Journal of Theoretical and Applied Information Technology
10th September 2015. Vol.79. No.1
© 2005 - 2015 JATIT & LLS. All rights reserved·

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

EFFICIENT ROUTING OF LOAD BALANCING IN GRID COMPUTING

MOHAMMAD H. NADIMI-SHAHRAKI*, FARAMARZ SAFI, ELNAZ SHAFIGH FARD

Department of Computer Engineering, Najafabad branch, Islamic Azad University, Najafabad, Iran
nadimi@iaun.ac.ir, fsafi@iaun.ac.ir, shafighfard@azaruniv.edu

ABSTRACT

In this decade, grid computing is a well-known solution for applying a large collection of connected heterogeneous systems and sharing various combinations of resources. It creates a simple but large, powerful and self-managing virtual computer, which leads to the problem of load balancing. The main goal of load balancing is to provide a distributed and low cost scheme that balances the load across all the processors. In this paper, a new load balancing algorithm named optimal anti-directed chord is proposed. In this overlay structured network that load information and processes among nodes have been organized for tradeoffs (load balancing), reaching time to a particular process or entity is reduced because of anticlockwise movement. Experimental results show that the proposed method reduces reaching time by 33% in comparison to simple chord and 16% ABC method.

Keywords: *Load Balancing, Grid Computing, Anti-Clockwise-Direction Chord.*

1. INTRODUCTION

Recent researches in computing architectures allowed the emergence of a new computing paradigm known as Grid computing. Grid is a type of distributed system which supports the shared and coordinated use of resources, independent of their physical type and location. This technology allows the use of geographically distributed and multi-owned resources to run large-scale applications like meteorological Simulations, data intensive applications, DNA research, and very important projects like SETI@home [1]. Grid computing [2] is a type of parallel and distributed system that enables the dynamic distribution, selection and aggregation of geological resources in run time depending on their availability, capability, performance, cost, and user quality of self-service requirements. In Grid computing, individual users can retrieve computers and data transparently without taking into account the location, operating system, account administration, and other details. Furthermore, in Grid computing, the details are abstracted and the resources are virtualized.

Reaching time is a very important factor in distributed systems. Considering this factor in chord [3] that is a structure looking for an item in clockwise direction, reaching time might suffer. Reaching time is important because it can influence the response time. Therefore, in this paper an efficient algorithm named optimal anti-directed chord is proposed to reduce the looking time to have a good response time. This algorithm firstly determines whether the search must be done clockwise or anti-clockwise. In fact the main objective of this work is reducing the reaching time in comparison to chord and ABC method which is a kind of P2P overlay network.

The rest of paper is organized as follows. In section 2, background of load balancing and its algorithms are considered. In section 3, related works are presented. In section 4, our proposed anti–directed chord algorithm is introduced. In section 5, we present the evaluation of proposed algorithm and its comparison with the simple chord and ABC method that dramatically reduces reaching time (hopes) in entities and processes among nodes in grid computing. Finally, the last section is to discuss about conclusions and future works.

2. RELATED WORK

The goal of load balancing is to fully utilize the computing power of multiple hosts without disturbing the user and improve the overall performance, regardless of the number of hosts available in the background. Besides, load balancing aims to ensure that the workload is fairly distributed among the nodes and that none of the nodes are overloaded or under loaded. Basically, there are two load balancing strategies which are static load balancing and dynamic load balancing. Static load balancing [4], [6] makes the balancing decision at compile time, and will remain constant.

However, the dynamic load balancing makes more informative decisions in sharing the system load based on runtime state. Comparatively, dynamic load balancing has the potential to provide better performance than static load balancing. Dynamic load balancing [5], [6] is based on runtime state, and needs to process the collected information with firm procedures. The balancing procedures are placed in the dynamic load balancing policy. It contains a set of rules referred by the system to run and to employ dynamic load balancing for better performance.

