On Optimal Replication of Data Object at Hierarchical and Transparent Web Proxies

Xiaohua Jia, Deying Li, Hongwei Du, and Jinli Cao

IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS (TPDS'05)

Outline

- Introduction
- Problem Formulation
- Computing the Induced Tree of Proxy Nodes
- Proxies with Unlimited Storage Capacities
- Proxies with Limited Storage Capacities
- Simulations

Overview

- This paper investigates the optimal replication of data objects at hierarchical and transparent web proxies.
- Two cases of data replication at proxies are studied: 1) proxies having unlimited storage capacities and 2) proxies having limited storage capacities.
- For the former case, an efficient algorithm for computing the optimal result is proposed.
- For the latter case, they prove the problem is NP-hard, and propose two heuristic algorithms.

Proxy and Data Replication

- Typical web caching techniques
- In this paper, they address the problem of data replication at proxies.
- Data replication proactively places a copy of data and anticipates many clients to make use of the copy at a proxy.
- By transparent, they mean the proxies are capable of intercepting users' requests and forwarding the requests to a higher level proxy if the requested data are not present in their local cache.

Outline

- Introduction
- Problem Formulation
- Computing the Induced Tree of Proxy Nodes
- Proxies with Unlimited Storage Capacities
- Proxies with Limited Storage Capacities
- Simulations

Problem Formulation

- The network is modeled by a connected graph $G(V, E)$.
- For a link $(u, v) \in E$, $d(u, v)$ is the distance of the link.
- Let s be the origin server. Suppose there are k proxies that can be used by s to host its contents, denoted by $P = \{p_1, p_2, \ldots, p_k\}$.
- The locations of the k proxies are given in prior.
- Each proxy, p_i, has a limited storage allocation to this server s, denoted by c_i.
- The origin server has a set of m data objects, denoted by $O = \{o_1, o_2, \ldots, p_m\}$. Each data object, o_i, has a size z_i.
- Every node, $u \in V$, has a read frequencey to data object o_i, denoted by $r(u, o_i), i \leq i \leq m$. Each data object, o_i, has an update frequency $w(o_i)$.

Problem Formulation (Cont.)

- The distance function to path $\pi(u, v)$ between nodes u and v is $d(u, v) = \sum_{(x,y)\in\pi(u,v)} d(x, y)$.
- Let $d(v, o_i)$ denote the distance from node v to object o_i.
- $d(v, o_i)$ is $d(v, p_j)$ if a request for retrieving o_i is served by proxy p_j and it becomes $d(v, s)$ if the request is missed by all the proxies and is finally served by the origin server s.
- The cost of the user at node v to retrieve o_i is $r(v, o_i) \times d(v, o_i) \times z_i$.
- The total retrieval cost of o_i by all cilents in the network is $readCost(o_i) = \sum_{v\in V} r(v, o_i) \times d(v, o_i) \times z_i$.

Problem Formulation (Cont.)

- The replicas of a data object at proxies need to be updated when the original copy is modified.
- When data object o_i is updated, the new version of o_i needs to be transmitted to the proxies that hold o_i.
- They assume the multicast model is used for the server to transmit updated data to proxies. That is, the route for multicasting from server s to proxies is a multicast tree (MT).
- Let $P(o_i)$ denote a set of proxies where o_i is replicated (including the server s) and $MT(s, P(o_i))$ the multicast tree rooted from s to reach all proxies in $P(o_i)$.
- The updating cost (write cost) of o_i is

$$writeCost(o_i) = w(o_i) \times z_i \times \sum_{(x,y) \in MT(s,P(o_i))} d(x, y).$$

Problem Formulation (Cont.)

- $readCost(o_i) + writeCost(o_i)$ is the total access cost to o_i by all clients in the network:
  $$Cost(o_i) = \sum_{v \in V} r(v, o_i) \times d(v, o_i) \times z_i + w(o_i) \times z_i \times \sum_{(x,y) \in MT(s, P(o_i))} d(x, y).$$
- The overall cost for all clients to access all m objects in the network is
  $$Cost = \sum_{i=1}^{m} Cost(o_i).$$

Problem Formulation (Cont.)

- Since each proxy has a limited storage capacity for s, the total size of s's data objects replicated at this proxy should not exceed this capacity. The constraint can be represented as:

$$\forall i : \sum_{j=1}^{m} \delta_{ij} \times z_j \leq c_i,$$

$$where|\delta_{ij} = \begin{cases} 1, & \text{if } p_i \in P(o_j) \\ 0, & \text{otherwise} \end{cases} , 1 \leq i \leq k.$$

Outline

- Introduction
- Problem Formulation
- Computing the Induced Tree of Proxy Nodes
- Proxies with Unlimited Storage Capacities
- Proxies with Limited Storage Capacities
- Simulations

Stable Routing

- If the routing in the network is stable, requests would always take the same route to the origin server.
- In this case, the routes from all clients to access the origin server s form a tree, where the root of the tree is s and all the leaf nodes are clients. Some (not all) nonleaf nodes are proxy nodes.
- Let T_s denote such a routing tree from all clients to the server s.
- Since we only concern about data replication at proxies and the proxy nodes are given in prior, we can focus on the induced tree that contains only proxy nodes.
- Let $T_s(P)$ denote the induced tree of T_s.

