OPTIMAL PLACEMENT OF IDENTICAL RESOURCES IN A DISTRIBUTED NETWORK

JAN 81 M J FISCHER, N D GRIFFETH, L J GUIBAS

UNCLASSIFIED TR-81-01-03
OPTIMAL PLACEMENT OF IDENTICAL RESOURCES
IN A DISTRIBUTED NETWORK

by

Michael J. Fischer
Nancy D. Griffeth
Leo J. Guibas
Nancy A. Lynch

Technical Report #81-01-03

†This research was supported in part by the National Science Foundation
under grants MCS77-02474, MCS77-15628, MCS78-01698, MCS80-03337, U.S.
Army Research Office Contract Number DAAG29-79-C-0155, and the Office
of Naval Research Contract Numbers N00014-79-C-0873 and N00014-80-C-0221.

††FEB 10 1981

Accession For
NTIS GRA Fin
DTIC TAB Unannounced
Justification

By
Distribution/
Availability Codes
Dist Avail and/or
Special
A
The problem is considered of locating a number of identical resources at nodes of a tree so as to minimize the total expected cost of servicing a set of random requests for the resources. The cost of servicing a request is the tree distance from the requesting node to the node at which the resource satisfying the request is located. An algorithm for finding an optimal placement of \( t \) resources is presented which runs in time \( O(t^2 | E |) \), where \( E \) is the edge set of the tree. In the special case of
OPTIMAL PLACEMENT OF IDENTICAL RESOURCES IN A DISTRIBUTED NETWORK

Michael J. Fischer
University of Washington

Leo J. Guibas
Xerox Palo Alto Research Center

Nancy D. Griffeth
Georgia Institute of Technology

ABSTRACT

The problem is considered of locating a number of identical resources at nodes of a tree so as to minimize the total expected cost of servicing a set of random requests for the resources. The cost of servicing a request is the tree distance from the requesting node to the node at which the resource satisfying the request is located. An algorithm for finding an optimal placement of \( t \) resources is presented which runs in time \( O(t^2 |E|) \), where \( E \) is the edge set of the tree. In the special case of a complete binary tree with requests uniformly distributed over the \( n \) leaves, another algorithm works in time \( O(t \log_2 n) \).

While optimal placements in general seem difficult to characterize, there is a very simple placement whose total cost differs from optimality by at most \( E \). The expected cost of this placement on a complete binary tree is \( O(n + \sqrt{tn}) \).

1. INTRODUCTION

We consider the problem of locating some number \( t \) of identical resources at nodes of a distributed network in such a way as to minimize the expected "cost" of servicing a random set of \( t \) requests for those resources. Various different costs are likely to be important in different situations.

For example, suppose the resources are processors and requests are to establish a virtual connection with a processor which will be used for a very long period of time. In this case one might want to minimize the expected total of all the network distances (measured in some appropriate way) between requesting users and their assigned processors. Because of the long holding times, it is probably reasonable to expend considerable effort to find a good matching of requests to resources. The total of the network distances provides a measure of the expected communication traffic introduced into the network by the computations.

For another example, suppose the resources are ticks to sporting events (or airline seats). A relevant cost is the expected waiting time until a buyer receives his ticket, or equivalently, the expected total waiting time for all buyers. In this situation, it is probably not reasonable to expend much effort in attaining a near-optimal matching, for the time to find the matching can easily exceed the eventual savings in locating a nearby ticket. Even so, the expected total distance in an optimal matching is significant as a lower bound for the expected total waiting time.

This paper investigates properties of optimal resource placements and provides fast algorithms for finding them. It also provides upper bounds for the costs of optimal placements and of certain nearly-optimal and easily-described placements. The network in all cases is assumed to be configured as a tree. Nevertheless, the upper bounds can often be applied to an arbitrary (connected) distributed network by constructing a spanning tree.

In this paper, we allow for the possibility that fractions of resources, and not only whole resources, might be located at some nodes of the network tree. This is reasonable if, for instance, the resources are large blocks of available data storage space. It would be perfectly permissible for a user to obtain parts of his needed storage from several different nodes. On the other hand, we assume that the requests are discrete -- each request is for one unit of resource.

In Section 2, we present our notation, definitions, and those results which apply to arbitrary trees and arbitrary probability distributions for arrivals of requests. §2.1 defines matchings and relates them to network flows. §2.2 defines the expected total cost of a placement in terms of expected total flow. Both the function describing the expected flow on each edge and the function describing the minimum possible expected total flow for any subtree are of a particularly simple form -- they are unbounded, convex, piecewise linear functions on the nonnegative reals, with
all singularities at integers. This immediately implies (in §2.3) that there are always optimal resource placements consisting of whole numbers of resources located at each node; it is never necessary to place fractions of resources at any node in order to achieve an optimal placement. An algorithm using $O(t^2 \cdot \text{edges})$ arithmetic operations is presented which always finds an optimal whole resource placement. This is the fastest algorithm we have which is completely general. Section 4 contains faster algorithms for special cases.

§2.4 considers what is required for a placement to optimize the expected flow over any particular edge in the tree -- i.e. to be locally optimal. Unfortunately, it is not possible in general to obtain a single placement which simultaneously optimizes the expected flow on all edges.

A fair whole placement is one which "almost optimizes" the flow on each edge -- that is, one in which the number of resources in each subtree is either the mean rounded up or down. It is always possible (and quite easy) to obtain a fair whole placement, and such placements are close to optimal.

