ENHANCING AVAILABILITY FOR DISTRIBUTED REPLICATED SERVICES CONSIDERING NETWORK EDGE AVAILABILITY

Manghui Tu[1], Liangliang Xiao[2] and Dianxiang Xu[3]

[1]Department of CITG, Purdue University Northwest, Hammond, Indiana
[2]Department of CSIT, Frostburg State University, Frostburg, Maryland
[3]Department of Computer Science, Boise State University, Boise, Idaho

ABSTRACT

Mechanism to improve data or service availability is critical for an enterprise to ensure the quality of service in terms of availability. Replication has been used to improve system availability. The number and location of the replicas are two impact factors on availability. In this paper, we will consider the impact of the node and network edge failures on the availability of replicated data or services. The Effective availability modeling approach is designed and efficient availability computing algorithms are developed to model and compute availability of replicated services for systems with the tree topology. The availability enhancement problem (maximizing the objective function) is transformed to a p-median problem (minimizing the objective function) through re-define the availability enhancement problem. An efficient replica allocation algorithm is developed to improve data availability in tree networks, with a runtime complexity of $O(K|V|^2)$, where K is the number of replicas and $|V|$ is the number of nodes in the tree network. Finally, experimental studies have been conducted to evaluate how efficient and effective the proposed availability computing algorithm and the availability enhancement algorithm on improving the availability of replicated data or services. The results show that the proposed solutions are efficient and effective on availability computing and availability enhancement.

KEYWORDS

Network Protocols, Wireless Network, Mobile Network, Virus, Worms &Trojon

1. INTRODUCTION

It is well known that the utility of the data and service is mainly limited by availability [1, 2, 3, 4], and the dynamic environment and unreliable internet connectivity raise the concerns on availability of distributed systems [5, 6, 7, 8, 9]. Replication and erasure coding techniques [7, 8, 9, 10, 11, 12, 30] have been used to enhance data availability. However, the data object that is replicated also impacts the effective availability. Replicating data at highly available sites will result in high data availability. Also, widely distributing data can result in higher availability than distributing data in local clusters. In [8, 9], it has shown that even 100% local availability does not necessarily provide high availability to end users. The network failures may, for 1.5-2 %of the time, prevent clients from successfully accessing a cluster [5, 6, 7, 13]. Thus, the location of data stored in the system plays a significant role in overall system availability.

Most of the existing research on system availability consider the availability of the storage nodes only [11, 12, 14]. Some other research also considers the impact of data consistency on service availability [8, 9]. All these research do not consider the reach availability of users' requests to service/data sites. As discussed earlier, availability of the network links also impacts service data

availability, especially the reach availability of users' requests. In [6, 7], network link availability is considered, but the evaluation algorithm itself requires exponential computation time relative to the number of storage nodes even in a tree graph with a single replica. Thus, it is not feasible to compute the availability of distributed systems with replications. Also, the work in [6, 7] does not provide an availability model in a system with more than one replica, i.e, the system has multiple service/data sites available to serve a user's request.

In this paper, we consider modeling the service/data reach availability (i.e., the availability of service/data to remote requests) in distributed systems in a tree network. Data or services are replicated in the system to improve both performance and availability. Since the system has multiple service/data sites available to serve a user's request, a single site failure does not lead to the unavailability to user's request. Therefore, our problem is fundamentally different from network reliability problem (reliable only ensures the access to a single site) [15]. To assess system availability, efficient algorithms are developed to compute system availability considering both node and link failures.

The remainder of the paper is organized as follows. System modeling is described in Section II. An algorithm on computing system availability in a tree network is presented in Section III. Section IV, issues on how to reduce the runtime complexity and how to improve the service/data availability will be discussed. Efficient replica allocation algorithms to improve system availability will be introduced in Section V. Experimental studies conducted will be described in Section VI to give some numerical results of the proposed replica allocation algorithm applied in a system with tree network topology. Section VII briefly discusses some related works in availability modeling and enhancement and Section VIII concludes the paper.

2. SYSTEM & AVAILABILITY MODELING

The Study shows that user access requests usually follow a constant path routing to data sites and most routes are stable in days or even weeks [16]. So the data access routes in such a distributed system can be modeled as a tree graph, $T = (V, E)$, as shown in Fig.1. Let $e(u, v)$ denote the edge that directly links two nodes u and v, and $\delta(u, v)$ denote a single path between u and v. Note that $e(u, v)$ itself is a path between u and v with a single edge. The system hosts a set of data objects replicated to different nodes. For simplicity, we consider a single data object d. The set of nodes hosting replicas is defined as the **residence set**, $R = \{u | u \in V$ and u holds a replica$\}$, and $|R|$ is the number of replicas. A read request needs to access the closest node to the request. If it is not available, then, the request will be further forwarded to another node. Let r denote a read request and $A^r(u)$ denote the number of read access requests from a node u over a time period. Majority of the user accesses are reading accesses, update accesses are not considered in this research.

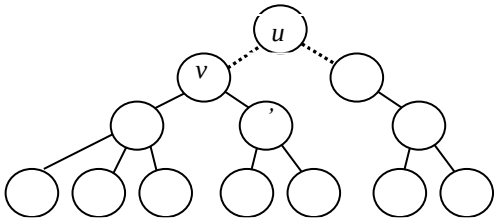


Fig.1 A sample tree network with replica.

In a distributed system, data or service availability may be affected by other factors such as node up/downtime, system load, data consistency, and network failure or congestion [5, 8, 9]. According to [8, 9], weaker consistency would result in higher availability. Here, we assume that users are allowed to read stale data and, hence, the effect of data inconsistency will have no impact on data availability. Also, due to replication, the overloading problem is not an issue, hence, the impact of system load on data availability is not considered.

2.1. Data Availability Analysis

If the preferred node in R is not available, then the read request will be forwarded and served at the next available node. This process repeats until it is finally served by one of the replicas, or finally none of the replica can serve the read request, by choosing different combinations of paths. The access path of such a read request can be modeled as a directed graph, and thus both the network topology and the location of the read requests originally issued will affect data availability. Therefore, to compute data availability, we need to consider requests originated from different nodes.

Let $S_A(u, G, R)$ denote the availability of d for a request r issued at node u with resident set R in G. If u hosts a replica, then d is available locally with probability $N_A(u)$. When data d is unavailable on u, a read request r needs to be forwarded to other nodes in R. Suppose that r is forwarded to v, where $v \in R$. The availability of data d on v to r can be computed as the product of $N_A(v)$ and the availability of path $\delta(u, v)$, denoted as $A_\delta(u, v)$. The availability of path $\delta(u, v)$ is computed as the product of the availabilities of all the edges in $\delta(u, v)$, i.e., $A_\delta(u, v) = \prod_{e(x,y)\in\delta(u,v)} A_e(x, y)$. As long as there is a node y such that $y \in R$ and the path $\delta(u, y)$ is available, the request can be served. If the availabilities of different paths are independent from each other, data d is unavailable for a request r issued in u only if all those nodes hosting replicas are unavailable. Therefore, the availability of a request issued from node u in G can be computed as the product of the node availability and path availability from node u to each node in resident set R, e.g.,

$$S_A(u, G, R) = 1 - \prod_{y\in R}(1 - A_\delta(u, y) \cdot N_A(y)) \quad (1)$$

2.2. Data Availability Modeling in a Tree Network

Consider a request originated from node u in the system with a tree graph $T = (V, E)$. Based on the user access protocols described, the tree network should be rooted at u, which is shown in Fig. 2. Consider a subtree T_w rooted at w, which is a subtree of T rooted at u (as shown in Fig.2). Data object d is not available for a request r issued by u if and only if d is not available at w or any of its child nodes. Let $S_{UA}(u, T_w, R)$ and $S_A(u, T_w, R)$ denote the unavailability and availability of data object d in T_w for request r. Let $ch(w)$ denote the set of children of w. Then,
$S_{UA}(u, T_w, R)=(1-N_A(w)) * \prod_{i\in ch(w)}(1 - A_e(w, i) \cdot N_A(i))$.

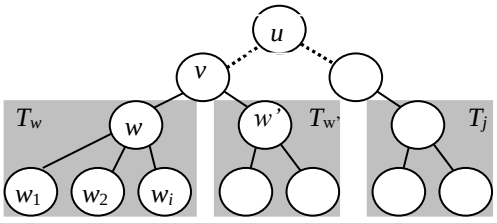


Fig.2. A sample tree graph rooted at u.

The unavailability of $T_{w'}$, …, and T_j, can be computed in the same way as T_w. The unavailability of these subtrees can be applied to the computation of the unavailability of the subtrees rooted at their parents. For example, the subtree rooted at v can be computed as $S_{UA}(u, T_v, R) = (1 - N_A(v)) * (1 - A_e(v, w)) * (1 - S_{UA}(u, T_w R))) * (1 - A_e(v, w')) * (1 - S_{UA}(u, T_{w'}, R)))$. For a request r issued in u, the unavailability of data d in the system can be computed recursively and, hence, $S_{UA}(u, T_u, R) = (1 - N_A(u)) * \prod_{v\in child(u)}(1 - (A_e(u, v) * (1 - S_{UA}(u, T_v, R))))$.

Subsequently, $S_A(u, T_u, R) = 1 - S_{UA}(u, T_u, R)$. The data availability for a request from another node x, e.g., $S_A(x, T_x, R)$ (note that the tree should now be rooted at x) can be computed in a similar way. Based on the analysis above, the system availability of d is computed as the average availability for requests in the entire system. Let $S_A(G, R)$ denote the system availability of data d. Then, we have,

$$S_A(G, R) = \sum_{x \in V} (A^r(x) \cdot S_A(x, T_x, R)) / \sum_{x \in V} A^r(x) \quad (2)$$

The algorithm $Tree\_S_A(T, R)$, which is developed to compute $S_A(T, R)$ in a tree graph $T = (V, E)$ is shown in Fig. 3. The algorithm works as what follows. It calls the recursive algorithm $T\_S_{UA}(u, T_u, R)$ for each node u in T to compute the unavailability for request issued at node u. The algorithm $T\_S_{UA}(u, T_u, R)$ recursively compute the data unavailability for the tree rooted at node u in T, by traversing each link and each vertex once. The computation uses the models we proposed earlier. Note that the children of node v are the nodes in the new tree rooted at node u (There are different implemetnations on building a new T_u in the original tree T, e.g., 1) physically re-build the tree rooted at T_u; 2) build a new virtual tree T_u by tracking the neighboring nodes for each node v in T, such that the unvisited neighboring nodes are child of v and the visited neighboring node is the parent node of v in new tree T_u.

Tree_ $S_A(T, R)$ {
 totalAvai = 0; $S_{UA}(u, R) = 0$; *totalRead* = 0;
 For $\forall\ u \in V$ {
  $S_{UA}(u, T_u, R) = T\_S_{UA}(u, T_u, R)$;
  $S_A(u, T_u, R) = 1 - S_{UA}(u, T_u, R)$;
  $totalAvai += A^r(u_x) * S_A(u, T_u, R)$;
  $totalRead += A^r(u)$;}
$S_A(T, R) = totalAvailable / totalRead$;}
$T\_S_{UA}(u, T_v, R)$ {
 ch(v) = {child nodes of v }; //all neighboring nodes
  if $(ch(v) == \phi)$ $S_{UA}(v, T_v, R) = 1 - N_A(v)$;
  else $S_{UA}(u, T_v, R) = (1 - N_A(v)) *$
   $\prod_{i \in ch(v)} 1 - (A_e(v, i) \cdot (1 - T\_S_{UA}(i, T_i, R)))$;
 return $S_{UA}(u, T_v, R)$; }

Fig. 3 . The algorithm $Tree\_S_A(T, R)$ in tree graph T .

In Theorem 1, we show that the availability d in tree T, $S_A(T, R)$, is monotonically increasing.

