A Greedy Algorithm for Capacity-Constrained Surrogate Placement in CDNs

Yifeng Chen[1], Yanxiang He[2], Jiannong Cao[3], and Jie Wu[4]

[1] State Key Laboratory of Water Resources and Hydropower Engineering Science, Wuhan University, Wuhan 430072, Hubei, China
[2] School of Computer, Wuhan University, Wuhan 430072, Hubei, China
{Csyfchen, Yxhe}@whu.edu.cn
[3] Department of Computing, Hong Kong Polytechnic University, Hung Hom, Kowloon, Hong Kong, China
Csjcao@comp.polyu.edu.hk
[4] Department of Computer Science & Engineering, Florida Atlantic University, Boca Raton, FL 33431, USA
Jie@cse.fau.edu

Abstract. One major factor that heavily affects the performance of a content distribution network (CDN) is placement of the surrogates. Previous works take a network-centric approach and consider only the network traffic. In this paper, we propose solutions to optimal surrogate placement, taking into consideration both network latency and capacity constraints on the surrogates. For CDNs with a tree topology, an efficient and effective greedy algorithm is proposed which minimizes network traffic while at the same time maximizing system throughput. Simulation results show that the greedy algorithm is far better than the existing optimal placement scheme that makes decisions based solely on network traffic. This suggests that capacity constraints on surrogates or server bottlenecks should be considered when determining surrogate placement, especially when the capacities of CDN servers are limited.

1 Introduction

A content distribution network (CDN) is a network optimized to deliver specific content such as static Web pages, streaming media, or real-time video or audio. The design of a CDN aims at quickly providing users with the most current content in a highly available fashion [1]. This is achieved by pushing hosted content from the origin server(s) to a set of surrogates located at the edge of the Internet closer to clients. For any client request, an appropriate surrogate is selected to deliver the requested content to the client on behalf of the origin server(s) [2]. Besides speeding up content delivery, CDNs can also reduce server workload and alleviate network congestion.

The performance of a CDN can be significantly affected by the decisions on 1) how many surrogates are needed, and 2) where they should be placed. Previous studies typically formulate this decision problem as the minimum *k*-median problem [3,4], the facility location problem [3], the minimum *k*-center problem [5], or for simple network topologies (e.g., line, ring, or tree), the dynamic programming problem [3,6-9].

All of these previous works [3-9], however, take a network-centric view of the issue of surrogate placement, assuming that a client's requests can always be directed to the surrogates closest to the client. Consequently, they consider only the network latency factor and the resultant placement scheme may very likely lead to an undesirable load concentration on some surrogates. In this paper, we argue that, in order to minimize the network traffic and maximize the system throughput, load balancing among surrogates should also be considered in surrogate placement. We call this problem the *capacity constrained surrogate placement problem* (CCSP).

In this paper, we focus on a simplified version of the CCSP problem in the context of *transparent data replication* [8-10], in which the access paths to a Web site are arranged as a tree with the origin server at the root. The aim of transparent data replication is to reduce the management overhead incurred by client redirection and to simplify the design of surrogate cooperation and load balancing. Fig. 1 illustrates a surrogate hierarchy for transparent data replication. A collection of surrogates, together with the origin server, is placed on $\{1, 6, 10, 15, 20\}$. The request issued from node 11 is forwarded toward the origin server along the unique path from 11 to 1. Normally, the surrogate placed on node 10 will intercept the request and immediately satisfy the request on behalf of the origin server. However, if the surrogate is overloaded, the request will be forwarded up the tree, until another available surrogate, say node 6, is able to serve the request. For any update activity, the update message will first be propagated from root, (i.e., the origin server node 1) to its immediate descendant surrogate node 6, and node 20. Then the update message will continue to be propagated down the surrogate hierarchy from node 6 to nodes 10 and 15.

