QoS-Aware Object Replica Placement in CDN networks

Zhiyong Xu
Suffolk University
zxu@mcs.suffolk.edu

Laxmi Bhuyan
University of California, Riverside
bhuyan@cs.ucr.edu

***Abstract*— Recently, Content Distribution Networks (CDNs) have attracted a great deal of attention from both the industry and academic communities. In this paper, we design efficient object replication algorithms to achieve the optimal performance while not violating clients' QoS requirements. We show that such a problem is NP-complete and solve it with a three-stage mechanism: In the first stage, object replication constraints to meet the QoS requirements are generated. Second, a minimal object replication set (MORS), which can satisfy the constraints with the minimal number of replicas on each server, is created. Finally, more objects are replicated on the servers with spare space to further improve the performance. We propose a number of heuristic algorithms and conduct extensive trace-driven simulation experiments to evaluate the performance of these algorithms.**

keyword **– Content Distribution Network (CDN), Replica Placement Algorithm, Quality of Service (QoS)**

I. INTRODUCTION

With the recent successes of the commercial CDNs, such as Akamai [1] and Digital Island [2], CDNs have attracted a great deal of attention from both the industry and academic communities. In CDN system, Replica Placement algorithm (RP) has great impact on the system overall performance. According to the replication granularity, CDN systems can be categorized into two types: full replication architecture (FD) and partial replication architecture (PD). Massive research efforts have been carried out on the replica placement problem on both FD and PD architectures [3], [4], [5], [6], [7], [8], [9]. Among them, Greedy algorithms are considered to be the best algorithms.

Existing heuristic algorithms solved the replica placement very well, and a near-optimal performance for the entire client community can be achieved. While an average performance metric is important, the current algorithm can not provide the QoS guarantees for individual clients. In this paper, we investigate the techniques of object replica placement algorithms to support Quality of Service (QoS) in PD architecture. Given certain amount of system resources, our objective is to optimize an average performance metric (such as access latency) for all the clients while meeting different QoS requirements of individual clients. A three-stage mechanism is introduced. First, we generate object replication constraints according to the QoS requirements. Second, we create a minimal object replication set (MORS) which can satisfy all the replication constraints with the minimal number of copies of each object and decide the location for each replica in MORS. This gives the CDN service provider an intuitive impression on how many servers and how many resources on each server are needed for providing certain level of QoS requirement. In the final stage, we store more replicas on the servers which still have spare space to further improve the performance. We show such a problem is a NP-complete problem. A number of heuristic algorithms are proposed. The main contributions of this paper are:

- We formulate the QoS-Aware object replica placement problem in CDN system and prove it is a NP-complete problem;
- We define the concept of MORS. A MORS represents the minimal system resource demand for a certain QoS guarantee. We define an optimal MORS is the MORS which achieves the best performance among all the MORSs;
- We develop several heuristic algorithms including random, popularity and greedy algorithms, to generate MORS. We conduct simulation experiments to evaluate the performance of the various algorithms;
- We propose a flexible three-stage mechanism to solve this problem. Several heuristic algorithm combinations are introduced. We evaluate and compare the performance of various algorithm combinations with the super optimal algorithm. The greedy algorithm is proved to be able to achieve the best performance.

The rest of the paper is organized as follows: Section II describes the problem formulation and the cost model. Section III presents the proposed heuristic algorithms. Section IV evaluates the performance of various algorithms. Section V summaries the previous related works, and finally, we conclude in Section VI.

II. PROBLEM FORMULATION

Object replication problem can be formulated as a graph problem that approximates the overall performance of a certain metric (Such as minimal storage cost, minimal user access latency or network bandwidth consumption).

Consider a content distribution system, a set of objects are replicated and distributed on these servers. For each object, at most one copy can be stored on each server. However, multiple copies may be stored on different servers. Each server only has limited storage capacity and can not hold all the objects. There are a large number of clients with different QoS requirements want to access those objects. We focus our attention on the design of an efficient object placement algorithm to determine the number of copies for each object and the locations for these copies subject to each server's capacity restriction. Our objective function is to minimize the end user retrieve cost while not violating the QoS requirement for each client.