Theorem 1. Let R_1 and R_2 denote two resident sets of T and $R_1 \subseteq R_2$, then $S_A(T, R_1) \leq S_A(T, R_2)$.
Proof: Consider $R_1 = R_2$. According to the definition of $S_A(T, R_1)$ and $S_A(T, R_2)$, we know that $S_A(T, R_1) = S_A(T, R_2)$.

Consider $R_1 \subset R_2$. For each request in a node u, it has at least one more node that can provide a replica of data object d to access. Without loss of generality, assume that $|R_2| = |R_1| + 1$, and $R_2 = R_1 \cup \{v\}$. According to the availability model, the unavailability of v, $1 - N_A(v) < 1$. According to the availability computing model, for each request r issued at node u, which means $S_{UA}(u, T_v, R_2) < S_{UA}(u, T_v, R_1)$, or $S_A(u, T_v, R_2) > S_A(u, T_v, R_1)$. Thus, we have $S_A(T, R_1) < S_A(T, R_2)$. It follows that $S_A(T, R_1) \leq S_A(T, R_2)$ if $R_1 \subseteq R_2$.

We now show that the runtime complexity of our proposed availability computing algorithm in a tree T is in the order of the square of the number of nodes in T.

Theorem 2. The runtime complexity of algorithm $Tree\_S_A(T, R)$ is $O(|V|^2)$.

Proof. It suffices to prove the correctness of recursive relations.

Let v_j be a leaf node. If $v_j \notin \{r_j^1, \ldots, r_j^i\}$ or $q \le 0$, the value of $G(v_j,q,r_j^i)$ should not be counted since the situation is not included in the definition. Therefore we set $G(v_j,q,r_j^i)=-\infty$. On the contrary, if $v_j=r_j^s \in \{r_j^1, \ldots, r_j^i\}$ and $q \ge 1$, $G(v_j,q,r_j^i)$ equals to the optimal value of the sub-problem defined on the sub-tree T_j, which is the node v_j. Therefore $G(v_j,q,r_j^i) = A^r(v_j)\cdot A_j^s$.

If $r_j^i \in V_j$ or $q < 0$, the value of $F(v_j,q,r_j^i)$ should not be counted since the situation is not included in the definition. Therefore we set $F(v_j,q,r_j^i)=-\infty$. If $r_j^i \notin V_j$ and $q = 0$, the leaf node v_j has to fetch data from node r_j^i. Therefore $F(v_j,q,r_j^i)=A^r(v_j)\cdot A_j^i$. If $r_j^i \notin V_j$ and $q \ge 1$, the leaf node v_j can either fetch data from node r_j^i or fetch data from itself. Therefore $F(v_j,q,r_j^i)=\max\{F(v_j,0,r_j^i),G(v_j,1,v_j)\}$.

Let v_j be an internal node. If $V_j \cap \{r_j^1, \ldots, r_j^i\}=\varnothing$ or $q \le 0$ or $i = 0$, the value of $G(v_j,q,r_j^i)$ should not be counted since the situation is not included in the definition. Therefore we set $G(v_j,q,r_j^i)=-\infty$. If $r_j^i \notin V_j$, at least one node must be selected in $\{r_j^1, \ldots, r_j^i\} \cap V_j$ is equivalent to at least one node must be selected in $\{r_j^1, \ldots, r_j^{i-1}\} \cap V_j$. Therefore $G(v_j,q,r_j^i)=G(v_j,q,r_j^{i-1})$. The rest situations can be separated into 3 cases: $r_j^i=v_j$ and $q \ge 1$, $r_j^i \in V_{j1}$ and $q \ge 1$, and $r_j^i \in V_{j2}$ and $q \ge 1$. We discuss them as follows.

In case 1 we consider that $r_j^i=v_j$ and $q \ge 1$. Without loss of generality, we assume that r_j^i corresponds to r_{j1}^{i1} for v_{j1} (the left child node of v_j) and r_{j2}^{i2} for v_{j2} (the right child node of v_j).We claim that
$G(v_j,q,r_j^i)=\max\{G(v_j,q,r_j^{i-1}),A^r(v_j)\cdot A_j^i+\max\{F(v_{j1},q_1,r_{j1}^{i1})+F(v_{j2},q_2,r_{j2}^{i2})\mid 0\le q_1,q_2\le q-1,\ q_1+q_2=q-1\}\}$.

In order to prove that, we must show (1) the optimal solution appears either in $G(v_j,q,r_j^{i-1})$ or $A^r(v_j)\cdot A_j^i+\max\{F(v_{j1},q_1,r_{j1}^{i1})+F(v_{j2},q_2,r_{j2}^{i2}) \mid 0\le q_1,q_2\le q-1,\ q_1+q_2=q-1\}$, and (2) the optimal solution is the maximum value.

First, if a node in $\{r_j^1, \ldots, r_j^{i-1}\} \cap V_j$ is selected in the optimal solution, then the optimal solution appears in $G(v_j,q,r_j^{i-1})$. Otherwise we assume that no node in $\{r_j^1, \ldots, r_j^{i-1}\} \cap V_j$ is selected but r_j^i is selected in the optimal solution. Then it is better for v_j to fetch data from node $r_j^i=v_j$ than other nodes. Let q_1 nodes will be selected from T_{j1} and q_2 nodes will be selected from T_{j2} where $q_1+q_2=q-1$. Hence the optimal solution appears in $A^r(v_j)\cdot A_j^i+\max\{F(v_{j1},q_1,r_{j1}^{i1})+F(v_{j2},q_2,r_{j2}^{i2}) \mid 0\le q_1,q_2\le q-1,\ q_1+q_2=q-1\}$.

Second, $A^r(v_j)\cdot A_j^i+\max\{F(v_{j1},q_1,r_{j1}^{i1})+F(v_{j2},q_2,r_{j2}^{i2}) \mid 0\le q_1,q_2\le q-1,\ q_1+q_2=q-1\}$ may contain some inconsistent values. For example, if $F(v_{j1},q_1,r_{j1}^{i1})$ or $F(v_{j2},q_2,r_{j2}^{i2})$ selects a node in $\{r_j^1, \ldots, r_j^{i-1}\} \cap V_j$, v_j should fetch data from that node instead of r_j^i. However since the term $A^r(v_j)\cdot A_j^i$ is used, it actually counts that v_j fetches data from node r_j^i. Or $F(v_{j1},q_1,r_{j1}^{i1})$ selects a node such that v_{j2} should fetch data from that node outside T_{j2} instead of r_{j2}^{i2}. However the term $F(v_{j2},q_2,r_{j2}^{i2})$ indicates that actually it counts that v_{j2} fetches data from node r_{j2}^{i2} outside T_{j2}. Fortunately, the inconsistent values are smaller than the corresponding consistent values, and hence are smaller than the optimal solution. Therefore the inconsistent values will not affect the optimal solution to be the maximum value.

In case 2 we consider $r_j^i \in V_{j1}$ and $q \ge 1$. Without loss of generality, we assume that r_j^i corresponds to r_{j1}^{i1} for v_{j1} (the left child node of v_j) and r_{j2}^{i2} for v_{j2} (the right child node of v_j). We claim that
$G(v_j,q,r_j^i)=\max\{G(v_j,q,r_j^{i-1}),A^r(v_j)\cdot A_j^i+\max\{G(v_{j1},q_1,r_{j1}^{i1})+F(v_{j2},q_2,r_{j2}^{i2}) \mid 1\le q_1\le q,\ 0\le q_2\le q,\ q_1+q_2=q,$ the associated set of $G(v_{j1},q_1,r_{j1}^{i1})$ does not contain $r_j^1, \ldots, r_j^{i-1}$ but contains $r_j^i\}\}$.

First, if a node in $\{r_j^1, \ldots, r_j^{i-1}\} \cap V_j$ is selected in the optimal solution, then the optimal solution appears in $G(v_j,q,r_j^{i-1})$. Otherwise we assume that no node in $\{r_j^1, \ldots, r_j^{i-1}\} \cap V_j$ is selected but r_j^i

is selected in the optimal solution. We claim that v_j will not be selected. Because if v_j is selected, then it cannot be in $\{r_j^1, \ldots, r_j^{i-1}\} \cap V_j$. Thus it is better for v_j to fetch data from node r_j^i than v_j. Consequently it is better for all nodes to fetch data from node r_j^i than v_j. It implies that the selection of v_j will not gain any benefit. Therefore v_j will not be selected. So q_1 nodes will be selected from T_{j1} and q_2 nodes will be selected from T_{j2} where $q_1+q_2=q$. Hence the optimal solution appears in $A^r(v_j)\cdot A_j^i+\max\{G(v_{j1},q_1,r_{j1}^{i1})+F(v_{j2},q_2,r_{j2}^{i2}) \mid 1\le q_1\le q,\ 0\le q_2\le q,\ q_1+q_2=q,$ the associated set of $G(v_{j1},q_1,r_{j1}^{i1})$ does not contain $r_j^1, \ldots, r_j^{i-1}$ but contains $r_j^i\}$.

Second, if the associated set of $G(v_{j1},q_1,r_{j1}^{i1})$ does not contain $r_j^1, \ldots, r_j^i$, then that $G(v_{j1},q_1,r_{j1}^{i1})$ should not be counted. If the associated set of $G(v_{j1},q_1,r_{j1}^{i1})$ contains $r_j^1, \ldots, r_j^{i-1}$, then the inconsistent value is smaller than the corresponding consistent value, and hence smaller than the optimal value. so that $G(v_{j1},q_1,r_{j1}^{i1})$ can be omitted. Therefore it suffices to consider $G(v_{j1},q_1,r_{j1}^{i1})$ whose associated set does not contain $r_j^1, \ldots, r_j^{i-1}$ but contains r_j^i.

In case 3 we consider $r_j^i \in V_{j2}$ and $q \ge 1$. Without loss of generality, we assume that r_j^i corresponds to r_{j1}^{i1} for v_{j1} (the left child node of v_j) and r_{j2}^{i2} for v_{j2} (the right child node of v_j). Analogous to the proof of case 2, it can be proved that
$G(v_j,q,r_j^i)=\max\{G(v_j,q,r_j^{i-1}),A^r(v_j)\cdot A_j^i+\max\{F(v_{j1},q_1,r_{j1}^{i1})+G(v_{j2},q_2,r_{j2}^{i2}) \mid 0\le q_1\le q,\ 1\le q_2\le q,\ q_1+q_2=q,$ the associated set of $G(v_{j2},q_2,r_{j2}^{i2})$ does not contain $r_j^1, \ldots, r_j^{i-1}$ but contains $r_j^i\}\}$.

If $r_j^i \in V_j$ or $q < 0$, the value of $F(v_j,q,r_j^i)$ should not be counted since the situation is not included in the definition. Therefore we set $F(v_j,q,r_j^i)=-\infty$. If $r_j^i \notin V_j$ and $q \ge 0$, without loss of generality, we assume that r_j^i corresponds to r_{j1}^{i1} for v_{j1} (the left child node of v_j) and r_{j2}^{i2} for v_{j2} (the right child node of v_j). We claim that
$F(v_j,q,r_j^i)=\max\{G(v_j,q,r_j^i),A^r(v_j)\cdot A_j^i+\max\{F(v_{j1},q_1,r_{j1}^{i1})+F(v_{j2},q_2,r_{j2}^{i2}) \mid 0\le q_1\le q,\ 0\le q_2\le q,\ q_1+q_2=q\}\}$
First, if the optimal solution selects node in $\{r_j^1, \ldots, r_j^i\} \cap V_j$, then the optimal solution appears in $G(v_j,q,r_j^i)$.