In §2.5, we show that if there is a probability less than $\frac{1}{2}$ that any request will arrive in a particular subtree, then it is always bad to place even a very small fraction of a resource anywhere in that subtree.

In Section 3, we analyze the cost of an optimal placement for the case of a complete binary tree of $n$ leaves but with an arbitrary probability distribution for request arrivals. An arbitrary fair whole resource placement has expected total cost at most $O(n + \sqrt{n} \cdot \sqrt{t})$. Note that in the very important case where $t$ is roughly proportional to $n$ (i.e. the number of resources in the network is proportional to the number of nodes), that the expected average cost per request is bounded by a constant, independent of the size of the network. This situation is very different from the case of centralizing the resources, where the average cost grows proportionately with the log of the number of nodes in the network.

2. NOTATION, DEFINITIONS AND GENERAL RESULTS

2.1. Trees, Matchings and Flows

A tree $T = (V, E)$ is an undirected acyclic graph, where $V$ is the vertex (node) set and $E$ is the edge set.

Let $\mathbb{N}$ denote the natural numbers, including 0, and let $\mathbb{R}^+$ denote the nonnegative reals.

A set of requests is described by a function $r : V \rightarrow \mathbb{R}^+$ such that $r(v)$ is the number of requests originating at node $v$. A placement of resources is described by a function $s : V \rightarrow \mathbb{R}^+$ such that $s(v)$ is the number of resources placed at node $v$. We always assume $\text{total}(r) = \text{total}(s)$, where for any $g : V \rightarrow \mathbb{R}^+$, $\text{total}(g) = \sum_{v \in V} g(v)$.

A matching is a function $m : V \times V \rightarrow \mathbb{R}^+$. $m(u, v)$ gives the number of requests at node $u$ which are satisfied by resources at node $v$. $m$ is exact for $r, s$ if for all $u, v \in V$, $\sum_{w \in V} m(u, w) = r(u)$ and $\sum_{w \in V} m(w, v) = s(v)$. Thus, an exact matching gives a complete correspondence between requests and resources. Clearly, an exact matching exists whenever $\text{total}(r) = \text{total}(s)$.

The cost of a matching is defined to be

$$\text{cost}(m) = \sum_{u, v \in V} m(u, v) \cdot d(u, v)$$

where $d(u, v)$ is the number of edges in the unique path from $u$ to $v$ in $T$. Let

$$\text{cost}(r, s) = \min \{ \text{cost}(m) \mid m \text{ is an exact matching for } r, s \}.$$

A matching $m$ is optimal for $r, s$ if $\text{cost}(m) = \text{cost}(r, s)$.

It is convenient to relate a matching to a flow in a directed graph. Choose a node $w$, and direct the edges of the tree to point away from $w$. Corresponding to the usual way of drawing trees with the root at the top, we let high($e$) be the endpoint of edge $e$ which is closest to $w$, and low($e$) be the endpoint farthest from $w$.

In the remainder of this paper, it will be convenient for $w$ itself to have an edge entering it. Therefore, we add a new node $v$ to serve as the root, and we let rootedge($T$) = $(v, w)$, an edge from $v$ to $w$. Resources will never be placed on $v$, and requests will never originate at $v$. rootedge($T$) is just notational convenience.

A flow is a function $f : E \rightarrow \mathbb{R}$ which describes the "movement" of resources from their initial placement to corresponding requests. A positive value of $f(e)$ denotes a flow along the direction of the edge (i.e. towards the leaves) whereas a negative value of $f(e)$ denotes a flow towards the root. A flow is stable for $r, s$ if for every $v \in V$,

$$r(v) + \sum_{e : \text{high}(e) = v} f(e) = s(v) + \sum_{e' : \text{low}(e') = v} f(e')$$

Thus, at every node, the flow out equals the flow in.

The cost of a flow is defined to be

$$\text{cost}(f) = \sum_{e \in E} |f(e)|.$$

Call a flow $f$ optimal for $r, s$ if it is stable.
for \( r,s \) and has minimal cost over all such flows.

The following theorem formalizes the intuition that the flows correspond to resources moving along edges and that in an optimal matching, resources do not move both directions along the same edge. We omit the straightforward but tedious proof.

**Theorem 2.1.1.** Let \( m \) be an optimal matching and \( f \) an optimal flow for \( r,s \). Then \( \text{cost}(m) = \text{cost}(f) \). Moreover, if either \( m \) or \( f \) has an integral range, then the other can be chosen so also.

The removal of any edge \( e \) of a tree \( T \) splits the tree into two disconnected components: \( B(e) \), the nodes "below" \( e \), and \( A(e) \), the nodes "above" \( e \). \( B(e) \) and \( A(e) \) are defined by:

\[
B(e) = \{ v \in V \mid v \text{ is in the subtree rooted by low}(e) \}
A(e) = V - B(e).
\]

The flow along \( e \) in any flow for \( r,s \) depends only on the total number of requests and the total number of resources in \( B(e) \). For any function \( g: V \to \mathbb{R}^+ \) and \( e \in E \), let

\[
\text{total}(e, g) = \sum_{v \in B(e)} g(v).
\]