A. System Cost Model

The object replication problem can be formally stated as follows: In a CDN system, we have *N* content servers. The storage capacity on each server is denoted by S_k, *k = 1,2,...,N)*. These servers are used to keep the copies of a set of objects. Assuming the number of objects in the system is *J*, and the size of

each object is denoted by O_j, *j = 1,2,...,J*. For simplicity, we assume the aggregated capacity of all the servers is much bigger than the overall storage requirement (which is the case in reality): $\sum_{j=1}^{J} O_j < \sum_{k=1}^{N} S_k$, and for any $k \in [1, N]$, $S_k < \sum_{j=1}^{J} O_j$. Thus, the server storage shortage problem seldom happens. Before we define any QoS related constraints, we give the storage constraints first. We define the following variable: x_{jk}, to denote the relations between the servers and the objects to be stored. If object j has a copy on server k, then x_{jk} = 1; otherwise, x_{jk} = 0. Based on these definitions, we have:

For any server k,

$$\sum_{j=1}^{J} O_j x_{jk} \leq S_k \tag{1}$$

For any object j,

$$1 \leq \sum_{k=1}^{N} x_{jk} \leq N \tag{2}$$

Assume we have M clients denoted by T_i, *i = 1,2,...,M)*. For each client, retrieving an object from any server will introduce some cost. This retrieve cost is determined by the network topology and the network conditions between the client and the server which used to fetch the object. For simplicity, we assume that all the objects have the same size, and the retrieve cost is proportional to the distance between the client and that server. Thus, we define another variable: C_{ik}, to represent the cost for the client i to retrieve an object from server k. For any client, it is desirable to retrieve all the requested objects from the server which has the minimal distance to it. We define the server $k^{'}$ is the designated server for client i if server $k^{'}$ has a smaller $C_{ik'}$ than any other servers. However, due to the storage capacity limitation, it is very likely that the designated server does not contain all the objects a client wants to access. This client has to retrieve some of these objects from other servers.

We define the variable: P_{ij} to represent the request probability that client i will request object j. If we do not consider QoS requirements of different clients, the objective of the object replica placement can be formulated as choosing the suitable values for all the x_{jk} in the matrix X = $(x_{jk}), j = 1, 2, ..., J$ and $k = 1, 2, ..., N$, that we can achieve the minimal average retrieve cost. Assume the current replica placement is A. Then, for client i to access an object j, the minimal cost is:

$$P_{ij} * min_{k \in [1,N] \wedge x_{ik}=1}(C_{ik}) \tag{3}$$

$min_{k \in [1,N] \wedge x_{ik}=1}(C_{ik})$ represents the cost for client i to fetch object j from the nearest server which has the object (this server may not be the designated server if it does not have the requested object). Thus, considering all the clients and all the objects, the minimal overall cost C(A) for an object replication deployment A is:

$$C(A) = \sum_{i=1}^{M} \sum_{j=1}^{J} P_{ij} * min_{k \in [1,N] \wedge x_{ik}=1}(C_{ik}) \tag{4}$$

B. QoS requirements

To provide QoS guarantees for different clients, we divide the clients into multiple service classes according to their different QoS requirements. From the clients' points of views, in general, higher QoS guarantee stands for smaller access latencies. Clients in the higher service classes should receive better performance guarantee than the clients in the lower service classes. We define the QoS requirement with the time of retrieving an object. Thus, QoS guarantee for a service class means: for any clients belongs to this class, retrieving an object must be finished within a certain period of time. Assume we have D service classes (1,2,...,D). A *maximal cost constraint* is set for each class, denoted by MCC_D. The higher the class, the lower the maximal cost constraint. We have the following equations:

$$MCC_D \leq MCC_{D-1} \leq ... \leq MCC_1 \tag{5}$$

Our goal is to find a matrix X = (x_{jk}) which can achieve the minimal cost C(A) subject to the following constraints:

For client i who belongs to class d, for any $P_{ij} > 0$, to meet the QoS requirement, the system must guarantee at least one server, which the cost for client i to access an object from it is less than MCC_d, exists, and that server has a copy of object j.

$$\exists k \in [1, N], (x_{jk} = 1) \wedge (C_{ik} \leq MCC_d) \tag{6}$$

C. NP-Complete Problem

In theory, the above problem is a decision problem and it is NP-complete. To prove this, first, we will show that it is a NP problem. Given a target overall cost $C^{'}(A)$, we need to find out whether a candidate solution exists such that

$$\sum_{i=1}^{M} \sum_{j=1}^{J} P_{ij} * min_{k \in [1,N] \wedge x_{ik}=1}(C_{ik}) \leq C^{'}(A) \tag{7}$$

can be computed in polynomial time.

