Hindawi Publishing Corporation
Mathematical Problems in Engineering
Volume 2016, Article ID 5641831, 9 pages
http://dx.doi.org/10.1155/2016/5641831

Research Article

A Hierarchical Load Balancing Strategy Considering Communication Delay Overhead for Large Distributed Computing Systems

Jixiang Yang,[1] Ling Ling,[2] and Haibin Liu[3]

[1]*School of Mathematics and Statistics, Chongqing Jiaotong University, Chongqing 400074, China*
[2]*School of Materials Science and Engineering, Chongqing Jiaotong University, Chongqing 400074, China*
[3]*College of Business Administration, Hebei Normal University of Science & Technology, Qinhuangdao 066004, China*

Correspondence should be addressed to Jixiang Yang; jixiang_yang@126.com

Received 22 November 2015; Accepted 6 April 2016

Academic Editor: Veljko Milutinovic

Load balancing technology can effectively exploit potential enormous compute power available on distributed systems and achieve scalability. Communication delay overhead on distributed system, which is time-varying and is usually ignored or assumed to be deterministic for traditional load balancing strategies, can greatly degrade the load balancing performance. Considering communication delay overhead and its time-varying feature, a hierarchical load balancing strategy based on generalized neural network (HLBSGNN) is presented for large distributed systems. The novelty of the HLBSGNN is threefold: (1) the hierarchy with optimized communication is employed to reduce load balancing overhead for large distributed computing systems, (2) node computation rate and communication delay randomness imposed by the communication medium are considered, and (3) communication and migration overheads are optimized via forecasting delay. Comparisons with traditional strategies, such as centralized, distributed, and random delay strategies, indicate that the HLBSGNN is more effective and efficient.

1. Introduction

In traditional load balancing strategies, as the task grain size and the number of hops to traverse are likely to be relatively small, communication overhead between any pair of processors in computing system is commonly assumed to be nearly the same or ignored [1–4]. In large distributed systems, the topology diameter, task grain size, and data scale to traverse are likely big. For example, network topology diameter is commonly big and schedule objects are generally virtual machine resources, and the task grain size to traverse and data scale may be big in cloud computing environment [5, 6]. Transmission of such large-scale information can lead to big communication delay, which will be able to undoubtedly reduce the accuracy of scheduling strategies and incur the aging problem of obtained information.

In centralized load balancing strategies, a dedicated "central" computer gathers global information about the state of the entire system and uses it to make global load balancing decisions. Centralized strategies are inherently nonscalable and have the following limitations when applied to large distributed systems [7–10]: (1) collection of global information may become a prohibitively expensive process; (2) the memory usage for storing the global state information on the central node can be prohibitively high; (3) the single central node can become the communication bottleneck with a large number of processors; (4) the execution overhead of a centralized strategy's decision-making algorithm can be very high, given the large number of processors; (5) greedy-based centralized strategies tend to lead to migrating almost all tasks away from their current location which can be very expensive in a large system. On the other hand, in a distributed strategy, each processor exchanges state information with other processors in its neighborhood. Compared to centralized strategies, a distributed load balancing strategy is designed to be scalable on large distributed systems [7, 11]. However, several problems can be found in practice. For example, distributed schemes can suffer from the aging of

load information. This is largely due to the nonpreemptive message scheduler. Other messages have to wait in the message queue. Thus the processing of critical messages that contain the load information may be delayed in the queue and get out-of-date when the execution time of the method being processed is long. This aging of load information may lead the load balancing runtime to make poor load balancing decisions. Also, the delay in processing the load messages may further delay the invocation of the load balancing strategies. This is because load balancing can only be triggered when all load information is received from neighboring processors. Any delay in receiving these messages may slow down the invocation of the load balancing strategies and response time. For user experience of cloud service, it depends heavily on the communication delay between end user and service instance the user accesses, which is mainly caused by Internet delay between the user and the data center hosting the instance [12].

Moreover, overall performance of large-scale distributed data processing is known to degrade when the system contains nodes that have a large communication delay [13]. For loosely coupled communication applications such as mpiBLAST, we can always see a considerable speedup as here what prevails is the computation gained by an increased parallelism over the communication overhead. Nevertheless, if the dataset is relatively small and the number of VMs utilized is too large, then the parallelism can be overwhelmed by the communication. We could see that as we include more than 32 VMs from three different domains, the performance did not obtain any speedup, which was mainly caused because of a higher communication delay, and the computation time was overwhelmed by the communication time [14]. The current virtual machine (VM) resources scheduling in cloud computing environment mainly considers current state of the system but seldom considers system variation and historical data, which always leads to load imbalance of the system [5]. However, CPU speed and communication delay are two factors that influence platform performance in high performance computing systems [15].

Hajjat et al. [16] show that communication delay is time-varying. This is attributable to uncertainties associated with the amount of traffic, congestion, and other unpredictable factors within the network [17, 18]. How to perceive and predict the communication state quickly and accurately in load balancing process is a key problem to be solved in large distributed computing system. At present, there exist some algorithms for solving this problem such as RW (Random Walk) [19], HA (Historical Average) [20], and IHA (Informed Historical Average) [21]. The RW judges the communication situation just based on the current network delay situation, HA performs this only according to the average situation of historical data, and the IHA combines RW and HA. However, the nonlinear and uncertain characteristics of communication delays are not considered and the influence of random factors cannot be avoided. Thus, the prediction accuracy of these methods is very low with the time interval getting shorter.