Induced Tree

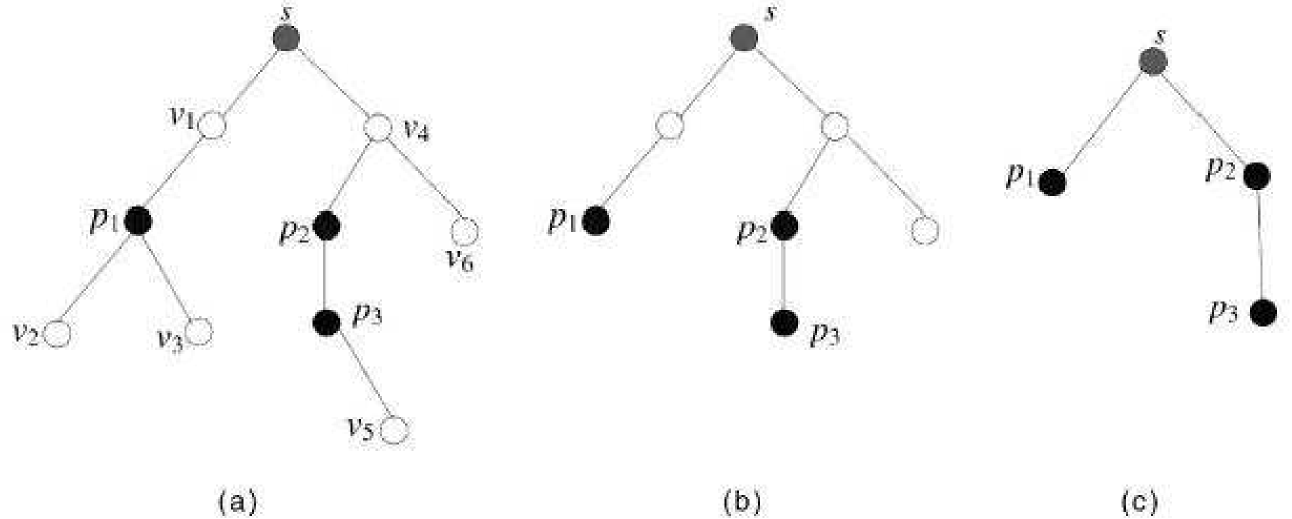


Fig. 1. An example of producing Induced Subtree (nodes in black are proxies). (a) T_s. (b) Intermediate Graph from T_s to $T_s(P)$. (c) $T_s(P)$.

Aggreate Read Frequency

- Let T_p denote a subtree of T_s and the root of T_p is p.
- p' is said to be a *direct child proxy* of p if p' is a proxy node in the subtree of T_p and there is no other proxy nodes in the path between p and p' along the tree.
- For any $p \in T_s(P)$, let $C(p)$ denote the set of *direct child proxies* of p in T_s.
- Fro any $p \in T_s(P)$, the aggregate read frequency of o_j is:

$$r^+(p, o_j) = \sum_{v \in T_p - \cup_{p' \in C(p)} T_{p'}} r(v, o_j).$$

Another Representation of $Cost(o_i)$

- Let $p(v)$, $v \in V$, be the first proxy node from v to root s along tree T_s. The distance $d(v, o_i)$ consists of two parts: $d(v, p(v))$ and $d(p(v), o_i)$. Notice that $d(p(v), o_i) = 0$ if $p(v)$ holds o_i,

$$
\begin{aligned}
Cost(o_i) &= \sum_{v \in V} r(v, o_i) \times (d(v, p(v)) + d(p(v), o_i)) \times z_i + writeCost(o_i) \\
&= \sum_{v \in V} r(v, o_i) \times d(v, p(v)) \times z_i + \\
&\quad \sum_{v \in V} r(v, o_i) \times d(p(v), o_i) \times z_i + writeCost(o_i) \\
&= \sum_{v \in V} r(v, o_i) \times d(v, p(v)) \times z_i + \\
&\quad \sum_{p \in T_s(P)} \left(\sum_{v \in T_p - \cup_{p' \in C(p)} T_{p'}} r(v, o_i) \times d(p(v), o_i) \times z_i \right) + writeCost(o_i) \\
&= \sum_{v \in V} r(v, o_i) \times d(v, p(v)) \times z_i + \\
&\quad \sum_{p \in T_s(P)} r^+(p, o_i) \times d(p, o_i) \times z_i + writeCost(o_i)
\end{aligned}
$$

An Example

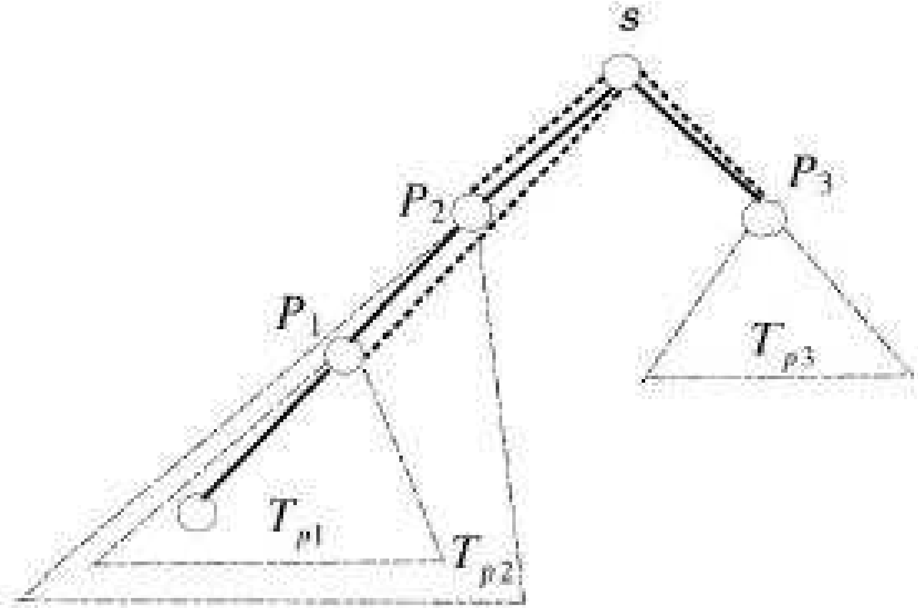


Fig. 2. An example of cost-saving by replicating data at proxies.

