International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Index Copernicus Value (2013): 6.14 | Impact Factor (2014): 5.611

A Novel Approach for Load Balancing in Distributed System using FIFO-Support Vector Machine (FIFOSVM)

Harsh Singhal[1], Neelendra Badal[2], Amit Kumar Gupta[3], Devesh Singh Sisodia[4], Gautam Kumar Singh[5], Hemant Kumar Singh[6]

[1]NIT Allahabad, India

[2] KNIT Sultanpur, India

[3] NIT Bhopal, India

[4]KNIT Sultanpur, India

[5]NIE Mysore, India

[6]Sastra University, Thanjavur, India

Abstract: ***Standard balancing algorithms used for load balancing in the distributed environment do not give best solution always. Latency in the system deter its performance, so it is important to optimize the load balancing strategies to get better performance .An optimum algorithm should distribute the load to the available servers based on their processing power. There are many load balancing algorithms available. Most load balancer dynamically chooses next eligible server. Simple load balancers use FIFO and round robin (RR) algorithms. Each algorithm would be efficient in one aspect and might be inefficient otherwise. In our proposed approach, we will use few algorithms such as FIFO with Support Vector Machine (SVM) and invoke them during a particular situation when they are efficient. Our simulation results show that our load balancer significantly improves the average and total response time of client tasks and thus increases the performance of the overall system.***

Keywords: Load balancing, FIFO, Distributed System, SVM, Training Set

1. Introduction

In distributed load balancing system, distribution of loads to the processing components is basically called the load balancing issue. In a system with multiple machines there is a very high chance that some machine will be idle while the other will be over loaded. The goal of the load balancing algorithms is to maintain the load to each processing element such that all the processing elements become neither overloaded nor idle that means each processing element ideally has equal load at any moment of time during execution to obtain the maximum performance (minimum execution time) of the system [9-11]. So the proper design of a load balancing algorithm may significantly improve the performance of the system. The computing power of any distributed system can be realized by allowing its constituent computational elements (CEs) or components of machine, to work cooperatively so that large loads are allocated among them in a fair and effective manner. Any strategy for load distribution among CEs is called load balancing (LB). An effective load balancing policy confirms best use of the distributed resources whereby no machines remains in an idle state while any other CE is being utilized. In many of today's distributed system environments, the machines are linked by a delay-limited and bandwidth limited communication medium that inherently inflicts tangible delays on internode communications and load exchange. Examples include distributed systems over wireless local-area networks (WLANs) as well as clusters of geographically distant CEs connected over the Internet, such as PlanetLab [1]. Although the majority of Load balancing strategies established heretofore take account of such time delays [2], [3], [4], [5], [6], they are predicated on the assumption that delays are deterministic. In actuality, delays are random in such communication media, especially in the case of WLANs. This is attributable to uncertainties associated with the amount of traffic, congestion, and other unpredictable factors within the network. Furthermore, unknown characteristics (e.g., type of application and load size) of the incoming loads cause the CEs to exhibit fluctuations in runtime processing speeds.

For example, if nodes have dated, inaccurate information about the state of other nodes, due to random communication delays between nodes, then this could result in unnecessary periodic exchange of loads among them. Consequently, certain nodes may become idle while loads are in transit, a condition that would result in prolonging the total completion time of a load.

A. Support Vector Machine

Support Vector Machines (SVMs) (shown in Fig. 1) consist of two major phases: training and classification. The user obtains the model file through the training process and uses it to make predictions in the classification process. Based on our performance profiling, the majority of SVM execution time is spent on the training phase which makes training the major performance bottleneck in SVM.

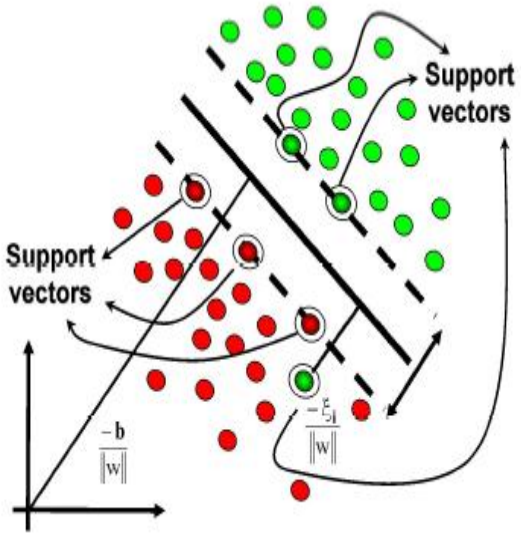

Figure 1: support vector machine

The figure shows Support Vector Machines, whose decision boundary is decided by Support Vector machine (SVMs).The left-lower dots represent -1 class and the right-upper dots represent +1 class. These two classes are classified by decision boundary (the dashed lines in this figure). We can observe that classification result is decided by the six Support Vectors.