Examine a candidate solution satisfies QoS requirement or not can be done within polynomial time since the shortest path between clients can servers can be computed in polynomial time. Given a candidate solution A, it is very easy to verify the total computation cost is less than the bound $C^{'}(A)$ or not within the polynomial time. Thus, this problem is NP problem. Next, if we consider a special case that all the clients are belonging to the same service class. Only one server $k^{'}$ can be used to store objects and the request probability for any object is the same. The problem is reduced to:

Minimizing:

$$C(A) = \sum_{i=0}^{M} C_{ik'} \tag{8}$$

subject to:

$$\sum_{j=1}^{J} O_j x_{jk'} \leq S_{k'} \tag{9}$$

Clearly, this problem is identical to the well-known NP-complete Knapsack problem [10]. Thus, such a problem is a NP-complete problem and finding the optimal solution is not feasible.

III. Object Placement Algorithms

In this section, we present a number of algorithms for solving the QoS-Aware object placement problem. Since no optimal solution is feasible, we have designed several heuristic algorithms that utilize the available information in different ways in order to get the best performance and meet the QoS requirements. We use the average retrieve cost as the metric to evaluate the performance. We take a three-stage approach to make the placement decision. First, we generate a set of object placement constraints by considering the QoS requirements. According to these constraints, in the second stage, we construct a small set of objects (MORS) to be deployed on the servers. Thus, after this stage, the QoS requirements are satisfied. The last stage is used to further reduce the retrieve cost by adding more object replicas on the servers with free space.

A. Determine the Object Placement Constraints

In the first stage, we must determine the replica placement constraints to meet the certain QoS requirements. Assume client i belongs to class d, the maximal cost allowed in class d is MCC_d, and the set of the objects that client i will retrieve is Q. Then, for any object $j \in Q$, a replica must be presented on a server which the cost for client i to retrieve an object from it is less than MCC_d. We determine the constraints as follows: each time, the algorithm selects a client, denoted by i. We divide the servers into two partitions: the servers which client i has the retrieve cost lower than MCC_d and the servers which client has the cost higher than MCC_d. We do not consider the second partition since it can not satisfy the QoS requirement for this client anyway. For the first set of servers, we will generate a constraint Cons(i,j) for client i and object j: For object j, which $P_{ij} \neq 0$, we must store a copy on at least one of these servers. This process continues until we generated all the placement constraints for all the clients and all the objects.

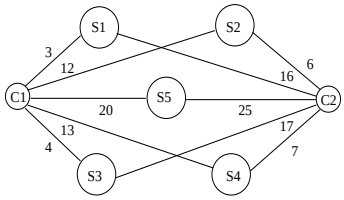

Fig. 1. Simple illustration of determining the object placement constraints

Figure 1 shows a simple example. Consider a CDN system which contains only two clients C1 and C2, and five servers S1, S2, S3, S4 and S5. C1 and C2 are belonging to the same service class d1, and MCC_{d1} is 10. The distances between clients and the servers (retrieve cost) are shown in the figure. Both Clients C1 and C2 have the same probability to retrieve object j1 in the future. Clearly, since the distance between C1 and S1 is 3, C1 and S2 is 4, both are less than MCC_{d1}. For client C1, to meet the QoS requirement, either S1 or S3 must keep a copy of j1, we denote this constraint with $x_{1,j1} \cup x_{3,j1}$. There's no constraint on servers S2, S4 or S5 since the distance between C1 and any of them is larger than MCC_{d1}. Whether they have a copy of j1 or not has no effect on satisfying C1's QoS requirement. Similarly, for client C2, either S2 or S4 must keep a copy. We can generate another constraint, represented by $x_{2,j1} \cup x_{4,j1}$. For client C2, no constraint is generated on servers S1, S3 or S5 as well. In summary, to meet the QoS requirements for object j1 and clients C1 and C2, we have to store at least two copies of j1. One copy must be resided on S1 or S3 and another copy must be stored on S2 or S4. Thus, we can create a placement constraint for object j1:

