CPI: A Novel Three-Phase Algorithm for QoS-Aware Replica Placement Problem[1]

Wei Fu[1], Yingjie Zhao[1], Nong Xiao[1], and Xicheng Lu[1]

[1] School of Computer, National University of Defense Technology, Changsha, P.R. China
lukeyoyo@tom.com

Abstract. QoS-aware replica placement decides how many replicas are needed and where to deploy them to meet every request from individual clients. In this paper, a novel three-phase algorithm, namely CPI, is proposed. By dividing candidate nodes into proper medium-scale partitions, CPI is capable to handle with large-scale QoS-aware replica placement problem. Pharos-based clustering algorithm obtains ideal grouping, and partition integrating method is developed to obtain final replica policy. Theoretical analysis and experiments show that CPI has lower computation complexity and good scalability. The replicating cost and updating cost remains acceptable under different simulating conditions.

1 Introduction

Replication is the process of sharing resources so as to ensure consistency between redundant copies. These copies are formally called replicas, which usually spread at geographically distributed locations. As a simple but effective technique, replication is widely employed in distributed systems [1-6]. Proper replica mechanism can speedup response time, reduce network traffic, balance overload, as well as enhance data reliability and fault-tolerance. Distributed databases [1], distributed file systems [2], content distributing network [3, 4], P2P systems [5] and Data Grids [6] are some of the most common scenarios to use replicas. Replica placement takes charge of a proper replica policy. It decides how many replicas should be deployed and where to locate them, which is very important to the effectiveness of replication.

Traditional replication researches aimed at optimize the global/average metrics as much as possible. For example, Qiu [7] tried to minimize the average accessing delay, and Cidon [8] aimed to spend the least communication messages. While an average performance measure may be important from the system's point of view, it does not differentiate the various performance requirements of the individuals [9]. With the rapid growth of time-critical applications, some researches [9-12] tried to provide QoS-guaranteed replica service. Instead of only concerning about average metrics, their first and foremost objective is to guarantee that EVERY individual request should meet its QoS requirement, usually response time. They named it as **Q**oS-

[1] This paper is supported by National 863 High Technology Plan (NO. 2006AA01A118, NO. 2006AA01A106), and Chinese NSF (NO.60573135, NO. 60736013).

Aware **R**eplica **P**lacement problem (QARP for short), which has been proved to be NP-Complete. Several heuristic algorithms have been presented to solve the problem, including Tang [9], Wang [10] and Fu [12]. However, they are all centralized methods and lack of scalability. Furthermore, the computation complexities are rather high. Theoretically, the time complexities are about $O(|V|^3)$ or even $O(|V|^{2l+2})$[9-11].

In order to overcome the difficulty, a novel three-phase algorithm CPI is presented to solve large-scale QoS-aware replica placement problem. CPI divides the entire problem into several medium-scaled problems. Then each sub-problem deals with its own placement problem in parallel. Finally, all sub-problem solutions are integrated to form a final solution. The main contributions of the paper are listed as follows:

1. A novel semi-distributed method CPI is proposed to solve large-scale QoS-aware replica placement problem;
2. A pharos-based algorithm are invented for node clustering;
3. A simple but effective integration mechanism is introduced to obtain global placement policy;

2 Related Work

In 2004, Tang and Xu put forward the QARP problem for the first time [9]. They proved the replica-aware QARP to be NP-complete. Meanwhile, two families of heuristic algorithms, named *l-Greedy-Insert* and *l-Greedy-Delete* respectively, are proposed for optimal solution. The selection of *l* reflects a tradeoff between the time complexity and the quality of solution. On the basis of their work, Jeon [11] gave another proof of NP-hard property. He deduced it to be a minimum set cover problem. With the help of matrix, a centralized algorithm based on the approximation algorithm for minimum set cover problem was presented. Wang [10] proposed another heuristic algorithm called *Greedy-Cover* inspired by set operations. Recently, Fu and Xiao et.al [12] utilized vector operations to accelerate computation. The output replica is organized into a ring structure for concurrent updating.

However, all these solutions are classified as centralized algorithms. A dominate node is required to collect communication cost between any two nodes, and the algorithm will be performed in this single node. If the scale of network is small or medium (e.g., < 1000), they work well. However, when the scale is a bit larger, the computation cost and memory cost will both increase sharply [10]. Either the time cost will be so long, or it will cause the out of memory exception. We can confirm this judgment from experimental results in section 6.