Otherwise we assume that no nodes in $\{r_j^1, \ldots, r_j^i\} \cap V_j$ is selected in the optimal solution. Then it is better for v_j to fetch data from node r_j^i than other nodes. Additionally v_j will not be selected because of the same reason in the proof of case 2. So q_1 nodes will be selected from T_{j1} and q_2 nodes will be selected from T_{j2} where $q_1+q_2=q$. Hence the optimal solution appears in

$$A^r(v_j)\cdot A_j^i + \max\{F(v_{j1},q_1,r_{j1}^{i1})+F(v_{j2},q_2,r_{j2}^{i2}) \mid 0\le q_1\le q,\ 0\le q_2\le q,\ q_1+q_2=q\}.$$

Finally, the inconsistent values in $A^r(v_j)\cdot A_j^i+\max\{F(v_{j1},q_1,r_{j1}^{i1})+F(v_{j2},q_2,r_{j2}^{i2}) \mid 0\le q_1\le q,\ 0\le q_2\le q,\ q_1+q_2=q\}$ are smaller than the corresponding consistent values, and hence are smaller than the optimal solution. Therefore the inconsistent values will not affect the optimal solution to be the maximum value.

3. SYSTEM AVAILABILITY ENHANCEMENT

In this research, we proposed to develop mechanisms to optimize data availability in a system with a fixed number of replicas, K, i.e., to find a resident set with a size of K such that the overall data availability in a tree graph is optimal. It has been shown in our previous work in [17, 18] that reducing average distance of the paths (number of hops) from user's requests to service/data sites can result in high availability, as the shorter the distance (the smaller the number of hops), the higher the path availability from a requester to its closest replica. Therefore, a potential approach to improve the overall system availability is to allocate data replicas in the system such the average distance of the paths from user's requests to service sites is minimal.

3.1 Availability Enhancement Problem Modeling

The replica allocation problem to optimize data availability is much more complex than the traditional optimal resident set problems [19, 20, 21]. For traditional resident set problem, each request will be served by a single site (usually the closest site) and the corresponding communication cost will be computed as the final communication cost for such a request. Therefore, each request will be associated with a single data replica site only. However, with the availability model defined in this research, a request may need to find another available site if the current accessing site is not available. This process will repeat till such access is served by a replica site or is not served after all replica sites have been tried but none of them are available. Thus, each request should be associated with all sites within the resident set for data availability computing. To find the resident set with maximal data availability in a tree network, the computation complexity could be too high to be feasible. For example, an intuitive approach is to try every combination of the resident set in the tree and the runtime complexity would be $|V|^{|R|}$. Thus, an approximate solution is to apply the property observed in [17, 18] that the smaller the number of hops from a request to the resident set, the higher the data availability.

First, since a request will be forwarded to a secondary replica only the primary replica node cannot serve such request, and such an event has a very low probability (node availability is usually very high), we can assume that a request is only associated with the primary replica node to reduce the complexity (note that this assumption is only valid for replica allocation purpose, a request is still associated with all replicas when computing availability). Thus, our problem can be reduced to a resident set problem **maximizing** the total of the product of the number of reading requests and the path availability of each request to the resident set. i.e., $\sum_{v \varepsilon V}(A^{\gamma}(v)*(A_{\delta}(v,R)*N_A(v)))$, where $A_{\delta}(v, R)$ is the path availability from a request issued at node v to a replica site in the resident set R.

Second, we all understand that the heterogeneity of the node (replica site) availability can have a big impact on the overall system availability. Here, we will show that the heterogeneity of the edge availability can have big impact on the overall system availability. In a system with line topology shown Fig. 4, the allocation of two replicas to optimize the availability without considering the heterogeneity of edge availability is shown in allocation scheme *A*, and the allocation considering the heterogeneity of edge availability is shown in allocation scheme *B* (with homogeneous node availability of 0.999). In Scheme *B*, the service to requests at node v_1 is available locally with availability of 0.999, and requests need to be forwarded to node v_3 only if node v_1 is not available (with a probability of 0.001). Requests at other nodes can be served by the replica at node v_3 with very high probability, and only need to be forwarded to node v_1 if node v_3 is not available or edge $e(v_2, v_3)$ is not available (for requests from node v_2). The overall normalized data availability in system with replica allocation scheme *B* is around 0.96 by applying the availability computing algorithm described in Fig. 3. In Scheme *A*, the service to requests at node v_1 is not available locally and thus need to be forwarded to node v_2 or v_3 where the availability of edge $e(v_1, v_2)$ is 0.5. The overall normalized data availability in system with replica allocation scheme *A* will definitely be lower than 0.90.

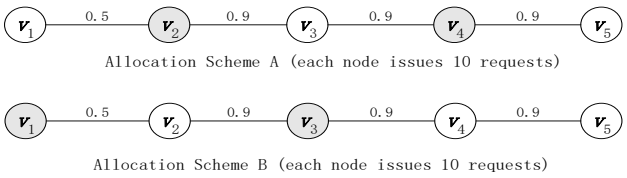


Fig. 4. The impact of the edge availability on system availability

Based on the above analysis, the availability optimization problem considering the heterogeneity of network link availability is desired to better improve the overall data availability in the system. In a system $T(V, E)$, $\exists e(v_i, v_j), e(v_k, v_m) \in E, A_e(v_i, v_j) \neq A_e(v_k, v_m)$. The avaialiblity of such system $T(V, E)$ can be computed as $(\sum_{v \in V}(A^r(v) * \prod_{e(x,y) \in \delta(v,R)} A_e(x,y) * N_A(u))) / \sum_{v \in V} A^r(v)$, where $u \in R$ and $u \in \delta(v, R)$. Hence, the replica allocation problem to maximize the availability considering heterogeneity of both node availability and network link availability can be defined as the following replica allocation problem (Problem 1).

Problem 1*: Given a tree network $T(V, E)$,* where $V = \{v_1, v_2, \ldots, v_n\}$, E is the set of all edges in T. $\exists v_i, v_j \in V, N_A(v_i) \neq N_A(v_j)$, and $\exists e(v_i, v_j), e(v_k, v_m) \in E, A_e(v_i, v_j) \neq A_e(v_k, v_m)$. *Find a subset of nodes* $R \subseteq V$ and $|R| = K$ that maximize the value of $\sum_{v \in V}(A^r(v) * \prod_{e(x,y) \in \delta(v,R)} A_e(x,y) * N_A(u))$, where $u \in \delta(v, R) \wedge u \in R$.

In our previous work in [18], solutions have been given to availability optimization problem, by considering the homogeneous of node availability or network link availability, or both. However, these solutions are only partial solutions to Problem 1.

3.2 The Availability Enhancement Algorithm

Problem 1 is an optimization problem similar to the *p*-median problem [22, 15]. In a *p*-median problem, it is given a graph $G(V, E)$ where $V = \{v_1, v_2, \ldots, v_n\}$. Each node v_i is associated with a nonnegative weight c_i, and each edge $e(v_i, v_j)$ is associated with a nonnegative weight $d(e(v_i, v_j))$. Additionally, it is given a real monotonically increasing function f_i for each node v_i. The goal is to find a subset $R \subseteq V$ containing at most p nodes (p is an arbitrary but pre-determined number) to minimize the objective

$$\sum_{v_j \in R} c_j + \sum_{v_i \in V} \min_{v_j \in R} f_i(d(e(v_i, v_j))) \tag{3}$$

Problem 1 and *p*-median problem have some common issues. They both consider weighted tree networks where each node/edge is assigned value(s). They both look for optimal solutions to the objective functions. Problem 1 looks for the maximum value of an objective function while *p*-median problem looks for the minimum value of another objective function. It is therefore possible for us to borrow some ideas of the solution to the *p*-median problem to develop the solution to Problem 1.

However, it is difficult to directly apply the solution to the *p*-median problem to Problem 1 since they are different. Consider the *p*-median problem. Suppose that a node v_j is selected and the edges $e(v_i, v_{i1})$, $e(v_{i1}, v_{i2})$, …, $e(v_{ik}, v_j)$ form a path connecting nodes v_i and v_j. Then the computation of the cost for v_i to access v_j includes k+1 piece of information: $f_i(d(e(v_i,v_{i1})))$, $f_{i1}(d(e(v_{i1},v_{i2})))$, …, $f_{ik}(d(e(v_{ik},v_j)))$, and c_j. While for Problem 1, the computation of the contribution for v_i to fetch data from v_j includes k+2 piece of information: $A^r(v_i)$, $A_e(v_i,v_{i1})$, $A_e(v_{i1},v_{i2})$, …, $A_e(v_{ik},v_j)$, and $N_A(v_j)$. Therefore the solution to Problem 1 should include and handle more information than that to *p*-median problem. Thus we need to modify the solution to *p*-median problem to re-develop the solution to Problem 1.

<u>Problem Transformation and Re-definition</u>

A three step solution is provided to Problem 1:

1. Transformation: reduce Problem 1 from an arbitrary tree network to a binary tree network

2. Pre-processing: compute A_j^i and r_j^i for each node v_j, $1 \le i, j \le n$
3. Computing function values: compute the values of two functions $G(v_j,q,r_j^i)$ and $F(v_j,q,r_j^i)$ based on their recursive relations.

Step 1. Transformation

Given an arbitrary tree network T(V, E), where $V = \{v_1, v_2, \ldots, v_n\}$, E is the set of all edges in T. It can be transformed to that of *a binary tree network T'(V',E').* We show that by considering the following two cases, Case 1 and Case 2.

Case1. Suppose that an internal node v_j has exactly one child node (Fig. 5(a)). Without loss of generality, we assume that v_j has a left child node v_{j1}. Then we introduce a right child node v_{j2} and set the availabilities $N_A(v_{j2}) = A_e(v_j, v_{j2}) = 0$ and the number of read requests $A^r(v_{j2}) = 0$ (Fig. 5(b)).

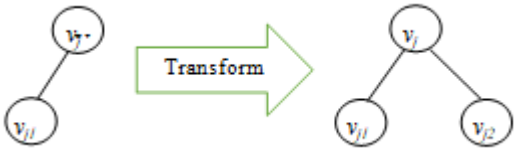


Fig. 5 Node with a single child node (a) is transformed to (b)

Since $N_A(v_{j2}) = A_e(v_j, v_{j2}) = A^r(v_{j2}) = 0$, there is no benefit to select v_{j2} and add v_{j2} to the resident set R. Therefore if R' is an optimal solution to Problem 1 for $T'(V',E')$, then $R = R' - \{v_{j2} \mid j\}$ is an optimal solution to Problem 1 for $T(V,E)$.

Case 2. Suppose that an internal node v_j has more than 3 childe nodes v_{j1}, v_{j2}, ..., v_{jt} where $t \ge 3$ (Fig 6(a)). Then we introduct t-2 nodes u_{js} where $2 \le s \le t - 1$, replace edge e(v_j, v_{js}) by edge $e(u_{js}, v_{js})$ for $2 \le s \le t - 1$, replace edge $e(v_j, v_{jt})$ by edge $e(u_{jt-1}, v_{jt})$, and add edges $e(v_j, u_{j2})$ and $e(u_{js'}, u_{js'+1})$ for $2 \le s' \le t - 2$, and set the availabilities $N_A(u_{js}) = 0$ for $2 \le s \le t - 1$, $A_e(u_{js}, v_{js}) = A_e(v_j, v_{js})$ for $2 \le s \le t - 1$, $A_e(u_{jt-1}, v_{jt}) = A_e(v_j, v_{jt})$, $A_e(v_j, u_{j2}) = A_e(u_{js'}, u_{js'+1}) = 1$ for $2 \le s' \le t - 2$, and the number of read requests $A^r(u_{js}) = 0$ for $2 \le s \le t - 1$ (Fig 6 (b)).

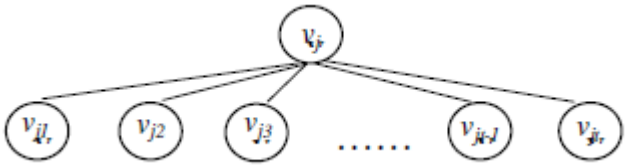


Fig. 6 (a). Case 2, the node with over 3 child nodes.