If \( \text{total}(e, r) \) exceeds \( \text{total}(e, s) \), then there are more requests than resources in the subtree \( B(e) \), so the excess must be satisfied by resources flowing into \( B(e) \) via the edge \( e \). On the other hand, if \( \text{total}(e, s) \) exceeds \( \text{total}(e, r) \), the excess resources in \( B(e) \) are needed by requests in \( A(e) \) (since \( \text{total}(r) = \text{total}(s) \)), so they must flow out of the subtree via \( e \). By our convention that the sign of the flow denotes its direction, the directed flow in either case is given by

\[
\text{dflow}_{r,s}(e) = \text{total}(e, r) - \text{total}(e, s).
\]

**Theorem 2.1.2.** \( \text{dflow}_{r,s} \) is the unique stable flow for \( r,s \).

**Proof.** The argument sketched above shows that there is at most one stable flow for \( r,s \). We leave to the reader to show that if \( \text{dflow}_{r,s} \) is stable for \( r,s \), then

\[
\text{cost}(r,s) = \text{cost}(\text{dflow}_{r,s}).
\]

Theorems 2.1.1 and 2.1.2 immediately yield:

**Theorem 2.1.3.** \( \text{cost}(r,s) = \text{cost}(\text{dflow}_{r,s}) \).

Thus, our original problem of studying optimal exact matchings reduces to the problem of studying a particular stable flow.

2.2. **Expected Costs**

In this section, we introduce probability distributions for sets of requests and define costs of placements in terms of their expected costs given a randomly chosen set of requests.

If \( \phi \) is a probability function on \( V \) and \( t \in \mathbb{N} \), then a random \( r: V \to \mathbb{R}^+ \) can be chosen according to a probability distribution determined as follows: for each \( i \) in turn, let \( \phi \) be used to select a vertex. Then \( r(v) \) is the total number of times \( v \) is selected for each \( v \in V \). We always assume \( \phi(\text{root}) = 0 \).

Fix \( \phi \), \( t \) as above, and let \( s: V \to \mathbb{R}^+ \) with total\( (s) = t \). Then \( \exp\text{cost}(s) \) denotes the expected value of \( \text{cost}(r,s) \), where \( r \) is chosen as described above, and

\[
\min\exp\text{cost} = \min \{ \exp\text{cost}(s) \mid s: V \to \mathbb{R}^+ \text{ and total}(s) = t \}.
\]

The flow along an edge \( e \) depends only on total\( (s) \), the number resources in the subtree below \( e \), and not on their particular placement. Let \( \exp\text{flow}(u) \) be the expected value of \( \text{dflow}_{r,s}(e) \), where \( r \) is chosen randomly as described above, and \( s \) is any placement with total\( (s) = u \).

Theorem 2.2.1. \( \exp\text{cost}(s) = \sum_{e \in E} \exp\text{flow}(\text{total}(e,s)) \).
Thus, \( \text{expcost}_e(s) \) gives the portion of the expected cost of placement \( s \) due to flow in the subtree below (and including) \( e \), and \( \text{minexpcost}_e(u) \) gives the least such cost, subject to the constraint that exactly \( u \) resources be placed in the subtree.

From these definitions and Theorem 2.2.1, it should be clear that

\[
\text{expcost}_e(s) = \text{expflow}_e(\text{total}(e,s))
\]

where \( e_1, \ldots, e_k \) are the immediate descendants of edge \( e \). Optimizing over all \( s \) with \( \text{total}(e,s) = u \), we get

**Theorem 2.2.4.** Let \( f, t, e \) be fixed. Let \( e_1, \ldots, e_k \) be the immediate descendant edges of \( e \), and let \( u \in \mathbb{R}^+ \). Then

\[
\text{minexpcost}_e(u) = \text{expflow}_e(u) + \min \left( \sum_{i=1}^k \text{minexpcost}_{e_i}(u) | u_i = u \right).
\]

In order to obtain more information about the values of the \( \text{minexpcost}_e \) function, we first show that it has a simple form.

**Theorem 2.2.5.** For any fixed \( f, t, e \), \( \text{minexpcost}_e \) is an unbounded, convex, piecewise linear function from \( \mathbb{R}^+ \) to \( \mathbb{R}^+ \), with all singularities occurring at integer values.

**Proof.** We use induction on edges in the tree, working from the leaves toward the root.

If \( e \) is a lowest edge, then there is only one edge, \( e \), in \( F_e \). Thus, \( \text{minexpcost}_e = \text{expflow}_e \), which has the needed properties by Theorem 2.2.3.

Now assume the result holds for edges below \( e \), and let \( e_1, \ldots, e_k \) denote the immediate descendant edges of \( e \). Consider the expression for \( \text{minexpcost}_e \) given in Theorem 2.2.4. The first term has the needed properties by Theorem 2.2.3. It remains to show that the second term is convex, piecewise linear and has all singularities at integers.

Write \( f_1 \) for \( \text{minexpcost}_e \), \( g(u) \) for

\[
\min \left( \sum_{i=1}^k f_1(u_i) | u_i = u \right).
\]

By the induction hypothesis, each \( f_1 \) is unbounded, convex and piecewise linear with all singularities at integers. Hence, the derivative \( f_1'(u) \) is defined at all non-singular points. We define \( f_1'(u) \) for \( u \) a singularity of \( f_1 \) to be the limit of \( f_1'(x) \) as \( x \) approaches \( u \) from above. By convexity and linearity of \( f_1, f_1' \) is a non-decreasing step function over the domain \( \mathbb{R}^+ \), with steps occurring at integer points.