Generally speaking, all of them lack of scalability. And the loads are imbalanced. While the dominate node is over-used, all the other nodes are almost idle. The powerful capabilities are not exploited.

3 Replication System Model and QoS-aware Replica Placement Problem Definition

In this section, a replication system model is introduced. Some servers are selected to hold replicas, which are called replica nodes. The other servers are called non-replica nodes. In this context, the terms "server" and "node" are regarded as the same thing.

Let an undirected graph $G = (V, E)$ represent the server network, where V is the set of servers, $E \subseteq V \times V$ denotes the set of links between these nodes. Each node is identified by a global unique identifier. Without loss of generality, we use integer 0, 1, 2, …, n, where n = $|V|-1$. A storage function $s(v)$ is assigned to node v, representing for the storage cost when a replica resides on it. Besides, Let $d(u, v)$ denote the communication cost between a pair of nodes u and v. If $(u, v) \in E$, $d(u, v)$ means communication cost of the link between u and v. Otherwise, $d(u, v)$ equals to the smallest cost among all possible path from u to v.

QARP problem is defined on the basis of this model. Given an original data in a source node labeled by s, the objective of QARP is to find a subset of nodes R (i.e., $R \subseteq V - \{s\}$). When each server in R holds a copy from s, any of the other nodes can arrive at a replica node without violating its QoS restriction. At the same time, the replicating cost should be minimized. Figure 1 illustrates a typical graph with communication costs.

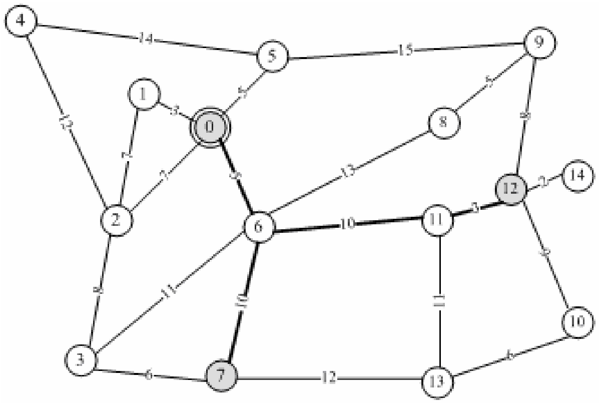

Figure 1. This is a graph with 15 nodes and 22 edges. Node 0 is the original server, and grey circles are replica nodes. Any node can reach to his nearest replica node within a distance restrict of QoS $\leqslant$ 19. Thick paths show the updating distributing tree rooted by 0, with an updating cost of 5+10+10+3 = 28.

The replicating cost of R is calculated by the following equation:

$$Cost(R) = \lambda \cdot Storage(R) + (1-\lambda) \cdot \mu \cdot Update(R) \quad (1)$$

where λ is a relative weight. Let μ be the update rate of data, $Storage(R)$ and $Update(R)$ are respectively represented by the follows:

$$\mathit{Storage}\ (R) = \sum_{v \in R} s(v) \tag{2}$$

$$\mathit{Update}\ (R) = \sum_{(u,p(u)) \in T} d(u, p(u)) \tag{3}$$

In equation (3), T denotes an update distribution tree [13] rooted by s, and token (u, $p(u)$) $\in T$ means that u and its parent node $p(u)$ are a pair of successive nodes in the tree T.

4 Three-phase Placement Algorithm

As discussed above, most of existing algorithms are centralized solutions. The lack of scalability and the rather high computation complexity make them incapable to solve a large-scale QARP problem. In this section, a novel QoS-aware replica placement algorithm CPI is introduced. Namely, the algorithm consists of three phases, illustrated by the pseudo-code in Figure 2.

```
**Input**:  G = (V, E); QoS; s
**Output**: Policy P; Update distributing tree T
Begin
    //Phrase 1: Node grouping
    Find 3 pharoses with Pharos Electing Policy;//Sec. 4.2
    Group all nodes into N Clusters $V_1, V_2, …, V_N$; //Sec. 4.1
    //Phrase 2: Find local replica placement policy
    For each Cluster $V_i$
        Find local replica policy $P_i$;   //Sec. 4.3
        Construct local update tree $T_i$; //Sec. 4.4
    End for;
    //Phrase 3: Integration of all local policies
    P = $P1 \cup P_2 \cup … \cup P_N$;
    Construct T from $T_1, T_2, …, T_N$;//Sec. 4.4
End
```

Figure 2. The pseudo-code of algorithm CPI: **C**lustering, **P**lacing and **I**ntegrating.