In large-scale distributed computing systems in which the computational elements (CEs) are physically or virtually distant from each other, there are a number of inherent time-delay factors that can seriously alter the expected performance of the load balancing policies that do not account for such delays [17]. Dhakal et al. [18] presented a random delay forecasting formula which combines RW and HA. However, the value of forgetting factor α can only be chosen with experience, and it is very difficult to predict for changing network, thereby having no practical usability. In addition, it does not have a complete prediction model and can only predict the average delay of the next time interval but not the delay after the next time interval.

A hierarchical dynamic load balancing strategy based on generalized neural network (HLBSGNN) is presented considering time-varying characteristics of communication delays in large distributed computing systems. This hierarchical load balancing strategy reduces the load balancing overhead in large distributed computing systems with communication-optimized hierarchy. In the new strategy, the computation rate of node and time-varying characteristics of communication delay are considered, and a delay prediction model based on generalized neural network (GNN) theory is constructed. It provides an effective optimization method for load balancing strategies considering delay overhead in large distributed systems.

The rest of this paper is organized as follows. After describing the hierarchical load balancing strategy (HLBSGNN) in Section 2, comparison experiments are provided in Section 3. We conclude this paper in Section 4.

2. The Hierarchical Load Balancing Strategy (HLBSGNN)

2.1. Intelligent Neuron Model. In traditional neural network models, the neuron's structure is very simple and its transfer function is not changeable, so the neuron only has information processing ability and the information storage ability of the whole neural network is limited. When dealing with large-scale problems, traditional neural network is hard to converge. In order to greatly increase the information storage ability, the model of generalized neural network (GNN) is presented. The neurons of GNN can be the simple neurons of traditional neural network or the intelligent neurons which have information storage ability or can be the neurons which consist of a neural network (multi-inputs/multioutputs). The neuron is constructed based on sample functions. In this paper, a new intelligent neuron model is constructed based on linearly independent functions. A GNN model is formed with these intelligent neurons and can greatly improve the performance of neural network. The GNN constructed by neurons can be applied to predict communication delays of large distributed computing systems having high practicability.

The intelligent neuron has information storage ability and adjusts its transfer function in a set of functions by some training algorithms. In previous research, Eck and Shih [22] used linearly independent functions to pretreat the neural network's inputs, which made neural network get a better mapping effect. Without importing new inputs, these functions can effectively increase the dimensions of input vectors and therefore can greatly accelerate the network's convergent speed. This paper imported linearly independent

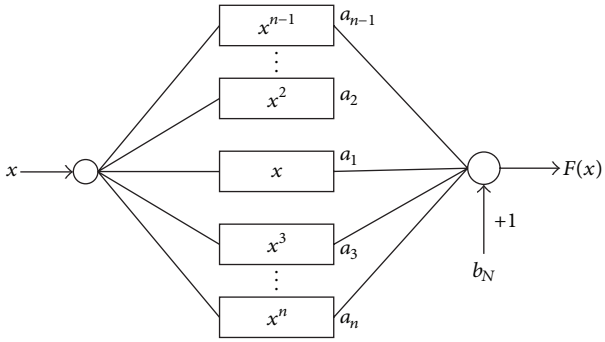


Figure 1: Model of intelligent neuron.

functions into interior constructions of intelligent neurons and expanded neuron's input x into linearly independent functions $x, x^2, x^3, \ldots, x^n$ (n is odd) and constructed a new intelligent neuron model, as shown in Figure 1, where x is neuron's input, a_r ($r = 1, 2, 3, \ldots, n$) is connected weight, $F(x)$ is neuron's output and can be formulated as

$$F(x) = \sum_{r=1}^{n} a_r x^r + b_N. \tag{1}$$

If the neuron has n inputs, the mapping relation is

$$XA + b_N = \Upsilon, \tag{2}$$

where Υ is neuron's output and X can be represented as

$$X = \begin{bmatrix} x_1 & x_1^2 & \cdots & x_1^n \\ x_2 & x_2^2 & \cdots & x_2^n \\ \vdots & \vdots & \ddots & \vdots \\ x_n & x_n^2 & \cdots & x_n^n \end{bmatrix}, \tag{3}$$

$$A^T = (a_1, a_2, \ldots, a_n).$$

Drawing the common factors $x_1, x_2, \ldots, x_n$ of matrix X, the following formulation can be gained:

$$X = \prod_{r=1}^{n} x_r \begin{bmatrix} 1 & x_1 & \cdots & x_1^{n-1} \\ 1 & x_2 & \cdots & x_2^{n-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & \cdots & x_n^{n-1} \end{bmatrix}. \tag{4}$$

By Vandermonde determinant we can have

$$\begin{vmatrix} 1 & x_1 & \cdots & x_1^{n-1} \\ 1 & x_2 & \cdots & x_2^{n-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & \cdots & x_n^{n-1} \end{vmatrix} = \prod_{1 \le i < j \le n} (x_j - x_i). \tag{5}$$

So the value of determinant of X is

$$\det X = |X| = \prod_{r=1}^{n} x_r \prod_{1 \le i < j \le n} (x_j - x_i). \tag{6}$$

If $x_i \neq x_j$, then $\det X \neq 0$; the rank of X is n. It shows that the dimensions of input vectors can be increased and the neuron's information storage ability is consequently improved without increasing the number of inputs in the interior of intelligent neurons by the way of expanding functions. All of that makes neural network have a good mapping effect.