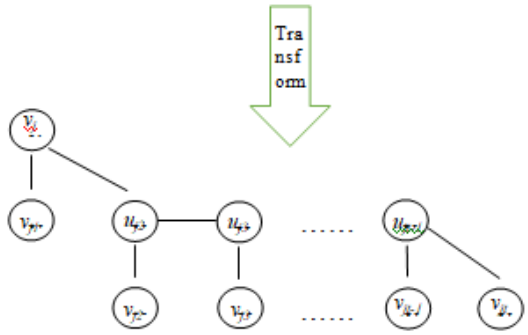


Fig. 6 (b). Transformed graph for the graph shown in Fig. 6 (a).

First, it can be verified that the success availability for node v_j (v_{js}) to fetch data from node v_{js} (v_j) does not change, $2 \le s \le t-1$. Second, the new assigned availabilities and number of requests guarantee that there is no benefit to select nodes u_{js}, $2 \le s \le t-1$. Therefore if R' is an optimal solution to Problem 1 for $T'(V',E')$, then $R = R' - \{u_{js} \mid j, s\}$ is an optimal solution to Problem 1 for $T(V,E)$.

Based on the previous discussion, Problem 1 can be efficiently transformed from an arbitrary tree network to a binary tree network. The solution to the transformed problem can be easily transformed back to the original problem.

Step 2. Pre-processing:

In this step we compute A_j^i and the corresponding r_j^i for each node v_j, $1 \le i, j \le n$. A_j^i and r_j^i are defined in the following definition.

Definition 1: A_j^1 is the greatest success probability for node v_j to fetch data from all nodes, …, A_j^n is the smallest success probability for node v_j to fetch data from all nodes. r_j^1 is the node corresponds to A_j^1, …, r_j^n is the node corresponds to A_j^n.

Given a binary tree network, we compute the success probability p_{ij} for node v_i to fetch data from node v_j. Specifically, if the edges $e(v_i, v_{i1})$, $e(v_{i1}, v_{i2})$, …, $e(v_{ik}, v_j)$ form the path connecting v_i and v_j, then the probability $p_{ij} = A_e(v_i,v_{i1}) \cdot A_e(v_{i1},v_{i2}) \ldots A_e(v_{ik},v_j) \cdot N_A(v_j)$. Then we sort p_{1j}, …, p_{nj} from the smallest to the largest, and let the results be A_j^1, …, A_j^n. Additionally, let node r_j^i corresponds to A_j^i, $1 \le i \le n$. The equality of success probabilities can be resolved by arbitrary set the order of nodes.

Step 3. Computing recursive function values

Let v_j be a node, q be an integer, and r_j^i and A_j^i be the values computed in the pre-processing step. The functions $G(v_j,q,r_j^i)$ and $F(v_j,q,r_j^i)$ are defined in the following definition.

Definition 2: $G(v_j,q,r_j^i)$ is defined to be the optimal value of the sub-problem defined on the sub-tree T_j given that total of at least 1 and at most q nodes can be selected in T_j, and at least one node must be selected in $\{r_j^1, \ldots, r_j^i\} \cap V_j$. The set of nodes for $G(v_j,q,r_j^i)$ to achieve the optimal value is called the associated set of $G(v_j,q,r_j^i)$.

Definition 3: $F(v_j,q,r_j^i)$ is defined to be the optimal value of the sub-problem defined on the sub-tree T_j under the constraints that a total of at most q nodes can be selected in T_j and there are

already some selected nodes in $V - V_j$, and the best amongst them that v_j can fetch data is r_j^i. The set of nodes for $F(v_j,q,r_j^i)$ to achieve the optimal value is called the associated set of $F(v_j,q,r_j^i)$.

We establish the recursive relations between $G(v_j,q,r_j^i)$ and $F(v_j,q,r_j^i)$ as what follows.

Leaf node v_i

For function $G(v_j,q,r_j^i)$,

$G(v_j,q,r_j^i)=-\infty$ if $v_j \notin \{r_j^1, \ldots, r_j^i\}$ or $q \le 0$

$G(v_j,q,r_j^i)=A^r(v_j)\cdot A_j^s$ if $v_j=r_j^s \in \{r_j^1, \ldots, r_j^i\}$ and $q \ge 1$

For function $F(v_j,q,r_j^i)$,

$F(v_j,q,r_j^i)=-\infty$ if $r_j^i \in V_j$ or $q < 0$

$F(v_j,q,r_j^i)=A^r(v_j)\cdot A_j^i$ if $r_j^i \notin V_j$ and $q = 0$

$F(v_j,q,r_j^i)=\max\{F(v_j,0,r_j^i),G(v_j,1,v_j)\}$ if $r_j^i \notin V_j$ and $q \ge 1$

Internal node v_i

For function $G(v_j,q,r_j^i)$,

$G(v_j,q,r_j^i)=-\infty$ if $V_j \cap \{r_j^1, \ldots, r_j^i\}=\varnothing$ or $q \le 0$ or $i = 0$

$G(v_j,q,r_j^i)=G(v_j,q,r_j^{i-1})$ if $r_j^i \notin V_j$

$G(v_j,q,r_j^i) = \max\{G(v_j,q,r_j^{i-1}),A^r(v_j)\cdot A_j^i + \max\{F(v_{j1},q_1,r_{j1}^{i1})+F(v_{j2},q_2,r_{j2}^{i2}) \mid 0\le q_1,q_2\le q-1, q_1+q_2=q-1\}\}$

if $r_j^i=v_j$ and $q \ge 1$, r_j^i corresponds to r_{j1}^{i1} for v_{j1} and r_{j2}^{i2} for v_{j2}

$G(v_j,q,r_j^i)=\max\{G(v_j,q,r_j^{i-1}),A^r(v_j)\cdot A_j^i+\max\{G(v_{j1},q_1,r_{j1}^{i1})+F(v_{j2},q_2,r_{j2}^{i2}) \mid 1\le q_1\le q, 0\le q_2\le q, q_1+q_2=q,$ the associated set of $G(v_{j1},q_1,r_{j1}^{i1})$ does not contain $r_j^1, \ldots, r_j^{i-1}$ but contains $r_j^i\}\}$

if $r_j^i \in V_{j1}$ and $q \ge 1$, r_j^i corresponds to r_{j1}^{i1} for v_{j1} and r_{j2}^{i2} for v_{j2}

$G(v_j,q,r_j^i)=\max\{G(v_j,q,r_j^{i-1}),A^r(v_j)\cdot A_j^i+\max\{F(v_{j1},q_1,r_{j1}^{i1})+G(v_{j2},q_2,r_{j2}^{i2}) \mid 0\le q_1\le q, 1\le q_2\le q, q_1+q_2=q,$ the associated set of $G(v_{j2},q_2,r_{j2}^{i2})$ does not contain $r_j^1, \ldots, r_j^{i-1}$ but contains $r_j^i\}\}$

if $r_j^i \in V_{j2}$ and $q \ge 1$, r_j^i corresponds to r_{j1}^{i1} for v_{j1} and r_{j2}^{i2} for v_{j2}

For function $F(v_j,q,r_j^i)$,

$F(v_j,q,r_j^i)=-\infty$ if $r_j^i \in V_j$ or $q < 0$

$F(v_j,q,r_j^i)=\max\{G(v_j,q,r_j^i),A^r(v_j)\cdot A_j^i+\max\{F(v_{j1},q_1,r_{j1}^{i1})+F(v_{j2},q_2,r_{j2}^{i2}) \mid 0\le q_1\le q, 0\le q_2\le q, q_1+q_2=q\}\}$

if $r_j^i \notin V_j$ and $q \ge 0$, r_j^i corresponds to r_{j1}^{i1} for v_{j1} and r_{j2}^{i2} for v_{j2}

The Algorithm for the Transformed Problem

We integrate the three steps described in previous subsections to develop the algorithm, *Alg_Enhan*, to solve the transformed Problem 1.

1. transforms the problem to a binary tree network if necessary,
2. pre-processes the binary tree network and compute A_j^i and r_j^i for each node v_j, $1 \le i, j \le n$,
3. computes the values of $G(v_j,q,r_j^i)$ and $F(v_j,q,r_j^i)$ based on the recursive relations following the post order of the nodes: first compute the function values for the leaf nodes, then the internal nodes, and finally the root node. The associated sets of $G(v_j,q,r_j^i)$ and $F(v_j,q,r_j^i)$ are recorded at the same time. If two sets of nodes both achieve the optimal value, then the set containing the node in V_j from which it is better for v_j to fetch data will be recorded, and
4. Returns $G(v_1,K,r_1^n)$ together with the associated set, where v_1 is the root node.

3.3 A Numerical Example for the Availability Enhancement Algorithm (Alg_Enhan)

We now use a numerical example to demonstrate the algorithm we developed for Problem 1. Consider the binary tree network $T(V, E)$ as shown in fig. 7, where $V = \{v_1, v_2, v_3, v_4, v_5\}$ and $E = \{e(v_1, v_2), e(v_1, v_3), e(v_2, v_4), e(v_2, v_5)\}$. The nodes availabilities $N_A(v_1) = 0.3$, $N_A(v_2) = 0.1$, $N_A(v_3) = 0.6$, $N_A(v_4) = 0.7$, $N_A(v_5) = 0.9$. The edges availabilities $A_e(v_1, v_2) = 0.5$, $A_e(v_1, v_3) = 0.4$, $A_e(v_2, v_4) = 0.2$, $A_e(v_2, v_5) = 0.8$. The numbers of read requests $A^r(v_1) = 100$, $A^r(v_2) = 200$, $A^r(v_3) = 300$, $A^r(v_4) = 400$, $A^r(v_5) = 500$. The tree network is illustrated in the following figure. We further assume that $K = 2$.

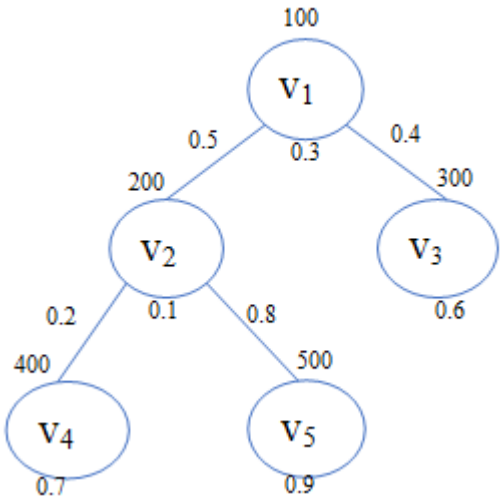


Fig. 7. A sample network for algorithm numerical demonstration.

Pre-processing:

Since it is a binary tree network, transformation is not needed. We compute A_j^i and r_j^i in this subsection.

Root node v_1

The success probability to fetch data from $v_1 = 0.3$.
The success probability to fetch data from $v_2 = 0.5 \cdot 0.1 = 0.05$.
The success probability to fetch data from $v_3 = 0.4 \cdot 0.6 = 0.24$.
The success probability to fetch data from $v_4 = 0.5 \cdot 0.2 \cdot 0.7 = 0.07$.
The success probability to fetch data from $v_5 = 0.5 \cdot 0.8 \cdot 0.9 = 0.36$.
We sort the success probabilities and record A_1^i and the corresponding r_1^i in the following table, $1 \le i \le 5$.

Table 1. Pre-processing the data access for root node v_1

$r_1^1=v_5$	$r_1^2=v_1$	$r_1^3=v_3$	$r_1^4=v_4$	$r_1^5=v_2$
$A_1^1=0.36$	$A_1^2=0.3$	$A_1^3=0.24$	$A_1^4=0.07$	$A_1^5=0.05$

Internal node v_2

The success probability to fetch data from $v_1 = 0.5 \cdot 0.3 = 0.15$.
The success probability to fetch data from $v_2 = 0.1$.

