CCIT Report #377 May 2002

# Optimal content location in overlay networks with updates

Israel Cidon
Department of Electrical Engineering
Technion - Israel Institute of Technology
Email: cidon@ee.technion.ac.il

Oren Unger
Department of Electrical Engineering, Technion
and Zoran Microelectronics
Email: unger@tx.technion.ac.il

**Abstract**

A major challenge for organizations and application service providers (ASP) is to provide high quality services over the network to geographically dispersed consumers at a reasonable cost. As the performance of the Global Internet and Corporate Intranets is unpredictable, such providers employ content delivery networks (CDN) and overlay networks to bring content and applications closer to their service consumers. The overlay network is composed of a set of distributed servers which serve local consumers and are updated and synchronized with the most recent information.

The architecture of overlay networks should encourage high-performance, high-scalability and reduced costs. This becomes more crucial as despite technological advances, communication, storage costs as well as service latencies grow with the exploding amounts of data exchanged and with the size and span of the overlay network. Examples for objects which incur high traffic, storage and update costs are interactive media objects as well as large design materials and software repositories. For that end, multicast methodologies can be used to deliver content from regional servers to end users, as well as for the timely and economical synchronization of content among the distributed servers. Another important architectural problem is the efficient allocation of objects to servers to minimize storage, delivery and update costs while maintaining the expected quality of service.

In this work, we suggest an IP multicast based architecture and address the optimal allocation and replication of objects that are both consumed and updated. Our model network includes consumers which are served using IP multicast or IP unicast transmissions and media sources (that may be also consumers) which update the objects within the servers using multicast communication. General costs are associated with distribution (download) and update traffic as well as the storage of objects in the servers. Optimal object allocation algorithms for tree networks are presented with complexities of $O(N)$ and $O(N^2)$ in case of multicast and unicast distribution respectively.

To our knowledge, the model of multicast distribution combined with multicast updates has not been analytically dealt before, despite its popularity in the industry.

## I. Introduction

Recent years have witnessed tremendous activity and development in the area of content and services distribution. Geographically dispersed consumers and organizations demand higher throughput and lower response time for accessing distributed content, outsourced applications and managed services. In order to enable high quality and reliable end-user services despite unpredictable Internet and Intranet conditions, organization and applications service providers (ASPs) employ content delivery networks (CDN) and overlay networks. These networks bring content and applications closer to their consumers, overcoming slow backbone paths, network congestions and physical latencies. Multiple vendors such as Cisco[1], Akamai[2] and Digital Fountain[3] offer CDN services and overlay technologies. Recently, more collaborative models such as distributed storage and peer-to-peer computational models require both consumption and modification of the content by multiple, geographically distributed users[5][6].

Naturally, organizations and ASPs try to optimize the overall cost of the overlay network mainly in terms of storage and communication costs. Efficient allocation of information objects to the overlay network servers reduces the operational cost and improves the overall performance. This becomes more crucial as the scale of services extend to a large number of users over international operation where communication and storage costs as well as network latencies are high. The optimization problem becomes more difficult as the service becomes dynamic and needs to be changed, updated and synchronized frequently.

Our model network includes consumers, which are served from servers using IP unicast or IP multicast communication. The network also includes media sources (that may also be consumers) that update and modify the objects within the servers. The update traffic between a media source and the relevant servers is most efficiently conducted using IP multicast communication (IP multicast communication within the overlay network is used by vendors such as Digital Fountain[3]). Using IP multicast can reduce significantly the overall update transport and the update latency. General costs are associated with distribution and update traffic as well as the storage of objects in the servers.

The popularity of IP multicast for distribution of the content is especially increasing for real-time and multimedia applications that consume high bandwidth and are delivered to a large number of consumers. There are several companies that offer multicast based servers as well as multicast based content delivery services[1][3][4]. A possible application of the new multicast distribution and multicast update model is in a collaborative environment where one or more media sources (which may also be consumers) constantly update the media servers in the overlay network, and the media servers distribute the media to all the consumers in the network using multicast.

Specifically, our initial overlay network model is a tree graph that has a server located at each of its vertices. The use of a tree graph does capture the IP multicast environment where a tree is pre-computed by the network itself (PIM-SM[7]). The vertices also include communication gateways to other vertices and optional entries to local consumers and media sources. Each server may be assigned an arbitrary storage costs and each edge may be assigned with an arbitrary communication cost (for distribution and update). The distribution demand of the consumers and the update requirements of the media sources are known a-priory (static).

We present two optimal object allocation algorithms for tree networks with computational complexities of $O(N)$ and $O(N^2)$ in case of multicast and unicast distribution respectively. In both cases updates are conducted via multicast. The algorithms fit the native model of IP multicast and are easily mapped to distributed implementation.

The object allocation problem, also referred as the file allocation problem (FAP) in storage systems or data management in distributed databases has been studied extensively in the literature. While the multicast distribution model of this paper is new and was never considered before, the combination of multicast update (write) and unicast distribution (read) was considered in the past. Dowdy and Foster[8] survey a number of models for the FAP. In the static FAP they present models in which the update traffic is point-to-point traffic (unicast). Wolfson and Milo[9] present a model of replicated data placement in distributed databases using multicast for writes and unicast for reads. Their model does not include the storage costs and has a unit traffic cost for both reads and writes. They prove that the object allocation problem is NP-Complete for general graphs. They provide an algorithm with $O(N)$ computational complexity for tree networks.

More recent works by Kalpakis et al.[10] and Krick et al.[11] present a model of a network with unicast reads, multicast write and storage costs. [10] presents a problem with additional constrains for a tree network and the algorithm they present for the similar case is less efficient than our algorithms (to be later described). [11] deal with more general networks and provides an optimal algorithm for the tree network which is also less efficient than our algorithm. These works don't solve the multicast reads multicast write problem. Moreover, the multicast protocol presented in [10] and [11] assumes that the media source sends a unicast message to the closest server (which stores an object) and the server itself forwards a multicast message over the minimum spanning tree to the other servers (i.e. - multicast is used only between the servers). This scheme does not employ properly the IP native multicast protocol where a single source can update directly all the servers via an IP native multicast tree. Therefore, our model uses an IP native multicast protocol in which the media source sends the message to all the servers using an IP multicast tree (i.e. - each media source manages an IP multicast tree for updates). The IP multicast that we present is common in the industry, for example in peer-to-peer multicast, in which each source sends data to its peers using multicast. The computational complexity for the similar (but not identical) object allocation problem on a tree network is $O(N^5)$ in [10] and $O(N \cdot diam(T) \cdot \log(deg(T)))$ in [11] (Worst case is $O(N^2 \cdot \log(N))$).

When omitting the update process from the related unicast distribution model, we end up with the classical "uncapacited plant location problem"[12] model with facilities replacing servers and roads replacing communication lines. The problem has been proved to be NP-complete for general graphs[12]. It was solved for trees in polynomial time[13][14]. The "uncapacited plant location problem" was mapped to content delivery networks[15]. Additional works that address the read only model address the server location problem given a fixed number of servers[16][17] and objects[18].

The two algorithms we present in this article share several mutual properties. They are based on dynamic programming and on the critical observation that when potential costs are computed at a server not all the possible states (object locations) of all the (down stream) servers should be taken into account, but it is enough to know if there is (or is not) a server that needs to be updated inside/outside a subtree. So only up to three different potential costs are computed for each servers pair in the unicast distribution case and up to four potential costs in the multicast distribution case. An additional important observation is the fact that the total update traffic generated by all the media sources, can be characterized and merged for each edge prior to running the algorithm with computational complexity of $O(N)$ (see sub-section II-E). The resulted algorithms are performed in two phases. First a *potential* cost calculation phase advances from the leaves of the tree up the root. After the termination of the first phase, a second backtrack phase for the actual allocation starts at the root and ends at the leaves of the distribution subtree. During the calculation phase, each vertex in the tree calculates multiple alternate costs based on its children's set of costs and its storage and communication costs. The optimal allocation is not known at vertices before the cost calculation phase ends at the root.