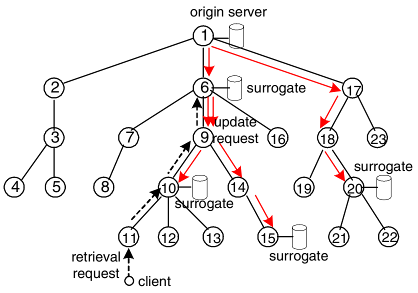

Fig. 1. Request-routing and consistency maintenance under transparent data replication

We employ queuing theory [11] to model server throughput and achieve load balancing among surrogates by redirecting part of the client requests initially assigned to the heavily loaded surrogates to other lightly loaded ones according to transparent data replication. We propose an efficient greedy algorithm to solve the CCSP problem. The performance of the proposed algorithm is evaluated in terms of communication cost and system throughput. We compare our algorithm with an existing optimal placement scheme that solely minimizes the communication cost and a random placement scheme that uniformly chooses sites to place surrogates. The simulation results show that our proposed CCSP approach significantly outperforms these two benchmarks.

2 Problem Formulation

In this section, we first develop a queuing model of the throughput of CDN servers, and then formulate the CCSP problem for tree networks.

We model each CDN server as an M/G/1/K*PS queuing system [11]. The arrival process of HTTP requests is assumed to be Poissonian with rate $(\lambda+\mu)$ (λ is the read rate, μ the write rate), whereas the service time has a general distribution with mean $\bar{x}$. The service discipline is processor sharing. The total number of requests that can be processed at one time is limited to K ($\bar{x}$ and K represent the processing power of each CDN server). Denoting the blocking probability by $P^{b\prime}$, we have

$$P^{b\prime} = \frac{(1-\rho)\rho^{K}}{(1-\rho^{K+1})} \tag{1}$$

where $\rho=(\lambda+\mu)\bar{x}$. Thus, the rate of blocked requests is given by $(\lambda+\mu)P^{b\prime}$. A CDN increases the throughput of the whole system by enabling the surrogates to cooperate for redirecting the overloaded amount of requests (i.e., $(\lambda+\mu)P^{b\prime}$) that have routed to one surrogate to other lightly loaded ones. Note that update requests should always be served locally and only retrieval requests can be redirected. Thus, if we define $P^{b}=(1+\mu/\lambda)P^{b\prime}$, the request blocking rate can be represented as λP^{b}. This transformation is reasonable, since the CDN servers are typically dominated by retrieval requests.

The network is modeled as a tree $T_r(V,E)$, where V is a set of nodes, $E\subseteq V\times V$ is a set of edges and $r\in V$ is the root where the origin server is located. Each node represents an autonomous system (AS) and each edge corresponds to a physical link connecting two AS's. For any node $v\in T_r$, we denote by T_v the subtree of T_r rooted at v.

Assume that the origin server holds N objects. The size of each object i is denoted by $s_i(1\leq i\leq N)$. For each object i, every node v is associated with a nonnegative retrieval rate $\lambda_{v,i}$. The origin server is responsible for propagating update information down the surrogate hierarchy and is additionally associated with a nonnegative update rate μ_i for object i. Any link (u,v) in E is associated with a distance metric $d(u,v)$, which could be interpreted as bandwidth, hop counts, link cost, etc. Assuming that $\pi_{x,y}$ is a path between node x and y, the distance associated with path $\pi_{x,y}$ could be represented as $d(x,y)=\sum_{(u,v)\in\pi_{x,y}} d(u,v)$. We use $f(s_i,d(u,v))$ to denote the data transmission cost for object i traverses link (u,v) or path $\pi_{u,v}$, which measures the resource utilization on that link or path for transferring object i from node u to v.

Suppose M surrogates are to be placed on a set of domains $P(P\subseteq V$, $r\in P$ and $|P|=M)$. For any node $v\in T_r$, we say a node is the *parent surrogate* of v, denoted by $C(v,P)$, if it is the first node in $P\backslash\{v\}$ that is seen while going up from v to the root r, i.e., the lowest ancestor of v which is contained in $P\backslash\{v\}$. Also, the *immediate descendant surrogates* of any v, denoted by $D(v)$, is defined as follows. If $v\notin P$, $D(v)=\{u\text{: } u\in P \wedge u\in T_v \wedge C(u,P)=C(v,P)\}$; if $v\in P$, $D(v) = \{u\text{: } u\in P \wedge C(u,P)=v\}$.