The success probability to fetch data from v_3 = 0.5·0.4·0.6 = 0.12.
The success probability to fetch data from v_4 = 0.2·0.7 = 0.14.
The success probability to fetch data from v_5 = 0.8·0.9 = 0.72.
We sort the success probabilities and record A_2^i and the corresponding r_2^i in the following table, $1 \le i \le 5$.

Table 2. Pre-processing the data access for internal node v_2

$r_2^1=v_5$	$r_2^2=v_1$	$r_2^3=v_4$	$r_2^4=v_3$	$r_2^5=v_2$
$A_2^1=0.72$	$A_2^2=0.15$	$A_2^3=0.14$	$A_2^4=0.12$	$A_2^5=0.1$

Leaf node v_3

The success probability to fetch data from v_1 = 0.4·0.3 = 0.12.
The success probability to fetch data from v_2 = 0.4·0.5·0.1 = 0.02.
The success probability to fetch data from v_3 = 0.6.
The success probability to fetch data from v_4 = 0.4·0.5·0.2·0.7 = 0.028.
The success probability to fetch data from v_5 = 0.4·0.5·0.8·0.9 = 0.144.
We sort the success probabilities and record A_3^i and the corresponding r_3^i in the following table, $1 \le i \le 5$.

Table 3.Pre-processing the data access for leaf node v_3

$r_3^1=v_3$	$r_3^2=v_5$	$r_3^3=v_1$	$r_3^4=v_4$	$r_3^5=v_2$
$A_3^1=0.6$	$A_3^2=0.144$	$A_3^3=0.12$	$A_3^4=0.028$	$A_3^5=0.02$

Leaf node v_4

The success probability to fetch data from v_1 = 0.2·0.5·0.3 = 0.03.
The success probability to fetch data from v_2 = 0.2·0.1 = 0.02.
The success probability to fetch data from v_3 = 0.2·0.5·0.4·0.6 = 0.024.
The success probability to fetch data from v_4 = 0.7.
The success probability to fetch data from v_5 = 0.2·0.8·0.9 = 0.144.
We sort the success probabilities and record A_4^i and the corresponding r_4^i in the following table, $1 \le i \le 5$.

Table 4.Pre-processing the data access for leaf node v_4

$r_2^1=v_5$	$r_2^2=v_1$	$r_2^3=v_4$	$r_2^4=v_3$	$r_2^5=v_2$
$A_2^1=0.72$	$A_2^2=0.15$	$A_2^3=0.14$	$A_2^4=0.12$	$A_2^5=0.1$

Leaf node v_5

The success probability to fetch data from v_1 = 0.8·0.5·0.3 = 0.12.
The success probability to fetch data from v_2 = 0.8·0.1 = 0.08.
The success probability to fetch data from v_3 = 0.8·0.5·0.4·0.6 = 0.096.
The success probability to fetch data from v_4 = 0.8·0.2·0.7 = 0.112.
The success probability to fetch data from v_5 = 0.9.
We sort the success probabilities and record A_5^i and the corresponding r_5^i in the following table, $1 \le i \le 5$.

Table 5. Pre-processing the data access for leaf node v_5

$r_5^1=v_5$	$r_5^2=v_1$	$r_5^3=v_4$	$r_5^4=v_3$	$r_5^5=v_2$
$A_5^1=0.9$	$A_5^2=0.12$	$A_5^3=0.112$	$A_5^4=0.096$	$A_5^5=0.08$

Recursive function computing:

We now compute the function values based on the pre-processing results and recursive relations developed above. We follow the post order of the nodes, which is $v_4 \rightarrow v_5 \rightarrow v_2 \rightarrow v_3 \rightarrow v_1$.

Leaf node v_4

The G and F function values for node v_4 is summarized in the Table 6. The associated set of nodes is also recorded.

Table 6. G and F function values for node v_4

$G(v_4,1,r_4^1)=280;$ v_4	$G(v_4,2,r_4^1)=280;$ v_4	$F(v_4,0,r_4^1)=-\infty$	$F(v_4,1,r_4^1)=-\infty$	$F(v_4,2,r_4^1)=-\infty$
$G(v_4,1,r_4^2)=280;$ v_4	$G(v_4,2,r_4^2)=280;$ v_4	$F(v_4,0,r_4^2)=57.6$	$F(v_4,1,r_4^2)=280;$ v_4	$F(v_4,2,r_4^2)=280;$ v_4
$G(v_4,1,r_4^3)=280;$ v_4	$G(v_4,2,r_4^3)=280;$ v_4	$F(v_4,0,r_4^3)=12$	$F(v_4,1,r_4^3)=280;$ v_4	$F(v_4,2,r_4^3)=280;$ v_4
$G(v_4,1,r_4^4)=280;$ v_4	$G(v_4,2,r_4^4)=280;$ v_4	$F(v_4,0,r_4^4)=9.6$	$F(v_4,1,r_4^4)=280;$ v_4	$F(v_4,2,r_4^4)=280;$ v_4
$G(v_4,1,r_4^5)=280;$ v_4	$G(v_4,2,r_4^5)=280;$ v_4	$F(v_4,0,r_4^5)=8$	$F(v_4,1,r_4^5)=280;$ v_4	$F(v_4,2,r_4^5)=280;$ v_4

Leaf node v_5

The G and F function values for node v_5 is summarized in the Table 7. The associated set of nodes is also recorded.

Table 7.G and F function values for node v_5

$G(v_5,1,r_5^1)=450;$ v_5	$G(v_5,2,r_5^1)=450;$ v_5	$F(v_5,0,r_5^1)=-\infty$	$F(v_5,1,r_5^1)=-\infty$	$F(v_5,2,r_5^1)=-\infty$
$G(v_5,1,r_5^2)=450;$ v_5	$G(v_5,2,r_5^2)=450;$ v_5	$F(v_5,0,r_5^2)=60$	$F(v_5,1,r_5^2)=450;$ v_5	$F(v_5,2,r_5^2)=450;$ v_5
$G(v_5,1,r_5^3)=450;$ v_5	$G(v_5,2,r_5^3)=450;$ v_5	$F(v_5,0,r_5^3)=56$	$F(v_5,1,r_5^3)=450;$ v_5	$F(v_5,2,r_5^3)=450;$ v_5
$G(v_5,1,r_5^4)=450;$ v_5	$G(v_5,2,r_5^4)=450;$ v_5	$F(v_5,0,r_5^4)=48$	$F(v_5,1,r_5^4)=450;$ v_5	$F(v_5,2,r_5^4)=450;$ v_5
$G(v_5,1,r_5^5)=450;$ v_5	$G(v_5,2,r_5^5)=450;$ v_5	$F(v_5,0,r_5^5)=40$	$F(v_5,1,r_5^5)=450;$ v_5	$F(v_5,2,r_5^5)=450;$ v_5

Internal node v_2

The G and F function values for node v_2 is summarized in the Table 8. The associated set of nodes is also recorded.

Table 8. G and F function values for node v_2

$G(v_2,1,r_2^1)=651.6$; v_5	$G(v_2,2,r_2^1)=874$; v_4,v_5	$F(v_2,0,r_2^1)=-\infty$	$F(v_2,1,r_2^1)=-\infty$	$F(v_2,2,r_2^1)=-\infty$
$G(v_2,1,r_2^2)=651.6$; v_5	$G(v_2,2,r_2^2)=874$; v_4,v_5	$F(v_2,0,r_2^2)=102$	$F(v_2,1,r_2^2)=651.6$; v_5	$F(v_2,2,r_2^2)=874$; v_4,v_5
$G(v_2,1,r_2^3)=651.6$; v_5	$G(v_2,2,r_2^3)=874$; v_4,v_5	$F(v_2,0,r_2^3)=-\infty$	$F(v_2,1,r_2^3)=-\infty$	$F(v_2,2,r_2^3)=-\infty$
$G(v_2,1,r_2^4)=651.6$; v_5	$G(v_2,2,r_2^4)=874$; v_4,v_5	$F(v_2,0,r_2^4)=81.6$	$F(v_2,1,r_2^4)=651.6$; v_5	$F(v_2,2,r_2^4)=874$; v_4,v_5
$G(v_2,1,r_2^5)=651.6$; v_5	$G(v_2,2,r_2^5)=874$; v_4,v_5	$F(v_2,0,r_2^5)=-\infty$	$F(v_2,1,r_2^5)=-\infty$	$F(v_2,2,r_2^5)=-\infty$

Leaf node v_3

The G and F function values for node v_3 is summarized in the Table 9. The associated set of nodes is also recorded.

Table 9. G and F function values for leaf node v_3

$G(v_3,1,r_3^1)=180$; v_3	$G(v_3,2,r_3^1)=180$; v_3	$F(v_3,0,r_3^1)=-\infty$	$F(v_3,1,r_3^1)=-\infty$	$F(v_3,2,r_3^1)=-\infty$
$G(v_3,1,r_3^2)=180$; v_3	$G(v_3,2,r_3^2)=180$; v_3	$F(v_3,0,r_3^2)=43.2$	$F(v_3,1,r_3^2)=180$; v_3	$F(v_3,2,r_3^2)=180$; v_3
$G(v_3,1,r_3^3)=180$; v_3	$G(v_3,2,r_3^3)=180$; v_3	$F(v_3,0,r_3^3)=36$	$F(v_3,1,r_3^3)=180$; v_3	$F(v_3,2,r_3^3)=180$; v_3
$G(v_3,1,r_3^4)=180$; v_3	$G(v_3,2,r_3^4)=180$; v_3	$F(v_3,0,r_3^4)=8.4$	$F(v_3,1,r_3^4)=180$; v_3	$F(v_3,2,r_3^4)=180$; v_3
$G(v_3,1,r_3^5)=180$; v_3	$G(v_3,2,r_3^5)=180$; v_3	$F(v_3,0,r_3^5)=6$	$F(v_3,1,r_3^5)=180$; v_3	$F(v_3,2,r_3^5)=180$; v_3

Root node v_1

The G and F function values for node v_1 is summarized in the Table 10. The associated set of nodes is also recorded.

Table 10.G and F function values for root node v_1

$G(v_1,1,r_1^1)=730.8$; v_5	$G(v_1,2,r_1^1)=953.2$; v_4,v_5	$F(v_1,0,r_1^1)=-\infty$	$F(v_1,1,r_1^1)=-\infty$	$F(v_1,2,r_1^1)=-\infty$
$G(v_1,1,r_1^2)=730.8$; v_5	$G(v_1,2,r_1^2)=953.2$; v_4,v_5	$F(v_1,0,r_1^2)=-\infty$	$F(v_1,1,r_1^2)=-\infty$	$F(v_1,2,r_1^2)=-\infty$
$G(v_1,1,r_1^3)=730.8$; v_5	$G(v_1,2,r_1^3)=953.2$; v_4,v_5	$F(v_1,0,r_1^3)=-\infty$	$F(v_1,1,r_1^3)=-\infty$	$F(v_1,2,r_1^3)=-\infty$
$G(v_1,1,r_1^4)=730.8$; v_5	$G(v_1,2,r_1^4)=953.2$; v_4,v_5	$F(v_1,0,r_1^4)=-\infty$	$F(v_1,1,r_1^4)=-\infty$	$F(v_1,2,r_1^4)=-\infty$
$G(v_1,1,r_1^5)=730.8$; v_5	$G(v_1,2,r_1^5)=953.2$; v_4,v_5	$F(v_1,0,r_1^5)=-\infty$	$F(v_1,1,r_1^5)=-\infty$	$F(v_1,2,r_1^5)=-\infty$

According to the entry ($G(v_1,2,r_1^5)$ = 953.2; v_4,v_5) in Table 10 function values for root node v_1, when selecting nodes v_4 and v_5, the objective function in Problem 2 achieves the maximum value 953.2.