This computation structure also enables an easy conversion of the placement procedure into a distributed algorithm. The first phase starts at the leaves of the tree and advances up the tree where each vertex sends its potential cost vector (a constant number of potential costs) to its parent. The second phase starts at the root where each vertex selects the optimal local placement decision and forwards it to its children.

It is also possible to use the algorithms in a quasi static environment, in which the distribution demand and the update requirements are slowly changing and periodically calculated. The algorithm is then performed in order to adopt the changes of demand and replace the copies of the objects.

## II. The model

### A. Objects

For each object $o$ of the objects set $O$, we determine the set of servers which store the object. The algorithm handles each object separately, so the costs described below are defined (and can be different) for each object $o$.

### B. The tree network

Let $T = (V, E)$ be a tree graph that represents a communication network, where $V = \{1, \ldots, N\}$ is the set of vertices and $E$ is the set of edges. The tree is rooted at any arbitrary vertex $r$ ($r$=1). Each vertex in the tree represents a network switch and a potential storage place for object copies. Each vertex in the tree is also an entry point of content consumers and/or media sources to the network. Distribution demands of a consumer are provided by the network from the server at the closest vertex $i$ which stores an object. An object update may be provided by any media source and is sent to all the vertices that store the object using multicast.

Denote the subtree of $T$ rooted at vertex $i$ as $T_i$.

Denote the set of edges in $T_i$ as $E_i$ (including edge $i$)

Denote the set of vertices in $T_i$ as $V_i$ ($V_r \equiv V$) and the set of children of vertex $i$ in $T$ as $Ch_i$ (For a leaf $i$ $Ch_i = \emptyset$).

Denote edge $i$ ($i \neq r$) the edge that connects vertex $i$ to its parent in $T$.

Figure 1. displays a typical tree with various costs related to its vertices and edges. These costs appear in the various formulas for storage, distribution and update traffic.

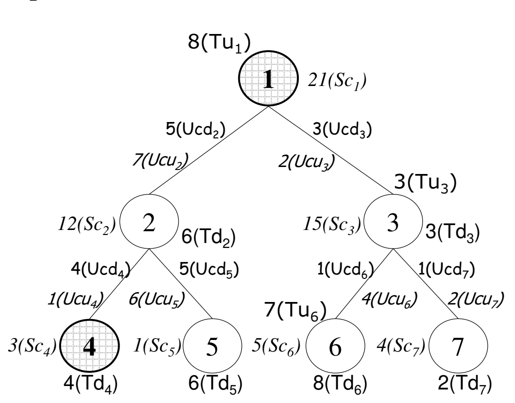

Fig. 1. An example of a tree network and various costs.

## C. Storage cost

Let the storage cost of the object at vertex $i$ to be $Sc_i$. Denote $\Phi$ is the set of vertices that store the object.

The total storage cost of the object in the network is $\sum_{i\in\Phi} Sc_i$.

## D. Distribution traffic cost

Denote the distribution cost per traffic unit at edge $i$ as $Ucd_i$ (The root edge $r$ ($r$=1) does not exist, therefore $Ucd_r \equiv 0$). The distribution cost per traffic unit along a path between vertices $j$ and $k$ is $Dd_{j,k} = \sum_{i\in P_{j,k}} Ucd_i$, where $P_{j,k}$ is the set of edges that connect vertex $j$ to vertex $k$ ($P_{j,k} = P_{k,j}$).

In the unicast case, the total distribution traffic generated by all the consumers at vertex $i$ is $Td_i$.

The total distribution traffic cost in the network is $\sum_{i\in V} Td_i \cdot \min_{j\in\Phi} Dd_{i,j}$.

In the multicast case, the total distribution traffic provided to vertex $i$ is $Td$ (the same for all vertices - since in multicast the server determines the transmission rate, not each customer as in the unicast case). Denote $Dmt_i$ the set of edges in the distribution multicast tree rooted at vertex $i$ (if $i$ does not store an object $Dmt_i \equiv \emptyset$).

The total distribution traffic cost in the network is $\sum_{i\in\Phi} Td \cdot (\sum_{e\in Dmt_i} Ucd_e)$.

## E. Multicast update traffic cost

Denote the update cost per traffic unit at edge $i$ as $Ucu_i$ (The root edge $r$ does not exist, $Ucu_r \equiv 0$).

The total update traffic generated by the media source at vertex $i$ is $Tu_i$.

Denote $Umt_{i,\Phi}$ as the set of edges of the multicast update tree from vertex $i$ to $\Phi$.

The total update traffic cost in the network is $\sum_{i\in V} Tu_i \cdot \left(\sum_{e\in Umt_{i,\Phi}} Ucu_e\right)$.

Since there may be more than one media sources, there may be multiple multicast update trees. For each edge $i$ we define $Tu_i^{out}$ and $Tu_i^{in}$. $Tu_i^{out}$ is the total update traffic that is outgoing via vertex $i$ out of $T_i$, in case there is at least one copy located outside that subtree. $Tu_i^{in}$ is the total update traffic that is incoming via vertex $i$ into $T_i$, in case there is at least one copy located in that subtree.

$$Tu_i^{out} \leftarrow \sum_{j\in V_i} Tu_j$$
$$Tu_i^{in} \leftarrow \sum_{j\notin V_i} Tu_j = Tu_r^{out} - Tu_i^{out}$$

## F. Formulas notations

The following chart describes the most important notations described above that are used in the formulas and algorithms.

| Notation | Explanation |
|---|---|
| $T_i$ | Subtree of $T$ rooted at vertex **i**. |
| $E_i$, $V_i$ | The set of **E**dges/**V**ertices in $T_i$. |
| $Ch_i$ | The **Ch**ildren of vertex **i** in $T$. |
| $\Phi$ | The set of vertices which Store an object |
| $Sc_i$ | **S**torage **C**ost of vertex **i**. |
| $Ucd_i$ | **U**nit **C**ost of **D**istribution for edge **i**. |
| $Td_i$ | **D**istribution **T**raffic at vertex **i**. |
| $Dmt_i$ | Edges of **D**istribution **M**cast **T**ree from **i** |
| $Ucu_i$ | **U**nit **C**ost of **U**pdate for edge **i**. |
| $Tu_i$ | **U**pdate **T**raffic at vertex **i**. |
| $Umt_{i,\Phi}$ | Edges of **U**pdate **M**cast **T**ree from **i** to $\Phi$ |

## III. The optimal algorithm for multicast distribution

The optimization problem is to minimize the total cost (storage and traffic):

$$\sum_{i\in\Phi} Sc_i + \sum_{i\in V} Tu_i \cdot \Big( \sum_{e\in Umt_{i,\Phi}} Ucu_e \Big) + \sum_{i\in\Phi} Td \cdot \Big( \sum_{e\in Dmt_i} Ucd_e \Big) \tag{1}$$

The algorithm calculates the optimal object allocation cost as well as the set of servers that will store the object. Before we turn to the algorithm we first prove a basic property of the optimal object allocation solution.

*Lemma 1:* In the optimal allocation, each vertex $i$ can only belong to one multicast distribution tree.

*Proof:* Suppose a vertex $i$ belongs to more than one multicast distribution trees, then by disconnecting it from the other trees and keeping it connected to only one multicast distribution tree we reduce the distribution traffic in contradiction to the optimality of the cost. Moreover, the optimal solution connects consumers only to the closest multicast distribution tree.

