that it too takes merely O(k) messages to select the optimal supernode.

V. Performance Results and Analysis

We evaluate the performance of our hybrid DLB algorithm using simulation. Simulation is an approach that is useful to setup an appropriate distributed environment in which the empirical evaluation can be carried out. It is practical approach to analyze the DLB algorithm and to reason about the behavior of DLB algorithm.

A. Experimental Setup

We use GridSim toolkit to simulate a distributed system that comprises of 50 nodes and 10 users. Nodes and users are connected via links and routers under hybrid network topology. Nodes are created with greater heterogeneity levels to show the impact of heterogeneity on performance. Users generate gridlets that are submitted to nodes for execution. There are several parameters to measure the performance of DLB algorithms. We use Average Response Time (ART), Average Round Trip Time (ARTT) in seconds, and Average Completion Time (ACT) as parameters for performance comparison. As it is not feasible to show the performance of all users individually, we compute average value for each of the performance parameters. Response time is defined as the time at which the response of first gridlet is received. Round trip time is defined as the total time the gridlet has spent in the network. Completion time is defined as the elapsed time between the submission of the first gridlet and the completion of the last gridlet. It is overall completion time per user. We consider the classical Centralized Algorithm (CA), as in [33], and the DeCentralized Algorithm (DCA), as in [9], as the base algorithms for the comparison of hybrid DLB algorithm. For complete comparison, here we show the results of hybrid algorithm considering all four supernode selection approaches. The performance of Hybrid algorithm utilizing SSA1, SSA2,

SSA3, and SSA4 is shown as HSSA1, HSSA2, HSSA3, and HSSA4 respectively in graphs.

B. Performance under Different Cluster Configurations

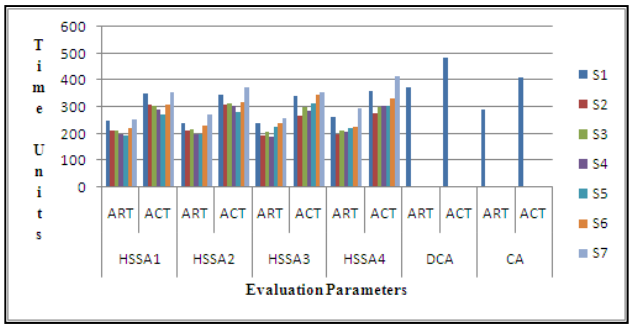
In this experiment, we analyze the performance of hybrid algorithm considering different number and sizes of the clusters. Specifically, we intend to test the performance for following cluster configurations.

- small number of clusters where size of each cluster is large
- large number of clusters where size of each cluster is small
- moderate number of clusters where size of each cluster is moderate

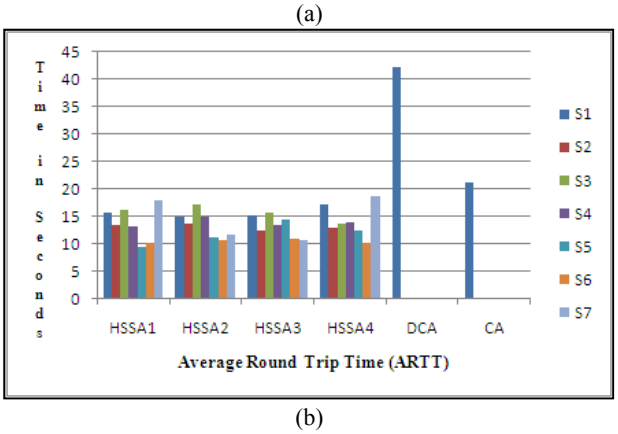
We have conducted experiments for cluster configurations of Table II. The readings of ART and ACT for various scenarios are depicted in Fig. 5(a) and the readings of ARTT for various scenarios are depicted in Fig. 5(b). It is observed that utilizing any of the SSAs, the hybrid algorithm outperforms the centralized algorithm and the decentralized algorithm irrespective of the number and sizes of the clusters. Under different cluster configurations, the performance improvement over DCA ranges from 21% to 49% for ART, 56% to 78% for ARTT, and 14% to 45% for ACT. Similarly, the performance improvement over CA ranges from 6% to 35% for ART, 12% to 56% for ARTT, and 10% to 35% for ACT.