B. Distributed Load Balancing System

This section present a distributed load balancing (DLB) system for SVM (support vector machine) on distributed systems. To address the processors of machine, each processor is assigned a relative weight. To deal with the SVM, the scheme divides the load balancing process into two steps: global load balancing phase and local load balancing phase. Further, the proposed scheme addresses distributed feature of machines by adaptively choosing an appropriate action according to the traffic on them. The details are given in the following subsections.

2. Description

First, we define a "group" as a set of processors which have the same performance and share an interconnected machine. A group can be a shared-memory parallel computer, a distributed-memory parallel computer, or a group of workstations. Communications within a group are referred as local communication, and those between different groups are remote communications. A distributed system is composed of two or more groups. This terminology is consistent with that given in [6] Our distributed Load Balancing scheme entails two steps to redistribute the workload: global load balancing phase and local load balancing phase, which are described in detail below.

i. Global Load Balancing Phase

After each time-step at level 0 only, the scheme evaluates the load distribution among the groups by considering both group and virtual features of the system. If imbalance is detected, a heuristic method described in the following subsections is invoked to calculate the computational gain of removing the imbalance and the overhead of performing such load redistribution among groups. If the computational gain is larger than the redistribution overhead, this step will be invoked. All the processors will be involved in this process, and both global and local communications are considered. Workload will be redistributed by considering the heterogeneity of number of processors and processor performance of each group.

ii. Local Load Balancing Phase

After each time-step at the finer levels, each group entails a balancing process within the group. The parallel distributed load balancing scheme as mentioned above is invoked, that is, the workload of each group is evenly and equally distributed among the processors. However, load balancing is only allowed within the group. An overloaded processor can migrate its workload to an under loaded processor of the same group only. In this manner, children grids are always located at the same group as their parent grids; thus no remote communication is needed between parent and children grids. There may be some boundary information exchange between sibling grids which usually is very small. During this step, load imbalance may be detected among groups at the finer levels; however, the global balancing process will not be invoked until the next time-step at level 0.

3. Problem and Motivation Statement

Most of the previous work on static load balancing considered as their main objective the minimization of overall expected response time. The fairness of allocation, which is an important issue for modern distributed systems, has received relatively little attention. In this chapter we consider the load balancing problem in single class job distributed systems. Our goal is to find a formal framework for characterization of fair load balancing schemes that are also optimal for each job. Using this framework we formulate the load balancing problem in single class job distributed systems as a support vector machine in computers. We show that the FIFO with SVM provides a Pareto optimal operation point for the distributed system. We give a characterization of SVM and an algorithm for computing it. We prove that the FIFO with SVM is a fair solution and we compare its performance with other existing solutions. SVM guarantees the optimality and the fairness of allocation.

4. System Model

Now in our system model we try to precise the SVM mathematically and for this tutorial we try to present a linear SVM. The goals of SVM are separating the data with hyper plane and extend this to non-linear boundaries using kernel trick [8][11]. For calculating the SVM we see that the goal is to correctly classify all the data. For mathematical calculations we have,

[a] If $Y_i = +1$; $wx_i + b \geq 1$
[b] If $Y_i = -1$; $wx_i + b \leq 1$
[c] For all i; $y_i (w_i + b) \geq 1$

In this equation x is a vector point and w is weight and is also a vector. So to separate the data [a] should always be greater than zero. Among all possible hyper planes, SVM

selects the one where the distance of hyper plane is as large as possible. If the training data is good and every test vector is located in radius r from training vector. Now if the chosen hyper plane is located at the farthest possible from the data [12]. This desired hyper plane which maximizes the margin also bisects the lines between closest points on convex hull of the two datasets. Thus we have [a], [b] & [c].