For each \( i, 1 \leq i \leq k \), let \( r_i \) be the smallest element of \( \mathbb{R}^+ \) with \( f_1'(r_i) \geq 0 \). \( r_i \) exists since \( f_1 \) is unbounded, and \( r_i \in \mathbb{N} \) by the properties of \( f_1' \). We consider two cases:

**Case 1.** \( u \geq \sum r_i \). Then \( g(u) = \sum f_1(r_i) \), so \( g \) is constant for all such \( u \).

**Case 2.** \( u < \sum r_i \). Let \( S = \{ f_1(x) | 1 \leq i \leq k, x \in [0, r_i) \} \). For \( a \in S \) and \( 0 < i < k \), define

\[
x_1^0 = \text{glb} \{ x | f_1(x) < a \}
\]

\[
y_1^0 = \text{lub} \{ x | f_1(x) \geq a \}
\]

By the properties of \( f_1 \), we have \( x_1^0, y_1^0 \in \mathbb{N} \), \( x_1^0 \leq y_1^0 \leq r_i \), and \( f_1(z) = 0 \) iff \( z \in [x_1^0, y_1^0] \). Hence, the intervals \( [x_1^0, y_1^0] \) for \( a \in S \) partition \( [0, r_i) \), so the intervals \( [x_1^0, y_1^0] \) for \( a \in S \) partition \( [0, \sum r_i) \).

Choose \( a \in S \). We now show that \( g \) is linear over \( [0, \sum r_i) \). Choose \( y = (u_1, \ldots, u_k) \) such that \( g(u) = \sum f_1(u_i) \) and \( \sum u_i = u \). By the choice of \( u \) and using the facts that \( f_1'(z) < 0 \) iff \( z < x_1^0 \) and \( f_1'(z) > 0 \) iff \( z > y_1^0 \), it is straightforward to show that \( x_1^0 < u_1 \leq y_1^0 \) and \( \sum u_i = u \). Hence, \( f_1(u_i) = f_1'(x_1^0) + \sum (u_i - x_1^0) \). Summing over all \( i \), we get

\[
g(u) = \sum f_1(u_i) = \sum f_1(x_1^0) + \sum (u_i - x_1^0)
\]

a linear function of \( u \), as desired.

The convexity of \( g \) follows from the monotonicity of the \( f_1' \).

Examination of the proof of Theorem 2.2.5 allows us to sharpen Theorem 2.2.4 by stating that the minimum cost can always be achieved by placing whole resources on all vertices.
Theorem 2.2.6. Let $p$, $t$, $e$ be fixed. Let $e_1, \ldots, e_i$ be the immediate descendant edges of $e$, and let $u \in \mathbb{N}$. Then $\minexpcost_e(u) = \expflow_e(u) + \min \left\{ \frac{1}{i} \sum_{i=1}^{i} \minexpcost(u_i) \mid u_i \leq u \right\}$. 

2.3. Optimal Placement

We are interested in determining the "best" placement functions $s : V \rightarrow \mathbb{R}^+$ in the following sense. We say $s : V \rightarrow \mathbb{R}^+$ is optimal for $\phi$, $t$ provided $\text{total}(s) = t$ and $\text{expcost}(s) = \minexpcost$. The first characterization result follows immediately from Theorem 2.2.6 and shows that there are optimal $s$ which take on integer values only.

Theorem 2.3.1. For any probability function $\phi$ and any $t \in \mathbb{N}$, there exists $s : V \rightarrow \mathbb{R}$ which is optimal for $\phi$, $t$.

Proof. Theorem 2.2.6 essentially provides an algorithm for producing such $s$. For any edge $e$ of $T$ with immediate descendants $e_1, \ldots, e_i$, and any $u \in \mathbb{N}$, one determines values of $s$ for all nodes below $e$ by considering all possible decompositions $u = u_1 + \ldots + u_i$ with $u_i \leq u$ and $u_i \in \mathbb{N}$ for all $i$. For each such decomposition, one recursively determines values of $s$ for all nodes below each $e_i$, and corresponding costs. The decomposition with the smallest total cost is chosen.

In order to analyze the cost of determining an optimal placement as above, we do not perform a straightforward recursive analysis of the algorithm described in the proof of Theorem 2.3.1. Rather, we take advantage of repeated work in various recursive calls. During the algorithms, one must calculate $\expflow_e(u)$ for all $e \in E$ and all $u$, $0 \leq u \leq t$. The number of arithmetic operations involved in one calculation of $\expflow_e(u)$ is $O(t)$ (if performed judiciously), independent of $e$ and $u$. It is these costs which dominate the total count of arithmetic operations, so that an $O(t^2 \cdot |E|)$ analysis results. We summarize this discussion in the following theorem.

Theorem 2.3.2. There is an algorithm using $O(t^2 \cdot |E|)$ arithmetic operations which, for any tree $T = (E,V)$ probability function $\phi$, and $t \in \mathbb{N}$, determines a placement of whole resources which is optimal for $\phi$, $t$.

2.4. Optimizing Flow on Individual Edges

In this section, we show that the flow on each individual edge is optimized for a number equal to the median of an appropriate binomial distribution. We use this to bound the distance from optimal of two simple placements.

Let $n \in \mathbb{N} - \{0\}$, $0 < p < 1$. Let $x$ be a random variable whose value is the number of successes in $n$ independent trials, each of whose probability of success is $p$. Define $\text{median}(n,p)$ as the smallest $r \in \mathbb{N}$ such that $\Pr[x \leq r] \geq \frac{1}{2}$.