3.4 Complexity and Correctness Analysis for the Availability Enhancement Algorithm (Alg_Enhan)

Complexity Analysis for the Algorithm (*Alg_Enhan*)

We now analyze the runtime complexity for the availability enhancement algorithm (*Alg_Enhan*). For simplicity, we assume that the computational complexity for arithmetic operations (+, –, ·, /) and comparison is $O(1)$.

Theorem 1: The computational complexity of the algorithm is $O(Kn^2)$.

Proof. In step 1 (transformation), $O(n)$ nodes and edges may be added to the original tree network. The corresponding parameters (e.g. probabilities and number of read requests) may also be added. So the computational complexity of step 1 is $O(n)$.

In step 2 (pre-processing), the success probabilities for each node to fetch data from all nodes are computed and sorted. It is possible that the computation of a success probability costs $O(n)$ multiplications (consider unbalanced binary tree). So the computational complexity of step 2 is $O(n)\cdot(O(n)\cdot O(n)+O(n\log n)) = O(n^3)$. However, the computational complexity can be improved to $O(n^2)$ based on the similar technique used in [25, 15].

In step 3 (computing function values), the values of two functions are computed based on the recursive relations. $O(n)$ tables are computed for $O(n)$ nodes. Each table contains $O(n)\cdot O(K) = O(Kn)$ entries. In the worst case, the computation of an entry requires $O(K)$ operations. So the computational complexity of step 3 is $O(n)\cdot O(Kn)\cdot O(K) = O(K^2n^2)$. However, the computational complexity can be improved to $O(Kn^2)$ based on the similar technique used in [26, 15].

Therefore the computational complexity of the algorithm is $O(n) + O(n^2) + O(Kn^2) = O(Kn^2)$ □

Correctness Analysis for the Algorithm(*Alg_Enhan*)

We now show the correctness in Theorem 2 for the availability enhancement algorithm (*Alg_Enhan*).

Theorem 2: The algorithm *Alg_Enhan* outputs the optimal solution of Problem 1.

Proof. It suffices to prove the correctness of recursive relations.

Let v_j be a leaf node. If $v_j \notin \{r_j^1, \ldots, r_j^i\}$ or $q \le 0$, the value of $G(v_j,q,r_j^i)$ should not be counted since the situation is not included in the definition. Therefore we set $G(v_j,q,r_j^i)=-\infty$. On the contrary, if $v_j=r_j^s \in \{r_j^1, \ldots, r_j^i\}$ and $q \ge 1$, $G(v_j,q,r_j^i)$ equals to the optimal value of the sub-problem defined on the sub-tree T_j, which is the node v_j. Therefore $G(v_j,q,r_j^i) = A^r(v_j)\cdot A_j^s$.

If $r_j^i \in V_j$ or $q < 0$, the value of $F(v_j,q,r_j^i)$ should not be counted since the situation is not included in the definition. Therefore we set $F(v_j,q,r_j^i)=-\infty$. If $r_j^i \notin V_j$ and $q = 0$, the leaf node v_j has to fetch data from node r_j^i. Therefore $F(v_j,q,r_j^i)=A^r(v_j)\cdot A_j^i$. If $r_j^i \notin V_j$ and $q \ge 1$, the leaf node v_j can either fetch data from node r_j^i or fetch data from itself. Therefore $F(v_j,q,r_j^i)=\max\{F(v_j,0,r_j^i),G(v_j,1,v_j)\}$.

Let v_j be an internal node. If $V_j \cap \{r_j^1, \ldots, r_j^i\}=\varnothing$ or $q \le 0$ or $i = 0$, the value of $G(v_j,q,r_j^i)$ should not be counted since the situation is not included in the definition. Therefore we set $G(v_j,q,r_j^i)=-\infty$. If $r_j^i \notin V_j$, at least one node must be selected in $\{r_j^1, \ldots, r_j^i\} \cap V_j$ is equivalent to at least one node must be selected in $\{r_j^1, \ldots, r_j^{i-1}\} \cap V_j$. Therefore $G(v_j,q,r_j^i)=G(v_j,q,r_j^{i-1})$. The rest situations can be

separated into 3 cases: $r_j^i=v_j$ and $q \geq 1$, $r_j^i \in V_{j1}$ and $q \geq 1$, and $r_j^i \in V_{j2}$ and $q \geq 1$. We discuss them as follows.

In case 1 we consider that $r_j^i=v_j$ and $q \geq 1$. Without loss of generality, we assume that r_j^i corresponds to $r_{j1}{}^{i1}$ for v_{j1} (the left child node of v_j) and $r_{j2}{}^{i2}$ for v_{j2} (the right child node of v_j).We claim that
$G(v_j,q,r_j^i)=\max\{G(v_j,q,r_j^{i-1}),A^r(v_j)\cdot A_j^i+\max\{F(v_{j1},q_1,r_{j1}{}^{i1})+F(v_{j2},q_2,r_{j2}{}^{i2})|\ 0\leq q_1,q_2\leq q-1,\ q_1+q_2=q-1\}\}$.
In order to prove that, we must show (1) the optimal solution appears either in $G(v_j,q,r_j^{i-1})$ or $A^r(v_j)\cdot A_j^i+\max\{F(v_{j1},q_1,r_{j1}{}^{i1})+F(v_{j2},q_2,r_{j2}{}^{i2})\ |\ 0\leq q_1,q_2\leq q-1,\ q_1+q_2=q-1\}$, and (2) the optimal solution is the maximum value.

First, if a node in $\{r_j^1, \ldots, r_j^{i-1}\} \cap V_j$ is selected in the optimal solution, then the optimal solution appears in $G(v_j,q,r_j^{i-1})$. Otherwise we assume that no node in $\{r_j^1, \ldots, r_j^{i-1}\} \cap V_j$ is selected but r_j^i is selected in the optimal solution. Then it is better for v_j to fetch data from node $r_j^i=v_j$ than other nodes. Let q_1 nodes will be selected from T_{j1} and q_2 nodes will be selected from T_{j2} where $q_1+q_2=q-1$. Hence the optimal solution appears in $A^r(v_j)\cdot A_j^i+\max\{F(v_{j1},q_1,r_{j1}{}^{i1})+F(v_{j2},q_2,r_{j2}{}^{i2})\ |\ 0\leq q_1,q_2\leq q-1,\ q_1+q_2=q-1\}$.

Second, $A^r(v_j)\cdot A_j^i+\max\{F(v_{j1},q_1,r_{j1}{}^{i1})+F(v_{j2},q_2,r_{j2}{}^{i2})\ |\ 0\leq q_1,q_2\leq q-1,\ q_1+q_2=q-1\}$ may contain some inconsistent values. For example, if $F(v_{j1},q_1,r_{j1}{}^{i1})$ or $F(v_{j2},q_2,r_{j2}{}^{i2})$ selects a node in $\{r_j^1, \ldots, r_j^{i-1}\} \cap V_j$, v_j should fetch data from that node instead of r_j^i. However since the term $A^r(v_j)\cdot A_j^i$ is used, it actually counts that v_j fetches data from node r_j^i. Or $F(v_{j1},q_1,r_{j1}{}^{i1})$ selects a node such that v_{j2} should fetch data from that node outside T_{j2} instead of $r_{j2}{}^{i2}$. However the term $F(v_{j2},q_2,r_{j2}{}^{i2})$ indicates that actually it counts that v_{j2} fetches data from node $r_{j2}{}^{i2}$ outside T_{j2}. Fortunately, the inconsistent values are smaller than the corresponding consistent values, and hence are smaller than the optimal solution. Therefore the inconsistent values will not affect the optimal solution to be the maximum value.

In case 2 we consider $r_j^i \in V_{j1}$ and $q \geq 1$. Without loss of generality, we assume that r_j^i corresponds to $r_{j1}{}^{i1}$ for v_{j1} (the left child node of v_j) and $r_{j2}{}^{i2}$ for v_{j2} (the right child node of v_j). We claim that
$G(v_j,q,r_j^i)=\max\{G(v_j,q,r_j^{i-1}),A^r(v_j)\cdot A_j^i+\max\{G(v_{j1},q_1,r_{j1}{}^{i1})+F(v_{j2},q_2,r_{j2}{}^{i2})\ |\ 1\leq q_1\leq q,\ 0\leq q_2\leq q,\ q_1+q_2=q,$ the associated set of $G(v_{j1},q_1,r_{j1}{}^{i1})$ does not contain $r_j^1, \ldots, r_j^{i-1}$ but contains $r_j^i\}\}$.

First, if a node in $\{r_j^1, \ldots, r_j^{i-1}\} \cap V_j$ is selected in the optimal solution, then the optimal solution appears in $G(v_j,q,r_j^{i-1})$. Otherwise we assume that no node in $\{r_j^1, \ldots, r_j^{i-1}\} \cap V_j$ is selected but r_j^i is selected in the optimal solution. We claim that v_j will not be selected. Because if v_j is selected, then it cannot be in $\{r_j^1, \ldots, r_j^{i-1}\} \cap V_j$. Thus it is better for v_j to fetch data from node r_j^i than v_j. Consequently it is better for all nodes to fetch data from node r_j^i than v_j. It implies that the selection of v_j will not gain any benefit. Therefore v_j will not be selected. So q_1 nodes will be selected from T_{j1} and q_2 nodes will be selected from T_{j2} where $q_1+q_2=q$. Hence the optimal solution appears in $A^r(v_j)\cdot A_j^i+\max\{G(v_{j1},q_1,r_{j1}{}^{i1})+F(v_{j2},q_2,r_{j2}{}^{i2})\ |\ 1\leq q_1\leq q,\ 0\leq q_2\leq q,\ q_1+q_2=q,$ the associated set of $G(v_{j1},q_1,r_{j1}{}^{i1})$ does not contain $r_j^1, \ldots, r_j^{i-1}$ but contains $r_j^i\}$.

Second, if the associated set of $G(v_{j1},q_1,r_{j1}{}^{i1})$ does not contain $r_j^1, \ldots, r_j^i$, then that $G(v_{j1},q_1,r_{j1}{}^{i1})$ should not be counted. If the associated set of $G(v_{j1},q_1,r_{j1}{}^{i1})$ contains $r_j^1, \ldots, r_j^{i-1}$, then the inconsistent value is smaller than the corresponding consistent value, and hence smaller than the optimal value. so that $G(v_{j1},q_1,r_{j1}{}^{i1})$ can be omitted. Therefore it suffices to consider $G(v_{j1},q_1,r_{j1}{}^{i1})$ whose associated set does not contain $r_j^1, \ldots, r_j^{i-1}$ but contains r_j^i.

In case 3 we consider $r_j^i \in V_{j2}$ and $q \geq 1$. Without loss of generality, we assume that r_j^i corresponds to r_{j1}^{i1} for v_{j1} (the left child node of v_j) and r_{j2}^{i2} for v_{j2} (the right child node of v_j). Analogous to the proof of case 2, it can be proved that
$G(v_j,q,r_j^i)=\max\{G(v_j,q,r_j^{i-1}),A^r(v_j)\cdot A_j^i+\max\{F(v_{j1},q_1,r_{j1}^{i1})+G(v_{j2},q_2,r_{j2}^{i2}) \mid 0\leq q_1\leq q,\ 1\leq q_2\leq q,\ q_1+q_2=q,$ the associated set of $G(v_{j2},q_2,r_{j2}^{i2})$ does not contain $r_j^1, \ldots, r_j^{i-1}$ but contains $r_j^i\}\}$.