Now, suppose a set of surrogates P are placed on the network, the reduction of data transfer cost, denoted by $Cost(T_r,P)$, is ready to be obtained by:

$$Cost(T_r,P)=\sum_{i=1}^{N}\sum_{v\in P\setminus\{r\}}((1-P_v^b)\lambda_{v,i}^t f(s_i,d(v,r))-u_i f(s_i,d(v,C(v,P)))) \tag{2}$$

where the first term corresponds to the total decrease of retrieval cost and the second term represents the total increase of update cost due to the placement of surrogates. $\lambda_{v,i}$ denotes the access rate to object i issued from node v. $\lambda_{v,i}^t$ denotes the total retrieval requests for object i that traverse node v:

$$\lambda_{v,i}^t=\lambda_{v,i}+\sum\nolimits_{u\in B_v}P_u^b\lambda_{u,i}^t \tag{3}$$

where B_v is the children of v, and P_v^b is the blocking probability of v. Here we extend the concept of blocking probability: If v is a surrogate node (i.e., $v\in P$), P_v^b would be derived via a queuing model; otherwise, P_v^b is set to one, meaning that, for the nodes where no surrogates are located, all the incoming requests will be forwarded to their parent nodes. We define $\lambda_v^t=\sum_{i=1}^{N}\lambda_{v,i}^t$ and $\mu^t=\sum_{i=1}^{N}\mu_i$ to compute P_v^b.

Under the given request-routing mechanism, the drop of requests occurs only if the origin server is overloaded. The total requests blocked in the CDN therefore is

$$Block\ (T_r,P)=P_r^b\lambda_r^t \tag{4}$$

where λ_r^t denotes the total retrieval request rate directed to r after placing a set of surrogates P. Now, we are ready to define the CCSP problem in tree topologies: *Given $T_r(V,E)$, traffic pattern, and surrogate capacity constraints, find a set of M surrogates $P(P\subseteq V,\ r\in P,\ |P|=M)$ such that the objective function* (5) *is satisfied.*

$$Obj(T_r,P)=\underset{P\subseteq V,|P|=M,r\in P}{Max}(Cost(T_r,P)-\gamma Block(T_r,P)) \tag{5}$$

γ in (5) is a penalty coefficient to make a tradeoff between traffic reduction and load balancing among surrogates.

3 A Greedy Algorithm

From the computation of P_v^b, it's easy to verify that the CCSP problem for tree topologies can not be solved via a dynamic programming approach similar to that used in [8]. In this section, we develop an efficient greedy algorithm.

The greedy algorithm is illustrated in *Algorithm 1*. Initially, we set $P=\{r\}$ and the network cost reduction to zero. The objective is determined by the dropped requests. Then the algorithm iterates and chooses one surrogate in each step until M surrogates are chosen. In each iteration, for $\forall v\in V\setminus P$, we compute the objective increment assuming v is added to P. The node that yields the maximum objective increment is chosen and added to P. The objective increment of candidate node v, besides the contribution of v itself, includes (a) modifying the retrieval cost reduction of v's ancestor surrogates (the ratio of the request directed to and the blocking probabilities of these surrogates will change when a surrogate is placed on v); and (b) modifying the update cost of immediate descendant surrogates whose parent surrogate is $C(v,P)$ (their parent surrogate has changed from $C(v,P)$ to v).

The objective increment can be computed in the following fashion. Suppose a set of surrogates $P(P\subset V,\ |P|<M,\ r\in P)$ has been placed over the network, and a candidate node $v(v\in V\backslash P)$ is intended to join P. Define by $A(v)$ the ordered ancestor nodes of v, $v\notin A(v)$, i.e., the elements in $A(v)$ are the nodes ordered as seen while going up from v to the root r. Obviously, the first element in $A(v)$ is the parent of v, and for any successive node u and w in $A(v)$, w is the parent of u. After the candidate v joins in P, the increment of data transfer cost reduction $\Delta Cost(T_r,P\cup\{v\})$ and that of objective $\Delta Obj(T_r, P\cup\{v\})$ can be obtained by the following steps.

Step 1: compute the contribution of v itself

$$\Delta Cost(T_r,P\cup\{v\})=\sum_{i=1}^{N}((1-P_v^{\,b})\lambda_{v,i}^{t}f(s_i,d(v,r))-\mu_i f(s_i,d(v,C(v,P)))$$

Step 2: modify the retrieval cost reduction of ancestor surrogates of v

Let $\Delta\lambda=-(1-P_v^{\,b})\lambda_v^{t}$, $\Delta\lambda_i=-(1-P_v^{\,b})\lambda_{v,i}^{t}$

Then obtain a node u from $A(v)$ in order until all the elements are traversed. Note that the variable with a superscript of *new* corresponds to the case where v has joined in P.