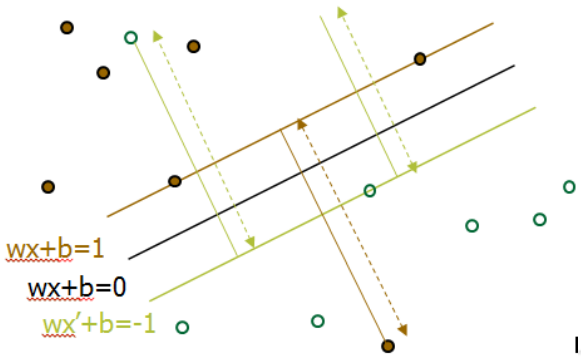

Figure 2: Representation of Hyper planes. [9]

Distance of closest point on hyper plane to origin can be found by maximizing the x as x is on the hyper plane. Similarly for the other side points we have a similar scenario. Thus solving and subtracting the two distances we get the summed distance from the separating hyper plane to nearest points. Maximum Margin = M = 2 / ||w||.Now maximizing the margin is same as minimum [8]. Now we have a quadratic optimization problem and we need to solve for w and b. To solve this we need to optimize the quadratic function with linear constraints. The solution involves constructing a dual problem and where a Langlier's multiplier $\alpha_i$is associated. We need to find w and b such that $\Phi(w) = \frac{1}{2} |w'||w|$ is minimized;
And for all $\{(x_i, y_i)\}$: $y_i (w * x_i + b) \geq 1$.
Now solving: we get that $w=\Sigma\alpha_i * x_i b = y_k - w * x_k$for any $x_k$such that $\alpha k \neq 0$
Now the classifying function will have the following form:
$f(x) = \Sigma\alpha_i y_i x_i * x + b$

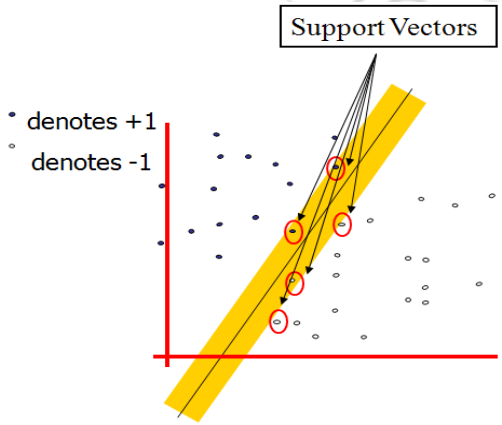

Figure 3: Representation of Support Vectors

5. Proposed Method

A load balancing algorithm should strive to achieve the following (often conflicting) goals:

• **Reduce the load imbalance.** To provide the best quality of service, every node would have the same utilization. Furthermore, for resources with a well-defined cliff in the load-response curve, it is of primary importance that no node's load is above the load at which the cliff occurs. We can take this point to be the capacity of the node.

• **Minimize the amount of load moved.** Moving a large amount of load uses bandwidth and may be infeasible if a node's load changes quickly in relation to the time needed to move objects.The flow control of this scheme is Shown in Figure 4. Here, time(i) denotes the iterative time for level i, and dt(i) is the time-step for level i. The left part of the figure illustrates the global load balancing step, while the right part represents the local load balancing step. Following each time-step dt (0) at level 0, there are several smaller time-steps dt(i) at a finer level i until the finer level i reaches the same physical time as that of level 0. Figure 5 illustrates the executing points of global balancing and local balancing processes in terms of the execution order given in Figure 2. It is shown that the local balancing process may be invoked after each smaller time-step while the global balancing process may be invoked after each time-step of the top level only. Therefore, there are fewer global balancing processes during the run-time as compared to local balancing processes.