$$(x_{1,j1} \cup x_{3,j1}) \cap (x_{2,j1} \cup x_{4,j1}) \quad (10)$$

The generated constraints will be used to assign object copies on certain servers in the second stage. We use (S1,S2) to denote that both S1 and S2 have a copy of j1. For the above example, to meet the QoS requirements, no matter how many copies are replicated on these five servers, one of the following server placement combinations: (S1,S2), (S1,S4), (S3,S2) and (S3,S4) must be chosen. Other possible placement decisions could be (S1,S2,S4), (S2,S3,S4,S5) or even (S1,S2,S3,S4,S5). However, only two replicas are necessary, the additional copies are redundant, they do not have any contribution to further improve the system performance.

B. Minimal Object Replication Set (MORS)

In the second stage, we will create a *Minimal Object Replication Set (MORS)* according to the object placement constraints. We define a MORS is the smallest replica set which containing the minimal number of copies for each object on different servers to meet all the QoS requirements. Actually, it can be viewed as the representation of the minimal resource (Here, it is the minimal storage capacity needed on individual servers) required for providing certain QoS guarantees. In the above example, any of the server placement combinations (S1,S2), (S1,S4), (S3,S2) and (S3,S4) which only has two servers is a MORS.

Different MORSs result in different performance. For example, in Figure 1, if we store a copy of object j1 on both server S1 and S2, the overall cost for client C1 and C2 to retrieve j1 is $3 + 6 = 9$. While in (S1,S4), (S3,S2) and (S3,S4), the corresponding costs are 10, 10 and 11. We define the MORS which has the minimal overall cost as the optimal SORS. In this example, (S1,S2) is the optimal MORS. Finding the optimal MORS is also a NP-complete problem (it can be easily derived from the previous proof). We develop several heuristic algorithms to generate near-optimal MORS placement decision.

C. Random MORS Algorithm (RA)

In random algorithm, the objects are assigned to the servers randomly subject to the QoS constraints and the server storage restrictions. Figure 2 shows the pseudo code of the random MORS algorithm. The algorithm is performed on the objects one by one. Each time, we pick an object (denoted as j) with the uniform probability. Then, we choose a client (denoted as i) which has $P_{ij} \neq 0$ with the uniform probability as well. By applying the constraint Cons(i,j), a set of servers which can satisfy the QoS requirement for client i and object j are determined. The system examines the availability of object j in this set, if none of these servers has a copy, it randomly chooses one server which has enough spare space from this set, and stores a copy of object j. If any of these servers has a copy of j, no operation is taken because the current replica deployment can satisfy this constraint already. We continue this procedure by randomly choosing another client until all the constraints related to this object are checked. After that, we move to another object (it is also picked with uniform probability), and repeat the same operation until all the objects are checked. Finally, a MORS which can satisfy all the constraints is generated.

```
Procedure  Random MORS Generation Algorithm

Input: Servers N, Clients M, Objects J
       Placement Constraints set  Cons(i,j)
Output: MORS
       FinishedClient=null;
       StartClientSet={1,2,...,M};
       FinishedObject=null;
       StartObjectSet={1,2,...,J};

       While(StartObjectSet!=NULL)
       {
          randomly choose one object i from StartObjectSet;
          while (StartClientSet != null)
          {
             choose one client j;
             serverset=apply Cons(i,j);
             if (any server in serverset has the object j stored)
                break;
             else
                randomly choose one server to store object j;
                FinishedClient+=i;
                StartClientSet−=i;
          }
         FinishedObject+=j;
         StartObjectSet−=j;
        StartObjectSet={1,2,...,J};
       }
 end procedure
```