TABLE II.
DIFFERENT CLUSTER CONFIGURATIONS

Scenario	Number of clusters	Cluster sizes
S1	2	25 25
S2	3	17 17 16
S3	3	20 20 10
S4	4	16 16 10 8
S5	8	7 7 7 7 7 7 5 3
S6	11	9 9 5 5 4 3 3 3 3 3 3
S7	14	5 4 4 4 4 4 4 3 3 3 3 3 3 3



(a)



(b)

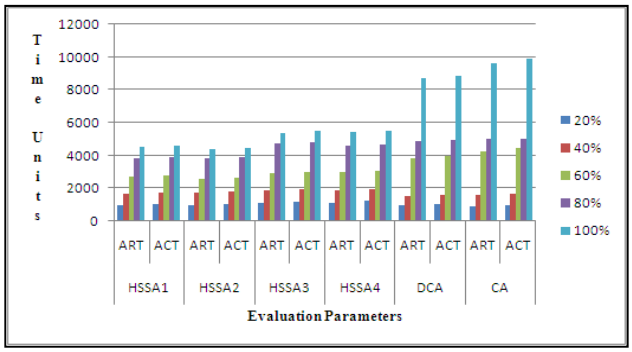
Figure 5. Performance of hybrid algorithm under different cluster configurations.

Further, we observe the readings to find out the SSA that is more suitable to a particular type of cluster configuration. Specifically, we observe that which SSA is more suitable to small, moderate, and large size clusters. It has been observed that when the clusters are moderate or large in size, all SSAs produce the improved performance that is nearly equivalent. However, when the clusters are small in size, SSA1 gives better performance amongst the all SSAs. This is so because the probability of selecting the lightly loaded node as the supernode from a small cluster is more than the probability of selecting the same from a large cluster. Thus, when the number of clusters is higher and the size of each cluster is small, it is highly probable that a lightly loaded node will be selected as the supernode. It is also noticed that the SSA1 has lesser communication overhead compared to other SSAs because it selects the supernode in random fashion. However, the later reason alone is not responsible for providing the better performance as it can be observed from the readings of large size clusters. Thus, when the lesser communication overhead incurred by the SSA1 is combined with the small cluster size, the improved performance is achieved.

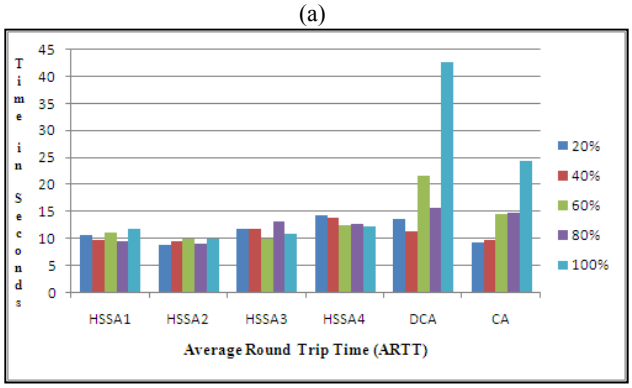
C. Performance under Different Load Scenarios

We have identified three different system states that are lightly loaded, moderately loaded, and highly loaded. To evaluate the performance of hybrid DLB algorithm under these system states, we have tested its performance under five system load set-ups: 20%, 40%, 60%, 80%, and 100%. From implementation point of view, 100% load is defined as the 1000 jobs in the system. The system with 80% to 100% load is defined as the highly loaded system while a system with 0% to 40% load is defined as the lightly loaded system. Accordingly, the system with 40% to 80% load is defined as the moderately loaded system. This experiment is conducted using the moderate number and sizes of the clusters. In particular, we have created 8 clusters with sizes 8, 7, 7, 6, 6, 6, 5, and 5 respectively. The readings of ART and ACT are shown in Fig. 6(a) and the readings of ARTT are shown in Fig. 6(b).

Experimental results show that when the system is lightly loaded, the hybrid algorithm gives marginally improved performance that is almost similar to that of the centralized algorithm or decentralized algorithm. This is so because when the system is lightly loaded, the majority of the system nodes are lightly loaded and very few nodes are highly loaded. Therefore, whether it is a cluster or an entire system, the communication overhead to find out the lightly loaded node will be logically less in lightly loaded system state compared to the communication overhead incurred in moderately loaded or highly loaded system state. Moreover, in lightly loaded

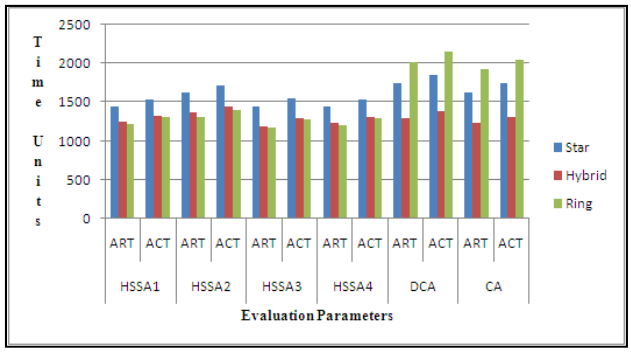


(a)