The sine and cosine functions ($x, \sin \pi x, \cos \pi x, \sin 2\pi x, \cos 2\pi x, \ldots$) are used to expand neural network's inputs, which can greatly improve neural network's performance [23]. This paper introduces sine and cosine functions into the constructions of intelligent neurons and constructs a new model of intelligent neuron as follows:

$$F(x) = \sum_{r=1}^{k} a_r S(r, x) + b_N. \tag{7}$$

In formulation (7), x is the neuron's input, $F(x)$ is the neuron's output, and $S(r, x)$ is linearly independent function in the intelligent neuron and can be formulated as

$$S(r, x) = \begin{cases} x, & \text{if } r = 1 \\ \sin\left(\dfrac{r}{2}\pi x\right), & \text{if } r \text{ is even} \\ \cos\left(\dfrac{r-1}{2}\pi x\right), & \text{if } r \neq 1,\ r \text{ is odd} \end{cases} \qquad r = 1, 2, \ldots, k. \tag{8}$$

The functions $x, \sin \pi x, \cos \pi x, \sin 2\pi x, \cos 2\pi x, \ldots$ are linearly independent. The neuron formed by these functions has good function mapping ability and is superior to the one formed by the functions $x, x^2, \ldots, x^n$ in terms of prediction accuracy and convergence rate [24].

2.2. Delay Prediction Model and Its Learning Algorithm Based on GNN. Generally speaking, time prediction can be divided into two main methods: data model and analysis model. Data model is basically characterized by data guidance, taking the historical and current delay time variable sequence as inputs. In Figure 2, we assume that current time is t, and the historical data is $f(t-1), f(t-2), \ldots, f(t-n)$ at time $t-1, t-2, \ldots, t-n$, respectively. We can forecast future time sequences $f(t+1), f(t+2), \ldots$ by analyzing historical data samples.

In this paper, a GNN method based on intelligent neuron model is used to predict the delays of nodes in future time intervals. GNN has many structural forms. The input layer of GNN is composed of ordinary neurons, and the hidden layer and the output layer are composed of intelligent neurons. For real-time prediction of node delay, there exists a certain relationship between the delay of current node and the delays of last several nodes, which thus can be used to predict the delay time of the node in next time interval. We choose the delay amount at times $t-6, t-5, t-4, t-3, t-2, t-1$ as

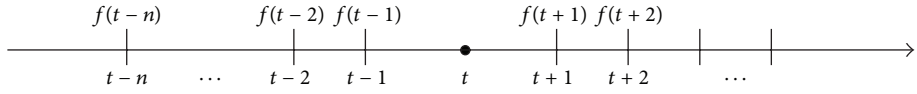


Figure 2: Forecasting via historical data.

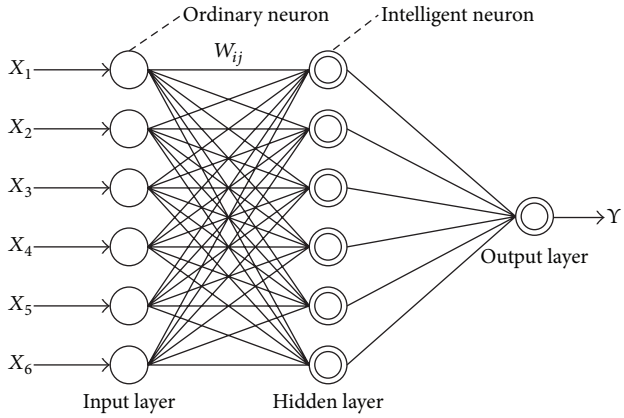


Figure 3: Delay forecasting model based on GNN.

the input of GNN and take the delay amount of prediction node at time $t + 1$ as output. The number of nodes in hidden layer nodes is set to 6, and the delay prediction model can be shown as in Figure 3.

The connection weights between layers can be learned by error back propagation algorithm, and the adjustable parameters of hidden layer and output layer can be learned with LMS algorithm [25]. A practical and complete learning process of GNN involves the following steps:

(1) Initializing the neural network.

(2) Calculating the actual output and the state of neuron in each layer for different samples.

(3) Computing the error of each neuron and back propagation error in output layer and hidden layer.

(4) Amending the connected weights and thresholds between input layer and hidden layer.

(5) Recalculating the state of GNN, error of node, and back propagation error in each layer.

(6) Amending the nonlinear transfer function of node in hidden layer.

(7) Recalculating the state of GNN, error of node, and back propagation error in each layer.

(8) Amending the connected weights and thresholds between output layer and hidden layer.

(9) Recalculating the state of GNN, error of node, and back propagation error in each layer.

(10) Amending the nonlinear transfer function of node in output layer.

(11) Calculating the whole neural network's error and verifying if this error meets the requirements. If true, then end the training algorithm; else go to (2).

