World Wide Web: Internet and Web Information Systems, 7, 315–336, 2004

Optimal Content Location in Multicast Based Overlay Networks with Content Updates [*]

OREN UNGER oren.unger@zoran.com
Department of Electrical Engineering, Technion and Zoran Microelectronics, Israel

ISRAEL CIDON cidon@ee.technion.ac.il
Department of Electrical Engineering, Technion – Israel Institute of Technology, Israel

Abstract

The architecture of overlay networks should support high-performance and high-scalability at low costs. This becomes more crucial when 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. 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. In this work, we suggest a multicast based architecture and address the optimal allocation and replication of dynamic objects that are both consumed and updated. Our model network includes consumers which are served using multicast or unicast transmissions and media sources (that may be also consumers) which update the objects 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.

Keywords: content distribution, multicast, overlay networks, tree networks

1. 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 [4], Akamai [1] and Digital Fountain [5] offer CDN services and overlay technologies. Recently, more collaborative models such as distributed storage and peer-to-

[*] Publisher's note: A partial preliminary version of this work can be found in the Proceedings of ICDCS 2003 Workshop.

peer computational models require both consumption and modification of the content by multiple, geographically distributed users [15,17].

An overlay network is a set of network edges that are connected through the general Internet Infrastructure. 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.

The popularity of multicast for distribution of the content is increasing with the introduction of real-time and multimedia applications that consume high bandwidth and are delivered to a large group of consumers. Although multicast is efficient for large groups, unicast can still be more effective for small groups, especially for a sparse distribution of consumers.

Our initial model is a tree graph that has a server located at each of its vertices. The vertices also include optional entries to local consumers and media sources. Each server is assigned with a storage cost and each edge is assigned with distribution and update communication costs. The distribution demand of the consumers and the update requirements of the media sources are given. The consumers are served from servers using multicast or unicast communication. The media sources update and modify the objects within the servers. The update traffic between a media source and the relevant servers is most efficiently conducted using multicast communication, since it can reduce significantly the overall update transport and the update latency.

Our goal is to find an optimal allocation, e.g., the set of servers which store an object, with the minimum overall (communication and storage) cost. The consumers are assigned to the servers in a way that each consumer is served by exactly one server for an object. It is clear that by changing the number of copies, we introduce a tradeoff between the storage/update costs that increase with the number of copies and the distribution cost that decreases with this number.

We present two optimal allocation algorithms for the tree network that have a computational complexity of $O(N)$ for the multicast distribution case and $O(N^2)$ for the unicast distribution case.

2. Related work

Application level multicast and overlay multicast protocols have been studied in recent years. Most of the works focus at the structure of the overlay topology (i.e., the way the multicast tree is constructed) for a single tree [7,16]. Our work assumes the overlay network topology is a tree, but instead of focusing on its construction, we focus on the way it should be partitioned to multiple regional multicast trees while optimizing the communication and storage cost.

The object allocation problem, also referred as the file allocation problem [6] has been studied extensively in the literature. While the multicast distribution model was never considered before, the combination of multicast update (write) and unicast distribution (read) without storage costs was considered in [18]. Kalpakis et al. [9] and Krick et al. [12] present a model of a network with unicast reads, multicast writes and storage costs. Article [9] presents a problem with additional constrains for a tree network and the algorithm they present for the similar case is less efficient than ours. [12] deals with general networks and suggests an optimal algorithm for tree networks which is also less efficient than our algorithm. These works do not solve the multicast reads multicast write problem. Moreover, the multicast update model presented in [9,12] is based on a single minimum spanning tree of the servers which store a copy (i.e., multicast is used only between the servers), while our model suggests an independent multicast tree from each media source to all the servers. The computational complexity for the similar (but not identical) object allocation problem in a tree network is $O(N^5)$ in [9] and $O(N\text{diam}(T)\log(\text{deg}(T)))$ in [12] (worst case is $O(N^2\log(N))$).

Without the update process in the unicast distribution model, we end up with the classical "uncapacited plant location problem" [13] model with facilities replacing servers and roads replacing communication lines. The problem has been proved to be NP-complete for general graphs [13]. It was solved for trees in polynomial time [2,11]. The "uncapacited plant location problem" was mapped to content delivery networks [3]. Additional works that address the read only model address the server location problem given a fixed number of servers [8,14] and objects [10].

3. The model

Objects

For each object o of the objects set O, we determine the set of servers which store copies the object. The algorithm handles each object separately, so the various costs described in figure 1 may differ for each object.

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. It is also an entry point of consumers and/or media sources to the network. Distribution demands of consumers connected to vertex i are satisfied by the network from the server at the closest vertex (or the closest multicast tree rooted at) j which stores an object copy. An update to an object may be generated 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; the parent vertex of vertex i in T ($i \neq r$) as P_i; the edge that connects vertex i to its parent in T, (i, P_i) as e_i ($e_r = \emptyset$); the set of

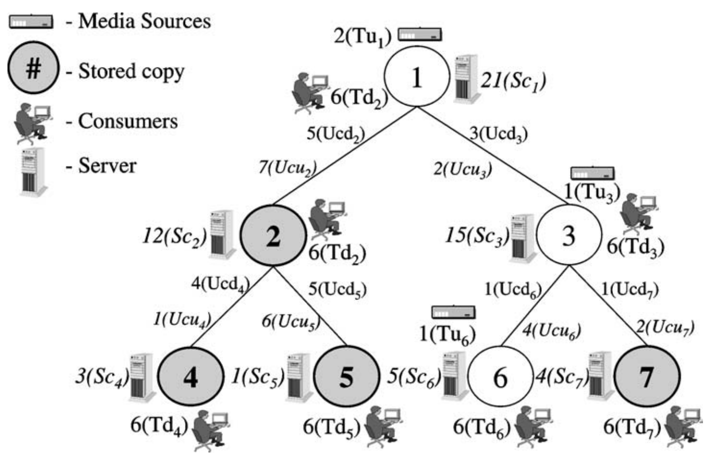

Figure 1. An example of a tree network. Edges are associated with unit distribution and update costs. Vertices may have servers with storage costs, media sources with update demands and consumers with distribution demands.

edges in $T_i \cup e_i$ (including edge e_i) as E_i ($E_r \equiv E$); the set of vertices in T_i as V_i ($V_r \equiv V$); the set of children vertices of vertex i in T as Ch_i (for a leaf i, $Ch_i = \emptyset$).