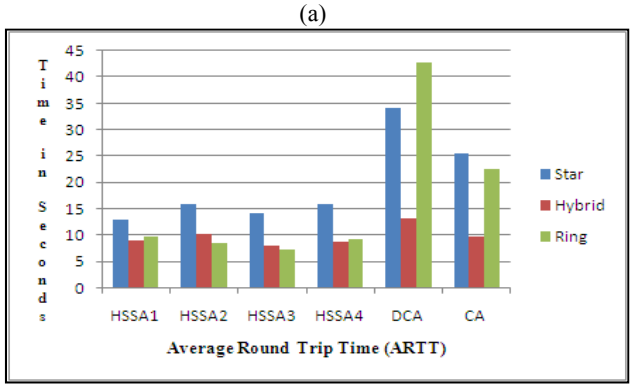


(b)

Figure 6. Performance of hybrid algorithm under different load scenarios.



(a)



(b)

Figure 7. Performance of hybrid algorithm under different topologies.

system state, the queuing delays at the destination node and the network delays are also less irrespective of a cluster or an entire system compared to the delays that occur in moderately loaded or highly loaded system. Thus, the performance provided by the hybrid algorithm is nearly similar to that of the centralized algorithm or decentralized algorithm when the system is lightly loaded. However, with increase in system workload, the performance provided by the hybrid algorithm significantly increases as compared to that of the centralized algorithm and decentralized algorithm. The performance improvement over DCA ranges from 3% to 49% for ART, 13% to 78% for ARTT, and 3% to 49% for ACT. Similarly, the performance improvement over CA ranges from 6% to 54% for ART, 4% to 60% for ARTT, and 4% to 55% for ACT.

D. Performance under Different Network Topologies

In this experiment, we analyze the performance of hybrid algorithm under three network topologies. Though the topology of nearly all distributed systems is hybrid, there are distributed systems with other topologies for various specific reasons. Hence, we analyze the performance of hybrid DLB algorithm under star, ring, and hybrid topologies. As shown in Fig. 7, the hybrid algorithm performs faster than the typical centralized algorithm and the decentralized algorithm irrespective of the underlying network topology. Under various network topologies, the performance improvement over DCA ranges from 4% to 49% for ART, 13% to 77% for ARTT, and 3% to 49% for ACT. Similarly, the performance improvement over CA ranges from 1% to 39% for ART, 8% to 68% for ARTT, and 1% to 38% for ACT. Thus, hybrid algorithm substantiates its adaptability to different network topologies.

VI. Conclusions

We have presented a novel framework for dynamic load balancing in distributed systems using hybrid approach. Our hybrid algorithm possesses three major features: an efficient load balancing approach that overcomes the several limitations of centralized and decentralized approaches, an effective load measurement policy to make the algorithm capable and applicable to a wide range of distributed applications, and a successful information policy that significantly reduces the communication overhead. We have shown that the hybrid algorithm actually overcomes the limitations of the centralized and decentralized approaches utilizing the notion of clustering. It performs competitively for heterogeneous distributed system.

The performance of hybrid algorithm is evaluated under different cluster configurations, different load scenarios, and different topologies. The experimental results show that the hybrid DLB algorithm potentially outperforms the traditional centralized and decentralized DLB algorithms in all situations. In particular, it provides significant improvement in response time, round trip time, and completion time compared to that of centralized and decentralized algorithms.

In our future work, we will examine the performance of hybrid algorithm with more number of nodes in distributed system. We also intend to analyze its performance with various real-time distributed applications. We will explore the other metrics and system scenarios for more complete comparison of

algorithms. Additionally, we aim to use a heuristic to choose the partition of n.

REFERENCES

[1] R. Shah, B. Veeravalli, and M. Misra, "On the design of adaptive and decentralized load balancing algorithms with load estimation for computational grid environment," *IEEE Tran. Parallel and Distributed Systems*, vol. 18, pp. 1675-1686, December 2007.

[2] D. Gupta, and P. Bepari, "Load sharing in distributed systems," in *Proc. National Workshop on Distributing Computing*, January 1999.

[3] R. Riedl, and L. Richter, "Classification of load distribution algorithms," in *4th Euromicro Workshop on Parallel and Distributed Processing*, pp. 404-413, 1996.