Saving Gain by Replicating o_i at p_1

$$\left(\sum_{u \in T_{p1}} r(u, o_i)\right) \times d(p_1, s) \times z_i - w(o_i) \times d(p_1, s) \times z_i$$

$$= \left(\sum_{u \in T_{p1}} r(u, o_i)\right) \times d(p_1, p_2) \times z_i + \left(\sum_{u \in T_{p1}} r(u, o_i)\right)$$

$$\times\, d(p_2, s) \times z_i - w(o_i) \times d(p_1, p_2) \times z_i - w(o_i) \times d(p_2, s) \times z_i$$

Saving Gain by Replicating o_i ***also at*** p_2

- When o_i is also replicated at p_2, all nodes in $(T_{p2} - T_{p1})$ will come to p_2 to read o_i.
- The distance saved for reading o_i at p_2 is $d(p_2, s)$.

$$\left(\sum_{u \in (T_{p2} - T_{p1})} r(u, o_i)\right) \times d(p_2, s) \times z_i$$

$$= \left(\sum_{u \in T_{p2}} r(u, o_i)\right) \times d(p_2, s) \times z_i - \left(\sum_{u \in T_{p1}} r(u, o_i)\right) \times d(p_2, s) \times z_i$$

Saving Gain by Replicating o_i also at p_3

- Considering also replicating o_i at p_3, which is similar to the case at p_1, the net gain by replicating o_i at p_3 is:
$$\left(\sum_{u \in T_{p3}} r(u, o_i)\right) \times d(p_3, s) \times z_i - w(o_i) \times d(p_3, s) \times z_i$$

Total Gain

- The net gain by replicating o_i at p_1, p_2, and p_3:
$$\left(\sum_{u \in T_{p1}} r(u,o_i) - w(o_i)\right) \times d(p_1,p_2) \times z_i +$$
$$\left(\sum_{u \in T_{p2}} r(u,o_i) - w(o_i)\right) \times d(p_2,s) \times z_i +$$
$$\left(\sum_{u \in T_{p3}} r(u,o_i) - w(o_i)\right) \times d(p_3,s) \times z_i$$
- Notice that the distances $d(p_1,p_2)$, $d(p_2,s)$, and $d(p_3,s)$ above are the distances from a proxy that has o_i to its first ancestor that also holds o_i in $T_s(P)$.
- Let $d(p,o_i)$, $p \in T_s(P)$, denote the distance from a proxy p to its first ancestor holding o_i in $T_s(P)$. For example, $d(p_1,o_i)$ is $d(p_1,p_2)$.
- The total net gain of replicating o_i at all proxies in $P(o_i)$ against the case of no replication of o_i is
$$\sum_{p \in P(o_i)} \left(\left(\sum_{u \in T_p} r(u,o_i) - w(o_i)\right) \times d(p,o_i) \times z_i\right).$$

Cost

- Assuming there is no replica of o_i placed at any proxies, i.e., only the server holds o_i, the total cost for accessing o_i is: $Cost^0(o_i) = \sum_{v \in V} r(v, o_i) \times d(v, p(v)) \times z_i + \sum_{p \in T_s(P)} r^+(p, o_i) \times d(p, s) \times z_i$.
- The cost of accessing o_i: $Cost(o_i) = Cost^0(o_i) - \sum_{p \in P(o_i)} \left(\left(\sum_{u \in T_p} r(u, o_i) - w(o_i) \right) \times d(p, o_i) \times z_i \right)$.
- The overall cost for the accesses to all m objects is:

$$
\begin{aligned}
Cost = \sum_{i=1}^{m} Cost(o_i) = \sum_{i=1}^{m} Cost^0(o_i) - \\
\sum_{i=1}^{m} \sum_{p \in P(o_i)} \left(\left(\sum_{u \in T_p} r(u, o_i) - w(o_i) \right) \times d(p, o_i) \times z_i \right)
\end{aligned}
$$

Outline

- Introduction
- Problem Formulation
- Computing the Induced Tree of Proxy Nodes
- Proxies with Unlimited Storage Capacities
- Proxies with Limited Storage Capacities
- Simulations

Independent between Each Other

- If proxies have unlimited storage, the optimal replication of all objects at proxies consists of the optimal replication of each object at proxies. The optimal replication of each object is independent from the others.