The above property leads us to the conclusion, that the optimal allocation is composed of a subgraph of $T$ which is a forest of multicast distribution subtrees. Each subtree is rooted at a vertex where a copy is located (The subtree rooted at vertex $i$ may not be $T_i$) and its leaves are vertices with distribution demands but no copy of the object stored. An edge in $T$ can be part of at most one multicast distribution tree. Figure 2 demonstrates the structure of a forest with the different possible

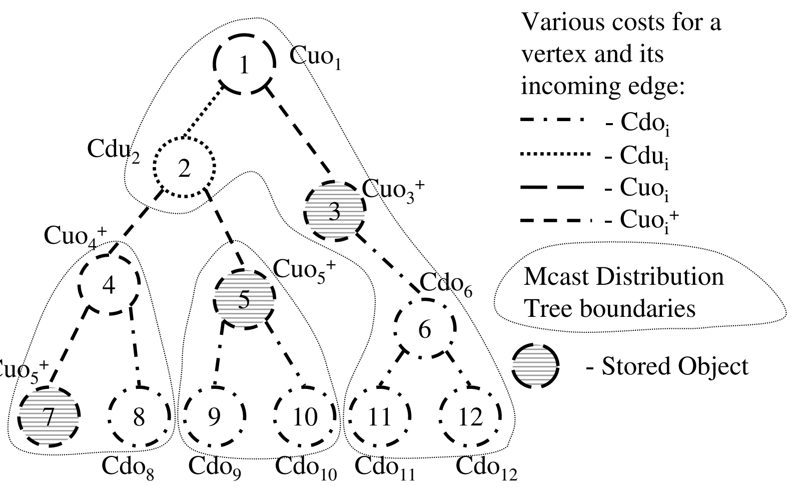

Fig. 2. Example of the multicast distribution forest.

Our algorithm is a recursive algorithm that at each step solves a new problem which is a subset of the problem for the subtree $T_i$ rooted at vertex $i$ and the edge $i$. The new problem also takes into account the possible existence of stored copies outside $T_i$ (results in outgoing update traffic and/or incoming distribution traffic). The new problem doesn't take into account the fact that consumers outside $T_i$ may be served through edge $i$.

The original optimization problem is solved by combining the solutions of the subsets of the problem. The algorithm starts at the leaves and ends at the root of $T$. The motivation to consider at each vertex only one upward edge (i.e. - the edge that connects the vertex to its parent in $T$) is that higher level edges of the hierarchy may be shared by several lower level edges. This principle is valid for both multicast distribution trees and multicast update trees. Since there may be multiple multicast update trees (one for each media source), each edge may have update traffic in both directions.

### A. The cost calculation phase

For each vertex $i$ the algorithm calculates for $T_i$ four alternate costs (we refer to traffic from $i$ towards its parent as outgoing, and from its parent as incoming), for the following possible scenarios:

$Cdo_i$ for the case when there is no copy located in $T_i$ ($i \neq r$) and the optimal cost is of incoming distribution and outgoing update traffic only.

$Cdo_i$ - **C**ost of **D**istribution **O**nly traffic in $T_i$.

$Cuo_i$ for the case when there is at least one copy of the object located in $T_i$ and no copy is located outside $T_i$ (and therefore only incoming update traffic flows in edge $i$).
$Cuo_i$ - **C**ost of **U**pdate **O**nly traffic in $T_i$.

$Cuo_i^+$ for the case when there is at least one copy of the object located in $T_i$ and all the internal consumers demand is supplied from copies in $T_i$, but additional copies are located outside $T_i$ (and therefore both incoming and outgoing update traffic and maybe outgoing distribution traffic flows in edge $i$).
$Cuo_i^+$ - **C**ost of **U**pdate **O**nly traffic **P**lus in $T_i$.

$Cdu_i$ for the case when there is at least one copy of the object located in $T_i$ but not all the internal consumers demand is supplied from copies in $T_i$ (and therefore both incoming distribution and incoming and outgoing update traffic flows in edge $i$).
$Cdu_i$ - **C**ost of both **D**istribution and **U**pdate traffic in $T_i$.

For calculating the minimal cost, the algorithm calculates the alternate costs as follows:

$$Cdo_i \leftarrow \begin{cases} Td \cdot Ucd_i + Tu_i^{out} \cdot Ucu_i + sum3, & \text{if } i \neq r \\ \infty, & \text{if } i = r \end{cases}$$

$$Cuo_i \leftarrow \begin{cases} Tu_i^{in} \cdot Ucu_i + \min\{min1, min2, min3\}, & \text{if } i \neq r \\ \min\{min1, min2, min3\}, & \text{if } i = r \end{cases}$$

$$Cuo_i^+ \leftarrow \begin{cases} \left(Tu_i^{in} + Tu_i^{out}\right) \cdot Ucu_i + \min\{min1, min2\}, & \text{if } i \neq r \\ \infty, & \text{if } i = r \end{cases}$$

$$Cdu_i \leftarrow \begin{cases} Td \cdot Ucd_i + Tu_i^{in} \cdot Ucu_i + Tu_i^{out} \cdot Ucu_i + sum1, & \text{if } i \neq r \& Ch_i \neq \emptyset \\ \infty, & \text{if } i = r \| Ch_i = \emptyset \end{cases}$$

where

$$\begin{aligned} sum1 &= \sum_{c \in Ch_i} \min\left\{Cdo_c, Cuo_c^+, Cdu_c\right\} \\ sum2 &= \sum_{k \in Ch_i, k \neq c} \min\left\{Cdo_k, Cuo_k^+, Cdu_k\right\} \\ sum3 &= \sum_{c \in Ch_i} Cdo_c \\ min1 &= Sc_i + sum1 \\ min2 &= \min_{c \in Ch_i} \left\{Td \cdot Ucd_c + Cuo_c^+ + sum2\right\} \\ min3 &= \min_{c \in Ch_i} \left\{Td \cdot Ucd_c + Cuo_c + \sum_{k \in Ch_i, k \neq c} Cdo_k\right\} \end{aligned}$$

note: $sum1, sum2, sum3$ equal 0 and $min2, min3$ equal $\infty$ if vertex $i$ is a leaf ($Ch_i$=$\emptyset$).

**The optimal total cost is $Cuo_r$.**

## *B. Backtracking for content allocation*

While calculating each alternate cost for each vertex $i$, the algorithm remembers for each alternate cost if a copy needs to be stored at $i$ and/or the states of each child $c$ which has at least one copy stored in its subtree. This is important for the backtracking phase, and allows accurate placement of the copies while backtracking.

The backtracking phase is recursive, starts at the root and ends at the leaves of $T$ (can stop earlier if no child has a copy in $V_c$). For each vertex the algorithm determines if a copy should be stored and if it is necessary to keep advancing towards the leaves of $T$. The algorithm uses the backtrack information that was saved along with the alternate optimal cost.

Appendix I-A presents the pseudo code of the algorithm. Backtrack details are shown there. An example of running the algorithm is shown in appendix II.

## C. *Proof of Optimality*

The proof is based on induction. Lemma 2 is the induction base.

*Lemma 2:* For all scenarios, the algorithm optimally allocates the object in $T_i$, when $i$ is a leaf of $T$

*Proof:* According to the definition of the new optimization problem, either one of the following possible scenarios holds.

1) Vertex $i$ is served from a vertex outside $T_i$ (through edge $i$), and no copy is stored in $T_i$. The optimal cost is constructed only from outgoing update and incoming distribution traffic will traverse through edge $i$. ($Cdo_i$).
2) Vertex $i$ is served from a vertex outside $T_i$ (through edge $i$), and at least one copy is stored in $T_i$. This is impossible for a leaf vertex, since if a copy is stored in $T_i$ it must be stored in $i$, and vertex $i$ will then be served from the copy stored in $i$ - contradiction. ($Cdu_i = \infty$).
3) Vertex $i$ is served from within $T_i$ and no copy of the object is allocated outside $T_i$. Vertex $i$ is the only vertex in $T_i$ and therefore there must be a copy allocated in vertex $i$. Since there are no copies allocated outside $T_i$ there will be no outgoing update traffic through edge $i$. The optimal cost is constructed from the storage cost and the cost of only the incoming update traffic that will traverse through edge $i$. ($Cuo_i$).
4) Vertex $i$ is served from within $T_i$ and at least one copy of the object is allocated outside $T_i$. Vertex $i$ is the only vertex in $T_i$ and therefore there must be a copy allocated in vertex $i$. Since there is a copy allocated in $T_i$ and at least one copy allocated outside $T_i$ there will be both incoming and outgoing update traffic through edge $i$. The optimal cost is constructed from the storage cost and the cost of both incoming and outgoing update traffic that will traverse through edge $i$. ($Cuo_i^+$).