The distributed hash tables or DHT is the basic core of the load balancing. The authors developed algorithms that completely rely on the implementation of the underlying DHT without making any programmable change to it. They are using CHORD in their example. CHORD was one of the first which used the virtual servers to improve node imbalance. Many literatures aim to improve the routing load on structured P2P based on Chord algorithm such as a collaborative file system CFS [7], low latency and high throughput user net DHT [8], Back-Up chord for P2P file sharing over MANETs [9], a geographic hash table (GHT) for data-centric storage [10], low overhead Usenet server User net DHT. In [11], M. Bienkowski et al utilize methods to manipulate the peer ID generating procedure to ensure peers' interval lengths differ at most by a constant factor to mitigate one of the designing issues. In [12], P. B. Godfrey and I. Stoica use virtual server's instantiated by physical node to act as peers in the network. Once a node becomes heavily loaded, it transfers some of its virtual servers to a proper node with fewer loads. It also imports virtual servers [13]. However, above proposals suffer from high maintenance overhead and extra complexity. Paper [14] presents a different way by selecting finger peer dynamically among certain local area. This simple enhancement of finger selection mechanism improves load fairness on Chord substantially and in last work [15] B. Hu, X. Zhang and X. Zhang called ABC method that is a dynamic load balancing mechanism, It also has the advantage of average 1-2 hops of lookup and better effect of load balancing.

Although there have many works have been proposed to distribute workload fairly among the nodes, they do not fulfill the need of short reaching time. Therefore, in the next section Chord protocol is investigated and then an efficient algorithm is proposed to reduce the reaching time in comparison to chord and ABC method.

3. ANTI-CLOCKWISE CHORD PROTOCOL

3.1 Chord Protocol

The Chord protocol supports just one operation: given a key, it maps the key onto a node. Depending on the application using Chord, that node might be responsible for storing a value associated with the key. Chord uses a variant of consistent hashing to assign keys to Chord nodes. Consistent hashing tends to balance load, since each node receives roughly the same number of keys, and involves relatively little movement of keys when nodes join and leave the system.

Chord protocol is a structured p2p that can balance processes or entities among all nodes which cooperate with each other. The paper shows how chord can work.

3.2 Chord Structure

Chord improves the scalability of consistent hashing by avoiding the requirement that every node should know about every other node. A Chord node needs only a small amount of "routing" information about other nodes. Because this information is distributed, a node resolves the hash function by communicating with a few other nodes. As shown Fig 1, in an N-node network, each node maintains information only about O (logN) other nodes, and a lookup requires messages. Chord must update the routing information when a node joins or leaves the network; a join or leave requires O (log n^2N) messages.

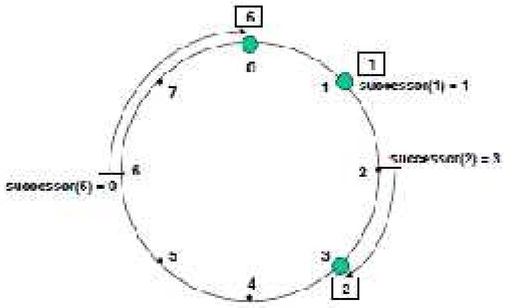

Figure1. Fingers Sketch

A very small amount of routing information suffices to implement consistent hashing in a distributed environment. Each node needs to be only aware of its successor node on the circle. Queries for a given identifier can be passed around the circle via these successor pointers until they first encounter a node that succeeds the identifier;

this is the node the query maps to. A portion of the Chord protocol maintains these successor pointers, thus ensuring that all lookups are resolved correctly. However, this resolution scheme is inefficient: it may require traversing all nodes to find the appropriate mapping. To accelerate this process, Chord maintains additional routing information. This additional information is not essential for correctness, which is achieved as long as the successor information is maintained correctly. As is shown in the Fig 1, successor is a node that stores information about other nodes; for example, node 0 is 6's successor.

4. OPTIMAL ANTI-DIRECTED CHORD

Chord offers an efficient structure for load balancing. Besides, it can lookup item from their key among other nodes. However, pure chord is not good because it is in one direction and its reaching time can be longer. Therefore an Anti-Clockwise Direction Protocol is proposed to solve this problem even in clockwise direction. It is expected that its results are equal to those of pure chord. This protocol is as follows:

1- In optimal chord, in all situations such as crashing, failure or leaving node, mobile agents present in any node can send messages to any successor and predecessor node to update their finger table.

2- For solving the second problem, second finger table must be created for anticlockwise direction according to some rules.

4.1 Assumptions

For using this protocol, we assume that:

- Each node n' maintains a routing table with up tom entries (the number of bits in identifiers), called finger table.