4.1 Pharos-based Clustering Algorithm

To divide all nodes into different clusters, the principle to be followed is that:

- If two nodes are close to each other, they should be in the same cluster;
- If two nodes are far away from each other, they should be in different clusters.

Therefore a technique is urgently needed to distinguish whether two nodes are close to or far away from each other. The idea is inspired by GPS [14]: the Global Positioning System. A typical GPS receiver can easily calculate its 3-D coordinate position using the distances to four or more GPS satellites. In our algorithm, since the graph G is in a 2-D coordinate system, it is easily to understand that 3 "satellites", here we called them pharoses[2], are enough to position a node.

The basic idea of the pharos-based clustering algorithm is to find out which nodes are close to each other, and then classify them into one cluster. As an simple example illustrated by Figure 3.

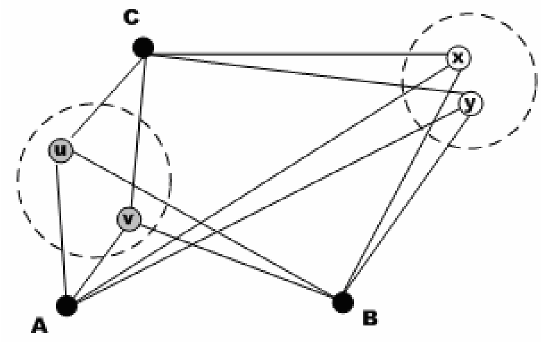

Figure 3. With the help of 3 pharoses A, B and C, it can be found out that node u is close to v, while x and y are far away from u or v. Thus, four nodes are clustered into two clusters, as the dashed circles shows.

Another important issue is how many clusters should be generated. Let N denote the number of clusters. The choice of N is a tradeoff between local replica placement cost and integrating cost. Generally, we decide N by controlling each cluster's size to be medium-sale. Therefore, N can be calculated by the following expression:

$$N = \lceil |V| / k \rceil \tag{4}$$

Where k is an integer, equal to the scale we want each cluster to be.

The clustering algorithm is illustrated by the pseudo-code in Figure 4.

```
Input:  $G=(V, E)$; Pharoses set{p1, p2, p3}; # of cluster: $N$
Output: Subsets $V_1, V_2, \cdots, V_N$
1.  Begin
      //Initialize $N$ subsets
2.    For each node u, let pha(u)=d(u,p1)$^2$+d(u,p2) $^2$+d(u,p3) $^2$;
3.    Sorting all pha(u), then divide all nodes into N
      subsets $V_1, V_2, \cdots, V_N$ according to their pha values;
      //Clustering processing
4.    For each node u, do loops:
5.      For each subset $V_i$, do loops:
6.          Calculate average distances from $V_i$ to pharoses:
```

[2] Pharos is a peninsula in the Mediterranean Sea at Alexandria, Egypt. It is the site of an ancient lighthouse. Ancient sailors used it to estimate their positions on the sea.

```
7.          $\overline{d_j} = \sum_{v \in Vi} d(v, p_j) / |Vi|$, j= 1, 2, 3;
8.          diff(u)=(d(u,p1)-$\overline{d_1}$)$^2$+(d(u,p2)-$\overline{d_2}$)$^2$+(d(u,p3)-$\overline{d_3}$)$^2$
9.        End of For each subset
10.       Move n to the $Vi$ which gets the minimal diff(u);
11.     End of For each node
12. End
```

Figure 4. The pseudo-code of Pharos-based Clustering Algorithm.

Finally, N un-intersect subsets $V1, V2, \cdots, VN$ are generated. They meet the following conditions:

$$V1 \cup V2 \cup \cdots \cup VN = V \quad (5)$$

$$\forall \, i, j \in \{1,2,\ldots,N\}, \; V1 \cap V2 = \varnothing \quad (6)$$

4.2 Pharos Determination Policy

In a 2-D coordinate system, 3 different reference points would be enough for positioning an unknown *u*. This can be easily confirmed by the left part of Figure 5.