If $u\notin P$, set $\lambda_u^{t,new}=\lambda_u^{t}+\Delta\lambda$, $\lambda_{u,i}^{t,new}=\lambda_{u,i}^{t}+\Delta\lambda_i$

Otherwise if $u\in P$, set $\lambda_u^{t,new}=\lambda_u^{t}+\Delta\lambda$, $\lambda_{u,i}^{t,new}=\lambda_{u,i}^{t}+\Delta\lambda_i$, compute $P_u^{\,b,new}$ by $\lambda_u^{t,new}$

$$\Delta\lambda=\Delta\lambda-(1-P_u^{\,b,new})\,\lambda_u^{t,new}+(1-P_u^{\,b})\lambda_u^{t}$$
$$\Delta\lambda_i=\Delta\lambda_i-(1-P_u^{\,b,new})\,\lambda_{u,i}^{t,new}+(1-P_u^{\,b})\lambda_{u,i}^{t}$$

$$\Delta Cost(T_r,P\cup\{v\})=\Delta Cost(T_r,P\cup\{v\})+\sum_{i=1}^{N}(((1-P_u^{\,b,new})\lambda_{u,i}^{t,new}-(1-P_u^{\,b})\lambda_{u,i}^{t})f(s_i,d(u,r)))$$

Step 3: modify the update cost of the immediate descendant surrogates of v

$$\Delta Cost(T_r,P\cup\{v\})=\Delta Cost(T_r,P\cup\{v\})+|D(v)|\sum_{i=1}^{N}\mu_i f(s_i,d(v,C(v,P)))$$

Step 4: compute $Block(T_r,P\cup\{v\})$ and $\Delta Obj(T_r,P\cup\{v\})$

$$Block(T_r,P\cup\{v\})=P_r^{\,b,new}\lambda_r^{t,new}$$
$$\Delta Obj(T_r,P\cup\{v\})=\Delta Cost(T_r,P\cup\{v\})-\gamma(Block(T_r,P\cup\{v\})-Block(T_r,P))$$

Algorithm 1. The greedy algorithm for surrogate placement

```
set $P=\{r\}$, set $\lambda_{v,i}^{t}=\sum_{u\in T_v}\lambda_{u,i}$ , $\lambda_v^{t}=\sum_{i=1}^{N}\lambda_{v,i}^{t}$ , for $\forall v\in T_r$, $\forall i$ $(1\le i\le N)$,
set $Cost(T_r,P)=0$, compute $Block(T_r,P)$, $Obj(T_r,P)$;
while($|P|\le M$){
    for $\forall v\in V\backslash P$, compute $\Delta Cost(T_r,P\cup\{v\})$, $Block(T_r,P\cup\{v\})$ and $\Delta Obj(T_r,P\cup\{v\})$;
    find $v\in V\backslash P$ such that $\Delta Obj(T_r,P\cup\{v\})$ is maximized;
    $P\leftarrow P\cup\{v\}$, $Cost(T_r,P)\leftarrow Cost(T_r,P)+\Delta Cost(T_r,P)$,
    $Obj(T_r,P)\leftarrow Obj(T_r,P)+\Delta Obj(T_r,P)$;
  for $\forall u\in A(v)$, update $\lambda_u^{t}$, $\lambda_{u,i}^{t}$ in order;
  for $\forall u\in D(v)$, $C(u,P)\leftarrow v$; }
```

4 Performance Evaluation

We have evaluated the performance of the proposed algorithm through simulations, in comparison with two baseline algorithms: a random algorithm and a throughput-oblivious dynamic programming (DP) algorithm.

We use synthetic tree topologies and traffic pattern to evaluate the algorithms, as in [8]. Tree topologies are created randomly in a breadth-first manner with two parameters: the total number of nodes (treeSize) and the maximum degree of a tree node (treeDegree). Each tree edge is associated with a distance randomly distributed in (0,1). Every node v is associated with a retrieval rate λ_v and values, $\bar{x}$ and K, uniformly distributed in (minSvTime, maxSvTime) and in (minJobLimit, maxJobLimit), respectively. The root r is further associated with an update rate of μ uniformly distributed in (minWtRate, maxWtRate).