Figure 1 demonstrates a tree network and its related costs.

Storage cost

Let the storage cost of the object at vertex i to be Sc_i. Sc_i represents the resources needed from the server for storing the object, like disk space, computational power and relative maintenance cost.

Denote Φ is the set of vertices that store the object. The total storage cost of the object in the network is $\sum_{i \in \Phi} Sc_i$.

Distribution traffic cost

Denote the cost per distribution traffic unit at edge e_i as Ucd_i ($Ucd_i > 0$). Since $e_r = \emptyset$, $Ucd_r \equiv 0$. Ucd_i represents the residual cost of traffic in a physical line or the relative cost of the connection to a public network.

Multicast distribution traffic cost. The multicast distribution traffic provided to vertex i, Tdm_i, is Td.[1] Td may be the bandwidth requirement, or other QoS related parameters.

Denote Dmt_i the set of edges in the distribution multicast tree rooted at vertex i. If vertex i does not store an object then $Dmt_i = \emptyset$. The total multicast distribution traffic cost in the network is $\sum_{i \in \Phi} Td \sum_{e \in Dmt_i} Ucd_e$.

Unicast distribution traffic cost. The cost per distribution traffic unit along a path between vertices i and j is $Dd_{i,j} = \sum_{e \in P_{i,j}} Ucd_e$, where $P_{i,j}$ is the set of edges that connect vertex i to vertex j. We define $P_{i,i} \equiv \emptyset$ and $Dd_{i,i} \equiv 0$. Since the tree is undirected, $P_{i,j} = P_{j,i}$.

The total distribution traffic demand (requirements) produced by all the consumers connected to vertex i is Tdu_i ($Tdu_i \geqslant 0$).

The total unicast distribution traffic cost in the network is $\sum_{i \in V} Tdu_i \min_{j \in \Phi} Dd_{i,j}$. (In case there exist $j, k \in \Phi$ such that $Dd_{i,j} = Dd_{i,k}$ and $j < k$ then we will select j, the smallest index.)

Multicast update traffic cost

Denote the cost per update traffic unit at edge e_i as Ucu_i ($Ucu_i > 0$). Since $e_r = \emptyset$, $Ucu_r \equiv 0$.

The total multicast update traffic generated by all the media sources connected to vertex i is Tu_i ($Tu_i \geqslant 0$). This traffic is delivered through $Umt_{i,\Phi}$, the set of edges of the multicast update tree from vertex i to Φ. The total update traffic cost in the network is $\sum_{i \in V} Tu_i \sum_{e \in Umt_{i,\Phi}} Ucu_e$.

4. Properties of the optimal solution on trees

Before we describe the common characteristics of the algorithms, we first emphasize some properties of the optimal solution. These properties are the foundations of our technique and are also required to explain its correctness.

4.1. Per edge update traffic

As described in Section 3, a vertex i which is a root of a multicast update tree generates Tu_i update traffic which is delivered through each edge $e \in Umt_{i,\Phi}$. The update traffic of such a tree is directed from i to Φ. Since the location of the media sources is known a-priory, when we look at a single edge, we can determine the update traffic that will pass through it in each direction, in case there are copies stored in the subtrees connected to it (in both ends of the edge).

For each edge e_i we define Tu_i^{out} and Tu_i^{in}. Tu_i^{out} is the total update traffic that is outgoing via vertex i and edge e_i out of T_i, in case there is at least one copy stored outside T_i. Tu_i^{in} is the total update traffic that is incoming via vertex i and edge e_i into T_i, in case there is at least one copy stored in T_i.

$$Tu_i^{\text{out}} \leftarrow \sum_{j \in V_i} Tu_j = Tu_i + \sum_{c \in Ch_i} Tu_c^{\text{out}},$$

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

4.2. Distribution traffic properties

Lemma 1. In the optimal allocation, in case of unicast distribution, if vertex i is served from vertex j, which satisfies $\min_{j\in\Phi} Dd_{i,j}$, and i is served through vertex k (i.e., $P_{i,j} = P_{i,k} \cup P_{k,j}$), then k must also be served from j.

Proof: Since $P_{i,j} = P_{i,k} \cup P_{k,j}$ then $Dd_{i,j} = Dd_{i,k} + Dd_{k,j}$. Suppose vertex k is not served from j, but from a different vertex l. Since the solution is optimal there must exist $Dd_{k,l} < Dd_{k,j}$. In that case we get $Dd_{i,l} = Dd_{i,k} + Dd_{k,l} < Dd_{i,k} + Dd_{k,j}$, which is a contradiction. □

Lemma 2. In the optimal allocation, in case of multicast distribution, each vertex i can only belong to at most one multicast distribution tree.

Proof: Suppose a vertex i belongs to more than one multicast distribution tree, 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. □

Lemma 3. In the optimal allocation, if vertex i is served through its neighbor k in T (either parent or child), then i and k are served from the same server.

Proof: The proof is a direct result of Lemmas 1 and 2. □

Corollary 1. The above properties lead us to the conclusion that the optimal allocation is composed of a subgraph of T which is a forest of unicast or multicast distribution subtrees. Each subtree is rooted at a vertex where a copy is located and its leaves are vertices were no copy is stored and there is distribution demand. Each edge and vertex in T can be part of at most one unicast or at most one multicast distribution subtree.

5. Common algorithms properties

The main idea behind the algorithms is the observation that in tree graphs, since there is only one edge from each vertex i to its parent, and due to Lemma 3, if we consider the influence of the optimal allocation outside T_i on the optimal allocation within T_i, it is narrowed to a very small number of possibilities. We just have to consider the following: are copies located outside T_i (and if i is served from an external copy, where is it located), are there no copies located within T_i (only when $i \neq r$) and is there multicast distribution demand outside T_i. We define scenarios that are possible for each vertex pair i, j in unicast distribution and for vertex i in multicast distribution, which cover all these possible external influences on the optimal allocation within T_i. In addition, due to the same Lemma 3, it is fairly easy and straight forward to calculate the optimal allocation for vertex i and T_i based on the optimal allocation calculated for each c and T_c, where $c \in Ch_i$.