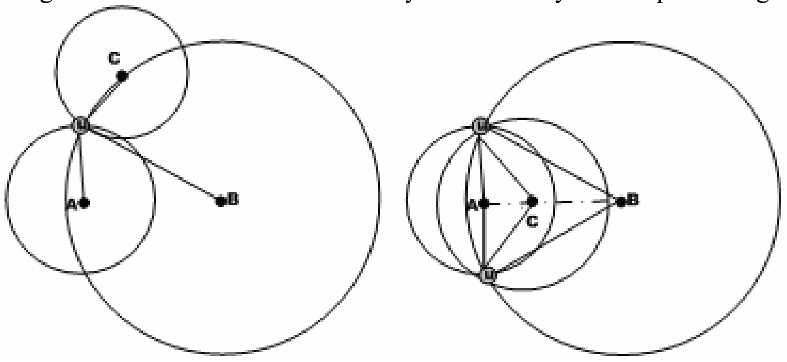

Figure 5 Left: Let d1, d2 and d3 denote the distances from *u* to three reference points A, B and C. Regard A as the centre, d1 as the radius, draws a circle Circle(A, d1). Similarly draw Circle(B, d2) and Circle(C, d3). Three circles can only intersect at one single point. Figure 5 Right: If three reference points are on a straight line, the position of *u* cannot be determinate, because it still has two alternatives.

Feasible pharoses should avoid being on the same line. Furthermore, geometrical theorem reveals that far-away reference points can improve positioning accuracy. Here introduce an elaborate method to meet these two requirements. Firstly, find out the diameter of G, denoted as D. Suppose two end nodes connecting the diameter are $p1$ and $p2$. Secondly, find out the set of nodes whose distances to $p1$ are larger than $2/D$, denoted as Set1. Similarly, find out another set of nodes whose distances to $p2$

are larger than $2/D$, denoted as Set2. Finally, let p3 be the node which makes the maximum sum: $d = d(p,p1)+d(p,p2)$, $p \in$ Set1 ∩ Set2. Then $\{p1, p2, p3\}$ are the selected pharoses. Since $d(p1, p3) + d(p2, p3) > D/2 + D/2 = D = d(p1,p2)$, it is surely that $p1$, $p2$ and $p3$ are not on a straight line. Meanwhile, these three nodes are as far away as possible from each other.

4.3 Local Replica Placing Process

After Clustering phase, a large-scale problem is divided into several medium-scale problems. It is noticeable that these problems are completely independent from each other. One cluster's result has none dependence with another's. Obviously, distributed and parallel Placing phrase can provide more scalability and speedup the whole algorithm. In each cluster Vi, any existing algorithm mentioned in section 2 can be applied to obtain local replica policy Pi with much smaller computation and memory overhead.

In order to minimize update cost, a shortest path tree is adopted to act as the update distributing tree T. Every Vi establishes its local tree Ti according to the location of source node s. In cluster Vs which contains the original node s, take s as the root of Ts. In other clusters, a root should be found through the following rule. Let ti denote the root of T_i. Then it can be picked out by calculate the distance between any node in Vi and original tree Ts. Let $d(u`, v`)$ denote the smallest one:

$$d(u`, v`) = \min_{u \in Ti}\{\min_{v \in Ts}\{d(u,v)\}\} \qquad (7)$$

Then $ti = u`$. And $v`$will be recorded as the attaching point, see Section 4.4.

4.4 Partition Integrating Mechanism

In order to obtain the global policy, every cluster submits its local policy to the original node. It is obviously that the finial replica policy is $P = P_1 \cup P_2 \cup \cdots \cup P_N$.

The last thing remaining is the integrating of update distributing trees. During the Integrating phase, every tree T_i is submitted to s. Then they are attached to Ts one by one, as Figure 6 illustrated. The attaching points also come from (7). This only needs constant time complexity.

5 Theoretical Analysis

The philosophical foundation of CPI algorithm is deduction and induction. At the beginning, a large-scale problem is deducted to several small-scale problems with the same essence. On finishing these small problems, all partial results are inducted to generate a complete result. Moreover, CPI is provided with parallel and distributed features in the second phase.