Theorem 2.4.1. Let $\phi$ be a probability function on $V$, $t \in \mathbb{N} - \{0\}$, $e \in E$. Then $\expflow_e(u)$ is minimized at $u = \text{median}(t,p)$, where $p = \sum_{v \in R(e)} \phi(v)$.

Proof. Write $f$ for $\expflow_e$. By Theorem 2.2.6, $f$ is minimized at an integer $u$ which is provided total$(s) = t$ and $\text{expcost}(s) = \minexpcost$. Theorem 2.3.1 provides total$(s) = t$ and $\text{expcost}(s) = \minexpcost$. Now $\text{median}(t,p)$ is the smallest $r \in \mathbb{N}$ with $f(r+1) - f(r) \leq 0$. Then $\text{median}(t,p)$ is the smallest $r \in \mathbb{N}$ with $2\Pr[x \leq r] > 1$. Thus, $u$ is the smallest $r \in \mathbb{N}$ with $2\Pr[x \leq r] > 1$ nonnegative, or $\Pr[x \leq r] = 1$; that is, $u = \text{median}(t,p)$.

The following theorem follows immediately from Theorem 0.2.7 and Corollary 0.3.1 of Jogdeo and Samuels [1]. It is also implicit in some earlier work by Uhlmann [2,3].

Theorem 2.4.2 (Jogdeo and Samuels). Let $n \in \mathbb{N}$, $n \geq 1$, $0 < p < 1$. Then $\text{median}(n,p) \in \{ \lfloor np \rfloor, \lceil np \rceil \}$.

Since $np$ is the mean number of successes in $n$ independent trials with success probability $p$, this theorem implies that the mean and median differ by less than one.

Thus, each edge individually has its flow optimized at a value which is either the mean rounded up or the mean rounded down. However, it is not always possible to achieve this local optimum consistently throughout the tree. In fact, in this very general setting, an optimal placement might require some subtree to contain a value other than the mean rounded up or down.
For any $T$, $\phi$, $t$, and for each edge $e \in E$, let $p_e = \frac{1}{\sum \phi(v)}$. $s: V \rightarrow K^+$ is an exact fair-share placement for $T$, $\phi$, $t$ if for each $e \in E$, $t_{\phi}(e) = t_{\phi}^e$. $s: V \rightarrow N$ is a fair whole placement for $T$, $\phi$, $t$ if for each $e \in E$, $\frac{1}{\sum \phi(v)} \leq \phi(v) \leq \frac{1}{\sum \phi(v)}$.

It is not difficult to see that for any $T$, $\phi$, $t$, an exact fair-share placement and a fair whole placement exist. Let $s(v) = t\cdot \phi(v)$ for $v \in V$. Then $s$ is an exact fair-share placement. Let $s'(v) = t\cdot \phi(v)$ if $v$ is a leaf. Let $s'(v) = 0$ if $v$ is the root. For $v$ not a leaf or root, let $e$ be the edge with $\text{low}(e) = v$, and let $e_1, \ldots, e_k$ be its immediate descendants. Let $s'(v) = \frac{1}{\sum \phi(v)} \cdot \sum_{i=1}^{k} t_{\phi}^e$. An easy induction shows that $s'$ is a fair whole placement for $T$, $\phi$, $t$, and in fact, $t_{\phi}(e,s') = t_{\phi}^e$ for every $e \in E$.

In the special case that $\phi$ is non-zero only on leaves, then there is a fair whole placement $s'$ which is non-zero only on leaves. It can be obtained by a simple recursive construction. Start at the root of $T$ with $t$ resources to distribute. Below any edge $e$ in the tree, we will have either $[t_{\phi}^e]$ or $[t_{\phi}^e]$ to distribute. Assume $e$ has descendants $e_1, \ldots, e_k$. Distribute $\frac{1}{k} [t_{\phi}^e]$ to the $i$th subtree. Since

$$\frac{1}{k} \sum_{i=1}^{k} t_{\phi}^e \geq \sum_{i=1}^{k} t_{\phi}^e \leq \sum_{i=1}^{k} t_{\phi}^e$$

the distribution is possible. We then proceed recursively on $e_1, \ldots, e_k$.

**Example 2.4.1.** Let $T$ be a 32-leaf complete binary tree (except for root edge of $T$), $\phi(i) = \frac{1}{256}$ for each of the leftmost 16 leaves $i$, $\frac{1}{256}$ for each of the rightmost 16 leaves, and 0 for all other nodes. Let $t = 16$. The placement $s$ which has $s(i) = 1$ for each of the leftmost 16 leaves $i$, and 0 elsewhere, has $t_{\phi}(e) = 16$ for $e$ the left descendant of root edge of $T$. However, the mean number of requests in the subtree below $e$ is $t \cdot \frac{15}{16} = 15$, so $s$ is not a fair whole placement. Nevertheless, one can determine by exhaustive searching that $s$ is optimal, and $\text{expcost}(s)$ is strictly smaller than $\text{expcost}(s')$ for any fair whole placement $s'$.

We note in general, however, that the optimal cost cannot be too much less than the cost of any exact fair-share placement or fair whole placement.

**Theorem 2.4.3.** Let $s$ be an exact fair-share placement or a fair whole placement for $T$, $\phi$, $t$. Then $\text{expcost}(s) \leq \text{minexpcost} + 1\cdot |E|$. 