As a result, our algorithms for tree graphs are recursive algorithms that find the optimal allocation for a new problem which is a subset of the original problem for vertex i and T_i, based on the optimal allocation computed by its children Ch_i for their subsets of the original problem. (There are different new problems for the multicast/unicast distribution cases.)

The algorithms are performed in two phases. The first phase is the cost calculation phase which starts at the leaves and ends at the root, while calculating the optimal allocation and its alternate cost for each vertex pair i, j in unicast distribution and each vertex i in multicast distribution and for each scenario, based on the optimal allocations calculated by the children of vertex i for all their possible scenarios. The second phase is a backtrack phase which starts at the root and ends at the leaves where the algorithm selects the scenario which is active in the optimal allocation (in the optimal solution there can be only one actual scenario possible for each vertex) and allocates the copies in the relevant servers. The second phase is needed since only in the root it is possible to find the optimal allocation of the entire tree, and since the algorithm works in a recursive way, the root does not know the entire optimal allocation, but only the actual scenarios of itself and its children as well as the cost of the optimal allocation.

The algorithms calculate the optimal object allocation cost as well as the set of servers that will store the object.

6. MDT – the optimal algorithm for multicast distribution

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

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

(*Storage cost* + *Total multicast update cost* + *Total multicast distribution cost*).

For the new problem we define a new tree, which is a subtree of T constructed of T_i edge e_i.

The new optimization problem is defined as follows. Find the optimal allocation and its alternate cost in T_i, given the following assumptions:

1. There are copies or there is no copy located inside T_i.
2. There are copies or there is no copy located outside T_i.
3. When there are copies outside T_i, is there a need for incoming multicast distribution. I.e., are there consumers inside T_i that are connected to a multicast distribution tree through edge e_i.
4. When there are copies inside T_i, is there a need for outgoing multicast distribution. I.e., are there consumers outside T_i that are connected to a multicast distribution tree through edge e_i.

Table 1. The possible scenarios of the optimal allocation for a subtree.

Abbr.	Scenario
xi_i	e**X**ternal only object allocation and **I**ncoming multicast distribution
io_i	**I**nternal only object allocation and optional **O**utgoing multicast distribution
bi_i	**B**oth sides object allocation and **I**ncoming multicast distribution
bo_i	**B**oth sides object allocation and optional **O**utgoing multicast distribution

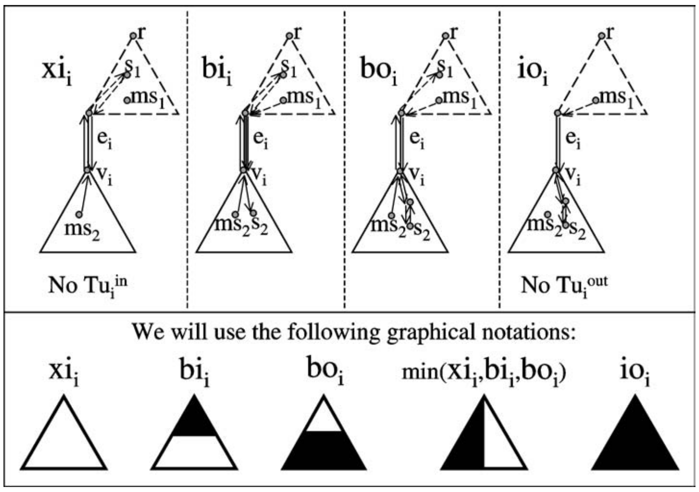

Figure 2. The possible scenarios of the optimal allocation for a subtree.

Based on the above assumptions we define the scenarios that are possible for each vertex i. Table 1 and figure 2 describe the possible scenarios.

6.1. The cost calculation phase

For each vertex i the algorithm calculates for T_i four alternate costs, for the scenarios in Table 1:

Cxi_i	There is no copy located inside T_i $(i \neq r)$. Edge e_i will carry incoming distribution and outgoing update traffic.
Cbi_i	Copies are located both inside and outside T_i but not all the internal consumers demand is supplied from copies in T_i. Edge e_i will carry incoming distribution and both incoming and outgoing update traffic.
Cbo_i	Copies are located both inside and outside T_i and all the internal consumers demand is supplied from copies in T_i. Edge e_i will carry both incoming and outgoing update (and maybe outgoing distribution) traffic.
Cio_i	All the copies of the object are located only inside T_i. Edge e_i will carry incoming update (and maybe outgoing distribution) traffic.

The algorithm calculates the alternate costs as follows:

$$Cxi_i \leftarrow \begin{cases} Td \cdot Ucd_i + Tu_i^{\text{out}} \cdot Ucu_i + sum4, & \text{if } i \neq r, \\ \infty, & \text{if } i = r, \end{cases}$$

$$Cbi_i \leftarrow \begin{cases} Td \cdot Ucd_i + \left(Tu_i^{\text{in}} + Tu_i^{\text{out}}\right)Ucu_i + sum1, & \text{if } i \neq r \text{ and } Ch_i \neq \emptyset, \\ \infty, & \text{if } i = r \text{ and } Ch_i = \emptyset, \end{cases}$$

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

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

where (various combinations of children scenarios):

$$\begin{aligned} min1 &= Sc_i + sum1, \\ sum1 &= \sum_{c \in Ch_i} \min\{Cxi_c, Cbo_c, Cbi_c\}, \\ min2 &= \min_{c \in Ch_i} \{Td \cdot Ucd_c + Cbo_c + sum2\}, \\ sum2 &= \sum_{k \in Ch_i, k \neq c} \min\{Cxi_k, Cbo_k, Cbi_k\}, \\ min3 &= \min_{c \in Ch_i} \{Td \cdot Ucd_c + Cio_c + sum3\}, \\ sum3 &= \sum_{k \in Ch_i, k \neq c} Cxi_k, \\ sum4 &= \sum_{c \in Ch_i} Cxi_c. \end{aligned}$$

Note. $sum1$, $sum2$, $sum3$, $sum4$ equal 0 and $min2$, $min3$ equal ∞ if vertex i is a leaf ($Ch_i = \emptyset$).