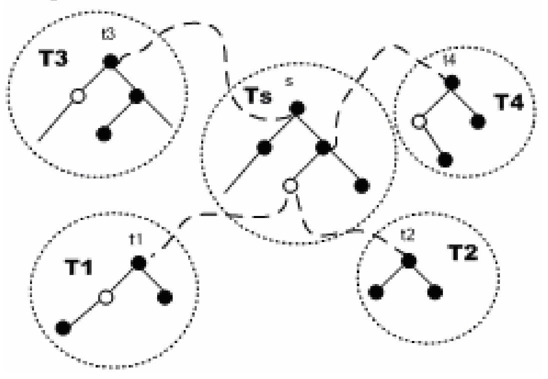

Figure 6. An example of tree integration during Integrating phase. Black dots represent for replica nodes, while whit dots for non-replica nodes. Four local update distributing trees are grafted to the original tree, thus obtaining the global update distributing tree T.

Recall the pseudo-code in Figure 2, we will analyze the time complexity of CPI line by line.

(1) Line 2: the pharos electing algorithm is performed at the time complexity $O(|V|^2)$.
(2) Line 3: the pharos-based clustering process is executed. It contains a sorting process, which has $O(|V|*\log|V|)$ time complexity [15]. About the nested For loops in Figure 4, there are $|V|$ nodes and N clusters, so the complexity is $O(N*|V|)$. Thus the total time complexity of line 3 is $O(|V|*\log|V| + N*|V|)$.
(3) In the For loops from line 4 to line 7, suppose the time complexity of the local replica placement algorithm is typically $O((|V|/N)^3)$, since constructing update distributing tree only needs a complexity of $O((|V|/N)^2)$ [13], so the total time complexity is $O(N*(|V|/N)^3)=O(|V|^3/N^2)$.
(4) Line 8: constant time.
(5) Line 9: according to equation (6), since the average sizes of V_s and V_i are all n/N, so the time complexity is $O(|V|^2/N^2)$.

Therefore, the total time complexity is $O(|V|^3/N^2 + |V|^2)$, depending on the choice of N. For example, in our experiments, let $N=|V|/k$, k is a constant. Then the final time complexity is $O(k^2*|V| + |V|^2) = O(|V|^2)$. It can be concluded that the computation cost is much cheaper than all existing ones.

6 Experiments and Evaluation

With Java language, we developed a simulating test-bed for replica algorithm validating. It consists of 5 parts, listed as follows:

- A famous network topology generator BRITE [16] was imbedded to produce networks graphs. Also corresponding functions were designed to read in BRITE output files and generate all-pairs shortest path matrix.
- A Java GUI graphic tool was developed to show how nodes and replicas are distributed in a square plane.
- A class was implemented to generate a shortest path tree and obtain update cost.
- A library included many existing replica placement algorithms, as well as CPI.
- Other utilities and assistant classes.

The test-bed is running at a personal computer with Intel Pentium M Process 1.7GHz, 1GB memory, 100GB disk and Windows XP OS. For justice, only "pure" costs of algorithms are recorded. For one graph size, BRITE generated 100 graphs to test effects of one algorithm. The result of this algorithm in our record is the average of 100 times of experiments. We chose the Waxman model [16] to generate network topology. The process can be outlined as follows: firstly N nodes are randomly placed into a square plane ordered by HS and LS. Then link is created between each pair of nodes *(u, v)* with the probability of $p(u, v) = \alpha \cdot e^{-d(u,v)/(\beta \cdot D)}$, where $d(u, v)$ is the Euclidean distance between u and v, D is the diameter of the graph, and α , β are both Waxman parameters. Larger β will generate more edges, and higher α will result more long edge. Finally a bandwidth is set to every edge. In our experiments, HS = 1000, LS = 1000, α = 0.15, β = 0.2. Additionally, the label of original node was generated randomly. Without loss of generality, $s(v) \equiv 1$ and $\lambda \equiv 0.5$.

6.1 The Choice of Parameter *N*

As we discussed in section 4.1, number of clusters *N* influences the clustering algorithm so much. In this section, different sizes of *k* are tested to find the proper rang of *N*. Let *k* varies from 100 to 800. At the same time, different sizes of *N* are tested. Since update cost is consistent with the time cost, we only recorded the time expended. Table 1 shows the influence of *k*.