Lemma 1

Lemma 1 *If* $P(o_i)$ *is the optimal replication of* o_i *in* $T_s(P)$*, for any path* $\pi(p_1, p_2)$ *in* $T_s(P)$*, where* p_1*,* $p_2 \in P(o_i)$*, then* $P(o_i) \cup \{p | p \in \pi(p_1, p_2)\}$ *is still an optimal replication.*

Minimal Number of Replicas

- Notice that Lemma 1 states that if o_i is replicated at p_1 and p_2 and they are in a path in $T_s(P)$, then o_i can be replicated at any proxy node in between p_1 and p_2, because it does not incur any extra cost for updating the object but save the cost for reading the object.
- However, if there is no read access to o_i at a proxy node, there is no need to replicate it.
- So an optimal set of replicas also contains only the minimal number of replicas.

Lemma 2

Lemma 2 *Let* $P(o_i) = \left\{p | \sum_{u \in T_p} r(u, o_i) > w(o_i), p \in T_s(P)\right\}$*. Then,* $P(o_i)$ *is the optimal replication of* o_i *in* $T_s(p)$*.*

Proof: Prove it by contradiction. Assume $P(o_i)$ is not optimal and P_i^{opt} is the optimal replication of o_i. Then, there must exist a proxy $p' \in P_i^{opt}$, but $p' \notin P(o_i)$. Since $p' \notin P(o_i)$, we have $\sum_{u \in T_{p'}} r(u, o_i) \leq w(o_i)$. Consider the two cases:

Case $\sum_{u \in T_{p'}} r(u, o_i) < w(o_i)$:

Case $\sum_{u \in T_{p'}} r(u, o_i) = w(o_i)$:

The two case contradict to the assumption. So the lemma follows.

Opt-replic Algorithm

Opt-replic{
$P(o_i) \leftarrow s$;
$C \leftarrow \{s$'s children$\}$;
while $C \neq \emptyset$ **do**
pick any node $p \in C$ and remove p from C;
if $\sum_{u \in T_p} r^+(u, o_i) > w(o_i)$, **then**
$P(o_i) \leftarrow P(o_i) \cup p$;
$C \leftarrow C \cup \{p$'s children$\}$;
end-while
}

Theorem 1

Theorem 1 $P(o_i)$ *produced by Opt-replic algorithm is the set of optimal replication of* o_i

Proof: According to Lemma 2, $P(o_i)$ is the optimal replication of o_i.

Theorem 2

Theorem 2 *Opt-replic algorithm can produce the optimal replication of o_i in time $O(|P|^2)$, and the optimal replication of m objects in time $O(m|P|^2)$, where $|P|$ is the number of nodes in $T_s(P)$.*

Proof:

- The algorithm searches nodes in $T_s(P)$ at most once.
- At each node p, it computes $\sum_{u \in T_p} r(u, o_i) > w(o_i)$, which takes time $O(|T_p|)$.
- Thus, Opt-replic has complexity of $\sum_{p \in T_s(P)} |T_p| = O(|P|^2)$.
- Because the optimal replication of objects is independent from each other, by using Opt-replic algorithm to compute the optimal replication of m objects, it takes time $O(m|P|^2)$.

Outline

- Introduction
- Problem Formulation
- Computing the Induced Tree of Proxy Nodes
- Proxies with Unlimited Storage Capacities
- Proxies with Limited Storage Capacities
- Simulations

Theorem 3

They refer the problem of optimal replication of data objects at proxies that have limited storage capacities as ORLS.

Theorem 3 *The ORLS problem is NP-hard.*

A Greedy Heuristic

- The greedy algorithm traverses the nodes in $T_s(P)$ in a breadth-first fashion, starting from root s.
- At each node p it traverses, it evaluates the gain of replicating o_i at p defined as: $gain(p, o_i) = (\sum_{v \in T_p} r(u, o_i) - w(o_i)) \times d(p, o_i) \times z_i$.
- The objects are sorted in descending order according to their gains (the objects with 0 or negative gains are dropped out).
- Then, the algorithm simply replicates at p the first k objects that have the largest gains and can be accommodated at p.
- For the rest of the storage at p, the objects from the rest of the list are chosen to fill it up.

A Greedy Heuristic (Cont.)

Greedy{
  append s's children to a list L in the order from left to right;
  $p = \text{get-head}(L)$;
  greedy-call(p);
}

greedy-call(p) {
  if $p = \text{nil}$ **then return;**
  sort all objects o_i in descending order of $gain(p, o_i) > 0$;
  // denote the sorted object list as $o_{i_1}, o_{i_2}, \ldots, o_{i_l}$

  if $z_{i_1} + z_{i_2} + \ldots + z_{i_k} \leq c_p$ but $z_{i1} + z_{i_2} + \ldots + z_{i_k} + z_{i_{k+1}} > c_p$ **then**
    replicate $o_{i_1}, o_{i_2}, \ldots, o_{i_k}$ at p;
    search the rest of objects in the list to fill up the rest of space in p;
  append p's children to L in the order from left to right;
  $p' = \text{get-head}(L)$;
  greedy-call(p');
}

Theorem 4

Theorem 4 *The time complexity of Greedy algorithm is* $O(|P|^2 m + |P| mlgm)$.