Lemma 3 constructs the induction step for the recursive proof of optimality.

*Lemma 3:* Assume that the algorithm optimally allocates the object to servers in every subtree rooted at vertex $c$ which is a child of $i$ ($T_c$, $c \in Ch_i$) for all the scenarios, then the algorithm optimally allocates the object in $T_i$ for all the scenarios.

*Proof:* According to the definition of the new optimization problem, either one of the following possible scenarios holds.

1) Vertex $i$ is served from a vertex outside $T_i$ (through edge $i$), and no copy is stored in $T_i$. This implies that no copy is allocated in $T_c$, $c \in Ch_i$. According to the induction, the problem was solved for $T_c$, $c \in Ch_i$, and the additional cost is the incoming distribution traffic through edge $i$ and the outgoing update traffic added at vertex $i$ and going through edge $i$ ($Cdo_i$).
2) Vertex $i$ is served from a vertex outside $T_i$ (through edge $i$), and at least one copy is stored in $T_i$. This implies that at least one copy is allocated in one or more subtrees $T_c$, $c \in Ch_i$. No copy is allocated for vertex $i$, since it's served from outside $T_i$. If there are vertices $c \in Ch_i$ which are served through edge $c$, then according to Lemma 1 these vertices must be served through edge $i$. Since vertex $i$ is served through edge $i$ there must be at least one copy stored outside $T_i$, and since at least one copy is allocated within $T_i$, then there must be additional incoming distribution traffic and both incoming and outgoing update traffic going through edge $i$. The optimal cost is constructed from the additional traffic through edge $i$, and the sum of optimal costs calculated for each $T_c$ (The minimum of the following legal scenarios for each $c \in Ch_i$: $Cdo_c$, $Cdu_c$ or $Cuo_c^+$). ($Cdu_i$).
3) Vertex $i$ is served from within $T_i$ and no copy of the object is allocated outside $T_i$. Since there are no copies allocated outside $T_i$ there will be no outgoing update traffic through edge $i$. Since $i$ is served from within $T_i$ there must be at least one copy allocated within $T_i$ and there will be incoming update traffic. There are three possibilities for object allocation in this scenario:
   a) Vertex $i$ stores a copy of the object (and served from that copy). The vertices in the subtrees $T_c$, $c \in Ch_i$ may be served either from vertex $i$ or from internal copies. (The minimum of the following legal scenarios for each $c \in Ch_i$: $Cdo_c$, $Cdu_c$ or $Cuo_c^+$). ($min1$).
   b) Vertex $i$ is served from $T_c$, $c \in Ch_i$ which stores a copy of the object, but other vertices in the subtrees $T_k$, $k \in Ch_i, k \neq c$ may be served either through vertex $i$ (through edge $c$ which supplies outgoing distribution traffic from $T_c$) or from internal copies. (The minimum of the following legal scenarios for each $k \in Ch_i$: $Cdo_k$, $Cdu_k$ or $Cuo_k^+$). ($min2$).
   c) Vertex $i$ is served from $T_c$, $c \in Ch_i$ which stores a copy of the object, and no other vertices in the subtrees $T_k$, $k \in Ch_i, k \neq c$ may store a copy of the object. All these subtrees must be served through vertex $i$ (through

edge $c$ which supplies outgoing distribution traffic from $T_c$). ($min3$).

The optimal cost is constructed from the incoming update traffic that will traverse through edge $i$ and from the minimum between $min1, min2, min3$. ($Cuo_i$).

4) Vertex $i$ is served from within $T_i$ and at least one copy of the object is allocated outside $T_i$. Since there is a copy allocated in $T_i$ and at least one copy allocated outside $T_i$ there will be both incoming and outgoing update traffic through edge $i$. There are two possibilities for object allocation in this scenario:
   a) Vertex $i$ stores a copy of the object (and served from that object). The vertices in the subtrees $T_c$, $c{\in}Ch_i$ may be served either from vertex $i$ or from internal copies. (The minimum of the following legal scenarios for each $c{\in}Ch_i$: $Cdo_c$, $Cdu_c$ or $Cuo_c^+$). ($min1$).
   b) Vertex $i$ is served from $T_c$, $c{\in}Ch_i$ which stores a copy of the object, but other vertices in the subtrees $T_k$, $k{\in}Ch_i{,}k{\neq}c$ may be served either through vertex $i$ (through edge $c$ which supplies outgoing distribution traffic from $T_c$) or from internal copies. (The minimum of the following legal scenarios for each $k{\in}Ch_i$: $Cdo_k$, $Cdu_k$ or $Cuo_k^+$). ($min2$).

The optimal cost is constructed from the incoming and outgoing update traffic that will traverse through edge $i$ and from the minimum between $min1, min2$. ($Cuo_i^+$).

*Theorem 1:* When the algorithm ends, $Cuo_r$ holds the optimal allocation cost and the allocation of copies is optimal.

*Proof:* The proof is conducted by the induction where Lemma 2 is the base and Lemma 3 is the step. In addition, the costs $Cdo_r$, $Cdu_r$ and $Cuo_r^+$ are illegal since there can't be copies allocated outside $T_r \equiv T$.

## D. Complexity

For each vertex in the tree $i{\in}V$ the algorithm calculates up to 4 alternate costs. Each cost calculation requires $O(|Ch_i|)$. Therefore the total complexity for vertex $i$ is $O(4 \cdot |Ch_i|)$. The total complexity for the entire tree is: $\sum_{i \in V} 4 \cdot |Ch_i|$. $|V|{=}N$ and the total number of children in the tree is $N{-}1$ (only the root $r$ is not a child). Therefore:

$$\sum_{i \in V} 4 \cdot |Ch_i| = 4 \cdot \sum_{i \in V} |Ch_i| = 4 \cdot (N-1) < 4N$$

**The algorithm computational complexity is $O(N)$.**

## IV. The optimal algorithm for unicast distribution

The optimization problem is to minimize the total cost (storage and traffic):

$$\sum_{i \in \Phi} Sc_i + \sum_{i \in V} Tu_i \cdot ( \sum_{j \in Umt_{i,\Phi}} Ucu_j) + \sum_{i \in V} Td_i \cdot \min_{j \in \Phi} Dd_{i,j} \tag{2}$$

The algorithm calculates the optimal object allocation cost as well as the set of servers that will store the object.

Before we turn to the algorithm we first prove basic properties of the optimal object allocation solution.

*Lemma 4:* In the optimal allocation, each vertex $i$ may only belong to one distribution unicast tree.

*Proof:* Suppose a vertex $i$ belongs to more than one distribution unicast trees, then by disconnecting it from the other trees and keeping it connected to only one distribution unicast tree we reduce the distribution traffic in contradiction to the optimality of the cost. Moreover, vertex $i$ can only be served from a single server.

*Lemma 5:* In the optimal allocation, if vertex $c$ is served through vertex $i$ then $c$ is served from the same server as $i$ does.

*Proof:* A direct result of Lemma 4. If $c$ is served from a different server, then $i$ belongs to two distribution unicast trees - a contradiction.

The above properties lead us to the conclusion, that the optimal allocation is composed of a subgraph of $T$ which is a forest of distribution subtrees. Each subtree is rooted at a vertex (The distribution subtree rooted at vertex $i$ may not be $T_i$) where a copy of the object is located and its leaves are vertices with distribution demands but no copy of the object is stored. An edge in $T$ can be part of at most one distribution tree. Moreover, in the optimal solution if a consumer at vertex $c$ is served through vertex $i$ which is an ancestor of $c$ in $T$, it will be served from the same server as $i$. Figure 3. demonstrates the structure of a forest with different possible cases of the vertices and edges in $T$.