2.3. The Hierarchical Load Balancing Algorithm Based on GNN. The computing nodes in large distributed systems can be mapped to a hierarchical tree with tree model [26] to take advantage of the architecture. A hierarchical tree can be built according to machine's topological hierarchy to minimize load balancing communication overhead. Assume that the transmission delay is low in large distributed computing system. Therefore it can be assumed that the transmission delay between two adjacent nodes is approximately equal. Let $\lambda_{ij,t}(0)$ be transmission delay of idle load between node i and node j at time t, and we have

$$\lambda_{ij,t}(0) \equiv ps_{ij}, \tag{9}$$

where $t = 0$, p is idle-load delay constant between adjacent nodes, and s_{ij} is the distance (in terms of number of hops) of the shortest path from node i to node j. For systems with N nodes, we set idle-load delay threshold of nodes in each layer by the equation

$$\lambda_{ij,t}(0) \leq \varphi. \tag{10}$$

Therefore we can determine the height of load balancing tree and construct a load balancing tree. In our load balancing strategy, an intermediate node at level l_i and its immediate children nodes at level l_{i-1} form a load balancing domain, with the root node as group leader (manager). Load balancing group leaders control the load balancing process inside their domain, playing the role similar to the central node in a centralized load balancing scheme. Load balancing domains periodically exchange the load of their processors. This process is triggered by leaf processors at level 0 of the tree, starting to send their local load up to the domain group leader processors which are at level 1. The same process continues by ascending the tree to top level in the tree. The manager load balancing algorithm is described in Algorithm 1.

2.4. Load Balancing Process

(1) *Load Balancing Initiation*. At a synchronous time, group leader collects information at each leaf node. Each leaf node reports the prediction of task completion time T_i to group leader. For example, a group leader manages n leaf nodes, and it controls a state vector $I = (i_1, i_2, \ldots, i_n)$, $i_i \in (0, 1)$. The vector is initialized to $(0, 0, \ldots, 0)$. If group leader receives T_i sent by node i, it then updates the value of i_i with 1.

(2) *Load Imbalance Calculation*. When the state vector I is $(1, 1, \ldots, 1)$, the expected completion time of each load is calculated to start load balancing if (10) is satisfied

$$\sigma(T) = \sqrt{\left|E\left(T^2\right) - \left[E(T)\right]^2\right|} \geq \eta. \tag{11}$$

```
(1) begin
(2)     Data: V_t (the set of tasks);
(3)           E_o (communication graph);
(4)           V_p (the set of processors);
(5)           G_p (the background load of processors);
(6)     Result: M: V_t → V_p; // A task mapping
(7)     nObjs ← NumObjs; // number of migratable objects
(8)     ObjectHeap objHeap(nObjs + 1); // declare a object heap objHeap
(9)     V_t → maxhAllObjs; // max for all migratable objects
(10)    ProcessorHeap lightProcessors(P); // declare a processor heap lightProcessors
(11)    G_p → lightProcessors;
(12)    for i ← 1 to nObjs do
(13)        minLoad ← MAX_DOUBLE; // Initially the minimum load minLoad are set to MAX_DOUBLE
(14)        o ← objHeap.deleteMax(); // Find the object o with high loads
(15)        cpuDonor ← lightProcessors->deleteMin(); // Find the migratable target (processor) with light loads
(16)        comm_cost ← ∑_{e_ob∈E_o∧M(b)=cpuDonor} GNN(c_ob^∧); // comm_cost represents communication cost
(17)        newLoad ← cpuDonor.load + comm_cost; // Recalculate new load considering communication cost
(18)        if newload < minLoad then
(19)            minLoad ← newLoad;
(20)            newDonor ← cpuDonor; // Find new migratable processor
(21)        end if
(22)        for cpuDonor ∈ P_comm do
(23)            comm_cost ← ∑_{e_ob∈E_o∧M(b)=cpuDonor} GNN(c_ob^∧);
(24)            newLoad ← cpuDonor.load + comm_cost;
(25)            if newLoad < minLoad then
(26)                minLoad ← newLoad;
(27)                newDonor ← cpuDonor; // newDonor: new migratable processor
(28)            end if
(29)        end for
(30)        o → newDonor; // Allocating task o to processor newDonor
(31)        Update(objHeap);
(32)        Update(lightProcessors);
(33)    end for
(34)    if objHeap.deleteMax() > sit_max then // sit_max: threshold
(35)        // failed
(36)        LB_TopManager(); // initiate load balancing on top level
(37)    else
(38)        // succeed;
(39)    end if
(40) end
```

Algorithm 1: LB_Manager // Load balancing for a manager.

(3) *Overhead Calculation*. The overhead of each leaf node can be represented as

$$L_j^{\text{ex}}(t) = Q_j(t) - \frac{\lambda_j^d}{\sum_{k=1}^{n} \lambda_k^d} \sum_{l}^{n} Q_l(t), \tag{12}$$

where $Q_j(t)$ denotes the CPU queue length in node j at time t, and λ_j^d denotes the calculation speed in node j. If $L_j^{\text{ex}}(t) > 0$ then the node is over-loaded and otherwise is low-loaded.

(4) Load balancing strategy GreedyCommLB is invoked to complete task migration.