- The ith entry in the table at node n contains the identity of the first node s that succeeds n by at least 2^{i-1} on the identifier circle.

- To look up O (logN), messages must be exchanged.

- For adding or removing a node from the network, it can get O (loglogN) messages.

4.2 The hypothesis

There is a hypothesis as follows: to reach to the key of any data item, there are two routes, one of which is related to the position of start node that can be more optimal than the other one.

- Before finding the key, algorithm under codes must be attached: The maximum active number (2^m) div 2=j

- If (key - node start)> j => looking up must move in anti-clockwise direction.

Fig 2 is to illustrate this hypothesis for looking up k54 in clockwise direction, there are three hopes, but in anti-clockwise direction, there is one hope. So, in one of the directions, the number of hopes can be decreased. Thus, to achieve this optimal reaching time, we need two finger tables: first anticlockwise direction finger table, and second, clockwise direction finger table.

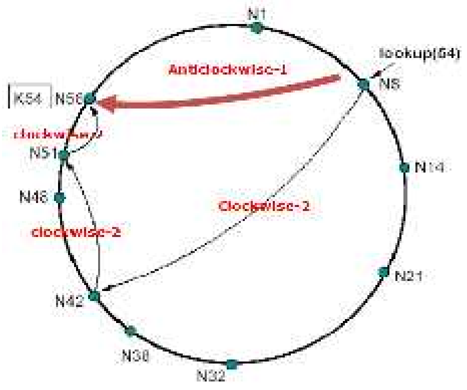

Fig 2. The Number Of Hopes For Looking Up K54 In Clockwise Direction Compared With Anti-Clockwise Direction

4.3 Creating anticlockwise direction

There are following steps to create the anticlockwise direction.

- Find m as the count of maximum number of identifier rings.

- Result = N –2^ (k-1), 1<=k<=m

- If result is negative then N= 2^m +result

To find key in clockwise direction, the start node tries to find address node which is the maximum in its finger table smaller than key. However, in anti-clockwise direction, the start node looks for another node in its finger table that is bigger, and the closest one to key. Table 1 shows new finger table for optimal chord. As chord protocol shown, in finger table, successor and predecessor are very important for look up keys in their finger table.

- Finger [k]: first node on circle that succeeds (n+2^ (k-1)) mod 2^m, 1<=k<=m

- Successor: the next node on the identifier circle: finger [1].node.

- Predecessor: the pervious node in the identifier circle.

Table1. Enhanced Finger Table For Optimal Chord

Finger table For anticlockwise	
N8-1	N7
N8-2	N6
N8-4	N4
N8-8	N0
N8-16	N56
N8-32	N40

In summarize, there are two following main differences between Anti-clockwise Algorithm and Clockwise Algorithm.

1- After considering m (identifier bit) in clockwise chord, active id node +2^ (k-1), reverse in anti-clockwise chord active id node, is subtracted from 2^ (k-1) because our chord tries to look up entity or processes that are anti-clockwise.

2- In anti–clockwise chord, it is important that the value of upper bound interval node be put in other nodes. But in clockwise chord, it is important that the value of lower bound interval node be put in other nodes.

5. EXPERIMENTAL EVALUATION

5.1 Experimental Setup

Experiments have been done in Mini laptop with the following properties: intel®Atom™ Cpu N270 @ 1.60GHz with memory of 1.00GB. Matlab is adaptive software for this kind of algorithms that need considerable computing function. Then, the algorithm has been implemented in such an environment.

5.2 The Number Of Hopes

In the first experiment, to prove that anti-clockwise has less hopes than simple chord in reaching the same point, we consider a distributed system in which m=7, the count of active nodes is 10 ,and from lower to upper active, nodes are [32,40,52,60,70,79,80,85,102,113]. The experiment was repeated four times. The input parameters were nodes, and they were independent variables. The numbers of hopes comprise our output. The variables were dependent and relative. Fig 3shows that, to reach node 39 from node 70 in anti-clock wise chord (by the propose algorithm), we need just one hope because by reaching to node 40 (active node), node 39 is saved there in clockwise chord with two hopes: 32 and 40. Then we can reach to node 39. As shown in Fig 4, the first experiment is run again for reaching to node 65 from node 80. In the proposed algorithm, node 70 should save information about node 65 in system load balancing. This is because, with calculation of node 65, we need two hopes. But in clockwise chord from node 80, node 32 must be met, after that node 70, and then node 65. So, in this process, there is one hope more than the number of hopes in anti-clock wise chord.