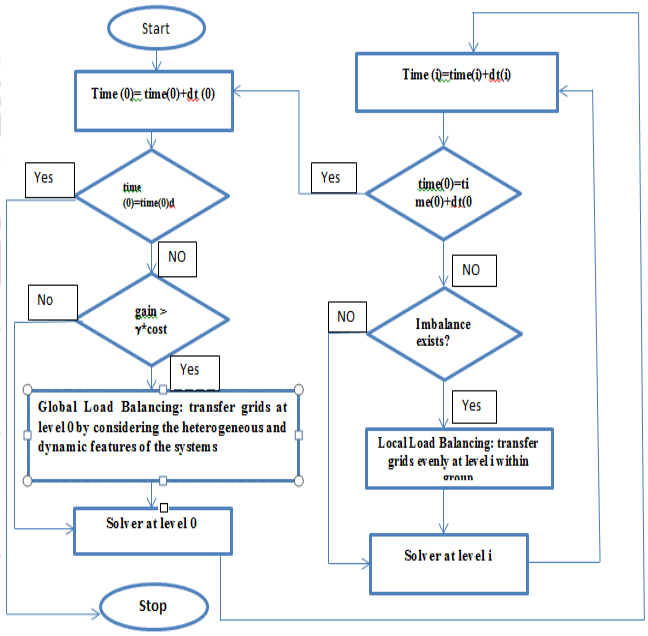

Figure 4: Flow Chart of Load Balancing

Training sets of support vector machine for distributed load balancing

- Input/output sets X, Y
- Training set (x 1, y 1)...(x m, y m)
- "generalization": given a previously seen x ∈X , find a suitable y ∈ Y.
- i.e., want to learn a classifier: y = f (x, α), where α are the parameters of the function.
- For example, if we are choosing our model from the set of hyper planes in R n, then we have: f (x, {w, b }) = sign (w · x + b).
- We can try to learn f (x, α) by choosing a function that performs well on training data:

$$R_{emp}(\alpha) = \frac{1}{m}\sum_{i=1}^{m} \ell(f(x_i, \alpha), y_i)$$ .....

…..Error of training

Where l is the zero-one loss function, l (y, yˆ)=1, if y 6= ˆy, and 0 otherwise. R_{emp} is called the empirical risk.

• By doing this we are trying to minimize the overall risk:

$$R(\alpha) = \int \ell(f(x, \alpha), y) dP(x, y)$$ .....

…… Test Error

Where P(x,y) is the (unknown) joint distribution function of x and y.

Classification of SVM in our work

1. Classification in SVM is an example of Supervised Learning. Known labels help indicate whether the system is performing in a right way or not. This information points to a desired response, validating the accuracy of the system, or be used to help the system learn to act correctly.
2. A step in SVM classification involves identification as which are intimately connected to the known classes. This is called feature selection or feature extraction. Feature selection and SVM classification together have a use even when prediction of unknown samples is not necessary.

The distributed system based load balancing support vector machine algorithm works as follows. The training set of the algorithm is split into subsets.

Each set within a distributed system classifies sub set locally via SVM algorithm and gets α values (i.e. support vectors (SVs)), and then passes the calculated SVs to global SVs to merge them. In load stage of load balancing job, the subset of training set is combined with global support vectors.

In Load balancing, the merged subset of training data is evaluated. The resulting new support vectors are combined with the global support vectors in Distributed system.

The algorithm can be explained as follows. First, each set in a distributed system reads the global support vectors set, then merges global SVs set with subsets of local training set and classifies using SVM algorithm. Finally, all the computed SVs set in load are merged. Thus, algorithm saves global SVs set with new ones. Our algorithm consists of the following steps. We showed our terminology at Table 1

Table 1: The notation we used in our work.

Notation	Description
T	Iteration number
T	Number of computers or Load size
h^t	Best hypothesis at iteration t
D_l	Sub data set at computer l
SV_l	Support vectors at computer l
SV_{global}	Global support vector

1) As initialization, the global support vector set as t= 0, $SV^t=\phi$
2) t = t + 1;
3) For any computer in l,l =1,……,L reads global SVs and merge them with subset of training data.
4) Train SVM algorithm with merged new set
5) Find out support vectors
6) After all computers in cloud system complete their training phase, merge all calculated SVs and save the result to the global SVs
7) If $h^t = h^{t-1}$ stop, otherwise go to step 2

For training SVM classifier functions, we used SVM with various kernels. Appropriate parameters and values were found by cross validation test.