[4] J. Baikerikar, S. Surve, and S. Prabhu, "Comparison of load balancing algorithms in a grid," in *Int. Conf. Data Storage and Data Engineering (DSDE)*, IEEE Press, Bangalore, 2010, pp. 20-23.

[5] M. Paksoy, and J. Prado, "Comparing centralized and decentralized distributed execution systems," Internet: http://www.sccs.swarthmore.edu/users/07/mustpaks/distsys_paper.pdf, November 2012.

[6] I. Psoroulas, I. Anagnostopoulos, V. Loumos, and E. Kayafas, "A study of the parameters concerning load balancing algorithms," *Int. J. Computer Science and Network Security*, vol. 7, pp. 202-214, April 2007.

[7] P. K. Chandra, and B. Sahoo, "Prediction based dynamic load balancing techniques in heterogeneous clusters," in *Proc. Int. Conf. Computer Science and Technology*, California, 2010, pp. 189-192.

[8] R. Mukhopadhyay, D. Ghosh, and N. Mukherjee, "A study on the application of existing load balancing algorithms for large, dynamic, heterogeneous distributed systems," in *9th WSEAS Int. Conf. Software Engineering, Parallel and Distributed Systems (SEPADS)*, Cambridge, 2010, pp. 238--243.

[9] I. Al-Azzoni, and D. G. Down, "Decentralized load balancing for heterogeneous grids," in *Computation World: Future Computing, Service Computation, Cognitive, Adaptive, Content, Patterns*, IEEE Computer Society, Athens, 2009, pp. 545-550.

[10] S. Dhakal, M. M. Hayat, J. E. Pezoa, C. Yang, and D. A. Bader, "Dynamic load balancing in distributed systems in the presence of delays: a regeneration – theory approach," *IEEE Tran. Parallel and Distributed Systems*, vol. 18, pp. 485-497, April 2007.

[11] G. D. Fatta, and M. R. Berthold, "Decentralized load balancing for highly irregular search problems," *ACM J. Microprocessors and Microsystems*, vol. 31, pp. 273-281, June 2007.

[12] P. Werstein, H. Situ, and Z. Huang, "Load balancing in a cluster computing," in *7th Int. Conf. Parallel and Distributed Computing, Applications and Technologies (PDCAT)*, IEEE Computer Society, Higashi Hiroshima 2006, pp. 569-577.

[13] T. N. Anitha, and R. Balakrishna, "An efficient and scalable content based dynamic load balancing using multi-parameters on load aware distributed multi-cluster servers," *Int. J. Engg. Science and Tech.*, vol. 3, pp. 6401-6411, August 2011.

[14] S. Penmasta, and A. T. Chronopoulos, "Dynamic multi-user load balancing in distributed systems," in *21st IEEE Int. Parallel and Distributed Processing Symp.*, IEEE Press, California, 2007, pp. 1-10.

[15] P. Jain, and D. Gupta, "An algorithm for dynamic load balancing in distributed systems with multiple supporting nodes by exploiting the interrupt service," *Int. J. Recent Trends in Engg*., vol. 1, pp. 232-236, May 2009.

[16] M. V. Gopalachari, P. Sammulal, and A. V. Babu, "Correlating scheduling and load balancing to achieve optimal performance from a cluster," in IEEE *Int. Conf. Advance Computing*, IEEE Press, Patiala, 2009, pp. 320-325.

[17] X. Qin, H. Jiang, and A. Manzanares, "Dynamic load balancing for i/o-intensive applications on clusters," *ACM Trans. Storage*, vol. 5, pp. 9:1--9:38, November 2009.

[18] G. F. Kabbany, N. M. Wanas, N. H. Hegazi, and S. I. Shaheen, "A dynamic load balancing framework for real-time applications in message passing systems," *Int. J. Parallel Prog.*, vol. 39, pp. 143-182, 2011.

[19] M. Mehta, and D. Jinwala, "A hybrid dynamic load balancing algorithm for heterogeneous environments," in *Int. Conf. Grid Computing and Applications*, CSREA Press, Las Vegas, 2011, pp. 61-65.

[20] N. Dimokas, D. Kastsaros, and Y. Manolopoulos, "Energy-efficient distributed clustering in wireless sensor networks," *ACM J. Parallel and Distributed Computing*, vol. 70, pp. 371-383, April 2010.

[21] N. Aslam, W. Phillips, W. Robertson, and S. Sivakumar, "A multi-criterion optimization technique for energy efficient cluster formation in wireless sensor networks," *Special Issue on Information Fusion*, vol. 12, pp. 202-212, July 2011.