Proof:

- Greedy algorithm computes the replication at each node in $T_s(P)$ by calling subroutine greedy-call(p).
- At each node p, it computes $gain(p, o_i)$ for all objects, which take time $O(|T_p|m)$.
- The sorting of $gain(p, o_i)$ for $1 \le i \le m$ takes time $O(mlgm)$.
- Therefore, computing the replication at all nodes in $T_s(P)$ takes time:

$$O\left(\sum_{p \in T_s(P)} (|T_p|m + mlgm)\right) = O(|P|^2 m + |P| mlgm).$$

A Knapsack-Based Heuristic

- The overall gain of replicating all possible objects is:

$$\sum_{i=1}^{m}\left(\left(\sum_{u\in T_p} r(u,o_i) - w(o_i)\right)\times d(p,o_i)\times z_i\right)x_i$$

$$wherex_i = \begin{cases} 1, & \text{if } o_i \text{ is replicated;} \\ 0, & \text{otherwise.} \end{cases} \tag{1}$$

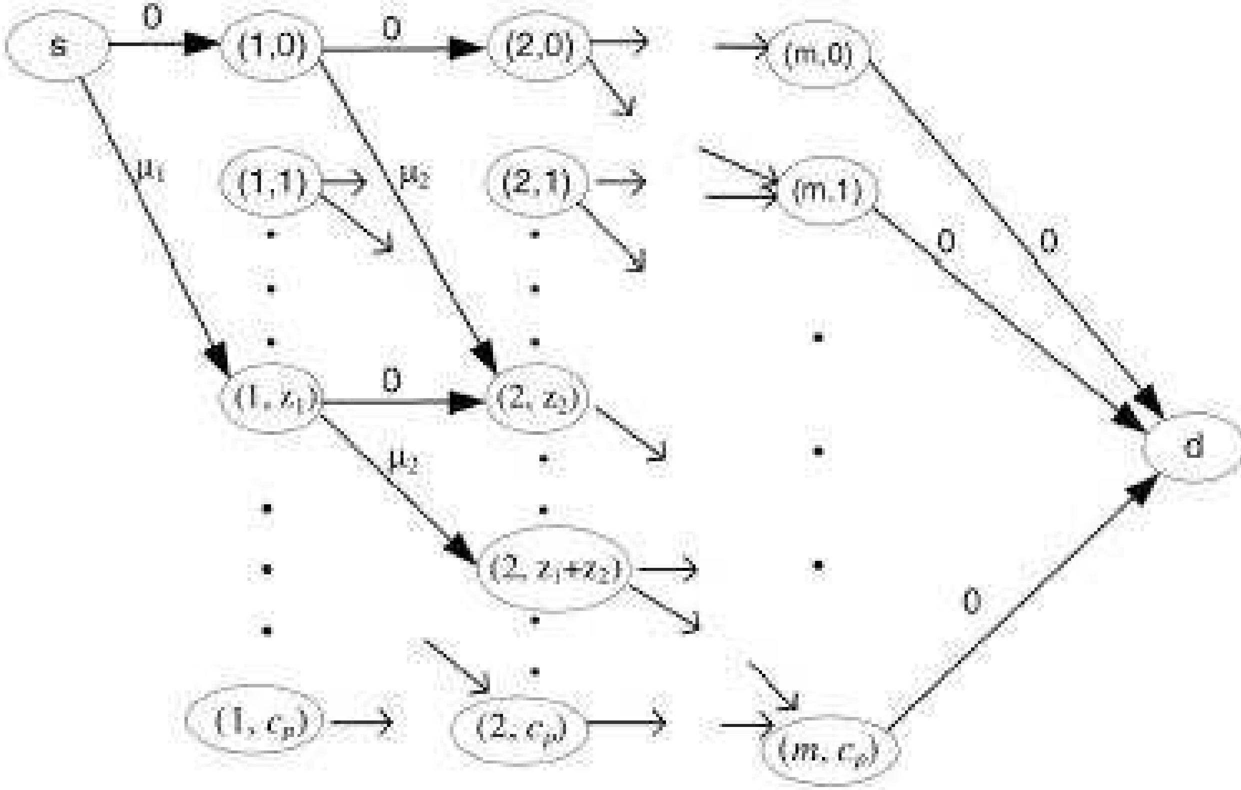
A Knapsack-Based Heuristic (Cont.)

- Let $\mu_i = \left(\sum_{u \in T_p} r(u, o_i) - w(o_i)\right) \times d(p, o_i) \times z_i$, which is a constant for a given o_i at p.
- The problem of finding the optimal replication of objects at p can be formulated as

$$max \sum_{i=1}^{m} \mu_i x_i$$
$$s.t. \begin{cases} z_1 x_1 + z_2 x_2 + \cdots + z_m x_m \leq c_p \\ x_i = 0 \text{ or } 1, \text{ for } i = 1, 2, \ldots, m. \end{cases} \tag{2}$$

- This is a typical knapsack problem. They use an auxiliary direct graph $G_A = (V_A, E_A)$ to solve this problem.

The Auxiliary Graph



Knapsack-h Algorithm

Knapsack-h{

append s's children to a list L in the order from left to right;

$p = \text{get-head}(L)$;

knapsack-call(p);

}

knapsack-call(p){

if $p = \text{nil}$ **then return**;

construct $G_A = (V_A, E_A)$ for p with capacity c_p;

compute the largest path from s to d and obtain values of $x_i, 1 \leq i \leq m$;

append p's children to L in the order from left to right;

$p' = \text{get-head}(L)$;

knapsack-call(p');

}

Theorem 5

Theorem 5 *The time complexity of Knapsack-h algorithm is* $O(m^2 \sum_{p \in T_s(P)} c_p^2)$.