All system is implemented with MATLAB

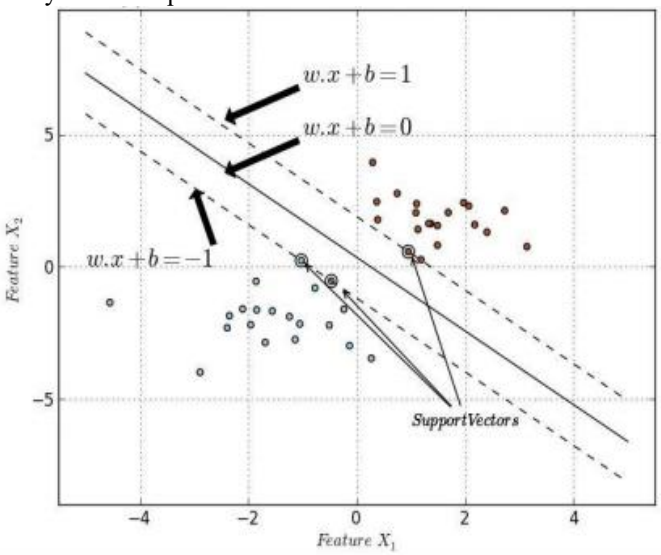

Figure 5: Classification of an SVM with Maximum-margin hyper plane trained with samples from two classes.

FIFO (First In First Out) Process with SVM

This block stores four types of data describing the training set, thus comprising of four memory sub-blocks as shown in Figure 6 (a). The first memory sub-block stores the complete 20 training set in the form of a matrix of a number of rows equal to the number of support vectors (SVs) and a number of columns equal to the number of features or dimensions (M).

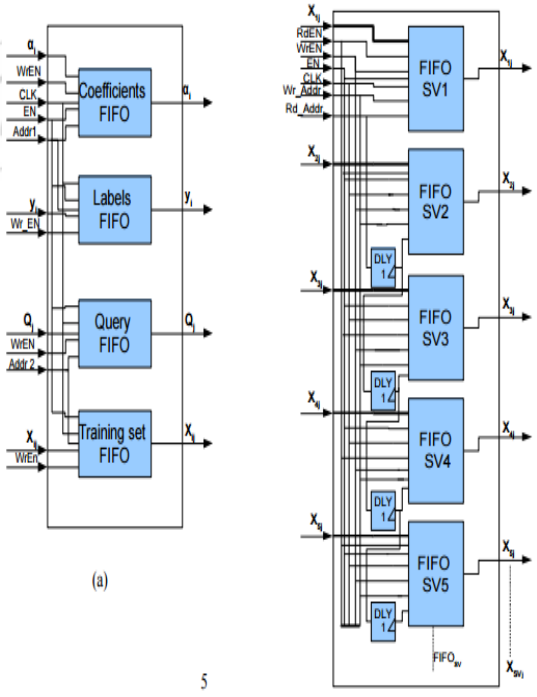

Figure 6: memory block of the SVM classifier

(a) illustrates the 8 components constituting the memory block highlighting that there are four different storage 9 sub-blocks to service the SVM classifier, and (b) illustrates the components of the Memory responsible for storing the training set in the SVM architecture

These are implemented as a set of FIFOs of a number equal to the number of SVs having a depth of M each, and a width of B (WL of each feature). The second memory sub-block is a FIFO used to store the class labels of the given SVs.

When these SVs are small, the associated 2 labels are stored inside the FPGA internal registers instead of wasting a complete Block RAM 3 due to the fact that each class label requires one bit only (to represent binary class labels).

Since M is expected to be significantly large for this architecture, the third memory sub-block is used as a FIFO to store the features of the query. Note that multiple queries can be stored 6 in this memory if enough Block RAMs are available. As for the fourth memory sub-block, it is used to store the training coefficients (αi's) computed off-line during the training phase, 8 and has a depth equal in number to the SVs.

6. Result

The ultimate objective of a distributed load balancing algorithm is to improve system performance. Therefore, a load balancing algorithm should adopt a performance index by which this performance improvement is measured. Since there is more than one index that can be utilized, the selection usually differs from one algorithm to another. Weexecuted SVM on a parallel machine with that executed on a distributed system. It is well-known that the distributed system will have a larger execution time than the parallel system with the same number of processors because of the performance of the SVM used to interconnect the machines in a distributed system. SVM generally have much larger latency than theinterconnects found in parallel machines. Our SVM attempts to reduce this overhead to make distributed systems more efficient. The experiments use small numbers of processors to illustrate the concepts, but it is assumed that in practice the distributed system will have a large number of processors to provide the needed compute power, which is beyond any single, available parallel machine.