If $r_j^i \in V_j$ or $q < 0$, the value of $F(v_j,q,r_j^i)$ should not be counted since the situation is not included in the definition. Therefore we set $F(v_j,q,r_j^i)=-\infty$. If $r_j^i \notin V_j$ and $q \geq 0$, without loss of generality, we assume that r_j^i corresponds to r_{j1}^{i1} for v_{j1} (the left child node of v_j) and r_{j2}^{i2} for v_{j2} (the right child node of v_j). We claim that
$F(v_j,q,r_j^i)=\max\{G(v_j,q,r_j^i),A^r(v_j)\cdot A_j^i+\max\{F(v_{j1},q_1,r_{j1}^{i1})+F(v_{j2},q_2,r_{j2}^{i2}) \mid 0\leq q_1\leq q,\ 0\leq q_2\leq q,\ q_1+q_2=q\}\}$
First, if the optimal solution selects node in $\{r_j^1, \ldots, r_j^i\} \cap V_j$, then the optimal solution appears in $G(v_j,q,r_j^i)$.

Otherwise we assume that no nodes in $\{r_j^1, \ldots, r_j^i\} \cap V_j$ is selected in the optimal solution. Then it is better for v_j to fetch data from node r_j^i than other nodes. Additionally v_j will not be selected because of the same reason in the proof of case 2. So q_1 nodes will be selected from T_{j1} and q_2 nodes will be selected from T_{j2} where $q_1+q_2=q$. Hence the optimal solution appears in

$$A^r(v_j)\cdot A_j^i + \max\{F(v_{j1},q_1,r_{j1}^{i1})+F(v_{j2},q_2,r_{j2}^{i2}) \mid 0\leq q_1\leq q,\ 0\leq q_2\leq q,\ q_1+q_2=q\}.$$

Finally, the inconsistent values in $A^r(v_j)\cdot A_j^i+\max\{F(v_{j1},q_1,r_{j1}^{i1})+F(v_{j2},q_2,r_{j2}^{i2}) \mid 0\leq q_1\leq q,\ 0\leq q_2\leq q,\ q_1+q_2=q\}$ are smaller than the corresponding consistent values, and hence are smaller than the optimal solution. Therefore the inconsistent values will not affect the optimal solution to be the maximum value.

4. EXPERIMENTAL STUDIES

4.1 Evaluating the Effectiveness of Availability Modeling in Tree Networks

In section II, we have shown that the proposed availability modeling algorithm in a tree network has a complexity of $|V|^2$, where $|V|$ is the number of nodes in the tree network. In this section, we investigate the practical complexity of our proposed algorithm. In this simulation, a variety of random tree topologies are generated. We use four parameters to control the generation of a tree: the total number of nodes ($|V|$), resident size ($|R|$), the number of reads issued from each node ($A^r(u)$), and the maximum node degree (*degree*). A random tree is generated in a breadth-first manner, i.e., starting from the root node, at each level, a random number of node are generated for the next level based on the randomly generated node degrees, until the number of nodes specified is reached. The root of the tree always hosts a replica and the rest of $|R|$-1 replicas are allocated to randomly selected $|R|$-1 nodes. The number read on each node is randomly generated following a uniform distribution, each in the range of (0 10). Link availability (*LA*) of each edge and node availability (*NA*) of each node in the system is generated randomly following a uniform distribution, each is in the range of (0.9 1.0).

Table 11. Performance of the Algorithm in a tree

Node Number	**Memory (KB)**	**Runtime**
100	1164	15 ms
1000	4787	1.34s
10000	7952	1.84 min.
100000	28922	2.06 hr

To study actual performance of the algorithm, the actual runtime and memory used are determined. The memory used is dynamic and the maximum memory used is measured in the middle of the recursive algorithm execution process. The runtime is determined as the time between the start of the execution of the algorithm and end of the algorithm. Note that the actual runtime and memory used here do not necessarily mean the lowest runtime and maximum memory used, and the algorithm is implemented in Java. The parameters are set as follows: *degree* = 5, each node can have 0 to 5 children, $|R| = 6$, and $|V|$ is ranged from 100 to 100000. As can be seen in Table 11, the memory used by the algorithm linearly increased along the increase of $|V|$, while the run time of the algorithm increases along with the increase of $|V|$, governed by a growth function of the square of $|V|$. The results confirmed our theoretic analysis in Section II such that the availability modeling algorithm in a tree network has a runtime complexity of $|V|^2$.

We then test our modeling approach against the simulated distributed environment to see how good it is. The impacts of three factors on $Tree\_S_A(T, R)$ are studied: (a) the graph size $|V|$; (b) the graph degree, which is the average number of neighbors of a node; and (c) the size of the resident set $|R|$. To avoid biased access patterns and topology structures, we repeat the experimental steps 100 times. The final result is the average of the 100 trials. The results are shown in Fig. 8 (a), Fig. 8 (b), and Fig. 8 (c).

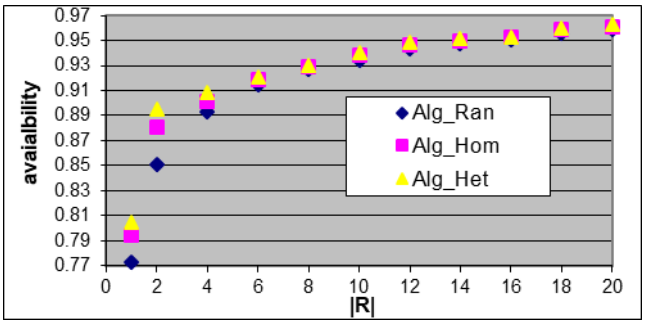


Fig. 8 (a). The impact of resident set size

Fig.8(a) shows the impact of resident set size on the effectiveness of the three replica allocation algorithms. The parameters are set as follows: $|V| = 100$, *degree* = 5, and $|R|$ is increased from 1 to 20. The result shows that data availabilities of the three replica allocation schemes increase along with the increase of $|R|$, which confirms the conclusion of Theorem 1. From Fig. 7(a), the two resident sets computed by the two replica allocation algorithms, *Alg_Hom* and *Alg_Het*, can always achieve higher availabilities than the availability achieved by the resident set that is computed by the randomized replication allocation algorithm. However, the difference on the availabilities of the resident sets between the randomized and the two algorithms, *Alg_Hom* and *Alg_Het*, decreases along with increase of $|R|$. Also, the availability increase rate decreases along with increase of $|R|$. The reason is obvious. With a larger $|R|$, the average distance from a user to service/data site (number of edges in the path) will be reduced and thus the service/data reach availability should be increased. Thus, the space of service availability enhancement will also be reduced when availability reaches a high level.

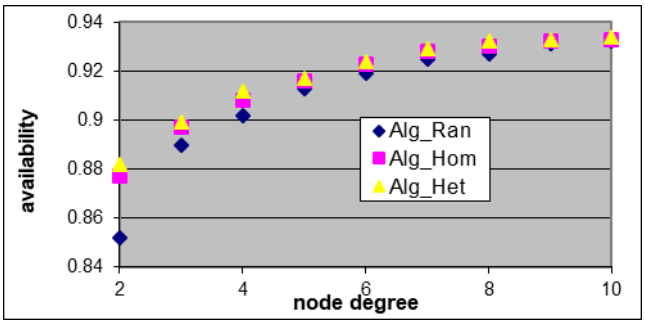


Fig. 8 (b). The impact of node degree

Fig.8(b) shows the impact of node degree on the effectiveness of the three replica allocation algorithms. The parameters are set as follows: $|V| = 100$, $|R| = 6$, and *degree* is increased from 2 to 10. The result shows that data availabilities of the three replica allocation schemes increase along with the increase of *node degree*. From Fig. 7(b), the two resident sets computed by the two replica allocation algorithms, *Alg_Hom* and *Alg_Het*, can always achieve higher availabilities than the availability achieved by the resident set that is computed by the randomized replication allocation algorithm. However, the difference on the availabilities of the resident sets between the randomized and the two algorithms, *Alg_Hom* and *Alg_Het*, decreases along with increase of node degree. The availabilities increase rate also slows down when node degree reach a certain size (e.g., *degree* = 5). The reason is obvious. With a larger node degree, the height of the tree will be reduced. Therefore, the average distance from a user to service/data site (number of edges in the path) will be reduced and thus the service/data reach availability should be increased. Thus, the space of service availability enhancement will also be reduced when availability reaches a high level.

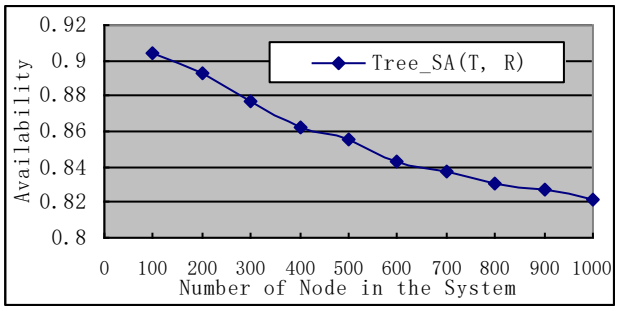


Fig. 8 (c). The impact of graph size on data availability

Fig.8(c) shows the impact of graph size on the data availability modeled and computed by $Tree\_S_A(T, R)$. The parameters are set as follows: $|R| = 6$, *degree* = 5 and $|V|$ is increased from 100 to 1000. The result shows that data availability decreases along with increase of graph size. The reason is obvious. With a larger $|V|$, the average distance from a user to service/data site (number of edges in the path) will be increased and thus the service/data reach availability should decrease.

4.2 Evaluating the Effectiveness of Availability Enhancement Algorithms in Tree Networks

In this section, we test two replica allocation algorithms we developed in this research in a simulated distributed environment to see how effective they are on enhancing service availabilities. Specifically, we compare the availability enhanced algorithm, *Alg_Enhan*, with the

randomized replica allocation scheme, *Alg_Ran*. In this simulation, a variety of random tree networks are generated. We use to simulate a tree with 30 nodes which have a variety of resident size (|*R*. A random tree is generated in a breadth-first manner, i.e., starting from the root node, at each level, a random number of the node are generated for the next level based on the randomly generated node degrees, until the number of nodes specified is reached. The entire tree hosts |*R*| replicas. The number read on each node is randomly generated following a uniform distribution, each is in the range of (0, 10). Link availabilities of all the edges and node availabilities of all the nodes in the system are generated randomly following a uniform distribution in the range of (0.9, 1.0). For the randomized allocation scheme, replicas are allocated to the randomly selected |*R*| nodes in the system. For algorithms *Alg_Enhan*, we will first implement the algorithm based on a tree network, and then use this algorithm to compute the resident sets in the simulated tree network. Finally, we will apply our availability computing algorithm shown in Fig. 3 to compute the service availabilities for the two resident sets in the simulated tree network.
Our goal is to test which replica allocation scheme is more effective in enhancing overall service availability in a tree network. The impacts of the resident set size |*R*| (from 1 to 6). The experiments are repeated 10 times and the results are shown in Fig. 9.