Proof:

- At node p, the construction of $G_A = (V_A, E_A)$ and the computing of the longest path takes time $O(|V_A||E_A|) = O(m^2 c_p^2)$. Because $|V_A|$ is $m \times (c_p + 1) + 2$, and $|E_A|$ is at most $2|V_A|$.
- The algorithm traverses every node p in $T_s(P)$. It takes time $O(\sum_{p \in T_s(P)} m^2 c_p^2 = O(m^2 \sum_{p \in T_s P} c_p^2)$.

Outline

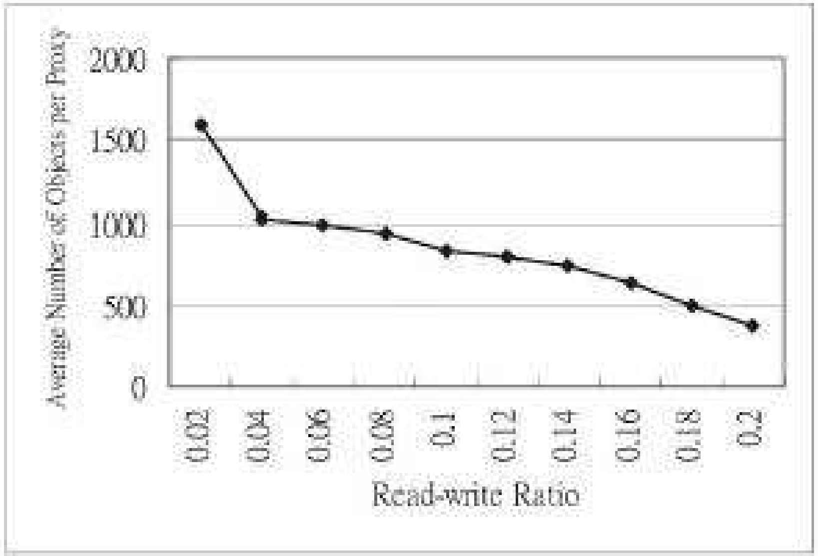
- Introduction
- Problem Formulation
- Computing the Induced Tree of Proxy Nodes
- Proxies with Unlimited Storage Capacities
- Proxies with Limited Storage Capacities
- Simulations

Simulation Parameters

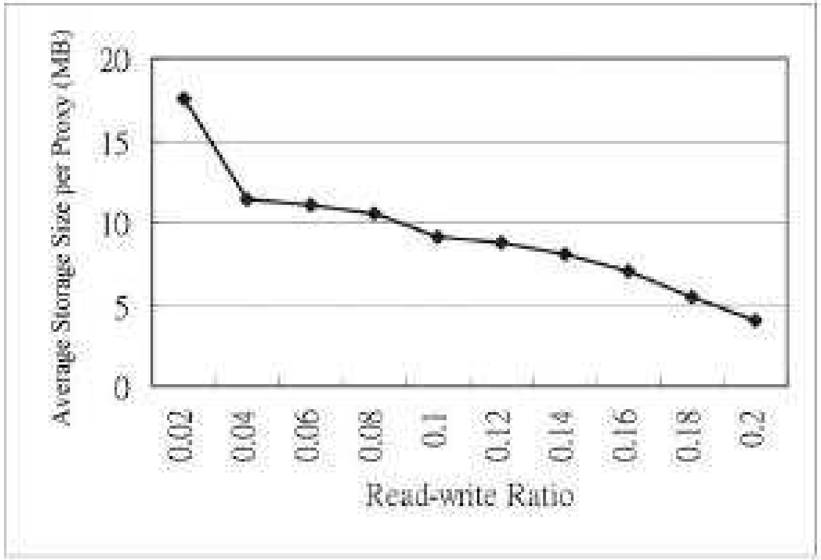
- Network topologies used in the simulations are generated using the Inet topology generator.
- The size of the networks used in the simulations is of 5,000 nodes.
- Server node, s, is randomly picked up from the graph.
- A set of proxy nodes are also randomly picked from the graph nodes. The default number of proxies in the simulations is 50.
- The total number of data objects stored at the Web server is 10,000.
- The distribution of object sizes follows a heavy-tailed characterization, which consists of a body and tail.
- The cut-off point of the body and the tail is approximately at 133K. By using this setting, more than 93 percent of objects fall into the body distribution. The mean size of objects is about 11K.

Simulation Parameters (Cont.)

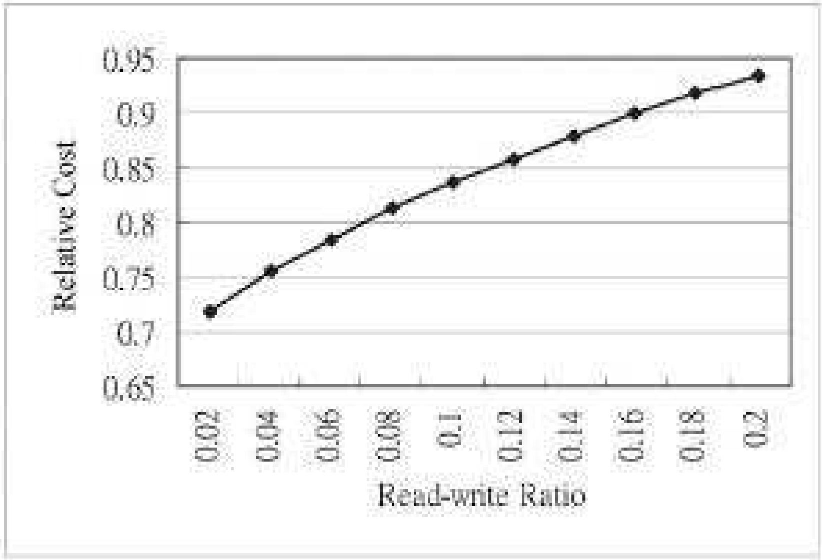
- The read frequency to data object o_i follows the Zipf-like distribution.
- Let $p(i)$ be the probability of accessing the ith most popular object. The Zipf-like distribution is: $p(i) = \frac{1}{i^\alpha}$, where α is typically between 0.6 and 0.8.
- During the simulations, the parameter α in the Zipf distribution is set to 0.75.
- They simply assume all objects have the same level of read-write ratio. Let α be the read-write ratio. The update frequency to o_i is $w(o_i) = \alpha \sum_{v \in T_s} r(v, o_i)$.