Figure 3 illustrates the combinations of children scenarios (and costs) that are used in the optimal cost calculation.

The cost of the optimal allocation in T is Cio_r.

6.2. Backtracking for content allocation

While calculating the alternate costs for each vertex i, the algorithm remembers for each alternate cost (scenario) if a copy needs to be stored at i and the relevant scenario of each child c which was used in the calculation (except for xi_c, since no copy is stored in its subtree).

The backtrack phase is recursive, starts at the root and ends at the leaves of T (can stop earlier if no child c has a copy in V_c). Based on the saved backtrack data, for each vertex i, the algorithm determines the actual scenario in the optimal allocation, if a copy should be

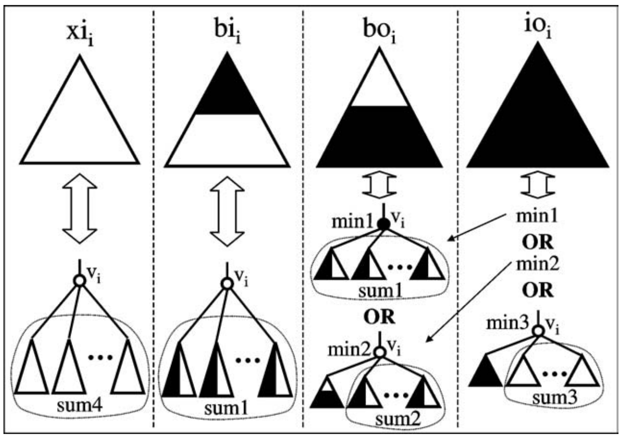

Figure 3. An illustration of the combinations of children scenarios used for cost calculation.

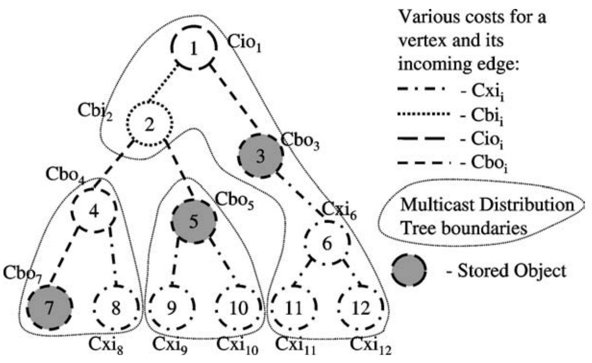

Figure 4. An optimal allocation example, the actual scenarios and the distribution forest.

stored at i (will happen if $min1$ was used in the actual scenario) and if it is necessary to keep advancing towards the leaves of T.

Figure 4 demonstrates an optimal allocation, the various actual scenarios selected during the backtrack phase, and the multicast distribution forest for that allocation.

Section 9.1 presents the pseudo-code of the algorithm. Backtrack details are also shown there.

6.3. *MDT computational complexity*

In the cost calculation phase, for each vertex in the tree $i \in V$ the algorithm calculates up to 4 alternate costs. Each cost calculation requires $O(|Ch_i| + 1)$. Therefore the total complexity of cost calculation for vertex i is $4O(|Ch_i| + 1)$.

The total complexity of the cost calculation phase for the entire tree is $\sum_{i\in V} 4O(|Ch_i| + 1)$.

The complexity of the backtrack phase for vertex i is $O(|Ch_i| + 1)$. $|V| = N$ and the total number of children in the tree is $N - 1$ (only the root r is not a child).

Therefore:

$$\begin{aligned} O_{\text{MDT}} &= \sum_{i\in V}(4+1)O\big(|Ch_i|+1\big) = O\Big(5\sum_{i\in V}\big(|Ch_i|+1\big)\Big) \\ &= O\big(5(2N-1)\big) = O(N). \end{aligned}$$

The computational complexity of MDT is $O(N)$.

6.4. *Proof of optimality*

The proof is based on induction. Lemma 4 is the induction base.

Lemma 4. For all the 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 copy outside T_i (through edge e_i), and no copy is stored in T_i. The optimal cost is outgoing update and incoming distribution traffic through edge e_i. (Cxi_i)
2. Vertex i is served from a copy outside T_i (through edge e_i), while a copy is stored in T_i. Impossible for a leaf, since leaf i must store a copy and it will be served from the local copy. ($Cbi_i = \infty$)
3. Vertex i is served from within T_i and copies are allocated both inside and outside T_i. As a leaf, vertex i must store a copy. Since copies are allocated both inside and outside T_i, edge e_i will carry both incoming and outgoing update traffic. The optimal cost is the storage cost at vertex i and update traffic through edge e_i. (Cbo_i)
4. All the copies are allocated inside T_i. As a leaf, vertex i must store a copy. Edge e_i will not carry outgoing update traffic, since there are no copies outside T_i. The optimal cost is the storage cost at vertex i and incoming update traffic through edge e_i. (Cio_i)

□

Lemma 5 constructs the induction step for the recursive proof of optimality.

Lemma 5. Assume that the algorithm optimally allocates copies 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 e_i), and no copy is stored in T_i. No copy can be allocated in T_c, $c \in Ch_i$. Based on the induction, the problem was solved for T_c, $c \in Ch_i$, and the additional cost in i is the incoming distribution and outgoing update traffic through edge e_i. (Cxi_i)
2. Vertex i is served from a copy outside T_i (through edge e_i), and copies are located both inside and outside T_i. At least one copy must be allocated in one or more subtrees T_c, $c \in Ch_i$. Vertex i cannot store a copy, and edge e_i will carry incoming distribution traffic, since it's served from outside T_i. If there are vertices $c \in Ch_i$ which are served through edge e_c, then according to Lemma 2 they must be served through edge e_i. Since copies are stored inside and outside T_i, edge e_i carries both incoming and outgoing update traffic. The optimal cost is the additional traffic through edge e_i, and the sum of optimal costs calculated for each T_c (the minimum of the following legal scenarios for each $c \in Ch_i$: Cxi_c, Cbi_c or Cbo_c). (Cbi_i)
3. Vertex i is served from within T_i and copies are allocated both inside and outside T_i. Since copies are allocated both inside and outside T_i, edge e_i will carry both incoming and outgoing update traffic. There are two 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 $c \in Ch_i$: Cxi_c, Cbi_c or Cbo_c). ($min1$)