The origin server holds a collection of N Web objects. The access popularity of the objects follows a Zipf-like distribution [12,13] with a parameter of θ_r for retrieval and θ_w for update. Each object i is assigned an object size of s_i, whose distribution has been found to be heavy-tailed [13]. The cumulative distribution function is given by

$$F(s)=1-(s_0/s)^{\beta} \quad \beta, s_0>0, s\geq s_0 \tag{6}$$

where β is known as the tail index, and s_0 represents the smallest possible value of the random object size in the heavy-tailed distribution. For simplicity, we set $f(s_i,d(u,v))= s_i*d(u,v)$. Default parameter settings are summarized in Table 1.

We vary the number of nodes from 60 to 1000 and examine the impacts on surrogate placement decision of the penalty coefficient, traffic volume, and server capacity. Fig. 2 shows a typical simulation result on a 600-node tree with M=0.3*treeSize.

We have made the following observations: (1) The greedy algorithm significantly outperforms the benchmarks in both network cost reduction and dropped request rate; (2) The greedy algorithm is not very sensitive to the penalty coefficient. A larger γ, however, will lead to a decrease in blocked requests at slight cost of network traffic; (3) When the traffic is relatively small, adding one more surrogate can absorb a significant amount of traffic and remarkably improve the performance of the system. As traffic increases, more surrogates are needed to achieve the same normalized performance; (4) When candidate surrogates are configured powerful (i.e., set $\bar{x}$ close to zero), the

Table 1. Default simulation parameter settings

Parameter	Setting	Parameter	Setting	Parameter	Setting
minRdRate	1	minSvTime	0.0001	θ_r	0.8
maxRdRate	80	maxSvTime	0.01	θ_w	0.4
minWtRate	1	minJobLimit	50	β	1.2
maxWtRate	80	maxJobLimit	300	s_0	4
treeDegree	6	N	1000	γ	10

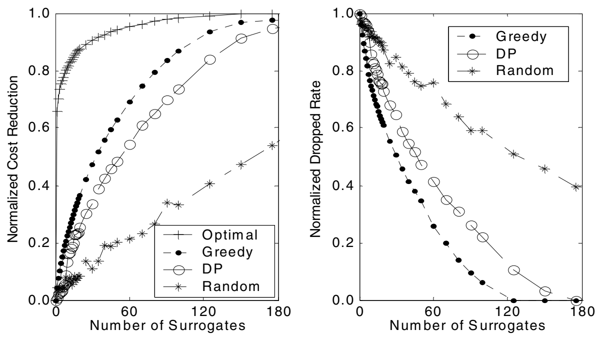

Fig. 2. Traffic reduction (normalized by "optimal" measure of placing a maximum of *M* surrogates) and blocked request rate (normalized by "optimal" measure of placing only a single surrogate at the root of the tree)

greedy algorithm can offer a performance close to optimal. Otherwise, the network cost reduction has to be traded off against the system throughput; (5) Heterogeneity in CDN servers and skewness in traffic pattern do not hurt the performance of the proposed greedy algorithm.

5 Conclusion

In this paper, we have investigated the capacity constrained surrogate placement problem (CCSP), aiming at minimizing the network traffic while maximizing the throughput of a CDN. An efficient greedy algorithm is developed to solve the problem in the context of transparent data replication.

The performance of the proposed algorithm is compared with a random solution and a dynamic programming based optimal solution, that makes decisions considering only data transmission cost. The simulation results demonstrate that the proposed greedy algorithm has a performance close to optimality and can find the placement scheme that remarkably increases the throughput of the system. Therefore, capacity constraints on surrogates or server bottlenecks should be integrated into the surrogate placement decision. This is especially the case when the power of CDN servers is limited for some reason. If the traffic volume increases roughly in proportion in the domains, an incremental or amortized surrogate placement scheme will be appropriate, just as the proposed greedy algorithm does.

Acknowledgement

This work is supported in part by the University Grant Council of Hong Kong under the CERG grant PolyU 5075/02E, the Hong Kong Polytechnic University under the grant G-YY41 and the National Natural Science Foundation of China under the grant 90104005.