Fig. 2. Random MORS Algorithm

D. Popularity MORS Algorithm (PA)

Random algorithm does not use workload characteristic information to make placement decision. While in popularity MORS algorithm, unlike random algorithm which picks the location for each replica with the uniform probability, the algorithm tries to determine the best locations for each object based on its popularity. First, the system sorts all the objects in decreasing order of popularity according to their aggregated request rates from the clients. The placement decision process starts from the object (denoted as j) which has the highest popularity. The algorithm also sorts the clients in decreasing order of popularity according to their request rates for object j. Then, it picks the client i with the highest request rate, and examines the corresponding constraint Cons(i,j), to generate the set of servers which can satisfy this QoS constraints. If any of these servers has a copy, we move to the next client which has the second highest request rate. If none of the servers has a copy of object j, we need to choose one server from the server set to store it. We use the following mechanism to determine this server: For each server in this set, we calculate the aggregated request rates for object j on clients which use this server as the designated server and sort these servers in decreasing order. The server which has the highest aggregated rate is chosen first. If this server does not have enough spare space, we will check the server which has the second highest aggregated request rate and store a copy. After all the client constraints for this object are checked, the replica placement decision for this object is made. We move to the next object and continue this procedure until all the objects are checked. This algorithm has higher computational overhead than the random algorithm. However, it can achieve better performance.

E. Greedy MORS Algorithm (GA)

The third algorithm we consider is greedy algorithm. In this algorithm, as Popularity algorithm, the system sorts the objects and the clients in the decreasing orders of popularity. Then, each time, we start from the first object j and the first client i, applying the placement constraint Cons(i,j). After getting the server set which contains all the servers satisfying the constraint, if any of these servers has a copy, system moves to the next client. If none of them has a copy, we calculate the retrieve cost for each server as if it is used to store object j. Then we pick the server which generates the lowest cost to store a copy of object j. If this server does not have enough spare space, the server which has the second lowest cost is chosen. This process continues until all the clients are checked. Thus, in each step, we choose the server which can introduce the maximum cost reduction. For example, in the previous example, for client C1, the retrieve cost for retrieve j1 on S1 is 3 and the cost on S3 is 4. Thus, a copy of j1 is replicated on S1. Then, for client C2, the cost for retrieve on S2 is 6 and S4 is 7. Thus, another copy should be kept on S2. Thus, among all the MORS configurations (S1,S2), (S1,S4), (S3,S2) and (S3,S4), (S1,S2) has the smallest overall cost, and it will be chosen by greedy algorithm.

F. Achieving Better Performance

By creating a minimal object placement set, the system defines the minimal resource required to meet the certain QoS requirements. However, it does not achieve the best performance. Thus, in the third stage, we use the additional storage space on each server to store more objects and improve the system performance even more.

The object placement problem for the additional copies is similar to the above problem. We could adopt the above heuristic algorithms directly for this purpose. There are two differences we should take into consideration. First, in previous cases, system starts from an empty set (no copy of any object exists on any server). Here, we start from the condition that each server already stored a bunch of objects determined by a MORS. Second, in previous cases, locations of the object copies are subjected to the constraints to meet the QoS requirements. Here, this restriction does not exist any more.

1) Random Algorithm (RA): In this algorithm, since no replica placement constraints need to apply, each time, we choose one object and one server with uniform probability, and store this object on that server in case no copy of this object exists.

2) Popularity Algorithm (PA): In this algorithm, each server stores the most popular objects for its clients. The placement decisions are made by the servers independently. Each time, we choose a server and generate the aggregated request probabilities of all the objects for the clients which use this server as the designated server. We sort these objects in the decreasing order of popularity. Then, we store as many objects as possible if they are not stored on this server yet.

3) Greedy Algorithm (GA): Here, we calculate all the retrieve costs by choosing one server and one object each time. The server-object pair which yields the lowest cost is chosen. And a copy of that object is stored on that server. After the new placement is generated, we do the same process by recalculating all the costs under this new placement. We iterate the procedure until all the storage spaces are exhausted.

G. Super Optimal Algorithm

We also propose a super optimal algorithm. In this algorithm, we do not set the storage capacity limitation. Each server can hold a full copy of all the objects. Thus, a client can get any object it wants from the designated server with the minimal retrieve cost introduced. We believe such a system can yield the best performance. This algorithm is used as an upbound reference in our experiments to evaluate the proposed solutions.

H. Summary

In summary, we develop a three-stage mechanism for QoS-Aware object replica placement. We could generate several strategies by using different algorithm combinations. For simplicity, we use RA-RA to denote the combination of using the random MORS algorithm in the second stage and also using the random algorithm in the third stage. The following combinations can be applied: RA-RA, RA-PA, RA-GA, PA-RA, PA-PA, PA-GA, GA-RA, GA-PA and GA-GA. We evaluate the performance of these strategies in Section IV.