   (b) Vertex i is served from T_c, $c \in Ch_i$, which stores a copy of the object (through e_c). The other vertices in the subtrees T_k, $k \in Ch_i$, $k \neq c$, may be served either through vertex i or from internal copies (the minimum of the following legal scenarios for each $k \in Ch_i$: Cxi_k, Cbi_k or Cbo_k). ($min2$)

   The optimal cost is the additional incoming and outgoing update traffic through edge e_i and the minimum between $min1$, $min2$. (Cbo_i)
4. All the copies are allocated inside T_i. Edge e_i will only carry incoming update traffic (for the internal copies). 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 $c \in Ch_i$: Cxi_c, Cbi_c or Cbo_c). ($min1$)

   (b) Vertex i is served from T_c, $c \in Ch_i$ which stores a copy of the object (through e_c). The other vertices in the subtrees T_k, $k \in Ch_i$, $k \neq c$, may be served either through vertex i or from internal copies (the minimum of the following legal scenarios for each $k \in Ch_i$: Cxi_k, Cbi_k or Cbo_k). ($min2$)

   (c) Vertex i is served from T_c, $c \in Ch_i$, which stores a copy (outgoing distribution traffic on edge e_c), and no other vertices in the subtrees T_k, $k \in Ch_i$, $k \neq c$, store a copy of the object. All these subtrees must be served through vertex i. ($min3$)

The optimal cost is the additional incoming update traffic through edge e_i and the minimum between $min1$, $min2$, $min3$ (Cio_i).

□

Theorem 1. When the algorithm ends, Cio_r holds the optimal allocation cost and the allocation of copies is optimal.

Proof: The proof is conducted by the induction where Lemma 4 is the base and Lemma 5 is the step. In addition, the costs Cxi_r, Cbi_r and Cbo_r are illegal since there cannot be copies allocated outside $T_r \equiv T$. □

7. UDT – the optimal algorithm for unicast distribution

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

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

(*Storage cost* + *Total multicast update cost* + *Total unicast distribution cost*).

For the new problem we define a new tree, $T_{i,j}$, which is a subtree of T constructed of T_i (the subtree of T rooted at i), and the additional set of edges e_i (connects vertex i to its parent) and $P_{i,j}$ (the string that connects vertex i to j), in case $j \notin T_i$.

The new optimization problem is defined as follows. Find the optimal allocation and its alternate cost in $T_{i,j}$, given the following assumptions:

1. A copy is located at vertex j, $j \in V$. If $j \in V_i$ vertex i is served by unicast distribution from j. If $j \notin V_i$ and vertex i is served by unicast distribution from outside T_i it is served from j. When $j \notin V_i$, since vertex j is not part of $T_{i,j}$ (just the path to it), its storage cost is ignored. Note: the difference between the case of $j \in V_i$ and $j \notin V_i$, is that i has the entire data (including storage cost) regarding copies allocation inside T_i, but only the unicast distribution cost from copies located outside T_i. For $j \notin V_i$, i may decide that it is less expensive to be served from a stored copy in an internal vertex l than to be served from j.
2. There are copies or there is no copy located inside T_i. Relevant for update multicast traffic.
3. There are copies or there is no copy located outside T_i. Relevant for update multicast traffic.

Based on the above assumptions we define scenarios that are possible for each vertex pair i, j. Table 2 lists the different scenarios for each vertex pair i, j.

Table 2. The possible scenarios of the optimal allocation for a subtree.

Abbreviation	Scenario
$xn_{i,j}\ (j \notin V_i)$	e**X**ternal only object allocation
$in_{i,j}\ (j \in V_i)$	**I**nternal only object allocation
$bn_{i,j}$	**B**oth sides object allocation

For each vertex pair i, j the algorithm calculates for $T_{i,j}$ three alternate costs:

$Cxn_{i,j}$	There is no copy located in T_i $(i \neq r)$ and the alternate cost is of distribution traffic only. Edge e_i will carry outgoing update traffic.
$Cin_{i,j}$	At least one copy is located inside and no copies are located outside T_i. Edge e_i will carry incoming update traffic.
$Cbn_{i,j}$	Copies are located both inside and outside T_i. Edge e_i will carry both incoming and outgoing update traffic.

The algorithm calculates the alternate costs as follows:

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

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

$$Cbn_{i,j} \leftarrow \begin{cases} Tdu_i \cdot Dd_{i,j} + Tu_i^{\text{in}} \cdot Ucu_i + Tu_i^{\text{out}} \cdot Ucu_i + sum3, & \text{if } j \in V_k,\ k \in Ch_i, \\ Tu_i^{\text{in}} \cdot Ucu_i + Tu_i^{\text{out}} \cdot Ucu_i + Sc_i + sum4, & \text{if } j = i, \\ \min\Big\{\min\limits_{l \in V_i} Cbn_{i,l},^{2}\ Tdu_i Dd_{i,j} + sum4 & \\ \qquad + \big(Tu_i^{\text{in}} + Tu_i^{\text{out}}\big) Ucu_i\Big\}, & \text{if } j \notin V_i, \end{cases}$$

where

$$sum1 = \sum_{k \in Ch_i} Cxn_{k,j},$$

$$sum2 = Cin_{k,j} + \sum_{l \in Ch_i, l \neq k} Cxn_{l,j},$$

$$sum3 = Cbn_{k,j} + \sum_{l \in Ch_i, l \neq k} \min\{Cxn_{l,j}, Cbn_{l,j}\},$$

$$sum4 = \sum_{k \in Ch_i} \min\{Cxn_{k,j}, Cbn_{k,j}\}.$$

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

The optimal total cost is $\min_{j \in V} Cin_{r,j}$.

7.1. *Backtracking for content allocation*

While calculating the alternate costs for each vertex pair i, j, the algorithm remembers for each alternate cost (scenario), if a copy needs to be stored at vertex i and the relevant scenario of each child k that was used in the calculation (unless the scenario is $xn_{k,j}$, since it has no copy stored in its subtree). This is important for the backtracking phase, and allows accurate placement of the copies while backtracking.