3. Experiments

3.1. Comparison to Traditional Centralized Load Balancing Strategy. The experiments were run on a 64-node Lenovo DeepComp 1800 installed at Dalian University of Technology, an SMP cluster machine formed from two 2.8 GHz Intel Xeon processors with 1 MB L2 cache and with 4 GB of physical RAM, each node connected by a Myrinet network, and running RedHat 9.0 operating system. We simulated the BlueGene/L (with 32 K–64 K nodes) on DeepComp 1800 with BigSim emulator [27] using 8 real nodes and used load balancing benchmark program (lb_test) to simulate the running of parallel programs. The program generated a certain amount of communication objects or tasks in a system with a mesh 2D topology and each object performs

Table 1: Load balancing overhead and memory overhead on 32 K emulated processors.

GreedyCommLB	Number of objects			
	128 k	256 k	512 k	1 m
Centralized				
Balancing overhead	10.2961 s	21.1509 s	38.1038 s	—
Memory overhead	41.4052 M	82.8662 M	165.8105 M	—
Hierarchical				
Balancing overhead at level 0	0.1958 s	0.4399 s	0.7824 s	1.7164 s
Memory overhead at level 0	5.1542 M	10.3281 M	24.9335 M	41.4052 M
Balancing overhead at level 1	0.4616 s	1.2910 s	3.6935 s	5.2248 s
Memory overhead at level 1	10.3281 M	24.9335 M	41.4052 M	82.8662 M

Table 2: Load balancing overhead and memory overhead on 64 K emulated processors.

GreedyCommLB	Number of objects			
	128 k	256 k	512 k	1 m
Centralized				
Balancing overhead	10.84 s	25.22 s	45.18 s	—
Memory overhead	41.40 M	82.87 M	165.81 M	—
Hierarchical				
Balancing overhead at level 0	0.42 s	1.08 s	3.95 s	4.95 s
Memory overhead at level 0	10.33 M	24.93 M	41.40 M	82.87 M
Balancing overhead at level 1	0.49 s	1.03 s	2.30 s	5.63 s
Memory overhead at level 1	12.45 M	20.68 M	41.41 M	82.87 M

a certain amount of iterations. For centralized and hierarchical load balancing strategies, the load balancing overhead and memory overhead at each iteration of the simulation on 32 K and 64 K emulated processors are shown in Tables 1 and 2.

The compared results of the centralized and hierarchical load balancing strategies in the same lb_test are shown in Tables 1 and 2. In the centralized strategy, the single central node can become the communication bottleneck in a computing system with a very large number of processors. However, in the hierarchical strategy, the height of the load balancing tree can be reduced by threshold setting, and thus the overheads on load balancing, memory, and the idle time of leaf node can be greatly reduced.

3.2. Comparison to Traditional Distributed Load Balancing Strategy. In order to compare the performance between HLBSGNN strategy and traditional distributed load balancing strategy, we ran the algorithm proposed in this paper and the nearest neighbor load balancing algorithm [28, 29] (migrating tasks from overloaded node to its underloaded neighboring nodes) and used the tool Projections to track the load balancing process. The results are shown in Figures 4(a) and 4(b).

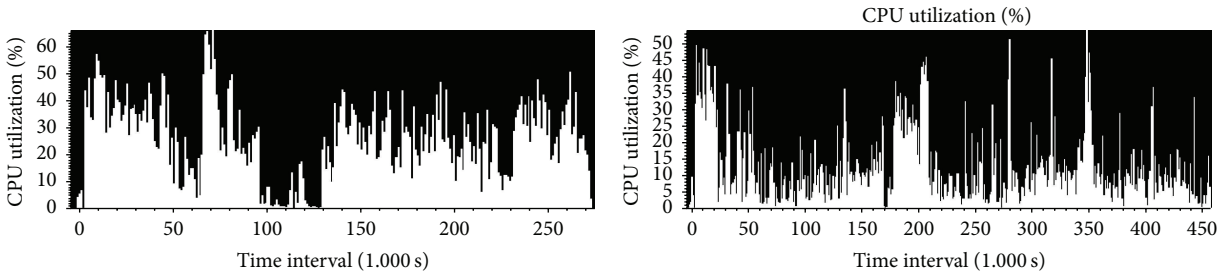
As shown in Figure 4(a), the average CPU utilization of the HLBSGNN algorithm is about 30%, while the average CPU utilization of the nearest neighbor load balancing algorithm is about 10%, as shown in Figure 4(b).

3.3. Comparison to Random Delay Strategies. In order to verify the communication prediction mechanism of hierarchical strategy, we ran communication test program Commbench of Charm++ on DeepComp 1800 machine using 2 nodes to take delay sampling ($\Delta t = 1$ m) to test the delay prediction performance of the IHA, GNN, and BP (back propagation) methods. According to the relationship between the square root error and the forgetting factor α (as shown in Figure 5), it is shown that the prediction results are best when α is 0.7 and thus let α be 0.7 in this paper.

Aim to compare the convergence time and number of learning times of GNN and traditional BP neural network, we ran two algorithms under the same conditions such as initial weight values, training samples, and parameters. The results are shown in Table 3.

As shown in Table 3, the prediction accuracy of BP is lower than that of GNN. The convergence time of BP is higher than that of GNN and it is even nonconvergent in the case of high precision to be required. In order to compare the prediction performance of BP, IHA, and GNN algorithms, we used 75 sets of data to perform training and forecasted 100 sets of data. The results are shown in Figures 6 and 7.