[22] A. Chamam, and S. Pierre, "A distributed energy-efficient clustering protocol for wireless sensor networks," *ACM J. Computers and Electrical Engg.*, vol. 36, pp. 303-312, March 2010.

[23] K. R. Bhakare, R. K. Krishna, and S. Bhakare, "An Energy-efficient grid based clustering technology for a wireless sensor network, *Int. J. Computer Applications*, vol. 39, pp. 24-28, February 2012.

[24] Z. Fan, and Z. Jin, "A multi-weight based clustering algorithm for wireless sensor networks, Internet: http://pe.org.pl/articles/2012/1b/4.pdf, November 2012.

[25] Y. Cao, and C. He, "A distributed clustering algorithm with an adaptive backoff strategy for wireless sensor networks," *IEICE Transactions*, vol. 89-B, pp. 609-613, February 2006.

[26] W. R. Heinzelman, A. Chandrakasan, and H. Balakrishnan, "Energy-efficient communication protocol for wireless microsensor networks," in *33rd Hawaii Int. Conf. System Sciences*, IEEE Computer Society, Washington, 2000, pp. 8020.

[27] O. Younis, and S. Fahmy, "Heed: a hybrid, energy-efficient, distributed clustering approach for ad-hoc sensor networks," *IEEE Tran. Mobile Computing*, vol. 3, pp. 366-369, October 2004.

[28] D. J. Dechene, A. E. Jardali, M. Luccini, and A. Sauer, "A survey of clustering algorithms for wireless sensor networks," *Project Report*, Department of Electrical and Computer Engineering, University of Western Ontario, Canada, 2006.

[29] S. R. Boselin Prabhu, and S. Sophia, "A survey of adaptive distributed clustering algorithms for wireless sensor networks," *Int. J. Comp. Science and Engg. Survey (IJCSES)*, vol. 2, no. 4, November 2011.

[30] D. J. Baker, and A. Epheremides, "The architectural organization of a mobile radio network via a distributed algorithm. *IEEE Tran. Communications*, vol. 29, pp. 1694-1701, November 1981.

[31] A. D. Amis, R. Prakash, T. H. P. Vuong, and D. T. Huynch, "Max-min d-cluster formation in wireless ad hoc networks," in *19th Annual Joint Conf. of the IEEE Computer and Communications Societies (INFOCOM)*, Dallas, 2000, pp. 32-41.

[32] B. Chen, K. Jamieson, H. Balakrishnan, and R. Morris, "Span: an energy-efficient coordination algorithm for topology maintenance in ad hoc wireless networks. *ACM J. Wireless Networks*, vol. 8, pp. 481-494, September 2002.

[33] H. Xiangchun, C. Duanjun, and C. Jing, "One centralized scheduling pattern for dynamic load balance in gird," in *Int. Forum on Information Technology and Applications*, IEEE Press, 2009, pp. 402-405.

[34] T. Kunz, "The influence of different workload descriptions on a heuristic load balancing scheme," *IEEE Transactions on Software Engineering*, vol. 17, pp. 725-730, July 1991.

[35] D. Ferrari, and S. Zhou, "An empirical investigation of load indices for load balancing applications," in *12th IFIP WG 7.3 Int. Symp. Computer Performance Modeling, Measurement and Evaluation*, North-Holland Publishing, Netherlands 1988, pp 515-528.

[36] M. A. Mehta, and D. C. Jinwala, "Analysis of significant components for designing an effective dynamic load balancing algorithm in distributed system," in *3rd IEEE Int. Conf. Intelligent Systems, Modeling and Simulation (ISMS 2012)*, IEEE Press, 2012, pp. 531-536.

[37] R. Buyya, and M. Murshed, "GridSim: a toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing," *J. Concurrency and Computation: Practice and Experience (CCPE)*, vol. 14, pp. 1175-1220, November 2002.

[38] M. A. Mehta, S. Agrawal, and D. C. Jinwala, "Novel algorithms for load balancing using hybrid approach in distributed systems," in *2nd IEEE Int. Conf. Parallel, Distributed and Grid Computing (PDGC 2012)*, IEEE Press, 2012, pp. 27-32.

[39] R. T. Marler, and J. S. Arora, "Survey of multi-objective optimization methods for engineering," *Structural and Multidisciplinary Optimization*, vol. 26, pp. 369-395, 2004.

[40] J. Barzilai, "Preference function modeling: the mathematical foundations of decision theory," in *Trends in Multiple Criteria Decision Analysis*, Springer, 2010, pp. 57-86.