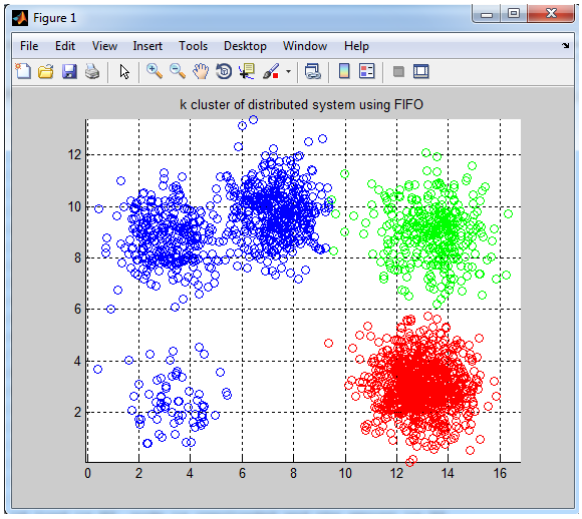

Figure 7: Number of cluster of distributed system using FIFO

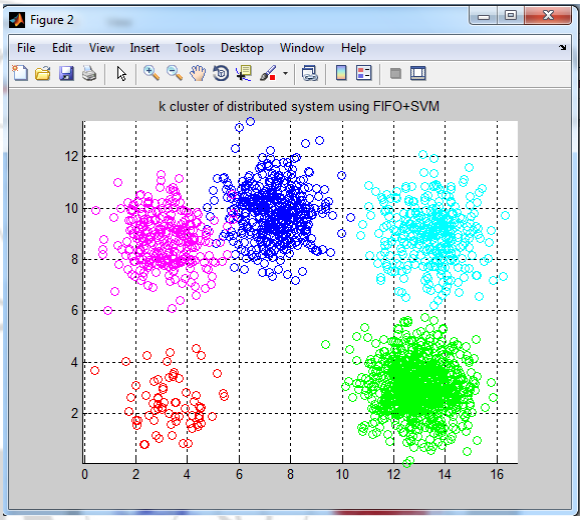

Figure 8: Number of cluster of distributed system using FIFO with support classifier

The load in clusters

for node 1 load is 204 node is overloaded and the amount is 154
for node 2 load is 226 node is overloaded and the amount is 176
for node 3 load is 32 node is balanced
for node 4 load is 228 node is overloaded and the amount is 178
for node 5 load is 158 node is overloaded and the amount is 108
for node 6 load is 24 node is balanced
for node 7 load is 70 node is overloaded and the amount is 20
for node 8 load is 137 node is overloaded and the amount is 87
for node 9 load is 239 node is overloaded and the amount is 189
for node 10 load is 241 node is overloaded and the amount is 191
for node 11 load is 39 node is balanced
for node 12 load is 243 node is overloaded and the amount is 193
for node 13 load is 239 node is overloaded and the amount is 189

for node 14 load is 121 node is overloaded and the amount is 71
for node 15 load is 200 node is overloaded and the amount is 150
for node 16 load is 35 node is balanced
for node 17 load is 105 node is overloaded and the amount is 55
for node 18 load is 229 node is overloaded and the amount is 179
for node 20 load is 240 node is overloaded and the amount is 190
for node 1 load is 138 node is overloaded and the amount is 88
for node 3 load is 8 node is balanced
for node 4 load is 154 node is overloaded and the amount is 104
for node 5 load is 91 node is overloaded and the amount is 41
for node 6 load is 12 node is balanced
for node 7 load is 122 node is overloaded and the amount is 72
for node 8 load is 48 node is balanced
for node 9 load is 31 node is balanced
for node 10 load is 51 node is overloaded and the amount is 1
for node 11 load is 37 node is balanced
for node 12 load is 47 node is balanced
for node 13 load is 11 node is balanced
for node 14 load is 159 node is overloaded and the amount is 109
for node 15 load is 70 node is overloaded and the amount is 20
for node 16 load is 135 node is overloaded and the amount is 85
for node 17 load is 174 node is overloaded and the amount is 124
for node 18 load is 125 node is overloaded and the amount is 75
for node 19 load is 134 node is overloaded and the amount is 84
for node 20 load is 111 node is overloaded and the amount is 61
for node 1 load is 184 node is overloaded and the amount is 134
for node 2 load is 16 node is balanced
for node 3 load is 215 node is overloaded and the amount is 165
for node 4 load is 234 node is overloaded and the amount is 184
for node 5 load is 246 node is overloaded and the amount is 196
for node 6 load is 215 node is overloaded and the amount is 165
for node 7 load is 196 node is overloaded and the amount is 146
for node 8 load is 128 node is overloaded and the amount is 78
for node 9 load is 44 node is balanced
for node 10 load is 100 node is overloaded and the amount is 50
for node 11 load is 33 node is balanced
for node 12 load is 8 node is balanced
for node 13 load is 235 node is overloaded and the amount is 185
for node 14 load is 75 node is overloaded and the amount is 25
for node 15 load is 74 node is overloaded and the amount is 24
for node 16 load is 83 node is overloaded and the amount is 33
for node 17 load is 117 node is overloaded and the amount is 67
for node 18 load is 162 node is overloaded and the amount is 112
for node 19 load is 6 node is balanced
for node 20 load is 211 node is overloaded and the amount is 161
Total number of overloaded node in Distributed System 554
Total Efficiency for Load Balancing is 70
Elapsed time is 16.507189 seconds.
Accuracy for Distributed environment using FIFO = 8.4704%
Accuracy for Distributed Environment using SVM+FIFO = 90.8470%