**Proof.** By the results of this section, $\text{expflow}_e$ is minimized at some $u$, where $|t_{\phi}^e| \leq |t_{\phi}^e|$. Thus, $|t_{\phi}(e,s) - u| \leq 1$. But then $|\text{expflow}_e(t_{\phi}(e,s) - \text{expflow}_e(u)| \leq 1$, by calculations similar to those in the proof of Theorem 2.4.1. Since no placement can do better than the optimal on each edge, $s$ incurs at most an extra cost of 1 per edge.

For some choices of $T$, $\phi$, $t$, of course, it is possible to consistently achieve the median on each edge -- for example, if the mean is an integer for every edge. But in general, we have no global optimality results. In §4 we obtain optimal results for a special case.

**Example 2.4.2.** If $T$ is a complete binary tree with leaf vertices $L$, $|L| = 2^k$, $t = a \cdot 2^k$ for integer $a$, and $\phi$ the uniform distribution on the leaves, then the placement $s$ such that $s(v) = a$ for each $v \in L$ and $s(v) = 0$ for $v \in V - L$ is optimal, because it achieves the integer mean on each edge.

**Example 2.4.3.** Let $T$ be the tree depicted below, $\phi$ as indicated on the leaves and 0 elsewhere, and $t = 12$.

![Diagram of a tree with node labels and resource distribution](attachment:image.png)

Then the means are indicated on each edge below, so all resources can be placed at the levels indicated by the circled numbers.
2.5. Nodes with Zero Placements

We conclude this section with a somewhat surprising characterization theorem. It says that if there is a probability less than \( \frac{1}{2} \) of any request arriving in a subtree, then it is bad to place even a small fraction of a resource anywhere in that subtree.

**Theorem 2.5.1.** Let \( e \in E \) satisfy \( (1-p)^t > \frac{1}{2} \), where \( p = \sum_{v \in B(e)} \phi(v) \). Let \( s \) be optimal for \( \phi \) with \( s(v) = 0 \) for all \( v \in B(e) \).

Proof. Assume not, and fix \( e, s \) exhibiting the contrary. Choose \( e' \) below \( e \) to be a lowest edge for which \( x = s(\text{low}(e')) > 0 \). Define a new placement \( s' \), where \( s'(\text{low}(e')) = 0 \), \( s'(\text{high}(e')) = s(\text{high}(e')) + x \), and \( s'(v) = s(v) \) otherwise. We show that \( \text{expcost}(s') > \text{expcost}(s) \), contradicting the optimality of \( s \).

By Theorem 2.2.1, \( e' \) is the only edge for which \( s \) and \( s' \) have different expected absolute flows, so

\[
\text{expcost}(s) - \text{expcost}(s') = \text{expflow}_{e'}(\text{total}(e', s)) - \text{expflow}_{e'}(\text{total}(e', s'))
\]

Let \( r = \sum_{v \in B(e')} \phi(v) \). Applying Theorem 2.2.2, the difference is

\[
\sum_{i=0}^{\log x} \left( \frac{1}{i} \right) r^i (1-r)^{t-i} (i x - 1 - i)
\]

\[
x(2(1-r)^t - 1).
\]

Since \( (1-r)^t > (1-p)^t > \frac{1}{2} \), the difference \( > 0 \), giving the needed contradiction. \( \square \)

3. BOUNDS ON THE COSTS OF OPTIMAL PLACEMENTS

In this section, we assume that \( T \) is a complete (balanced) binary tree (except for \( \text{rootedge}(T) \)) with leaves \( L \), and let \( n = |L| \). Assume \( \phi \) is arbitrary, except that as always \( \phi(\text{root}) = 0 \). We show that optimal placements are much better than centralized placements; in fact, their expected cost is linear in the number of leaves of the tree.

### 3.1. Cost of Exact Fair-Share Placements

Since we do not have a direct characterization of optimal placements, we instead bound the expected cost of an arbitrary exact fair-share placement. This upper bound, of course, provides an upper bound for optimal placements. Moreover, by Theorem 2.4.1, the costs cannot differ by more than \( Jn \).

**Theorem 3.1.1.** Let \( T, \phi, L, n \) be as above. Let \( t \in N \), \( t \geq 1 \), and let \( s : V \rightarrow R^+ \) be an exact fair-share placement with \( \text{total}(s) = t \). Let \( e \in E \) and \( p_e = \sum_{v \in B(e)} \phi(v), \quad \phi = \sum_{e \in B(e)} \phi(e) \). Then

\[
\text{expcost}_e(s) \leq c \sqrt{tp_e}
\]

Here \( c \) is a fixed constant independent of the choice of \( T, \phi, t, s, \) or \( e \).

Before proving the theorem, we need some technical lemmas.