In the second experiment, our identifying space has 6 bits, and our input parameters (active nodes) are [1, 8, 14, 21, 32, 38, 42, 48, 51, and 56]. Like the previous experiment, nodes are independent variables, and the number of hopes is our output. Also, the variables are dependent and relative. In this evaluation, in anti-clockwise direction, reaching from N54 to N8 with 1 hope is possible. On the contrary, in clockwise direction, the process is possible only with 3 hopes because N56 contains the information about N54. Fig 5 shows that, to reach to node 54 from node 8 in by using the proposed algorithm, we need 1 hope. But in clockwise direction chord, we need 3 hopes.

5.3 The Effectiveness Of Proposed Method

The goal of this experiment is to evaluate the effectiveness of proposed method and effect of load balancing. We have implemented both Chord and ABC and in each experiment, addition load are timed by a random Poisson process ranged in [1...1000]. The network delay is set to 10. The main operations of our Mechanism are as follows.

In the third experiment, we investigate the efficiency of proposed work in compared to ABC algorithm and chord method. We study the average lookup hops with varying the initial number of peers at different rate of joining peers. The load is varying with inserting 5 objects per second. Fig 6 shows the average lookup hops with peers in the steady state. We observe that the average lookup hops is 1-2 hops round from Fig 6, the reason is that every peer has almost the whole ring routing information. Thus, the average lookup is obviously shorter than O (logn) in Chord protocol.

In the fourth experiment, we consider a network consisting of 1,000 peers and the load distributed in [1 ... 1,000] by random. For each value, we pick up the average of three times in experiments. To assess the performance of proposed work, we compare our algorithm with ABC and Chord algorithm. Fig. 7 shows the maximum number of loads under continuing to insert objects while peers are in steady state.

6. CONCLUSION AND FUTURE WORK

In this paper, the concept of grid computing and the challenge of load balancing has been addressed. In load balancing, we showed that chord structured system has a very important role in representing load balancing information among nodes. This report proposed anti-clockwise direction lookup algorithm based on the Open Chord simulation platform. We built anti finger table by changing some algorithms derived from clockwise chord that we simulated in Matlab. Our proposed method can reduce reaching hopes down to 1/3 times, about 33% improvement in comparison to simple chord and 16% improvement in compared to ABC method. The future works can be to improve the algorithm, and to implement lists of paths where the source node can be connected to target node by choosing the shortest path among its connected nodes using that list.

Acknowledgment: The authors would like to thank Islamic Azad University, Najafabad branch for supporting this research.

REFRENCES:

[1] I. Foster, and C. Kesselman, the Grid: Blueprint for a new Computing Infrastructure, 2nd ed.: Morgan Kauffman publishers, 2004.

[2] M. Bote-Lorenzo, Y. Dimitriadis, and E. Gomez-Sanchez, “Grid characteristics and uses: a grid definition,” in Proc. 1st European Across Grids Conference (ACG’03), 2004, pp. 291-298.

[3] I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan.Chord: A scalable peer-to-peer lookup service forinternet applications. In Proceedings of ACM SIGCOMM, 2001, pp.149–160.

[4] A.N.Tantawi, and D. Towsley, “ Optimal static load balancing in distributed computer systems,” Journal of Association for ComputingMachinery, Vol. 32, No. 2, April 1985, pp. 445-465.

[5] J. 1Xu, and K. Hwang, “Heuristic methods for dynamic load balancing in a message-passing multicomputer,” Journal of Parallel and Distributed Computing, Vol. 18, 1993 pp. 1-13.

[6] S. K. Goyal, R. B. Patel, and M. Singh, “Adaptive and dynamic load balancing methodologies for distributed environment: a review,”International Journal of Engineering Science and Technology (IJEST), vol. 3, no. 3, 2011, pp. 1835-1840.

[7] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris and I.Stoica, "Wide-area cooperative storage with CFS," in Proceedings of the eighteenth ACM symposium on Operating systems principles, 2001, pp. 202-215.