7. Conclusion

This research evaluates a SVM to balance load with distributed system.Load balancing algorithms is totally dependent upon in which situations workload is assigned, during compile time or execution time. Above comparison shows that static load balancing algorithms are more stable than normal distributed system. But dynamic load balancing algorithms are always better than static as per as overload rejection, reliability, adaptability, cooperativeness, fault tolerant, resource utilization, response & waiting time and overall accuracy of system.

References

[1] Abhijit A. Rajguru, S.S. Apt" A Comparative Performance Analysis of LoadBalancing Algorithms in Distributed System using Qualitative Parameters"International Journal of Recent Technology and Engineering (IJRTE) ISSN:2277-3878, Volume-1, Issue-3, August 2012.
[2] Abirami M S1, Niranjana G2," Dynamic Load Balancing in DistributedSystems", International Conference on Computing and Control Engineering(ICCCE 2012), 12 & 13 April, 2012.
[3] Ali M. Alakeel, "Load Balancing in Distributed Computer Systems", International Journal of Computer Science and Information Security, Vol. 8, No. 4, 2010.
[4] Jasma Balasangameshwara, Nedunchezhian Raju, "A Decentralized Recent Neighbour Load Balancing Algorithm for Computational Grid", The International Journal of ACM Jordan (ISSN 2078-7952), Vol. 1, No. 3, September, 2010.
[5] Ms.NITIKA, Ms.SHAVETA, Mr. GAURAV RAJ; "Comparative Analysis of Load Balancing Algorithm in Cloud Computing" International Journal of Advance Research in Computer Engineering & Technology Volume 1, 2012
[6] Parveen Jain, Daya Gupta, "An Algorithm for Dynamic Load Balancing in Distributed Systems with Multiple

Supporting Nodes by Exploiting the Interrupt Service", ACEEE, ACADEMY PUBLISHER, Delhi College of Engineering, New Delhi, 2009.

[7] Quang Hieu Vu, Member, IEEE, Beng Chin Ooi, Martin Rinard, and Kian-Lee Tan, Histogram-Based Global Load Balancing in Structured Peer to Peer Systems, Knowledge and Data Engineering, IEEE Transaction,April 2009, Volume:21, Issue:4, Pages:595-608.

[8] Wenqiu Zeng, Ying Li, Jian Wu, Qingqing Zhong, Qi Zhang, Load rebalancing in Large-Scale Distributed File System, Information Science and Engineering (ICISE), 2009 1st International Conference, Dec 2009, Pages:265- 269.

[9] Belabbas Yagoubi and Meriem Meddeber "Distributed Load Balancing Model for Grid Computing", Revue ARIMA, vol. 12, pp.43 -60 2010 .

[10] A. Menendez LC and H. Benitez-Perez "Node Availability for Distributed Systems considering processor and RAM utilization for Load Balancing", Int. J. of Computers, Communications & Control, vol. V, no. 3, pp.336 -350 2010

[11] Kabalan K Y, Smari W and Hakimian J Y. Adaptive Load Sharing in Heterogeneous System: Policies, Modifications and Simulation.CiteSeerx, 2008.

[12] J.P.Lewis, Tutorial on SVM, CGIT Lab, USC, 2004.