**Lemma 3.1.2.** For \( t \in N \), \( t \geq 1 \), \( s \geq t - 1 \), \( 0 \leq s \leq 1 \), it is the case that

\[
\sum_{i=0}^{\infty} \left( \frac{1}{i} \right) p_i^t (1-p)^{t-1} (tp-i) = t \sum_{i=0}^{\infty} \left( \frac{1}{i} \right) p_i^t (1-p)^{t-1} (tp-i).
\]

Proof. \( \square \)

**Lemma 3.1.3.** If \( t \in N, t \geq 1 \), \( 0 \leq s \leq 1 \), then

\[
\sum_{i=0}^{\infty} \left( \frac{1}{i} \right) p_i^t (1-p)^{t-1} (tp-i) = 2t \sum_{i=0}^{\infty} \left( \frac{1}{i} \right) p_i^t (1-p)^{t-1} (tp-i).
\]
Therefore,
\[
\begin{align*}
(1-p)^{t-1} & \leq \expflow_{e}(tp) \\
& \leq (1-p)^{t-1} \cdot \frac{tp}{\sqrt{2\pi tp}} \\
& \leq (1-p)^{t-1} \cdot \frac{1}{\sqrt{2\pi tp}} (tp) \\
& \leq (1-p)^{t-1} \cdot \frac{1}{\sqrt{2\pi tp}} (tp) \\
& = \sqrt{\frac{tp}{tp}} \cdot (1-p)^{t-1} \cdot \frac{tp}{\sqrt{2\pi tp}} (tp) \\
& = \sqrt{\frac{tp}{tp}} \cdot (1-p)^{t-1} \cdot \frac{tp}{\sqrt{2\pi tp}} (tp) \\
& \leq \expflow_{e}(tp).
\end{align*}
\]

But
\[
\begin{align*}
\frac{tp}{tp} & = \left(1 + \frac{tp}{tp} \right) \cdot \frac{tp}{\sqrt{2\pi tp}} (tp) \\
& \leq (1 + \frac{1}{tp}) \cdot \frac{tp}{\sqrt{2\pi tp}} (tp) \\
& \leq e, \quad \frac{tp}{tp} \leq 1
\end{align*}
\]

The lemma follows.

**Lemma 3.1.5.** Let $T$, $C$, $t$ be as in Theorem 3.1.1. Let $e \in E$ and let $p = \sum v \in B(e)$. Then $\expflow_{e}(tp) < b \cdot \sqrt{tp}$.

**Proof.** If $p = 0$ or $p = 1$, then $\expflow_{e}(tp) = 0$. Assume $0 < p < 1$. By Theorem 2.2.2 and Lemma 3.1.3,

\[
\begin{align*}
\expflow_{e}(tp) & = \sum_{i=0}^{t-1} \frac{(1-p)^{t-i} \cdot \frac{tp}{\sqrt{2\pi tp}} (tp) \cdot \left(\frac{tp}{tp} \right)}{\sqrt{2\pi tp}} (tp) \\
& = 2tp \cdot \frac{1}{\sqrt{2\pi tp}} (tp) \cdot \left(\frac{tp}{tp} \right) \\
& \leq 2 \cdot \frac{1}{\sqrt{2\pi tp}} (tp) \cdot \left(\frac{tp}{tp} \right) \\
& < b \cdot \sqrt{tp}.
\end{align*}
\]

On the other hand, if $tp \leq 1$, then

\[
\expflow_{e}(tp) = 2tp \cdot (1-p)^{t-1} \cdot \frac{tp}{\sqrt{2\pi tp}} (tp) < b \cdot \sqrt{tp}.
\]

In any case,

\[
\expflow_{e}(tp) < b \cdot \sqrt{tp}.
\]
Proof of Theorem 3.1.1. Define a function $G: N+ \rightarrow R^+$ recursively:

$G(0, u) = 6 \cdot \sqrt{u}$;

$G(k, u) = \max \{G(k-1, u_1) + G(k-1, u_2) + 6 \cdot \sqrt{u_1 + u_2} \mid u_1 + u_2 \leq u \}, \ k \geq 1$.

We first show by induction on $m$ that $\text{expcost}(s) \leq G(\log_2 m, t_{pe})$. Since $s$ is an exact fair-share placement, $\text{total}(e, s) = t_{pe}$.

$m=1$: Then $\text{low}(e)$ is a leaf, so

$$\text{expcost}_e(s) \leq 6 \cdot \sqrt{t_{pe}} \text{ by Lemma 3.1.5} = G(0, t_{pe}).$$

$m>1$: Then $m = 2^k$ since $T$ is a complete binary tree. Let $e_1, e_2$ be the two immediate descendant edges of $e$. Then

$$\text{expcost}_e(s) = \text{expflow}_e(\text{total}(e, s)) + \frac{2}{k} \text{expcost}_e(s_1),$$

by induction,

$$\text{expcost}_e(s) \leq G(k-1, t_{pe}), \ i=1, 2.$$  

Again using Lemma 3.1.5, we have

$$\text{expcost}_e(s) \leq 6 \cdot \sqrt{t_{pe}} + \sum_{i=1}^{2} G(k-1, t_{pe}) \leq G(k, t_{pe}) = G(\log_2 m, t_{pe})$$

since $t_{pe_1} + t_{pe_2} \leq t_{pe}$.

We complete the proof of the theorem by showing that

$$G(k, u) \leq c \cdot 2^{k/2} \cdot \sqrt{u}$$

for some constant $c$.

First observe that for each $k \in N$, $G(k, u)$, regarded as a function of $u$, is concave and monotonically increasing. This is clearly true for $k=0$. For $k \geq 0$, we know inductively that $G(k-1, u)$ is concave and monotonically increasing. Hence, $G(k-1, u_1) + G(k-1, u_2) \leq 2 \cdot G(k-1, (u_1 + u_2)/2)$ whenever $u_1 + u_2 \leq u$, so

$$G(k, u) = 2 \cdot G(k-1, u/2) + 6 \cdot \sqrt{u},$$

and $G(k, u)$ is concave and increasing.