It can be seen from Figures 6 and 7 that the delay prediction of the generalized neural network is better than that of IHA algorithm and BP algorithm. In order to evaluate the prediction performance, we introduce the following evaluating metrics: (1) RME: relative mean error; (2) RMSE: root-mean-squared error; (3) EC: error change rate; (4) mxarer: maximum absolute relative error; (5) mrerr: mean relative error. In order to compare the prediction accuracy of various algorithms, we calculated the prediction errors of various algorithms, as shown in Table 4.



(a) CPU utilization projections graph of HLBSGNN strategy

(b) CPU utilization projections graph of nearest neighborhood strategy

Figure 4: CPU utilization.

Table 3: Comparison of convergence time and learning times between BP and GNN.

Algorithms	Data			
	50	100	150	200
BP				
Time	11.875 s	7.937 s	3.406 s	0.281 s
Steps	156667	63049	15683	771
GNN				
Time	0.281 s	0.296 s	0.328 s	0.078 s
Steps	629	367	225	49

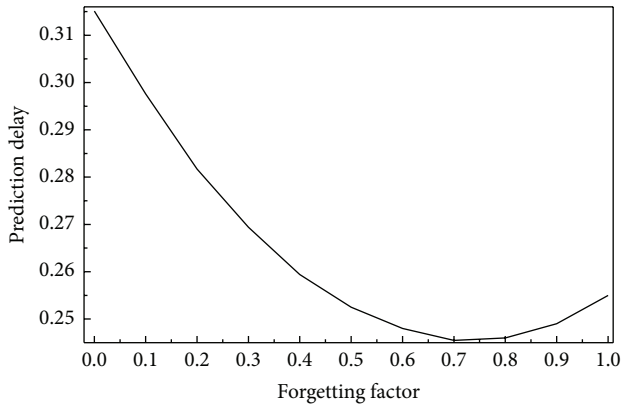


Figure 5: The function relation between forgetting factor and square root error of prediction delay.

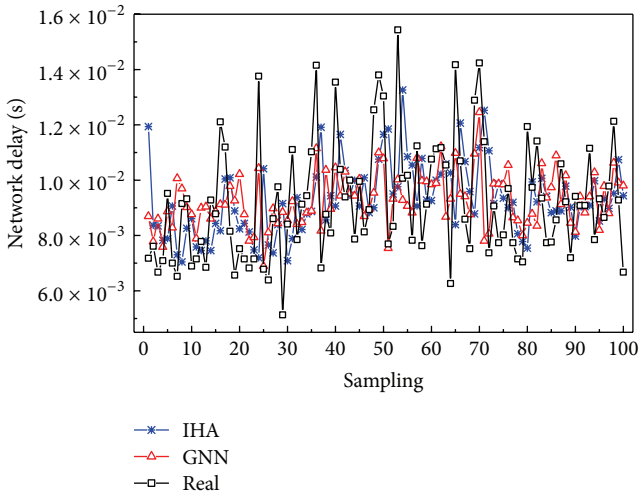


Figure 6: Comparison of GNN and IHA algorithm.

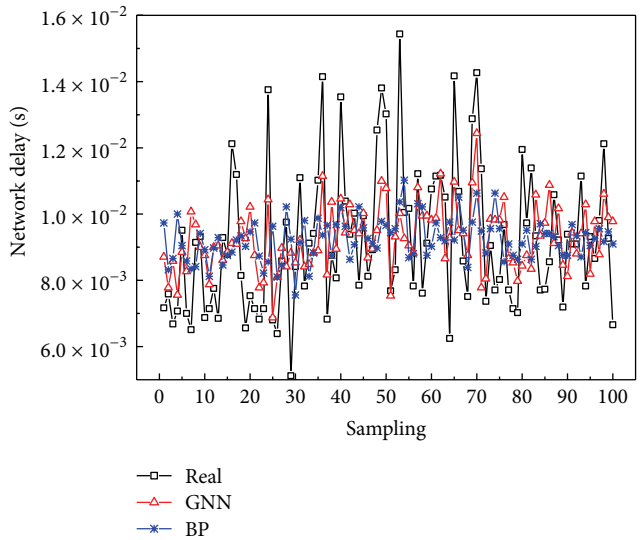


Figure 7: Comparison of GNN and BP algorithm.

In Table 4, the prediction accuracy of GNN algorithm is better than that of IHA and BP algorithm when the delay is sharply changed. In the load balancing process of HLBSGNN strategy, the accuracy of makespan can be improved by prediction results when considering communication cost, as shown in Table 5.

In order to investigate the effect of GNN method on load balancing, assuming a distributed system with 2D mesh topology and each node with 10 tasks (or objects), each object communicating with the nearest neighbor node 100 times, we tested the performance of the GreedyCommLB algorithm based on the GNN and IHA and performed performance comparison of GNN and IHA algorithms. The results are shown in Figure 8. It can be seen that the GNN algorithm is better than IHA algorithm and can improve the prediction precision of makespan.

Table 4: Prediction error comparison of various algorithms.