The backtrack 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_k). For each vertex i, the algorithm determines the actual scenario in the optimal allocation, if a copy should be stored at i (will happen if (i, i) pair was selected for an actual scenario) and if it is necessary to keep advancing towards the leaves of T. The algorithm uses the backtrack information that was saved earlier.

Section 9.2 presents the pseudo-code of the algorithm. Backtrack details are also shown there.

7.2. *Complexity*

In the cost calculation phase, for each vertex in the tree $i \in V$ the algorithm calculates up to $3N$ alternate costs. Each cost calculation requires $O(|Ch_i| + 1)$. Therefore the total complexity of cost calculation for vertex i is $3N \cdot O(|Ch_i| + 1)$.

The total complexity of the cost calculation phase for the entire tree is $\sum_{i \in V} 3N \cdot O(|Ch_i| + 1)$.

The complexity of the backtrack phase for vertex i is $O(|Ch_i| + 1)$.

$|V| = N$ and the total number of children in the tree is $N - 1$ (only the root r is not a child).

Therefore:

$$\begin{aligned} O_{\mathrm{UDT}} &= \sum_{i \in V} (3N+1) O\big(|Ch_i| + 1\big) \\ &= O\Big((3N+1) \sum_{i \in V} \big(|Ch_i| + 1\big)\Big) \\ &= O\big((3N+1)(2N-1)\big) = O\big(N^2\big). \end{aligned}$$

The computational complexity of UDT is $O(N^2)$.

7.3. *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 ($Cxn_{i,i}$ cannot exist, set to ∞). The optimal cost is constructed from the storage cost and the cost of update traffic (incoming only for $Cin_{i,i}$, both incoming and outgoing for $Cbn_{i,i}$).
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 $Cin_{i,j}$ is illegal (set to ∞) 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 $Cxn_{i,j}$; in this case the algorithm also sets $Cbn_{i,j}$ to $Cxn_{i,j}$ + the incoming update cost, which insures that if a copy of the object is not stored at T_i, $Cxn_{i,j} \leqslant Cbn_{i,j}$), otherwise the decision is to store a copy at vertex i (represented by $Cbn_{i,j}$ which is set to $Cbn_{i,i}$, in this case $Cbn_{i,j} \leqslant Cxn_{i,j}$).

□

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 $Cxn_{i,i}$ cannot exist (set to ∞), 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 $k \in Ch_i$: $Cxn_{k,i}$, $Cbn_{k,i} \Rightarrow sum4$.) 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 $Cin_{i,i}$ both incoming and outgoing for $Cbn_{i,i}$).
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 $Cxn_{i,j}$ cannot exist (set to ∞), and there must be incoming update traffic through edge i.
   For the scenario where no copy of the object is allocated outside T_i ($Cin_{i,j}$), 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 ($Cbn_{i,j}$), 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 $Cin_{i,j}$, both incoming and outgoing for $Cbn_{i,j}$).

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 $Cin_{i,j}$ is illegal (set to ∞) 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 1 all vertices in T_i are served from j. The optimal cost is the distribution cost from j to i plus the optimal $Cxn_{i,j}$ costs calculated for the children of i ($sum1$) plus the outgoing update traffic. (Represented by $Cxn_{i,j}$.)

   (b) Vertex i (and according to Lemma 1 all vertices in T_i) is served from within T_i. In this case the optimal allocation is represented by the minimal cost $\min_{l \in V_i} Cbn_{i,l}$ ($Cbn_{i,j}$ since there is a copy located outside T_i). (One possibility for $Cbn_{i,j}$.)

   (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 $k \in Ch_i$: $Cxn_{k,j}$, $Cbn_{k,j} \Rightarrow sum4$.) 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 $Cbn_{i,j}$.)

□

Theorem 2. When the algorithm ends, $\min_{j \in V} Cin_{r,j}$ 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 $j \in V$, $Cin_{r,j}$ represents an optimal allocation of the objects where r is served from j. The minimal $Cin_{r,j}$ is the optimal cost of the original optimization problem. In addition, the costs $Cbn_{r,j}$, $Cxn_{r,j}$ are illegal since there cannot be copies allocated outside $T_r \equiv T$. □

8. 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 multicast based update for unicast or multicast content distribution.

We developed optimal content allocation algorithms for tree networks 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 requires 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 Section 9 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 additional work may be in the direction of adjusting the above static algorithms into a dynamic environment where the demands and various costs may change over time. Since the algorithms are 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.

9. Pseudo-code of the algorithms

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 ∞ 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.

9.1. Pseudo-code of MDT

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 $i = n, n-1, n-2, \ldots, 2, 1$ do
    if $Ch_i = \emptyset$ then /* a leaf */
        $Cxi_i \leftarrow Td \cdot Ucd_i + Tu_i^{\text{out}} \cdot Ucu_i$; $Cbi_i \leftarrow \infty$; $BT\text{-}Cbi_i \leftarrow \emptyset$
        $Cbo_i \leftarrow (Tu_i^{\text{in}} + Tu_i^{\text{out}}) \cdot Ucu_i + Sc_i$; $BT\text{-}Cbo_i \leftarrow (i, \text{“}local\text{”})$
        $Cio_i \leftarrow Tu_i^{\text{in}} \cdot Ucu_i + Sc_i$; $BT\text{-}Cio_i \leftarrow (i, \text{“}local\text{”})$
    else /* Not a leaf */
        /* calculate $sum1$, $sum4$ ($sum1$ derives $sum2$, $sum4$ derives $sum3$!) */
        $sum1 \leftarrow 0$; $BT\text{-}sum1 \leftarrow \emptyset$; $sum4 \leftarrow 0$
        foreach $c \in Ch_i$ do
            $sum4 \leftarrow sum4 + Cxi_c$; $Cmin_c \leftarrow Cxi_c$; $Cmin_{c,type} \leftarrow \text{“}none\text{”}$
            if $(Cbo_c < Cmin_c)$ then $Cmin_c \leftarrow Cbo_c$; $Cmin_{c,type} \leftarrow \text{“}bo\text{”}$ end if
            if $(Cbi_c < Cmin_c)$ then $Cmin_c \leftarrow Cbi_c$; $Cmin_{c,type} \leftarrow \text{“}bi\text{”}$ end if
            $sum1 \leftarrow sum1 + Cmin_c$; $BT\text{-}sum1 \leftarrow BT\text{-}sum1 \cup (c, Cmin_{c,type})$
```