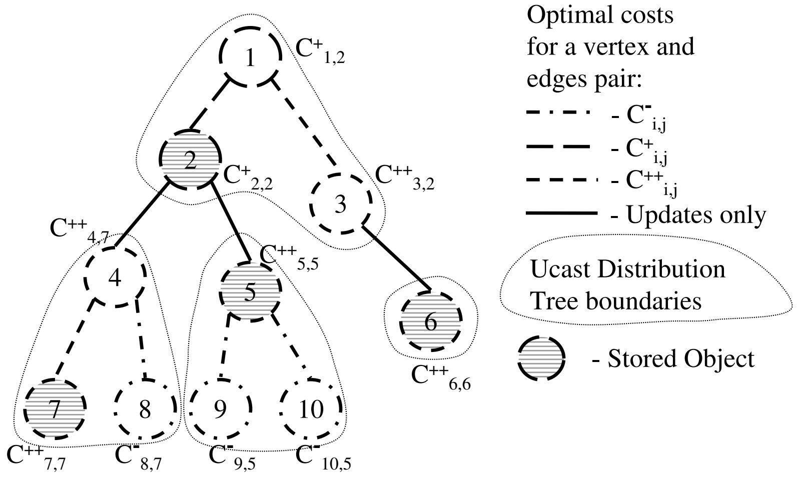

Fig. 3. Example of the unicast distribution forest

As suggested in [14] and [15], the optimization problem can be solved by defining a new set of optimization problems and combining their solutions to solve the original optimization problem. A subset of the problem at a given vertex $i$ is a tree $T_{i,j}$ that combines $T_i$ and the string that connects vertex $i$ to $j$ ($P_{j,i}$), in case $j \notin V_i$. The optimal solution of the subset of the problem assumes that vertex $j$ stores a copy of the object in the optimal solution of the original optimization problem. In case $j \notin V_i$, the storage cost of the copy at vertex $j$ is not included in the optimal cost of the tree $T_{i,j}$, but the distribution traffic cost along $P_{j,i}$ is included.

For the multicast update traffic there will be an additional one or more multicast trees for updating the servers which store copies of the object.

Unlike previous works, for each vertex pair $i, j$ the algorithm calculates for $T_{i,j}$ three alternate costs (we refer to traffic from $i$ towards its parent as outgoing, and from its parent as incoming):

$C^-$ for the case when there is no copy located in $T_i$ ($i \neq r$) and the alternate cost is of distribution traffic only.
$C^+$ for the case when at least one copy is located inside and no copies are located outside $T_i$.
$C^{++}$ for the case when copies are located both inside and outside $T_i$.

For calculating the minimal cost, the algorithm calculates the alternate costs as follows:

$$C^-_{i,j} \leftarrow \begin{cases} \infty, & \text{if } j \in V_i \\ Td_i \cdot Dd_{i,j} + Tu_i^{out} \cdot Ucu_i + sum1\,, & \text{if } j \notin V_i \end{cases}$$

$$C^+_{i,j} \leftarrow \begin{cases} Td_i \cdot Dd_{i,j} + Tu_i^{in} \cdot Ucu_i + \min\{sum2, sum3\}\,, & \text{if } j \in V_k, k \in Ch_i \\ Tu_i^{in} \cdot Ucu_i + Sc_i + sum4, & \text{if } j = i \\ \infty, & \text{if } j \notin V_i \end{cases}$$

$$C^{++}_{i,j} \leftarrow \begin{cases} Td_i \cdot Dd_{i,j} + Tu_i^{in} \cdot Ucu_i + Tu_i^{out} \cdot Ucu_i + sum3\,, & \text{if } j \in V_k, k \in Ch_i \\ Tu_i^{in} \cdot Ucu_i + Tu_i^{out} \cdot Ucu_i + Sc_i + sum4\,, & \text{if } j = i \\ \min\left\{\min\limits_{l \in V_i} C^{++(*)}_{i,l}, Td_i \cdot Dd_{i,j} + Tu_i^{in} \cdot Ucu_i + Tu_i^{out} \cdot Ucu_i + sum4\right\}, & \text{if } j \notin V_i \end{cases}$$

($*$) The minimum value can be calculated once during the calculation of each $C^{++}_{i,j}$, $j \in V_i$, given that these values are calculated prior to calculating any $C^{++}_{i,j}$, $j \notin V_i$

where

$$sum1 = \sum_{k \in Ch_i} C^-_{k,j}$$

$$sum2 = C^+_{k,j} + \sum_{l \in Ch_i, l \neq k} C^-_{l,j}$$

$$sum3 = C^{++}_{k,j} + \sum_{l \in Ch_i, l \neq k} \min\left\{C^{-}_{l,j}, C^{++}_{l,j}\right\}$$

$$sum4 = \sum_{k \in Ch_i} \min\left\{C^{-}_{k,j}, C^{++}_{k,j}\right\}$$

note: $sum1, sum2, sum3$ and $sum4$ equal 0 if vertex $i$ is a leaf ($Ch_i{=}\emptyset$).

**The optimal total cost is** $min_{j \in V} C^{+}_{r,j}$**.**

## A. Backtracking for content allocation

While calculating each alternate cost for each pair $i, j$, the algorithm remembers for each alternate cost if a copy needs to be stored at $i$ and/or the states of each child $k$ which has at least one copy stored in its subtree (donates $C^{++}$ or $C^{+}$). This is important for the backtracking phase, and allows accurate placement of the copies while backtracking.

The backtracking phase is recursive, starts at the root and ends at the leaves of $T$ (can stop earlier of no child has a copy in $V_c$). For each vertex the algorithm determines if a copy should be stored and if it is necessary to keep advancing towards the leaves of $T$. The algorithm uses the backtrack information that was saved along with the alternate optimal cost.

Appendix I-B presents the pseudo code of the algorithm. Backtrack details are also shown there.

## B. Proof of Optimality

The proof is based on induction. Lemma 6 is the induction base.

*Lemma 6:* For all scenarios, and for all vertices $j{\in}V$ the algorithm optimally allocates the object in $T_{i,j}$, when $i$ is a leaf of $T$

*Proof:* According to the definition of the new optimization problem, either one of the following possible scenarios holds.

1) $j{=}i$ (no string is connected), the algorithm allocates the object at vertex $i$. This is the only possibility, which is optimal ($C^{-}_{i,i}$ can't exist, set to $\infty$). The optimal cost is constructed from the storage cost and the cost of update traffic (incoming only for $C^{+}_{i,i}$, both incoming and outgoing for $C^{++}_{i,i}$).
2) $j{\neq}i$ ($j{\notin}V_i$). A string $P_{j,i}$ is connected to $T_i$. According to the definition of the problem, there's a copy of the object located at vertex $j$ - therefore $C^{+}_{i,j}$ is illegal (set to $\infty$) and there must be outgoing update traffic through edge $i$. If the distribution cost from $j$ to $i$ is less than the storage cost + incoming update cost, the optimal decision is to be served from $j$ (Represented by $C^{-}_{i,j}$. In this case the algorithm also sets $C^{++}_{i,j}$ to $C^{-}_{i,j}$ + the incoming update cost, which insures that if a copy of the object is not stored at $T_i$, $C^{-}_{i,j}{\leq}C^{++}_{i,j}$), otherwise the decision is to store a copy at vertex $i$ (Represented by $C^{++}_{i,j}$ which is set to $C^{++}_{i,i}$, in this case $C^{++}_{i,j}{\leq}C^{-}_{i,j}$).

Lemma 7 constructs the induction step for the recursive proof of optimality.

*Lemma 7:* Assume that the algorithm optimally allocates the object to servers in every subtree rooted at vertex $c$ which is a child of $i$ ($T_c$, $c{\in}Ch_i$) for all scenarios and for all vertices $j{\in}V$, then the algorithm optimally allocates the object in $T_i$ for all the scenarios and for all vertices $j{\in}V$.