Table 1. Time complexity of CPI under different parameter *k* (unit: second)

N	100	200	400	800	1600	3200
k=100	2.47	2.86	23.16	63.03	241.37	328.73
k=200	2.14	23.86	24.67	41.93	125.42	245.32
k=400	3.03	3.86	11.12	34.17	63.25	183.25
k=800	3.39	5.97	33.67	242.27	445.13	456.31

When *k* is large, the time cost is mainly decided by the local replica algorithm. On the contrary, it is mainly generated from the clustering algorithm. From table 1 we can conclude that *k*=400 is a proper value.

6.2 Time complexity and Space

Four different algorithms were operated on the test-bed, with the number of nodes growing from 100 to 3200. The QoS is fixed to $0.2*D$, where D is the diameter of graph G. *0-Greedy-Insert* and *0-Greedy-Delete* are introduced in Tang and Xu's paper [9]. *GC* stands for *Greedy-Cover* algorithm [10]. In CPI algorithm, *0-Greedy-Insert* is adopted to solve cluster replica placing problem. The number of clusters k is set to 400, as discussed in Section 6.1. If $N < 400$, no clustering operation occurs. And the CPI algorithm will be degraded to be a normal *0-Greedy-Insert*. Note: if running time exceeds 2 hours (7200 seconds), it will be marked as the symbol E/T, which means OutOfTime exception. Comparatively, another symbol E/M represents for the OutOfMemory exception.

Table 2 shows how the number of nodes influenced the time complexity of traditional algorithms. As the node number doubled, the time costs of *0-Greedy-Insert*, *0-Greedy-Delete* and *GC* increased by an order of magnitude. *GC* got better results than *0-Greedy-Insert* or *0-Greedy-Delete* because it didn't have to repeat calculating the update cost during placing. The construction of update distributing tree can be finished at the very end of the algorithm.

Table 2. Time & space costs of different algorithms (unit: second)

# of Nodes	*0-Greedy-Insert*	*0-Greedy-Delete*	*GC*	*CPI*
100	0.19	5.78	0.11	0.35
200	2.78	151.45	0.38	3.66
400	35.78	5285	3.16	37.85
800	542.27	E/T	29.50	42.72
1600	E/T	E/T	259.91	192.65
3200	E/M	E/M	E/M	472.23

Only CPI can handle a placement problem with over 3000 nodes. When N is less than 400, time cost of CPI is almost the same with a local algorithm. In this case, CPI is degenerated to a *0-Greedy-Insert* algorithm. When N is larger than 400, the cost of CPI keeps stable at several tens of seconds, which is approximate the cost of *0-Greedy-Insert* handling 400 nodes. Even when $N = 3200$, the cost does not exceed 500 seconds. This result accords with the theoretical analysis in section 5.

6.3 The Effect of Replica Placement

In this section we will compare the effect of different algorithms. Metrics includes the number of replicas and the update costs, and thus the replica cost calculated by the expression (1).

Suppose that CPI still use *0-Greedy-Insert* as the local cluster replica placement solution. From figure 7 we know that CPI needs more replicas than *0-Greedy-Insert* to satisfy all clients' QoS requirements. However, the replica number of CPI is much less than that of GC. This is because GC only considered the nodes nearby, while CPI will check the whole node sets to find proper clusters. The increment of storage cost is also acceptable. From the experiment we also found that update cost is closely relative with the number of replicas. It has the similar curves as in Figure 7. For the limitations of space, we omitted to describe it.

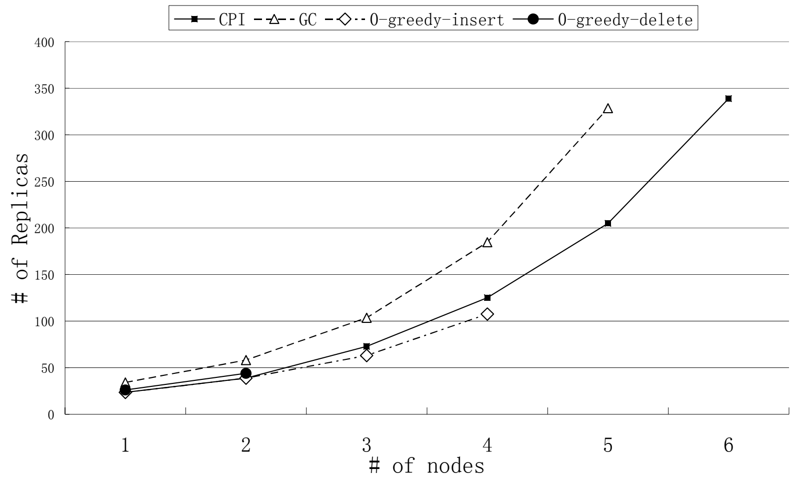

Figure 7. With the growth of nodes, increments of replica numbers of different algorithms remain stable. The coordinates 1-6 at x- axis stand for 100 nodes, 200 nodes, 400 nodes, 800 nodes, 1600 nodes and 3200 nodes respectively.