References

1. Lazar, I., Terrill, W.: Exploring Content Delivery Networking. IEEE IT Pro. (2001) 47-49
2. Day, M., Cain, B., Tomlinson, G., Rzewski, P.: A Model for Content Internetworking. RFC 3466. Network Working Group (2003)
3. Qiu, L., Padmanabhan, V.N., Voelker, G.M.: On the Placement of Web Server Replicas. Proc. IEEE INFOCOM'01, Vol. 3 (2001) 1587-1596
4. Li, Y., Liu, M.T.: Optimization of Performance Gain in Content Distribution Networks with Server Replicas. Proc. 2003 Symp. Applications and the Internet (2003)
5. Cronin, E., Jamin, S., Jin, C., Kurc, A.R., Raz, D., Shavitt, Y.: Constrained Mirror Placement on the Internet. IEEE J. Select. Areas Commun., Vol. 20. **7** (2002) 1369-1381
6. Li, B., Golin, M.J., Italiano, G.F., Deng, X., Sohraby, K.: On the Optimal Placement of Web Proxies in the Internet. Proc. IEEE INFOCOM'99 (1999) 1282-1290
7. Jia, X., Li, D., Hu, X., Du, D.: Placement of Read-Write Web Proxies on the Internet. Proc. IEEE ICDCS'01 (2001) 687-690
8. Xu, J., Li, B., Lee, D.L.: Placement Problems for Transparent Data Replication Proxy Services. IEEE J. Select. Areas Commun., Vol. 20. **7** (2002) 1383-1398
9. Krishnan, P., Raz, D., Shavitt, Y.: The Cache Location Problem. IEEE/ACM Trans. Networking, Vol.8. **5** (2002) 568-582
10. Heddaya, A., Mirdad, A.: WebWave: Globally Load Balanced Fully Distributed Caching of Hot Published Documents. Proc. IEEE ICDCS'97 (1997) 160-168
11. Cao, I., Andersson, M., Nyberg, C., Kihl, M.: Web Server Performance Modeling Using an M/G/1/K*PS Queue. Proc. 10th Int'l Conf. Telecommunications, Vol. 2. (2003) 1501-1506
12. Breslau, L., Cao, P., Fan, L., Phillips, G., Shenker, S.: Web Caching and Zipf-like Distributions: Evidence and Implications. Proc. IEEE INFOCOM'99, New York (1999) 126-134
13. Mahanti, A., Williamson, C., Eager, D.: Traffic Analysis of a Web Proxy Caching Hierarchy. IEEE Network (2000) 16-23

Appendix: Correctness Proof for Algorithm 1

Theorem 1. Algorithm 1 is correct, and can be computed in $O(ML)$ time. L is the *path length* of T_r, which is defined as the sum over T_r of the number of ancestors of each node.

Proof. The time complexity of the algorithm is straightforward. The proof for the correctness can be reduced to proving that for $\forall u \in A(v)$, the computation of $\lambda_u^{t,new}$ (and similarly $\lambda_{u,i}^{t,new}$ for $\forall i(1 \le i \le N)$) is correct. We first prove the following Lemma.

Lemma 1. For any surrogate placement scheme P in tree topologies, there holds

$$\lambda_v^t = \lambda_v^{t,0} - \Sigma_{u \in D(v)} \lambda_u^{t,0} + \Sigma_{u \in D(v)} P_u^b \lambda_u^t, \text{ for } \forall v \in T_r \qquad (7)$$

where $\lambda_v^{t,0}$ is the corresponding result after execution of the first step in *Algorithm 1.*

Proof: The proof is done by induction.
(1) Basis: when $P=\{r\}$, we have $D(v)=\varnothing$, $\lambda_v^t=\lambda_v^{t,0}$ for $\forall v \in T_r$. Thus (7) trivially holds.
(2) Induction: Suppose (7) holds when a set of surrogates $P(P \subset V, |P|<M, r \in P)$ is placed over the network. Now we prove that (7) still holds after any node $v(v \in V \backslash P)$ is added to P.