[8] F. Dabek, J. Li, E. Sit, J. Robertson, M. F. Kaashoek and R.Morris, "Designing a DHT for low latency and high throughput," in Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation - Volume 1, 2004, pp. 7.

[9] H. Jeong, D. Kim, J. Song, B. Kim and J. Park, "Back-Up chord: chord ring recovery protocol for p2p file sharing over MANETs," in Proceedings of the 5th international conference on Computational Science - Volume Part II, 2005, pp. 477-484.

[10] S. Ratnasamy, B. Karp, L. Yin, F. Yu, D. Estrin and R.Govindan, et al., "GHT: a geographic hash table for data centric storage," in Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications, 2002, pp. 78-87.

[11]M. Bienkowski, M. Korzeniowski, F. M. a. d. Heide, and F. M. Heide,"Dynamic Load Balancing in Distributed Hash Tables," in Proc.IPTPS, 2005, pp. 217-225.

[12] P. B. Godfrey and I. Stoica, "Heterogeneity and load balance in distributed hash tables," in INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies, vol. 1, 2005, pp. 596-606.

[13] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica," Widearea cooperative storage with CFS," SIGOPS Oper. Syst.Rev., vol. 35, 2001, pp. 202-215.

[14] R. Cuevas, M. Uruena, and A. Banchs, "Routing Fairness in Chord:Analysis and Enhancement," in Proc. INFOCOM 2009, IEEE, 2009, pp.1449-1457.

[15] B. Hu, X. Zhang and X. Zhang 2013 Fourth International Conference on Emerging Intelligent Data and Web Technologies, 2013.

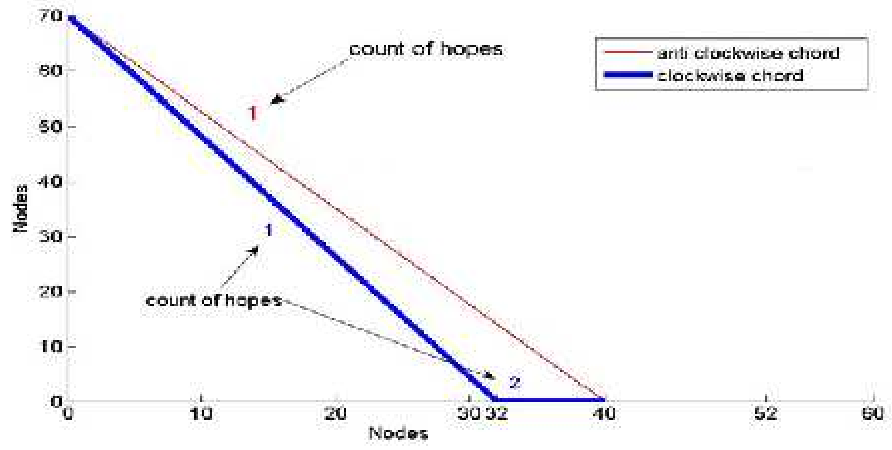

Fig 3. Reaching Hopes To Node 39 From Node 70.