.

IV. Performance Evaluation

We use Transit-Stub topology (TS model) introduced in [11] as the major network model. GT-ITM topology generator [12] is used to generate the emulated network model. We generate network models with two different sizes: a TS network with 300 nodes and another one with 1000 nodes. The average node distance in 300-node network is 161 and it is 266 in 1000-node network.

We evaluate the performance of three different MORS generating algorithms. We also compare the overall performance of nine different algorithm combinations: RA-RA, RA-PA, RA-GA, PA-RA, PA-PA, PA-GA, GA-RA, GA-PA and GA-GA.

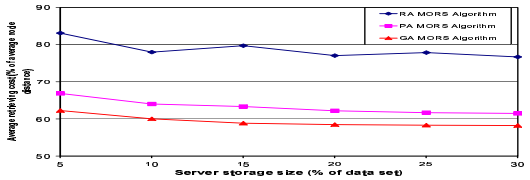

Fig. 3. MORS Performance , 30 servers, 150 clients

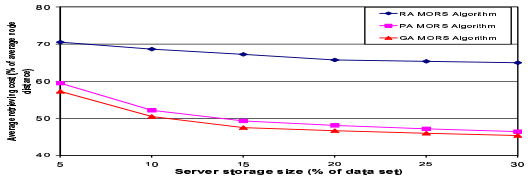

Fig. 4. MORS Performance, 200 servers, 500 clients

In our experiments, we take the web proxy logs obtained from the National Laboratory for Applied Network Research (NLANR) as the workload traces. The trace data we use is collected from *UC* server between Feb 21st and Feb 24th, 2003.

A. Minimal Object Replication Set

We start by identifying the performance of MORS generated by various algorithms. To avoid the biased results, in this experiment, we repeat the simulation three times for each algorithm. Each time, the sets of servers and clients are randomly chosen. The results are shown in figures 3 and 4. To make it more clearly, we show the results as the ratio between the average retrieve cost to the average node distance.

From the figures, we observe that the greedy MORS algorithm (GA) achieves the smallest average retrieve cost. For 300-node network, the average retrieve cost is about 58% to 63% of the average node distance. Popularity MORS algorithm (PA) also achieve good performance, which is slightly worse than GA algorithm. Random MORS algorithm (RA) has the worst performance. The average retrieve cost in RA is about 30-40% more than in GA and PA algorithms. Clearly, the object replication strategy used in the second stage has great impact on the system efficiency. This result shows, even with the smallest set of objects replicated, the locations of the replicas must be carefully selected. It is proven that utilizing workload characteristic information to make the replication decision is important. Both GA and PA take this information into account and achieve better performance. We also observe that as we increase the storage capacity on each server, all the three algorithms can not obtain much performance gain. This shows that, only small amount of system resources is enough to meet certain QoS requirements. Compare two different size networks, for each algorithm, the average retrieve cost achieved in 1000-node network is much less than in 300-node network if we measured with the ratio between the average retrieve cost and the average node distance. For example, with RA algorithm applied, this ratio is between 76-84% in 300-node network, while in 1000-node network, the ratio is reduced to 64-71%. This is because of the higher density of servers in the second case. In 1000-node network, 20% of the nodes are chosen as servers while only 10% of the nodes are chosen in 300-node network. In 1000-node network, the clients have more servers which satisfy the QoS requirements can be chosen. Thus, it has more storage space can be used to keep replicas, this is very important for achieving good performance especially when the storage size on each server is small.