```
      end do
      /* calculate $min1$, $min2$, $min3$ */
      $min1 \leftarrow Sc_i + sum1$; $BT\text{-}min1 \leftarrow BT\text{-}sum1 \cup (i, \text{“}local\text{”})$
      $min2 \leftarrow \infty$; $BT\text{-}min2 \leftarrow \emptyset$; $min3 \leftarrow \infty$; $BT\text{-}min3 \leftarrow \emptyset$
      foreach $c \in Ch_i$ do
         $sum2 \leftarrow sum1 - Cmin_c$; $tmp \leftarrow Td \cdot Ucd_c + Cbo_c + sum2$
         if $(tmp < min2)$ then
            $min2 \leftarrow tmp$; $BT\text{-}min2 \leftarrow (c, \text{“}bo\text{”}) \cup (BT\text{-}sum1 \setminus (c, Cmin_{c,type}))$
         end if
         $sum3 \leftarrow sum4 - Cxi_c$; $tmp \leftarrow Td \cdot Ucd_c + Cio_c + sum3$
         if $(tmp < min3)$ then $min3 \leftarrow tmp$; $BT\text{-}min3 \leftarrow (c, \text{“}io\text{”})$ end if
      end do
      if $i \neq 1$ then /* not root */
         $Cxi_i \leftarrow Td \cdot Ucd_i + Tu_i^{\text{out}} \cdot Ucu_i + sum4$
         $Cbi_i \leftarrow Td \cdot Ucd_i + (Tu_i^{\text{in}} + Tu_i^{\text{out}}) \cdot Ucu_i + sum1$; $BT\text{-}Cbi_i \leftarrow BT\text{-}sum1$
         if $min1 \leqslant min2$ then
            $Cbo_i \leftarrow (Tu_i^{\text{in}} + Tu_i^{\text{out}}) \cdot Ucu_i + min1$; $BT\text{-}Cbo_i \leftarrow BT\text{-}min1$
         else
            $Cbo_i \leftarrow (Tu_i^{\text{in}} + Tu_i^{\text{out}}) \cdot Ucu_i + min2$; $BT\text{-}Cbo_i \leftarrow BT\text{-}min2$
         end if
      end if
      /* calculate optimal $Cio_i$ cost and $BT$ data */
      if $min1 \leqslant min2$ & $min1 \leqslant min3$ then
         $Cio_i \leftarrow Tu_i^{\text{in}} \cdot Ucu_i + min1$; $BT\text{-}Cio_i \leftarrow BT\text{-}min1$
      else if $min2 \leqslant min3$ then
         $Cio_i \leftarrow Tu_i^{\text{in}} \cdot Ucu_i + min2$; $BT\text{-}Cio_i \leftarrow BT\text{-}min2$
      else
         $Cio_i \leftarrow Tu_i^{\text{in}} \cdot Ucu_i + min3$; $BT\text{-}Cio_i \leftarrow BT\text{-}min3$
      end if
   end if
end do
```

The optimal cost is Cio_1.

Backtrack phase The backtrack phase for allocation of copies is recursive and can easily be described using a recursive function. The recursion starts by calling **allocate**(1, “*io*”).

```
proc allocate $(i, type)$ {
   if $type =$ “$io$” then
      foreach $(c, ctype) \in BT\text{-}Cio_i$ do call allocate$(c, ctype)$ end do
   else if $type =$ “$bo$” then
      foreach $(c, ctype) \in BT\text{-}Cbo_i$ do call allocate$(c, ctype)$ end do
   else if $type =$ “$bi$” then
      foreach $(c, ctype) \in BT\text{-}Cbi_i$ do call allocate$(c, ctype)$ end do
   else if $type =$ “$local$” then
```