First, according to the request-routing mechanism, placing a surrogate on v can only affect the retrieval requests of its ancestor nodes $A(v)$. Therefore, based on the induction hypothesis, (7) holds for $\forall u\in T_r\backslash A(v)$, and $\lambda_u^{t,new}=\lambda_u^t$. For $A(v)$, we first consider the first element u in $A(v)$, i.e., the parent of v. Obviously, $D(v)\subseteq D(u)$. Now due to the join of v, $D^{new}(u)=(D(u)\text{-}D(v))\cup\{v\}$. According to the algorithm, $\lambda_u^{t,new}=\lambda_u^t+\Delta\lambda=\lambda_u^t\text{-}(1\text{-}P_v^b)\lambda_v^t$. By $\lambda_u^t=\lambda_u^{t,0}\text{-}\Sigma_{w\in D(u)}\lambda_w^{t,0}+\Sigma_{w\in D(u)}P_w^b\lambda_w^t$ (induction hypothesis)

$$\begin{aligned}\lambda_u^{t,new}&=\lambda_u^{t,0}\text{-}\Sigma_{w\in D(u)\text{-}D(v)}\lambda_w^{t,0}+\Sigma_{w\in D(u)\text{-}D(v)}P_w^b\lambda_w^t+(\lambda_v^{t,0}\text{-}\Sigma_{w\in D(v)}\lambda_w^{t,0}+\Sigma_{w\in D(v)}P_w^b\lambda_w^t\text{-}\lambda_v^{t,0})\text{-}(1\text{-}P_v^b)\lambda_v^t\\&=\lambda_u^{t,0}\text{-}\Sigma_{w\in D^{new}(u)}\lambda_w^{t,0}+\Sigma_{w\in D^{new}(u)}P_w^b\lambda_w^t\\&=\lambda_u^{t,0}\text{-}\Sigma_{w\in D^{new}(u)}\lambda_w^{t,0}+\Sigma_{w\in D^{new}(u)}P_w^{b,new}\lambda_w^{t,new}\end{aligned}$$

(7) holds. Then for the successive element x of u in $A(v)$, if $u\notin P$, it is completely the same as u. Otherwise, if $u\in P$, there is evidently $u\in D(x)$, but $D(u)\not\subset D(x)$. Therefore, $D^{new}(x)=D(x)$. According to the algorithm, $\lambda_x^{t,new}=\lambda_x^t+\Delta\lambda=\lambda_x^t\text{-}(1\text{-}P_v^b)\lambda_v^t\text{-}(1\text{-}P_u^{b,new})\lambda_u^{t,new}+(1\text{-}P_u^b)\lambda_u^t$. By $\lambda_x^t=\lambda_x^{t,0}\text{-}\Sigma_{w\in D(x)}\lambda_w^{t,0}+\Sigma_{w\in D(x)}P_w^b\lambda_w^t$ (induction hypothesis)
$\lambda_x^{t,new}=\lambda_x^{t,0}\text{-}\Sigma_{w\in D(x)}\lambda_w^{t,0}+\Sigma_{w\in D(x)\text{-}u}P_w^b\lambda_w^t+P_u^b\lambda_u^t\text{-}(1\text{-}P_v^b)\lambda_v^t\text{-}(1\text{-}P_u^{b,new})\lambda_u^{t,new}+(1\text{-}P_u^b)\lambda_u^t$

$$\begin{aligned}&=\lambda_x^{t,0}\text{-}\Sigma_{w\in D^{new}(x)}\lambda_w^{t,0}+\Sigma_{w\in D^{new}(x)}P_w^{b,new}\lambda_w^{t,new}+\lambda_u^t\text{-}(1\text{-}P_v^b)\lambda_v^t\text{-}\lambda_u^{t,new}\\&=\lambda_x^{t,0}\text{-}\Sigma_{w\in D^{new}(x)}\lambda_w^{t,0}+\Sigma_{w\in D^{new}(x)}P_w^{b,new}\lambda_w^{t,new}\end{aligned}$$

(7) holds. Based on this approach, we can prove one by one that (7) holds for all the elements in $A(v)$. Thus *Lemma 1* is true. Noticing that (7) is equivalent to $\lambda_v^t=\sum_{i=1}^{N}\lambda_{v,i}^t$, where $\lambda_{v,i}^t$ is computed by (3), it can be trivially inferred that *Theorem 1* is true.