Algorithms	RME	RMSE	EC	mxarer	mrerr
RW	0.2338	0.3158	0.8508	1.2601	0.0362
HA	0.2061	0.2550	0.8774	0.8979	0.0078
IHA	0.1970	0.2457	0.8805	0.9067	0.0043
BP	0.186842	0.229921	0.889735	0.363908	−0.048636
GNN	0.164918	0.205186	0.904193	0.470729	−0.044529

Table 5: Error comparison of IHA and GNN to predict makespan of load balancing.

Algorithms	RME	RMSE	EC	mxarer	mrerr
IHA	0.013957	0.014348	0.992736	0.017143	0.013957
GNN	0.011165	0.011479	0.994197	0.013714	0.011165

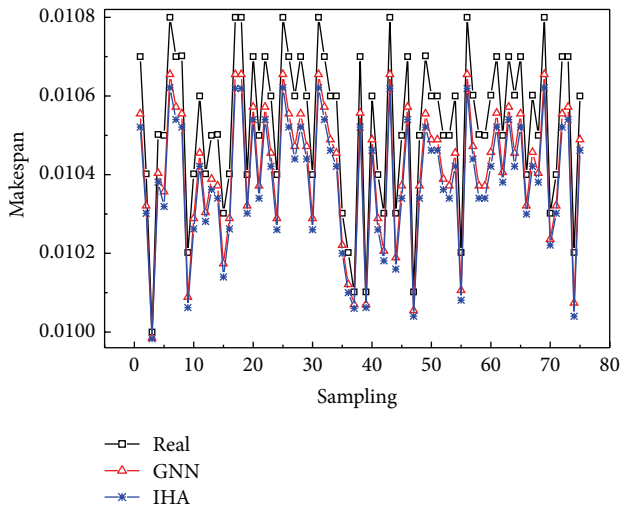


Figure 8: Makespan comparison of IHA and GNN algorithms.

4. Conclusions

In this paper, a hierarchical load balancing strategy based on generalized neural network is proposed, considering the load balancing overhead and time-varying characteristics of network delay in large distributed systems. The proposed strategy can reduce load balancing overhead in large-scale systems with communication-optimized hierarchy and optimize the delay of communication and migration, thereby improving performance of load balancing. Experimental results have shown it to be relatively effective and efficient against traditional centralized and distributed load balancing strategies in large distributed computing systems.

In fact, there is no "one size fits all" when it comes to load balancing. This means that the performance of load balancing is heavily dependent on the particular applications, system architectures, and numerous other variables, and there may be one or more viable solutions. This is also true of the work in this paper, which is mainly suitable for the case that the delay is large and time-varying. Research on load balancing for applications with high real-time requirement will be our future work.

Competing Interests

The authors declare that they have no competing interests.

Acknowledgments

This work was supported by Chongqing Natural Science Foundation (no. KJ1400316), partially supported by National Natural Science Foundation of China (no. 11401061) and Natural Science Youth Fund of Hebei Province (no. F2015407039).

References

[1] G. Cybenko, "Dynamic load balancing for distributed memory multiprocessors," *Journal of Parallel and Distributed Computing*, vol. 7, no. 2, pp. 279–301, 1989.

[2] C.-C. Hui and S. T. Chanson, "Hydrodynamic load balancing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 10, no. 11, pp. 1118–1137, 1999.

[3] T. L. Casavant and J. G. Kuhl, "A taxonomy of scheduling in general-purpose distributed computing systems," *IEEE Transactions on Software Engineering*, vol. 14, no. 2, pp. 141–154, 1988.

[4] Z. L. Lan, V. E. Taylor, and G. Bryan, "Dynamic load balancing for adaptive mesh refinement application," in *Proceedings of the International Conference on Parallel Processing (ICPP '01)*, pp. 571–579, IEEE, Washington, DC, USA, 2001.

[5] J. Hu, J. Gu, G. Sun, and T. Zhao, "A scheduling strategy on load balancing of virtual machine resources in cloud computing environment," in *Proceedings of the 3rd International Symposium on Parallel Architectures, Algorithms and Programming (PAAP '10)*, pp. 89–96, IEEE, Dalian, China, December 2010.

[6] R. W. Ahmad, A. Gani, S. H. A. Hamid, M. Shiraz, F. Xia, and S. A. Madani, "Virtual machine migration in cloud data centers: a review, taxonomy, and open research issues," *Journal of Supercomputing*, vol. 7, no. 7, pp. 2473–2515, 2015.

[7] J. O. Gutierrez-Garcia and A. Ramirez-Nafarrate, "Agent-based load balancing in Cloud data centers," *Cluster Computing*, vol. 18, no. 3, pp. 1041–1062, 2015.

[8] A. Bhadani and S. Chaudhary, "Performance evaluation of web servers using central load balancing policy over virtual machines on cloud," in *Proceedings of the 3rd Annual ACM Bangalore Conference (COMPUTE '10)*, pp. 1–4, ACM, New York, NY, USA, January 2010.

[9] W. Y. Zhou, S. B. Yang, J. Fang, X. L. Niu, and H. Song, "VMCTune: a load balancing scheme for virtual machine cluster based on dynamic resource allocation," in *Proceedings of the 9th International Conference on Grid and Cloud Computing*, pp. 81–86, IEEE, Nanjing, China, November 2010.