7 Conclusion

By testing various existing QoS-aware replica placement algorithms on our self-designed test-bed, we found out that they all lacks of scalability and can only handle small-scale and medium-scale QARP problems. In this paper, we propose a novel three-phase algorithm CPI to overcome the embarrassment. The ideal of CPI is to divide a large-scale problem into several medium-scale partial problems with an effective clustering algorithm. After each partial problem is solved, the integration of all partial results will generate the complete result.

There are several original ideas in the algorithm, including the pharos-based clustering methods and the integrating mechanism of multiple update distributing trees. In order to investigate and test different algorithms, we designed and implemented a general-purposed test-bed by ourselves. Elaborate plans and sufficient experiments make our work solid and convincible.

References

1. Marta Patiño-Martinez, Ricardo Jiménez-Peris, et.al. MIDDLE-R: Consistent database replication at the middleware level. ACM Transactions on Computer Systems (TOCS) ,Volume 23 , Issue 4 (November 2005), Pages: 375 – 423.

2. Zhang, Jiaying Honeyman, Peter, Hierarchical Replication Control in a Global File System. Seventh IEEE International Symposium on Cluster Computing and the Grid, P 155-162. 2007. CCGRID 2007.

3. Gkantsidis, C. Rodriguez, P.R. Network coding for large scale content distribution. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. 2005. Vol(4), P: 2235- 2245.

4. Balachander Krishnamurthy, Craig Wills, Yin Zhang. On the use and performance of content distribution networks. Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement table of contents. Pages: 169 – 182.

5. Dahlia Malkhi, Lev Novik, Chris Purcell. P2P replica synchronization with vector sets. ACM SIGOPS Operating Systems Review archive, Volume 41 , Issue 2, Pages: 68 - 74 . 2007.

6. Ann Chervenak, I.F., Carl Kesselman, Charles Salisbury, Steven Tuecke, The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientic Datasets. Journal of Network and Computer Applications 2001. 23: p. pp. 187-200.

7. L. Qiu, V.N. Padmanabhan, and G.M. Voelker, "On the Placement of Web Server Replicas," Proc. IEEE INFOCOM '01, pp. 1587-1596, 2001.

8. I. Cidon, S. Kutten, and R. Soffer, "Optimal Allocation of Electronic Content," Proc. IEEE INFOCOM '01, pp. 1773-1780, Apr. 2001.

9..Xueyan Tang, Jianjiang Xu., Qos-aware replica placement for content distribution. IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, 2005. Vol(10). pp. 921-932.

10..Hsiangkai Wang, P.L., Jan-Jan Wu. A QoS-Aware Heuristic Algorithm for Replica Placement. in Grid Computing Conference 2006. 2006.

11. Won J. Jeon, I.G., and Klara Nahrstedt. QoS-aware Object Replication in Overlay Networks. in IPTPS 2005. 2005.

12. Wei Fu, Nong Xiao, Xicheng Lu. QoS-Guaranteed Ring Replication Management with Strong Consistency. ApWeb/WAIM Workshops, Huangshan China 2007.

13. WANG Tao, LI Wei-Sheng, A Fast Low-Cost Shortest Path Tree Algorithm, Journal of Software (China). Vol.15, No.5. P:660-665.

14. http://en.wikipedia.org/wiki/Global_Positioning_System, 2008

15. C P Schnorr, A Shamir. An optimal sorting algorithm for mesh connected computers. Proceedings of the eighteenth annual ACM symposium on Theory of computing table of contents. Pages: 255 - 263.1986.

16. Medina, A. Lakhina, A. Matta, I. Byers, J. BRITE: an approach to universal topology generation. Ninth International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, page(s): 346-353. 2001.