```
        allocate a copy at $i$
    **end if**
}
```

9.2. Pseudo-code of UDT

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 $i = n, n-1, n-2, \ldots, 2, 1$ do
    /* calculate costs for $i, i$ */
    $sum4 \leftarrow 0$; $BT\text{-}sum4 \leftarrow \emptyset$
    foreach $k \in Ch_i$ do
        if $Cxn_{k,i} \leqslant Cbn_{k,i}$ then
            $sum4 \leftarrow sum4 + Cxn_{k,i}$
        else
            $sum4 \leftarrow sum4 + Cbn_{k,i}$; $BT\text{-}sum4 \leftarrow BT\text{-}sum4 \cup (k, i, \text{“}bn\text{”})$
        end if
    end do
    $Cin_{i,i} \leftarrow Tu_i^{\text{in}} \cdot Ucu_i + Sc_i + sum4$; $BT\text{-}Cin_{i,i} \leftarrow BT\text{-}sum4 \cup (i, i, \text{“}local\text{”})$
    $Cbn_{i,i} \leftarrow Cin_{i,i} + Tu_i^{\text{out}} \cdot Ucu_i$; $BT\text{-}Cbn_{i,i} \leftarrow BT\text{-}sum4 \cup (i, i, \text{“}local\text{”})$
    $Cxn_{i,i} \leftarrow \infty$; $j_{\min} \leftarrow i$
    /* calculate costs for $i, j$ where $j \in V_i$ */
    foreach $k \in Ch_i$ do
        foreach $j \in V_k$ do
            $sum2 \leftarrow Cin_{k,j}$; $BT\text{-}sum2 \leftarrow (k, j, \text{“}in\text{”})$
            $sum3 \leftarrow Cbn_{k,j}$; $BT\text{-}sum3 \leftarrow (k, j, \text{“}bn\text{”})$
            foreach $l \in Ch_i \setminus k$ do
                $sum2 \leftarrow sum2 + Cxn_{l,j}$
                if $Cxn_{l,j} \leqslant Cbn_{l,j}$ then
                    $sum3 \leftarrow Cxn_{l,j}$
                else
                    $sum3 \leftarrow Cbn_{l,j}$; $BT\text{-}sum3 \leftarrow BT\text{-}sum2 \cup (l, j, \text{“}bn\text{”})$
                end if
            end do
            $Cbn_{i,j} \leftarrow Cin_{i,j} + Tu_i^{\text{out}} \cdot Ucu_i + sum3$; $BT\text{-}Cbn_{i,j} \leftarrow BT\text{-}sum3$
            $Cxn_{i,j} \leftarrow \infty$; $Cin_{i,j} \leftarrow Tdu_i \cdot Dd_{i,j} + Tu_i^{\text{in}} \cdot Ucu_i$
            if $sum2 \leqslant sum3$ then
                $Cin_{i,j} \leftarrow Cin_{i,j} + sum2$; $BT\text{-}Cin_{i,j} \leftarrow BT\text{-}sum2$
            else
                $Cin_{i,j} \leftarrow Cin_{i,j} + sum3$; $BT\text{-}Cin_{i,j} \leftarrow BT\text{-}sum3$
            end if
            /* update $\min_{l \in V_i} Cbn_{i,l}$ */
            if $Cbn_{i,j} < Cbn_{i,j_{\min}}$ then $j_{\min} \leftarrow j$ end if
        end do
    end do
    /* calculate costs for $i$, $j$ where $j \notin V_i$ */
    foreach $j \in V \setminus V_i$ do
        $sum1 \leftarrow 0$; $sum4 \leftarrow 0$; $BT\text{-}sum4 \leftarrow \emptyset$
        foreach $k \in Ch_i$ do
            $sum1 \leftarrow sum1 + Cxn_{k,j}$
            if $Cxn_{k,j} \leqslant Cbn_{k,j}$ then
                $sum4 \leftarrow sum4 + Cxn_{k,j}$
            else
                $sum4 \leftarrow sum4 + Cbn_{k,j}$; $BT\text{-}sum4 \leftarrow BT\text{-}sum4 \cup (k, j, \text{“}bn\text{”})$
            end if
        end do
        $Cin_{i,j} \leftarrow \infty$; $Cxn_{i,j} \leftarrow Tdu_i \cdot Dd_{i,j} + Tu_i^{\text{out}} \cdot Ucu_i + sum1$
        $Cbn_{i,j} \leftarrow Tdu_i \cdot Dd_{i,j} + Tu_i^{\text{in}} \cdot Ucu_i + Tu_i^{\text{out}} \cdot Ucu_i + sum4$
        $BT\text{-}Cbn_{i,j} \leftarrow BT\text{-}sum4$
            if $Cbn_{i,j_{\min}} < Cbn_{i,j}$ then $Cbn_{i,j} \leftarrow Cbn_{i,j_{\min}}$; $BT\text{-}Cbn_{i,j} \leftarrow BT\text{-}Cbn_{i,j_{\min}}$; end if
    end do
end do
/* find the optimal cost */
$j_{\min} \leftarrow 1$
for $j = n, n-1, n-2, \ldots, 2$ do
    if $Cin_{1,j} < Cin_{1,j_{\min}}$ then $j_{\min} \leftarrow j$ end if
end do
```

The optimal cost is $Cin_{1,j_{\min}}$.

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, $j_{\min}$, “*in*”).

```
proc allocate ($i$, $j$, $type$) {
    if $type$ = “in” then
            foreach $(k, l, ntype) \in BT\text{-}Cin_{i,j}$ do call allocate($k, l, ntype$) end do
    else if $type$ = “bn” then
        foreach $(k, l, ntype) \in BT\text{-}Cbn_{i,j}$ do call allocate($k, l, ntype$) end do
    else if $type$ = “local” then
        allocate a copy at $i$
    end if
}
```

Notes

1. The reason for using the same traffic rate for all vertices is the fact that the server determines the transmission rate, not each customer as in the unicast case.
2. The minimum value can be calculated once during the calculation of each $Cbn_{i,j}$, $j \in V_i$, given that these values are calculated prior to calculating any $Cbn_{i,j}$, $j \notin V_i$.

References

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

[2] A. Billionnet and M.-C. Costa, "Solving the uncapacited plant location problem on trees," *Discrete Applied Mathematics* 49(1–3), 1994, 51–59.

[3] I. Cidon, S. Kutten, and R. Sofer, "Optimal allocation of electronic content," in *Proceedings of IEEE Infocom*, Anchorage, AK, April 22–26, 2001.

[4] Cisco, `http://www.cisco.com/`

[5] Digital Fountain, `http://www.digitalfountain.com/`

[6] L. W. Dowdy and D. V. Foster, "Comparative models of the file assignment problem," *ACM Computing Surveys* 14(2), 1982, 287–313.

[7] P. Francis, "Yoid: Extending the Internet multicast architecture," April 2000.

[8] S. Jamin, C. Jin, A. R. Kurc, D. Raz, and Y. Shavitt, "Constrained mirror placement on the Internet," in *Proc. of IEEE INFOCOM*, 2001.

[9] 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), 2001, 628–637.

[10] J. Kangasharju, 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.

[11] A. Kolen, "Solving covering problems and the uncapacited plant location problem on trees," *European Journal of Operational Research* 12, 1983, 266–278.

[12] C. Krick, H. Räcke, and M. Westermann, "Approximation algorithms for data management in networks," in *Proc. of the Symposium on Parallel Algorithms and Architecture*, July 2001, pp. 237–246.

[13] P. B. Mirchandani and R. L. Francis, *Discrete Location Theory*, Wiley, 1990.

[14] L. Qiu, V. N. Padmanabham, and G. M. Voelker, "On the placement of web server replicas," in *Proc. 20th IEEE INFOCOM*, 2001.

[15] Scale8, `http://www.scale8.com/`

[16] S. Shi and J. Turner, Routing in overlay multicast networks, in *Proc. of IEEE INFOCOM*, June 2002.

[17] WebDAV, `http://www.webdav.org/`

[18] O. Wolfson and A. Milo, "The multicast policy and its relationship of replicated data placement," *ACM Transactions on Database Systems* 16(1), 1991, 181–205.