[10] L. M. Vaquero, L. Rodero-Merino, and R. Buyya, "Dynamically scaling applications in the cloud," *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 1, pp. 45–52, 2011.

[11] M. Randles, D. Lamb, and A. Taleb-Bendiab, "A comparative study into distributed load balancing algorithms for cloud computing," in *Proceedings of the 24th IEEE International Conference on Advanced Information Networking and Applications Workshops (WAINA '10)*, pp. 551–556, IEEE, Perth, Australia, April 2010.

[12] Y. Kang, Y. F. Zhou, Z. B. Zheng, and M. R. Lyu, "A user experience-based cloud service redeployment mechanism," in *Proceedings of the IEEE International Conference on Cloud*

Computing (CLOUD '11), pp. 227–234, IEEE, Washington, DC, USA, July 2011.

[13] N. Oguchi and S. Abe, "Reconfigurable TCP: an architecture for enhanced communication performance in mobile cloud services," in *Proceedings of the 11th IEEE/IPSJ International Symposium on Applications and the Internet (SAINT '11)*, pp. 242–245, Munich, Germany, July 2011.

[14] J. C. Martinez, L. X. Wang, M. Zhao, and S. M. Sadjadi, "Experimental study of large-scale computing on virtualized resources," in *Proceedings of the 3rd International Workshop on Virtualization Technologies in Distributed Computing (VTDC '09)*, pp. 35–41, ACM, New York, NY, USA, June 2009.

[15] H. Song, J. Li, L. Li et al., "PerHPC: design and implement personal high performance computing platform using cloud computing technology," in *Proceedings of the 6th Annual Chinagrid Conference*, pp. 28–34, IEEE, Liaoning, China, August 2011.

[16] M. Hajjat, S. Narayanan, D. Maltz, S. Rao, and K. Sripanidkulchai, "Dealer: dynamic request splitting for performance-sensitive applications in multicloud environments," Tech. Rep. TR-ECE-11-10, School of Electrical and Computer Engineering, Purdue University, West Lafayette, Ind, USA, 2011.

[17] C. T. Abdallah, M. M. Hayat, S. Dhakal, J. D. Birdwell, and J. Chiasson, *Dynamic Time Delay Models for Load Balancing, Part II: A Stochastic Analysis of the Effect of Delay Uncertainty*, Springer, Berlin, Germany, 2004.

[18] S. Dhakal, M. M. Hayat, J. E. Pezoa, C. D. Yang, and D. A. Bader, "Dynamic load balancing in distributed systems in the presence of delays: a regeneration-theory approach," *IEEE Transactions on Parallel and Distributed Systems*, vol. 18, no. 4, pp. 485–497, 2007.

[19] N. Bisnik and A. A. Abouzeid, "Optimizing random walk search algorithms in P2P networks," *Computer Networks*, vol. 51, no. 6, pp. 1499–1514, 2007.

[20] B. L. Smith, B. M. Williams, and R. Keith Oswald, "Comparison of parametric and nonparametric models for traffic flow forecasting," *Transportation Research Part C: Emerging Technologies*, vol. 10, no. 4, pp. 303–321, 2002.

[21] P. K. Hoong, I. K. T. Tan, O. K. Chien, and C.-Y. Ting, "Road traffic prediction using Bayesian networks," in *Proceedings of the IET International Conference on Wireless Communications and Applications (ICWCA '12)*, pp. 1–5, IEEE, Kuala Lumpur, Malaysia, October 2012.

[22] J. T. Eck and F. Y. Shih, "An automatic text-free speaker recognition system based on an enhanced Art 2 neural architecture," *Information Sciences*, vol. 76, no. 3-4, pp. 233–253, 1994.

[23] D. Park, L. R. Rilett, and G. Han, "Spectral basis neural networks for real-time travel time forecasting," *Journal of Transportation Engineering*, vol. 125, no. 6, pp. 515–523, 1999.

[24] R. Hu, D. Li, Z. Xu, and T. Yao, "Generalized information storing principle and higher-order generalized neural networks," *Acta Electronica Sinica*, vol. 24, no. 7, pp. 59–65, 1996.

[25] B. Widrow, Y. Kim, and D. Park, "The Hebbian-LMS learning algorithm," *IEEE Computational Intelligence Magazine*, vol. 10, no. 4, pp. 37–53, 2015.

[26] B. Yagoubi and Y. Slimani, "Dynamic load balancing strategy for grid computing," *Enformatika Transactions on Engineering*, vol. 13, pp. 260–265, 2006.

[27] Parallel Programming Laboratory (PPL), *Runtime Systems and Tools: BigSim—Simulating PetaFLOPS Supercomputers*, Parallel Programming Laboratory (PPL), 2015, http://charm.cs.uiuc.edu/research/bigsim.

[28] A. B. Sinha and L. V. Kale, "A load balancing strategy for prioritized execution of tasks," in *Proceedings of the 7th International Parallel Processing Symposium (IPPS '93)*, pp. 230–237, IEEE, Newport Beach, Calif, USA, April 1993.

[29] W. Shu, "Dynamic scheduling of medium-grained processes on distributed memory computers," in *Proceedings of the 27th Hawaii International Conference on System Sciences*, pp. 435–444, IEEE, January 1994.