Average Number of Objects per Proxy
2000
1500
1000
500
0
0.02
0.04
0.06
0.08
0.1
0.12
0.14
0.16
0.18
0.2
Read-write Ratio

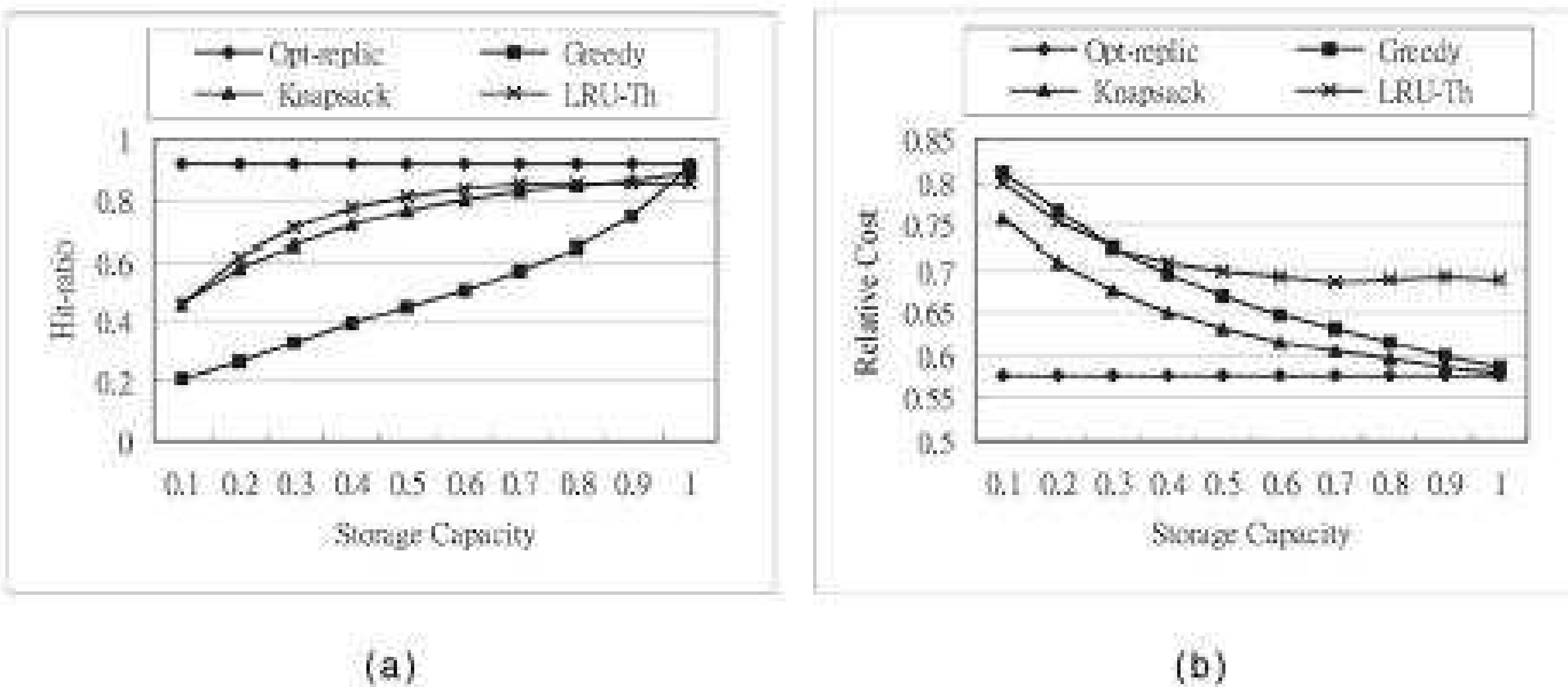
(a)


Average Storage Size per Proxy (MB)
20
15
10
5
0
0.02
0.04
0.06
0.08
0.1
0.12
0.14
0.16
0.18
0.2
Read-write Ratio

(b)


Relative Cost
0.95
0.9
0.85
0.8
0.75
0.7
0.65
0.02
0.04
0.06
0.08
0.1
0.12
0.14
0.16
0.18
0.2
Read-write Ratio

(c)



Opt-replic
Greedy
Knapsack
LRU-Th
Hit-ratio
1
0.8
0.6
0.4
0.2
0
0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1
Storage Capacity
(a)
Opt-replic
Greedy
Knapsack
LRU-Th
Relative Cost
0.85
0.8
0.75
0.7
0.65
0.6
0.55
0.5
0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1
Storage Capacity
(b)