B. QoS effect in MORS

We also evaluate the MORS property with different QoS values. Figure 5 shows the results in RA algorithm. We vary the minimal QoS cost from 0.5 to 2 times of the average node distance. The storage size on each server varies from 5% to 20% of the whole data set. We discard some cases which the QoS requirements can not be satisfied (5% storage size, with QoS value 0.5, etc). We observe that, as the QoS value increases from 0.5 to 0.75 of the average node distance, RA algorithm can achieve a little performance improvement. However, as the QoS value continues increase, the performance becomes worse. The reason is, in such conditions, because the system looses the QoS requirement, more servers which include some servers that have long distance to the client can satisfy the QoS constraints now. Because RA algorithm choose the location of a replica with equal possibility, it is very likely that a remote server is chosen. Thus, the average retrieve cost may even increase. However, for both GA and PA algorithms, system uses the workload information to make the decision, this problem does not happen. No matter what the QoS constraint is, the MORSs determined by GA and PA algorithms do not change a lot, and the MORS performance does not change as well. Thus, we do not show the results here.

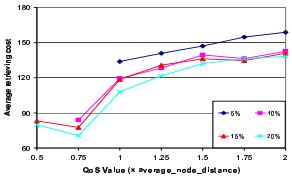

Fig. 5. MORS Performance with different QoS (30 servers, 150 clients)

C. Overall Performance

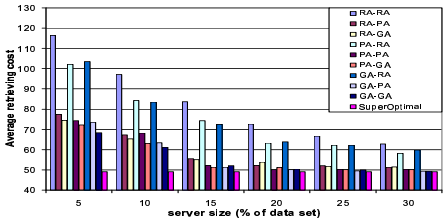

Fig. 6. Overall performance comparison (30 servers, 150 clients)

MORS is not enough to achieve the best performance. In the second experiment, we evaluate the overall retrieve cost. We compare nine different heuristic algorithm combinations with super optimal solution and the results for 300-node network are shown in Figure 6. We do not show the results for 1000-node network experiments because it generates the similar behavior. We observe that, the super optimal algorithm achieves the best performance because it does not have any storage constraints. It outperforms all the other algorithms. Among all the other approaches, using GA algorithm in the third stage yields good results. GA-GA achieves the best performance. We believe this is because GA algorithm maximizes the performance gain in each iteration when a replica placement decision is made. From the results, we also observe that, by using GA in the third stage, it is not very important which algorithm is used in the second

stage. Even RA-GA can obtain very good performance (only slightly worse than GA-GA). This proofs that greedy algorithm is the best solution which can achieve the near-optimal performance. Using PA algorithm can also achieve good performance, although it is a little worse than GA algorithm, the computational complexity is much lower. Thus, using PA algorithm in the third stage can be chosen as the best cost-effective option. On the other hand, using RA algorithm in the third stage is totally inadequate since it yields the worst performance no matter which algorithm is used in the second stage. As shown in the figure, all the RA-RA, PA-RA and GA-RA combinations have much worse performance than any other algorithm combinations. Even with a very large server storage size (30% of the data set). The performance for these three algorithms is still 20-25% more than any other solutions. Among them, RA-RA is the worst.

From the figure, we also find that using GA algorithm in the second stage can yield better overall performance than using PA algorithm in case the storage size is relatively small (5% or 10% of data set). This proofs that GA can make the better decision than PA algorithm. As the server storage size increases, this difference diminishes. This shows with a larger storage space can be utilized, an approximately accurate decision is good enough to achieve good performance. As the storage size increased to 25-30% of the data set, using GA or PA in the third stage can achieve nearly the same performance as the super optimal solution. Thus, from this experiment, we can conclude that *taking the workload characteristic information into account in object placement decision is important for the system to achieve the near-optimal performance*.

V. Related Works

There has been a considerable amount of researches to investigate the replica placement problem [3], [5], [13], [7], [9]. This problem can be further divided into server placement problem and object placement problem according to the replica granularity.

In server placement problem, the replica granularity is a mirror server. Actually, this problem is a variant of the traditional facility location problem. In [3], Liu *et al* considered the problem of placing replicas on a tree-based topology. In [7], Qiu *et al* formulated this problem as uncapacitated minimal K-median. They restrict the maximum number of replicas but do not restrict the number of requests served by each replica. The study conducted by Jamin *et al* [13] is similar to [7]. Their work found, regardless of the placement strategies, increasing the number of replicas is effective only for a very small number of replicas. The results suggested that the AS-level fanout-based algorithm should work as well as greedy algorithms, [5] used this approach by exploring the impact of various replica and client placement methods.

In [9], the authors investigated the object replication problem. The replica entity is a copy of object. The authors proven that it is a NP-complete problem and proposed several heuristic algorithms.