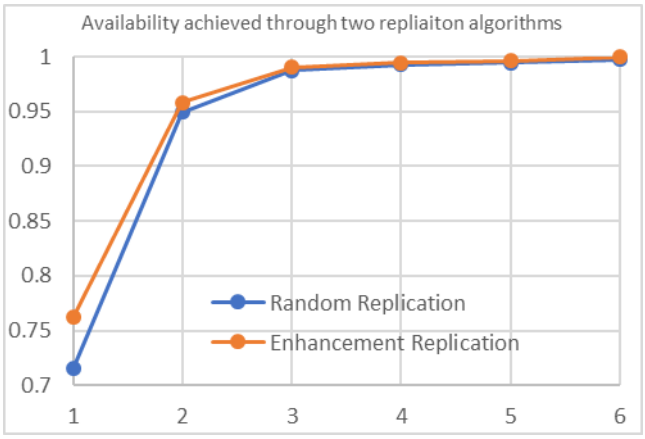


Fig.9. The impact of resident set size on improved availability

Fig.9 shows the impact of resident set size on the effectiveness of the three replica allocation algorithms. The parameters are set as follows: |*V*| = 30, *degree* = 5, and |*R*| is increased from 1 to 6. The result shows that data availabilities of the two replica allocation schemes increase along with the increase of |*R*|, which confirms the conclusion of Theorem 1. From Fig. 8, the resident set computed by *Alg_Enhan* can always achieve higher availabilities than the availability achieved by the resident set that is computed by *Alg_Ran*. However, the difference on the availabilities of the resident sets between *Alg_Ran* and *Alg_Enhan* decreases along the increase of |*R*| (from 0.05 to 0.002). Also, the availability increase rate decreases along the increase of |*R*|. The reason is obvious. With a larger |*R*|, the average distance from a user to service/data site (number of edges in the path) will be reduced and thus the service/data reach availability should be increased. Thus, the space of service availability enhancement will also be reduced when availability reaches a high level. From the above experimental study, we can conclude that when the replica size is small, the replica location has the bigger impact on availability. Hence, it is more critical to employ a better replica allocation algorithm to improve service availability, such as *Alg_Enhan.*

5. RELATED WORK

Network reliability has been well studied and a large amount of works have been conducted on this topic [15, 23, 24, 31]. The modeling and computation of network reliability is similar to (but not the same as) a specific case of network availability modeling for a distributed system, i.e., modeling the availability of a distributed system with no replication. For such a system, the data availability can be computed by computing the reach availability from each node to the destination node (the node with the data). The difference between availability and reliability modeling is that the unavailability of a node does not affect the reaching probability in our data availability modeling (in reliability, the node unavailability will affect the reliability). However, the availability modeling in a distributed system with data replication is very different from reliability modeling. The data availability in the system to a node *w* (actually the clients associated with such node) is affected by multiple nodes (each with a copy of the data) at the same time, i.e., if a copy of data on node *u* is not available to node *w*, then another node *v* can provide availability to *w*. Therefore, computing and modeling data availability to node *w* not only needs to compute the reach probability to multiple destination nodes (for example *u* and *v*), but also needs to consider their impact on each other.

Availability has long been considered as a critical issue for systems providing replicated services. In [8, 9], the authors explored the benefits of dynamically trading consistency for availability using a continuous consistency model. The availability model is a function of consistency, workload, and fault load. If consistency requirement is greatly relaxed, availability can be very high. In data storage systems such as Ocean store [14, 27] and PASIS [28], both pure replication and data fragmentation approaches are considered. Their availability models only consider node availability. [11] and [12] both present an analytical model for determining the overall availability of data stored in a data grid or P2P system. The availability model captures the characteristics of peer-to-peer systems and grid systems. The availability of replica location service (metadata service) of data grids or P2P systems can be considered independent of data storage availability.
All the above works on data availability computation only consider the availability of the service or data sites and do not consider network link availability. In [6, 7], the authors propose a model for studying the availability of replicated systems. The system availability is computed in a stochastic graph, i.e. each node and each link are assigned a specific failure probability. The overall system availability is computed exhaustively using a state enumeration method. However, the runtime complexity of the state enumeration method is exponential in the number of nodes in the system in a tree network. Also, it does not give a solution to compute data availability with multiple replicas in the system. In our previous research in [17, 29], we proposed effective availability modeling and computing approaches that consider both node and network link failures, for systems with the tree, ring, and general graph topologies. In [18], we provided two availability enhancement algorithms for a tree network with homogeneous node availability and link availability, which is a special case for the problem we consider in this research. Hence, a solution is needed for the general case, i.e., each node availability and link (network edge) availability are heterogeneous.

6. CONCLUSIONS

In this paper, we address the system service or data availability modeling and enhancement approaches for distributed systems with data replication. We first provide an efficient availability computing algorithm for tree networks. We then analyze the availability enhancement problem and formalized availability enhancement in a general tree network as Problem 1. Then, the problems are transformed into a special p-median problem and a recursive algorithm-based solution, Alg_Enhan is developed to maximize the service availabilities for distributed systems with a tree topology. The algorithm Alg_Enhan considers the impact of replica location on overall

service availability and the impact of network node and edge heterogeneity. Theorems have shown that the algorithm Alg_Enhan is efficient with a runtime complexity of O(K|V|2), where K is the number of replicas and |V| is the number of nodes in the network. Finally, experimental studies show that the algorithm Alg_Enhan can compute resident sets with higher availabilities than those computed by the randomized algorithm, but it is more effective when the number of replicas is small.

REFERENCES

[1] J. Hennessy, "The future of systems research", Computer, vol. 32, no. 8, pp. 27-33, 1999.

[2] J. Liu and H. Shen, "A Low-Cost Multi-failure Resilient Replication Scheme for High Data Availability in Cloud Storage", In High Performance Computing (HiPC), 2016 IEEE 23rd International Conference on (pp. 242-251),2016.

[3] K. Meenakshi and S.B. Singh, "Availability Assessment of Multi-State System by Hybrid Universal Generating Function and Probability Intervals", International Journal of Performability Engineering, Vol.12, No. 4, pp. 321-339, 2016.

[4] S. Ren, Y. Yang, Y. Chen and Y. Du, "Fluctuation analysis of instantaneous availability under specific distribution", Neurocomputing, vol. 270, pp. 152-158, 2017.

[5] M. Dahlin, B. Chandra, Lei Gao and A. Nayate, "End-to-end WAN service availability", IEEE/ACM Transactions on Networking, vol. 11, no. 2, pp. 300-313, 2003.

[6] G. On, J. Schmitt, and R. Steinmetz, "On availability Qos for replicated multimedia service and content", In Proceedings of International Workshop on Interactive Distributed Multimedia Systems, 2002.

[7] G. On, J. Schmitt, and R. Steinmetz, "Quality of availability: replica placement for widely distributed systems", In IWQos'03, 2003.

[8] H. Yu and A. Vahdat, "The costs and limits of availability for replicated services", ACM SIGOPS Operating Systems Review, vol. 35, no. 5, p. 29, 2001.

[9] H. Yu and A. Vahdat, "The costs and limits of availability for replicated services", ACM Transactions on Computer Systems, vol. 24, no. 1, pp. 70-113, 2006.

[10] M. Aguilera, R. Janakiraman, and L. Xu, "Using Erasure Codes Efficiently for Storage in Distributed System," 2005 International Conference on Dependable Systems and Networks (DSN05).

[11] K. Ranganathan, A. Iamnitchi, and I. Foster, "Improve data availability through dynamic model-driven replication in large peer-to-peer communities", In proceedings of 2nd IEEE/ACM International symposium on Cluster Computing and the Grid. 2002.

[12] F. Schintke and A. Reinefeld, "Modeling Replica Availability in Large Data Grids", Journal of Grid Computing, vol. 1, no. 2, pp. 219-227, 2003.

[13] B. Tang, N. Jaggi, and M. Takahashi, "Achieving data K-Availability in intermittently connected sensor networks", In Proceedings of the 23rd International Conference on Computer Communication and Networks (ICCCN'14), Shanghai, China, 2014.

[14] H. Weatherspoon and J. Kubiatowicz, "Erasure coding vs. replication: a quantitative comparison", In proceedings of Peer-to-Peer Systems: First International Workshop (IPTPS), 2002.0

[15] A. Tamir, "An O(pn2) algorithm for the p-median and related problems on tree graphs", Operations Research Letters, vol. 19, no. 2, pp. 59-64, 1996.

[16] V. Paxson, "End-to-end routing behavior in the Internet", IEEE/ACM Transactions on Networking, vol. 5, no. 5, pp. 601-615, 1997.

[17] M. Tu, D. Xu, Z. Xia, and J, Fu, “Reach availability of replicated services”, In Proceedings of the 35th IEEE International Computer Software and Application Conference, July 2011.

[18] M. Tu, L. Xiao, and D. Xu, “Maximizing the availability of replicated services in widely distributed systems”, In Proceedings of IEEE International Conference on Software Security and Reliability, Washington DC, June, 2013.

[19] K. Kalpakis, K. Dasgupta and O. Wolfson, "Optimal placement of replicas in trees with read, write, and storage costs", IEEE Transactions on Parallel and Distributed Systems, vol. 12, no. 6, pp. 628-637, 2001.

[20] O. Wolfson and A. Milo, "The multicast policy and its relationship to replicated data placement", ACM Transactions on Database Systems, vol. 16, no. 1, pp. 181-205, 1991.

[21] O. Wolfson, S. Jajodia and Y. Huang, "An adaptive data replication algorithm", ACM Transactions on Database Systems, vol. 22, no. 2, pp. 255-314, 1997.

[22] O. Kariv and S. Hakimi, "An Algorithmic Approach to Network Location Problems. II: The p-Medians", SIAM Journal on Applied Mathematics, vol. 37, no. 3, pp. 539-560, 1979.

[23] L. Fratta and U. Montanari, "A Recursive Method Based on Case Analysis for Computing Network Terminal Reliability", IEEE Transactions on Communications, vol. 26, no. 8, pp. 1166-1177, 1978.

[24] J. Fussell, "How to Hand-Calculate System Reliability and Safety Characteristics", IEEE Transactions on Reliability, vol. -24, no. 3, pp. 169-174, 1975.

[25] "On the location of a tree-shaped facility", Location Science, vol. 5, no. 1, p. 63, 1997.

[26] M. M. Halldórsson, K. Iwano, N. Katoh, and T. Tokuyama, “Finding Subsets Maximizing Minimum Structures,” SIAM Journal on Discrete Mathematics, vol. 12, no. 3, pp. 342–359, 1999.

[27] J. Kubiatowicz, C. Wells, B. Zhao, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea and H. Weatherspoon, "OceanStore", ACM SIGARCH Computer Architecture News, vol. 28, no. 5, pp. 190-201, 2000.

[28] J. Wylie, M. Bakkaloglu, V. Pandurangan, M. Bigrigg, S. Oguz, K. Tew, C. Williams, G. Ganger, and P. Khosla, “Selecting the right data distribution scheme for a survivable storage system”, Technical Report CMU-CS-01-120, Carnegie Mellon University, 2000.

[29] M. Tu, H. Ma, I. Yen, F. Bastani, and D. Xu, “Availability, security, access performance and load balance in P2P data grid”, Journal of Grid Computing, Vol. 11, No. 1, 2013.

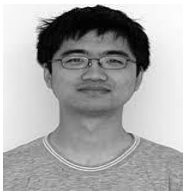
[30] A. Rahmani, Z. Fadaie and A. Chronopoulos, "Data placement using Dewey Encoding in a hierarchical data grid", Journal of Network and Computer Applications, vol. 49, pp. 88-98, 2015.

[31] O. Theologou and J. Carlier, "Factoring and reductions for networks with imperfect vertices", IEEE Transactions on Reliability, vol. 40, no. 2, pp. 210-217, 1991.

AUTHORS

Michael Tu is a Ph.D. in Computer Science, Associate Professor of Computer Information Technology, director of the Center for Cyber Security and Infrastructure Protection at Purdue University Northwest. Dr. Tu's areas of expertise are information assurance, digital forensics, and cybersecurity education. He has published over 40 peer reviewed papers in prestigious journals and peer reviewed conference proceedings. Dr. Tu is a Certified Information System Security Professional (CISSP), Certified Ethical Hacker (CEH), Certified Hacking and Forensics Investigator (CHFI) and AccessData Computer Examiner (ACE).

Dianxiang Xu, received the B.S., M.S., and Ph.D. degrees in computer science from Nanjing University, Nanjing, China. He is a Professor with the Department of Computer Science, Boise State University, Boise, ID, USA. His research interests include software security and safety, software engineering, access control, and software-defined systems. Dr. Xu has published over 100 peer reviewed papers in prestigious journals and peer reviewed conference proceedings.



Liangliang Xiao, received the M.S., and Ph.D. degrees in computer science from The University of Texas at Dallas, Richardson, USA. He is an Associate Professor with the Department of Computer Science & Information Technologies, Frostburg State University, Frostburg, MD, USA. His research interests include data security and distributed systems. Dr. Xiao has published over 30 peer reviewed papers in prestigious journals and peer reviewed conference proceedings.