*Proof:* According to the definition of the new optimization problem, either one of the following possible scenarios holds.

1) $j{=}i$ (no string is connected), the algorithm allocates the object at vertex $i$. There is a copy located at $T_i$ therefore $C^{-}_{i,i}$ can't exist (set to $\infty$), and there must be incoming update traffic through edge $i$. The vertices in each subtree $T_k$, $k{\in}Ch_i$ may be served either from vertex $i$ or from copies located internally in the subtree. (The minimum of the following legal scenarios for each $k{\in}Ch_i$: $C^{-}_{k,i}$, $C^{++}_{k,i} \Rightarrow sum4$). The optimal cost is constructed from the storage cost at $i$, the optimal costs calculated in the children ($sum4$) and the cost of update traffic (incoming only for $C^{+}_{i,i}$, both incoming and outgoing for $C^{++}_{i,i}$).
2) $j{\in}V_k$, $k{\in}Ch_i$ (no string is connected), the algorithm allocates the object at vertex $j$. There is a copy located at $T_i$ therefore $C^{-}_{i,j}$ can't exist (set to $\infty$), and there must be incoming update traffic through edge $i$.
   For the scenario where no copy of the object is allocated outside $T_i$ ($C^{+}_{i,j}$), two possibilities hold:

   a) There are copies allocated only within $T_k$ (at least at vertex $j$). In this case there are no copies allocated in $T_l$, $l{\in}Ch_i$, $l{\neq}k$. ($sum2$)
   b) There are copies allocated within $T_k$ (at least at vertex $j$) and also within at least another $T_l$, $l{\in}Ch_i$, $l{\neq}k$. ($sum3$)

   For the scenario where at least one copy of the object is allocated outside $T_i$ ($C^{++}_{i,j}$), only one possibility holds: There are copies allocated within $T_k$ (at least at vertex $j$) and maybe within at least another $T_l$, $l{\in}Ch_i$, $l{\neq}k$. ($sum3$). The optimal cost is constructed from the distribution cost from $j$ to $i$, the optimal costs calculated in the children ($sum2$ or $sum3$) and the cost of update traffic (incoming only for $C^{+}_{i,j}$, both incoming and outgoing for $C^{++}_{i,j}$).

3) $j{\notin}V_i$. A string $P_{j,i}$ is connected to $T_i$. According to the definition of the problem, there's a copy located at vertex $j$ - therefore $C^{+}_{i,j}$ is illegal (set to $\infty$) and there must be outgoing update traffic through edge $i$. There are three possibilities for object allocation in this scenario:
   a) Vertex $i$ is served from $j$, and no copy of the object is located within $T_i$. According to Lemma 5 all vertices in $T_i$ are served from $j$. The optimal cost is the distribution cost from $j$ to $i$ plus the optimal $C^{-}$ costs calculated for the children of $i$ ($sum1$) plus the outgoing update traffic (Represented by $C^{-}_{i,j}$).
   b) Vertex $i$ (and according to Lemma 5 all vertices in $T_i$) is served from within $T_i$. In this case the optimal allocation is represented by the minimal cost $\min_{l\in V_i} C^{++}_{i,l}$ ($C^{++}$ since there is a copy located outside $T_i$). (One possibility for $C^{++}_{i,j}$).
   c) Vertex $i$ is served from Vertex $j$, and at least one copy of the object is located within $T_i$. The vertices in each subtree $T_k$, $k{\in}Ch_i$ may be served either from vertex $j$ or from copies located internally in the subtree. (The minimum of the following legal scenarios for each $k{\in}Ch_i$: $C^{-}_{k,j}$, $C^{++}_{k,j} \Rightarrow sum4$). The optimal cost is constructed from the distribution cost from $i$ to $j$, the optimal costs calculated in the children ($sum4$) and the cost of outgoing update traffic (Other possibility for $C^{++}_{i,j}$).

*Theorem 2:* When the algorithm ends, $min_{j\in V} C^{+}_{r,j}$ holds the optimal allocation cost and the allocation of copies is optimal

*Proof:* The proof is conducted by the induction where Lemma 6 is the base and Lemma 7 is the step. For each $j{\in}V$, $C^{+}_{r,j}$ represents an optimal allocation of the objects where $r$ is served from $j$. The minimal $C^{+}_{r,j}$ is the optimal cost of the original optimization problem. In addition, the costs $C^{++}_{r,j}$, $C^{-}_{r,j}$ are illegal since there can't be copies allocated outside $T_r \equiv T$.

## C. Complexity

For each vertex in the tree $i{\in}V$ the algorithm calculates up to $3{\cdot}N$ alternate costs. Each cost calculation requires $O(|Ch_i|)$. Therefore the total complexity for vertex $i$ is $O((3 \cdot N) \cdot (|Ch_i|))$. The total complexity for the entire tree is: $\sum_{i\in V}(3 \cdot N) \cdot |Ch_i|$. $|V|{=}N$ and the total number of children in the tree is $N{-}1$ (only the root $r$ is not a child). Therefore:

$$\sum_{i\in V}(3 \cdot N) \cdot |Ch_i| = 3 \cdot N \cdot \sum_{i\in V} |Ch_i| = 3 \cdot N \cdot (N-1) < 3 \cdot N^2$$

**The algorithm computational complexity is $O(N^2)$.**

## V. Conclusion and Future work

In this work, we addressed overlay networks with update from multiple media sources and content distribution to users that employ native IP multicast based update for unicast or multicast content distribution.

We developed optimal content allocation algorithms for tree networks the with computational complexity of $O(N)$ and $O(N^2)$ for multicast and unicast distribution respectively. We showed that adding update traffic to the original unicast distribution problem require new algorithmic observations and techniques but has a minor effect on the computational complexity.

The presented algorithms can easily be transformed into distributed algorithms. The transformation is out of the scope of this paper but one can think of the way to turn the cost calculations described in Appendix I into calculations performed in each node, and the data produced by the calculations as the data that has to be passed from children to their parents and vice versa.

Our current work focus on the generalization of the problem to general graphs. Most of the related problems in general graphs are NP-hard. An interesting scenario is when all vertices of the general graph are part of the distribution forest (i.e. - consumers are connected to every vertex), it is easy to show that the distribution forest (that connects the servers to the consumers) is a subgraph of the minimal spanning tree, while each multicast update tree is a Steiner tree (which is NP-complete on general graphs). Using known approximations for the construction of the Steiner tree we can limit the total update cost (as well as the total cost) to a ratio of 2 from the optimal.

An additional work may be in the direction of adjusting the above static algorithm into a dynamic environment where the demands and various costs may change over time. Since the algorithm is most efficient, an easy adjustment may be to calculate the costs periodically and run the algorithm from scratch. Please note that an incremental change in the costs may not incur an incremental change in the result of the algorithm, there may be a need to restart all the calculations due to one cost change.

## APPENDIX I
## PSEUDO CODE OF THE ALGORITHMS

In this section we assume that the vertices are ordered by breadth first ordering. Vertex $1$ is the root and $n$ must be a leaf.
We also assume the $\infty$ is the maximal number that exists in the computer.
Variables starting with $BT$ are used for the backtrack process, and store a vertex number or the cost/vertex data.

### *A. Centralized multicast distribution algorithm*

The algorithm is performed in two phases. The first one is for calculating the optimal cost and the backtrack info for later.