Karlsson *et al.* performed extensive evaluation of different replica placement algorithms [14]. They concluded that web caching is much more efficient approach if the update is frequent. However, for content distribution problem, replica placement algorithms are still necessary once properties such as availability, reliability, performance and bounded update propagation have to be guaranteed.

Tang *et al*. [15] is the first work conducted on QoS-Aware replica placement problem. They studied two classes of service models: replica-aware and replica-blind service. l-Greedy-Insert and l-Greedy-Delete heuristic algorithms are proposed. While in this paper, we investigate the problem of minimizing the client retrieve cost and meet the client QoS requirements. Our work can be viewed as the compliments with theirs.

VI. Conclusion and Future Work

In this paper, we studied the QoS-Aware object replication placement problem in content distribution network. We proved that it is a NP-complete problem and an optimal solution is not feasible. To solve this problem, we proposed a three-stage mechanism to achieve the best performance while meeting the QoS requirements of different clients. We defined a minimal object replication set (MORS) to represent the minimal system resource requirement for certain QoS guarantee. After a MORS is generated, more objects are replicated on the spare server space. A combination of three heuristic algorithms: random, popularity and greedy are introduced to achieve the near-optimal performance. We did extensive experiments to evaluate the performance of different algorithm combinations and compare with the super optimal solution. We observed that it is important to take client access behavior into account when the system make the object replica deployment decision. The neglect of this information will cause at least 30% to 40% system performance degradation.

References

[1] Akamai, "http://www.akamai.com."

[2] Digital Island, "http://www.digitalisland.net."

[3] B. Li, M. Golin, G. Italiano, X. Deng, and K. Sohraby, "On the optimal placement of Web proxies in the Internet," in *Proceedings of the INFOCOM '99 conference*, Mar. 1999.

[4] P.Krishnan, D. Raz, and Y. Shavitt, "The Cache Location Problem," *IEEE/ACM Transactions on Networking*, vol. 8, pp. 568–582, Oct. 2000.

[5] P. Radoslavov, R. Govindan, and D. Estrin, "Topology-informed internet replica placement," in *Proc. of the Sixth International Workshop on Web Caching and Content Distribution*, 2000.

[6] S. Jamin, C. Jin, D. Raz, Y. Shavitt, and L. Zhang, "On the placement of Internet instrumentation," in *Proceedings of the INFOCOM '00 conference*, pp. 295–304, Mar. 2000.

[7] L. Qiu, V. N. Padmanabhan, and G. M. Voelker, "On the Placement of Web Server Replicas," in *Proceedings of IEEE INFOCOM*, (Anchorage, AL), pp. 1587–1596, April 2001.

[8] S. Jamin, C. Jin, A. Kurc, D. Raz, and Y. Shavitt, "Constrained mirror placement on the internet," in *Proceedings of the INFOCOM '01 conference*, pp. 31–40, Apr. 2001.

[9] J. Kangasharju, J. W. Roberts, and K. W. Ross, "Object replication strategies in content distribution networks," in *Proceedings of the 6th Web Caching and Content Distribution Workshop*, (Boston, MA), June 2001.

[10] M. R. Garey and D. S. Johnson, "Computers and Intractability: A Guide to the Theory of NP-Completeness, Freeman," 1979.

[11] E. W. Zegura, K. L. Calvert, and S. Bhattacharjee, "How to model an internetwork," in *Proceedings of the IEEE Conference on Computer Communication, San Francisco, CA*, pp. 594–602, Mar. 1996.

[12] G.-I. T. Generator, "http://www.cc.gatech.edu/projects/gtitm."

[13] S. Jamin, C. Jin, Y. Jin, D. Raz, Y. Shavitt, and L. Zhang, "On the placement of internet instrumentation," in *Proceedings of IEEE INFOCOM*, (Anchorage, AL), pp. 295–304, April 2000.

[14] M. Karlsson and M. Mahalingam, "Do we need replica placement algorithms in content delivery networks," in *7th International Workshop on Web Content Caching and Distribution (WCW)*, August 2002.

[15] X. Tang and J. Xu, "On Replica Placement for QoS-Aware Content Distribution," in *Proceedings of IEEE INFOCOM*, (HongKong), March 2004.