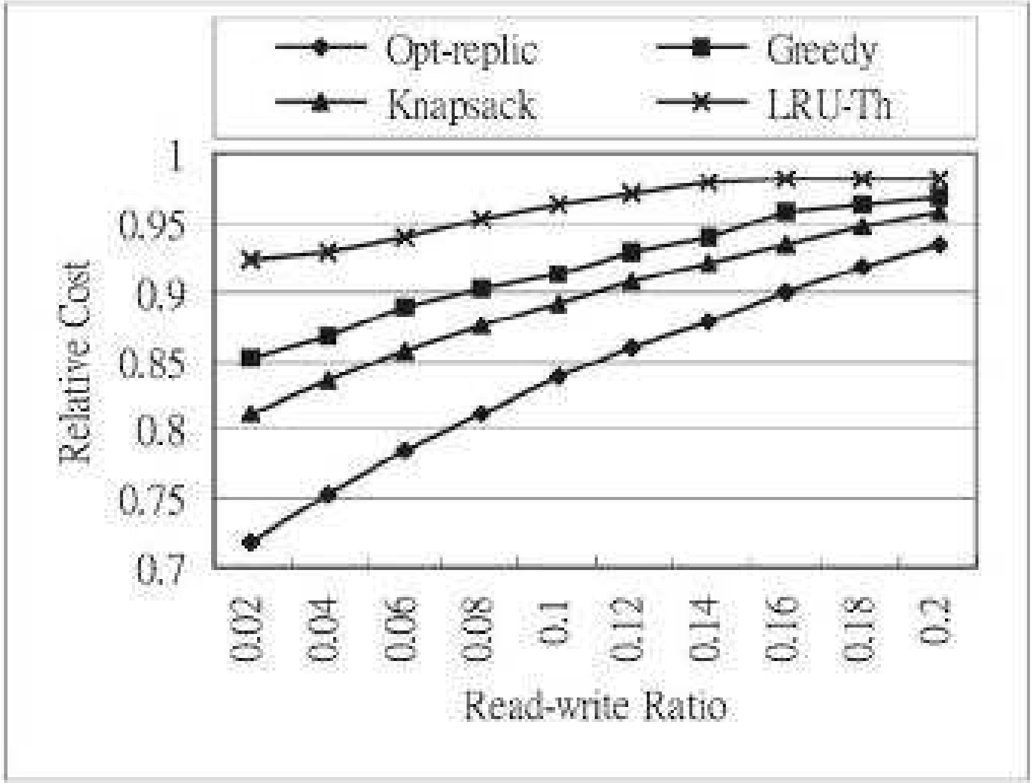


Fig. 7. Relative cost versus read-write ratio.

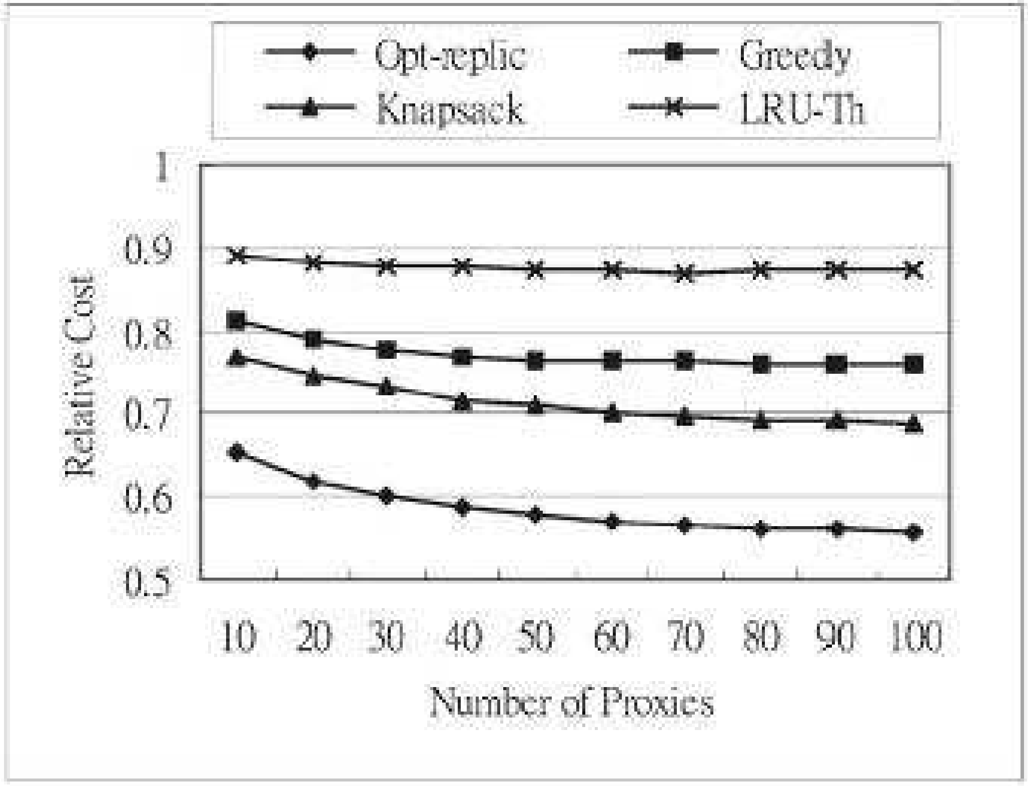


Fig. 8. Relative cost versus number of proxies.