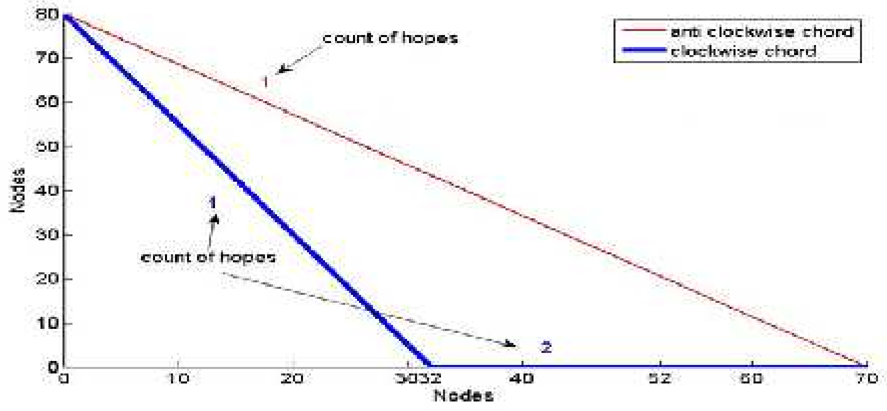

Fig 4. *Reaching Hopes To Node 65 From Node 80.*

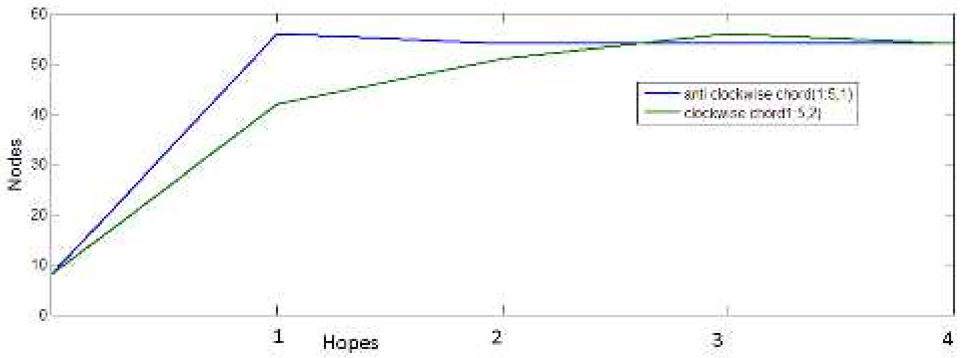

Fig 5. Reaching Hopes To Node 54 From Node 8.

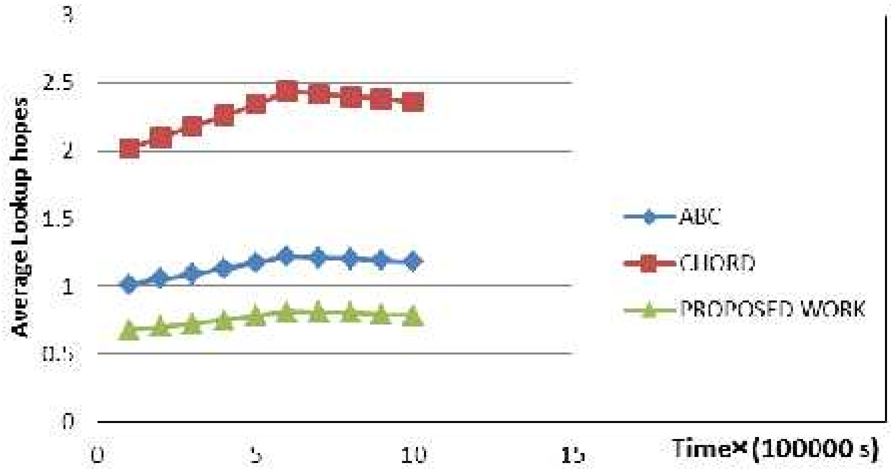

Fig 6. The Average Lookup Hops As Peers In Steady State.

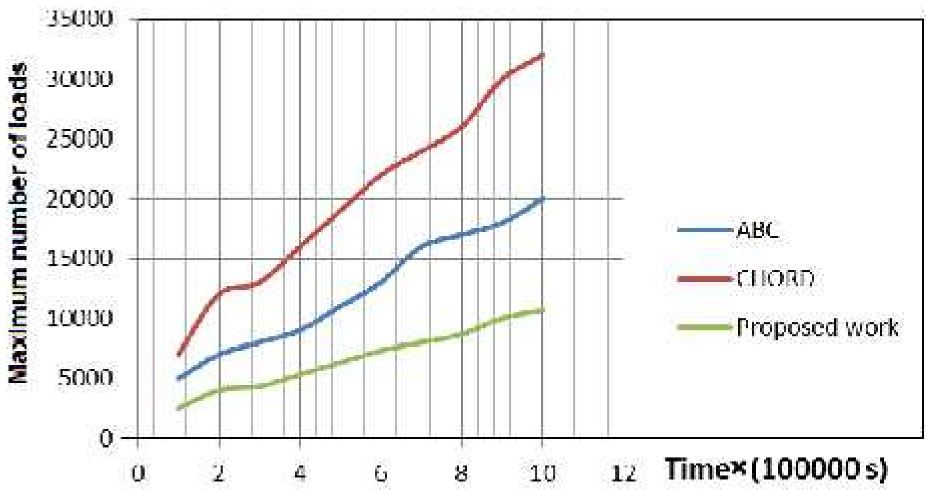

Fig 7. Maximum Number Of Loads Under Continuing To Insert Objects As Peers In Steady State.