**Cost calculation phase**

```
for $i = n, n-1, n-2, ..., 2, 1$ do
    if $Ch_i = \emptyset$ then /* a leaf */
        $Cdo_i \leftarrow Td \cdot Ucd_i + Tu_i^{out} \cdot Ucu_i$
        $Cuo_i \leftarrow Tu_i^{in} \cdot Ucu_i + Sc_i$ ; $BT\text{-}Cuo_i \leftarrow (i, "local")$
        $Cuo_i^{+} \leftarrow \left(Tu_i^{in} + Tu_i^{out}\right) \cdot Ucu_i + Sc_i$ ; $BT\text{-}Cuo_i^{+} \leftarrow (i, "local")$
        $Cdu_i \leftarrow \infty$ ; $BT\text{-}Cdu_i \leftarrow \emptyset$
    else
        /* calculate sum1, sum3 (sum1 derives sum2!) */
        $sum1 \leftarrow 0$ ; $BT\text{-}sum1 \leftarrow \emptyset$
        $sum3 \leftarrow 0$
        foreach $c$ in $Ch_i$ do
            $sum3 \leftarrow sum3 + Cdo_c$
            $Cmin_c \leftarrow Cdo_c$ ; $Cmin_{type} \leftarrow "local"$
            if ($Cdo_c <= Cuo_c^{+} \& Cdo_c <= Cdu_c$) then
                $sum1 \leftarrow sum1 + Cdo_c$
            else if ($Cdu_c <= Cuo_c^{+}$) then
                $sum1 \leftarrow sum1 + Cdu_c$ ; $BT\text{-}sum1 \leftarrow BT\text{-}sum1 \cup (c, "du")$
                $Cmin_c \leftarrow Cdu_c$ ; $Cmin_{type} \leftarrow "du"$
            else
                $sum1 \leftarrow sum1 + Cuo_c^{+}$ ; $BT\text{-}sum1 \leftarrow BT\text{-}sum1 \cup (c, "uo^{+}")$
                $Cmin_c \leftarrow Cuo_c^{+}$ ; $Cmin_{type} \leftarrow "uo^{+}"$
            end if
        end do
        /* calculate min1, min2, min3 */
        $min1 \leftarrow Sc_i + sum1$ ; $BT\text{-}min1 \leftarrow BT\text{-}sum1 \cup (i, "local")$
        $min2 \leftarrow \infty$ ; $BT\text{-}min2 \leftarrow \emptyset$
        $min3 \leftarrow \infty$ ; $BT\text{-}min3 \leftarrow \emptyset$
        foreach $c$ in $Ch_i$ do
            $tmp \leftarrow Td \cdot Ucd_c + Cuo_c^+ + sum1 - Cmin_c$
            if ($tmp < min2$) then
                $min2 \leftarrow tmp$ ; $BT\text{-}min2 \leftarrow (c, "uo^+") \cup \Big(BT\text{-}sum1 \setminus (c, Cmin_{type})\Big)$
            end if
            $tmp \leftarrow Td \cdot Ucd_c + Cuo_c + sum3 - Cdo_c$
            if ($tmp < min3$) then
                $min3 \leftarrow tmp$ ; $BT\text{-}min3 \leftarrow (c, "uo")$
            end if
        end do
        /* calculate optimal $Cuo_i$ cost and BT data */
        if $min1 <= min2 \& min1 <= min3$ then
            $Cuo_i \leftarrow Tu_i^{in} \cdot Ucu_i + min1$ ; $BT\text{-}Cuo_i \leftarrow BT\text{-}min1$
        else if $min2 <= min3$ then
            $Cuo_i \leftarrow Tu_i^{in} \cdot Ucu_i + min2$ ; $BT\text{-}Cuo_i \leftarrow BT\text{-}min2$
        else
            $Cuo_i \leftarrow Tu_i^{in} \cdot Ucu_i + min3$ ; $BT\text{-}Cuo_i \leftarrow BT\text{-}min3$
        end if
        if $i \neq 1$ then /* not root */
            $Cdo_i \leftarrow Td \cdot Ucd_i + Tu_i^{out} \cdot Ucu_i + sum3$
            if $min1 <= min2$ then
                $Cuo_i^+ \leftarrow \Big(Tu_i^{in} + Tu_i^{out}\Big) \cdot Ucu_i + min1$ ; $BT\text{-}Cuo_i^+ \leftarrow BT\text{-}min1$
            else
                $Cuo_i^+ \leftarrow \Big(Tu_i^{in} + Tu_i^{out}\Big) \cdot Ucu_i + min2$ ; $BT\text{-}Cuo_i^+ \leftarrow BT\text{-}min2$
            end if
            $Cdu_i \leftarrow Td \cdot Ucd_i + \Big(Tu_i^{in} + Tu_i^{out}\Big) \cdot Ucu_i + sum1$ ; $BT\text{-}Cdu_i \leftarrow BT\text{-}sum1$
        end if
    end if
end do
```

The optimal cost is $Cuo_1$.

**Backtrack phase**

The backtrack copies allocation phase is recursive and can easily be described using a recursive function. The recursion starts by calling **allocate**$(1, "uo")$.

```
proc **allocate** $(i, type)$ {
    if $type = "uo"$ then
        foreach $(c, ctype)$ in $BT\text{-}Cuo_i$ do ; call **allocate**$(c, ctype)$ ; end do
    else if $type = "uo^+"$ then
        foreach $(c, ctype)$ in $BT\text{-}Cuo_i^+$ do ; call **allocate**$(c, ctype)$ ; end do
    else if $type = "du"$ then
        foreach $(c, ctype)$ in $BT\text{-}Cdu_i$ do ; call **allocate**$(c, ctype)$ ; end do
    else if $type = "local"$ then
        allocate a copy at $i$
    end if
    return
}
```

## *B. Centralized unicast distribution algorithm*

The algorithm is performed in two phases. The first one is for calculating the optimal cost and the backtrack info for later.

**Cost calculation phase**

```
for $i = n, n-1, n-2, ..., 2, 1$ do
    /* calculate costs for $i, i$ */
    $sum4 \leftarrow 0$ ; $BT\text{-}sum4 \leftarrow \emptyset$
    foreach $k$ in $Ch_i$ do
        if $C^{-}_{k,i} <= C^{++}_{k,i}$ then
            $sum4 \leftarrow sum4 + C^{-}_{k,i}$
        else
            $sum4 \leftarrow sum4 + C^{++}_{k,i}$ ; $BT\text{-}sum4 \leftarrow BT\text{-}sum4 \cup (k, i, "{++}")$
        end if
    end do
    $C^{-}_{i,i} \leftarrow \infty$
    $C^{+}_{i,i} \leftarrow Tu^{in}_i \cdot Ucu_i + Sc_i + sum4$ ; $BT\text{-}C^{+}_{i,i} \leftarrow BT\text{-}sum4 \cup (i, i, "local")$
    $C^{++}_{i,i} \leftarrow C^{+}_{i,i} + Tu^{out}_i \cdot Ucu_i$ ; $BT\text{-}C^{++}_{i,i} \leftarrow BT\text{-}sum4 \cup (i, i, "local")$
    $j_{min} \leftarrow i$
    /* calculate costs for $i, j$ where $j \in V_i$ */
    foreach $k$ in $Ch_i$ do
        foreach $j$ in $V_k$ do
            $sum2 \leftarrow C^{+}_{k,j}$ ; $BT\text{-}sum2 \leftarrow (k, j, "{+}")$
            $sum3 \leftarrow C^{++}_{k,j}$ ; $BT\text{-}sum3 \leftarrow (k, j, "{++}")$
            foreach $l$ in $Ch_i \setminus k$ do
                $sum2 \leftarrow sum2 + C^{-}_{l,j}$
                if $C^{-}_{l,j} <= C^{++}_{l,j}$ then
                    $sum3 \leftarrow C^{-}_{l,j}$
                else
                    $sum3 \leftarrow C^{++}_{l,j}$ ; $BT\text{-}sum3 \leftarrow BT\text{-}sum2 \cup (l, j, "{++}")$
                end if
            end do
            $C^{-}_{i,j} \leftarrow \infty$
            $C^{+}_{i,j} \leftarrow Td_i \cdot Dd_{i,j} + Tu^{in}_i \cdot Ucu_i$
            $C^{++}_{i,j} \leftarrow C^{+}_{i,j} + Tu^{out}_i \cdot Ucu_i + sum3$ ; $BT\text{-}C^{++}_{i,j} \leftarrow BT\text{-}sum3$
            if $sum2 <= sum3$ then
                $C^{+}_{i,j} \leftarrow C^{+}_{i,j} + sum2$ ; $BT\text{-}C^{+}_{i,j} \leftarrow BT\text{-}sum2$
            else
                $C^{+}_{i,j} \leftarrow C^{+}_{i,j} + sum3$ ; $BT\text{-}C^{+}_{i,j} \leftarrow BT\text{-}sum3$
            end if
            /* update $\min_{l \in V_i} C^{++}_{i,l}$ */
            if $C^{++}_{i,j} < C^{++}_{i,j_{min}}$ then
                $C^{++}_{i,min} \leftarrow C^{++}_{i,j}$
            end if
        end do
    end do
    /* calculate costs for $i, j$ where $j \notin V_i$ */
    foreach $j$ in $V \setminus V_i$ do
        $sum1 \leftarrow 0$
        foreach $k$ in $Ch_i$ do
```

```
            $sum1 \leftarrow C^{-}_{k,j}$
        end do
        $C^{-}_{i,j} \leftarrow Td_i \cdot Dd_{i,j} + Tu^{out}_i \cdot Ucu_i + sum1$
        $C^{+}_{i,j} \leftarrow \infty$
        $sum4 \leftarrow 0$ ; $BT\text{-}sum4 \leftarrow \emptyset$
        foreach $k$ in $Ch_i$ do
            if $C^{-}_{k,j} <= C^{++}_{k,j}$ then
                $sum4 \leftarrow sum4 + C^{-}_{k,i}$
            else
                $sum4 \leftarrow sum4 + C^{++}_{k,i}$ ; $BT\text{-}sum4 \leftarrow BT\text{-}sum4 \cup (k, j, "{++}")$
            end if
        end do
        $C^{++}_{i,j} \leftarrow Td_i \cdot Dd_{i,j} + Tu^{in}_i \cdot Ucu_i + Tu^{out}_i \cdot Ucu_i + sum4$ ; $BT\text{-}C^{++}_{i,j} \leftarrow BT\text{-}sum4$
        if $C^{++}_{i,j_{min}} < C^{++}_{i,j}$ then
            $C^{++}_{i,j} \leftarrow C^{++}_{i,j_{min}}$ ; $BT\text{-}C^{++}_{i,j} \leftarrow BT\text{-}C^{++}_{i,j_{min}}$
        end if
    end do
end do
/* find the optimal cost */
$j_{min} \leftarrow 1$
for $j = n, n-1, n-2, ..., 2$ do
    if $C^{+}_{1,j} < C^{+}_{1,j_{min}}$ then
        $j_{min} \leftarrow j$
    end if
end do
```

The optimal cost is $C^{+}_{1,j_{min}}$.

**Backtrack phase**
The backtrack copies allocation phase is recursive and can easily be described using a recursive function.
The recursion starts by calling **allocate**$(1, j_{min}, "{+}")$.

```
proc **allocate** $(i, j, type)$ {
    if $type = "{+}"$ then
        foreach $(k, l, ntype)$ in $BT\text{-}C^{+}_{i,j}$ do
            call **allocate**$(k, l, ntype)$
        end do
    else if $type = "{++}"$ then
        foreach $(k, l, ntype)$ in $BT\text{-}C^{++}_{i,j}$ do ; call **allocate**$(k, l, ntype)$ ; end do
    else if $type = "local"$ then
        allocate a copy at $i$
    end if
    return
}
```

## Appendix II
## Run example of the multicast distribution algorithm

An example of executing the algorithm on the following tree network: Figure 4. describes a tree network and the costs associated with its vertices and edges.

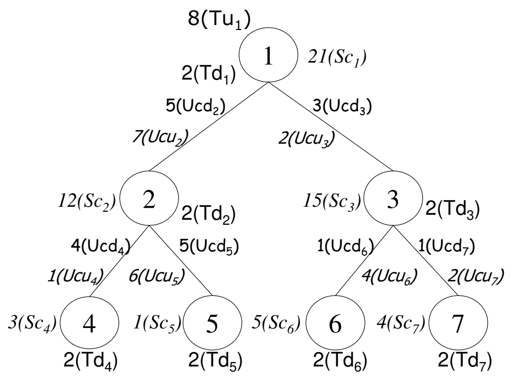

Fig. 4. A tree network for the run example

## A. Phase one

Calculating the optimal cost for each subtree and each scenario:

| $i$ | 7 | 6 | 5 | 4 | 3 | 2 | 1 |
|---|---|---|---|---|---|---|---|
| $Cdo_i$ | 2 | 2 | 10 | 8 | 10 | 28 | $\infty$ |
| $Cuo_i$ | 20 | 37 | 49 | 11 | 35 | 85 | 59 |
| $BT\text{-}Cuo_i$ | 7 | 6 | 5 | 4 | 3 | $4, uo$ | $1, loc$ |
| $Cuo_i^+$ | 20 | 37 | 49 | 11 | 35 | 85 | $\infty$ |
| $BT\text{-}Cuo_i^+$ | 7 | 6 | 5 | 4 | 3 | $4, uo^+$ | - |
| $Cdu_i$ | $\infty$ | $\infty$ | $\infty$ | $\infty$ | 26 | 84 | $\infty$ |
| $BT\text{-}Cdu_i$ | - | - | - | - | - | - | - |

## B. Phase two

The optimal cost is 59, and since the backtrack information of $Cuo_1$ ($BT\text{-}Cuo_r$) is only $1, local$ the copy is stored at vertex 1 and no more backtracks are needed.

## References

[1] Cisco, http://www.cisco.com/
[2] Akamai, http://www.akamai.com/
[3] Digital Fountain, http://www.digitalfountain.com/
[4] Talarian, SmartPGM, http://www.talarian.com/index2.shtml
[5] WebDAV, http://www.webdav.org/
[6] Scale8, http://www.scale8.com/
[7] PIM-SM, Protocol Independent Multicast-Sparse Mode, http://www.ietf.org/rfc/rfc2362.txt
[8] L. W. Dowdy and D. V. Foster. Comparative Models of the File Assignment Problem. ACM Computing Surveys, 14(2) pp. 287-313, 1982.
[9] O. Wolfson and A. Milo. The Multicast Policy and Its Relationship of Replicated Data Placement. ACM Transactions on Database Systems, 16(1) pp. 181-205, March 1991.
[10] 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, 12(6) pp. 628-637, June 2001.
[11] C. Krick, H. Räcke, and M. Westermann. Approximation Algorithms for Data Management in Networks. In Proceedings of the Symposium on Parallel Algorithms and Architecture, pages 237-246, July 2001.
[12] Pitu B. Mirchandani, Richard L. Francis, "Discrete Location Theory", John Wiley & Sons, Inc. 1990.
[13] Kolen A., "Solving covering problems and the uncapacited plant location problem on trees", European Journal of Operational Research, Vol. 12, pp. 266-278, 1983.
[14] Alain Billionnet, Marie-Christine Costa, "Solving the Uncapacited Plant Location Problem on Trees", Discrete Applied Mathematics, Vol. 49(1-3), pp. 51-59, 1994.
[15] I. Cidon, S. Kutten, and R. Sofer. "Optimal allocation of electronic content". In Proceedings of IEEE Infocom, Anchorage, AK, April 22-26, 2001.
[16] L. Qiu, V. N. Padmanabham, and G. M. Voelker. "On the placement of web server replicas". In Proc. 20th IEEE INFOCOM, 2001.
[17] Sugih Jamin, Cheng Jin, Anthony R. Kurc, Danny Raz and Yuval Shavitt, "Constrained Mirror Placement on the Internet", IEEE Infocom 2001.
[18] J. Kangasharju and J. Roberts and K. Ross, "Object Replication Strategies in Content Distribution Networks", In Proceedings of WCW'01: Web Caching and Content Distribution Workshop, Boston, MA, June 2001.