We proceed to solve the recurrence equation $G(k, a \cdot 2^k) = 2 \cdot G(k-1, a \cdot 2^{k-1}) + 6 \cdot 2^{k/2} \cdot \sqrt{a}$.

Now, divide both sides by $2^k$ and express in terms of $G'$, where $G'(k, a) = G(k, a \cdot 2^k)/2^k$, to get

$$G'(k, a) = G'(k-1, a) + 6 \cdot \sqrt{a}.$$  

Also,

$$G'(0, a) = G(0, a) = 6 \cdot \sqrt{a}.$$  

Hence,

$$G'(k, a) = \sqrt{2} \sum_{i=0}^{k} \frac{6 \cdot \sqrt{a}}{2^{i/2}}.$$  

The series is convergent, so let $c = 6 \cdot \sqrt{a}$.

Then $G'(k, a) \leq 2^{-k/2}$. Substituting back, we get

$$G(k, u) = 2^k \cdot G'(k, u/2) \leq c \cdot 2^{k/2} \cdot \sqrt{u}$$

as desired.

We conclude that

$$\text{expcost}_e(s) \leq G(\log_2 m, t_{pe}) \leq c \cdot \sqrt{t_{pe}}.$$  

Corollary 3.1.6. Let $T, \ t, \ L, \ n, \ t$ be as in Theorem 3.1.1. Let $s: V \rightarrow R^+$ be an exact fair-share placement for $T$, $t$, and let $s': V \rightarrow N$ be a fair whole placement for $T$, $t$, $t$. Then

$$\text{expcost}(s) \leq c \cdot \sqrt{nt}$$

and

$$\text{expcost}(s') \leq c \cdot \sqrt{nt} + 2n - 1.$$  

Proof. The first bound comes from a direct application of Theorem 3.1.1 to $\text{rootedge}(T)$. The second bound follows from the first using Theorem 2.4.3 and that fact that an $n$-leaf binary tree has at most $2n-1$ edges (including $\text{rootedge}(T)$). 

3.2. Cost of Centralized Placement

$s$ is a centralized placement if $s(v) = 0$ everywhere except at $\text{low}(\text{rootedge}(T))$, and $s(\text{low}(\text{rootedge}(T))) = t$. For such an $s$,

$$\text{expcost}(s) = t \cdot \log_2(n).$$

Note that the ratio of expected cost for a centralized placement to a fair whole placement is at least

$$\frac{t \cdot \log_2(n)}{c \cdot \sqrt{nt} + 2n}.$$  

When $t$ is small relative to $n$, then the centralized placement is superior, for reasons similar to those discussed in Section 2.5.
However, for \( t = \Omega(n) \), the fair whole placement is better by a factor of \( \Omega(\log n) \), and for \( t \gg n \), the ratio approaches \( n^2 \), that is, the centralized placement is worse by a square.

Note also that for \( t = \Omega(n) \), the fair whole placement has linear expected cost, so the expected cost per request is a constant, whereas the cost per request for a centralized placement is \( \log^2 n \).

4. OPTIMAL PLACEMENTS FOR SPECIAL DISTRIBUTION

In this section, we give several characterization results and algorithms for optimal placements for a further restriction of the case considered in Section 3 in which we assume \( \phi \) is the same on all leaves and zero elsewhere. Specifically, we assume the following for this section:

(a) \( T = (V, E) \) is a complete binary tree with leaves \( L \subseteq V \), \( n = |L| \), except that, as before, the root has a single emanating edge, \( \text{rootedge}(T) \).

(b) \( \phi(v) = 1/n \) for all \( v \in L \), and \( \phi(v) = 0 \) for all \( v \in V \setminus L \).

(c) \( t \in \mathbb{N}, \ t \geq 1 \).

We define the level of a vertex to be its distance from the root. By our conventions regarding \( \text{rootedge}(T) \), the leaves are at level \( \log_2 n + 1 \).

For the special case being considered in this section, certain of the relevant definitions can be generalized to reflect the symmetry in the tree. For example, if \( e_1 \) and \( e_2 \) are edges with \( \text{high}(e_1) \) and \( \text{high}(e_2) \) at the same level \( k \), then \( \text{expflow}_{e_1} = \text{expflow}_{e_2} \). Therefore, we write \( \text{expflow}_k \) in place of either. Similarly, we write \( \text{minexpcost}_k \) for \( \text{minexpcost}_{e_k} \), where \( k \) is the level of \( \text{high}(e) \).

4.1. A Bound on Levels with Nonzero Placements

Theorem 2.5.1 can be used to bound the level at which nonzero placement can occur in an optimal placement.

Theorem 4.1.1. Let \( v \in V \) be at level \( k \geq \lceil \log_2 t \rceil + 2 \). Let \( s \) be optimal for \( \phi, t \). Then \( s(v) = 0 \).

Proof. By Theorem 2.5.1, it suffices to show that \( (1 - \frac{1}{2^{\lceil \log_2 t \rceil + 1}}) t > \frac{1}{2} \) for \( t \in \mathbb{N} \). It also suffices to consider \( t \) a power of 2. In this case, the inequality is just \( (1 - \frac{1}{2^k} t > \frac{1}{2} \), which follows by an easy induction on \( \log_2 t \).

Let \( T_t = (V_t, E_t) \) denote the tree consisting of the vertices of \( T \) at levels from 0 to \( \lceil \log_2 t \rceil + 1 \) inclusive and the edges of \( T \) between them. The leaves \( L_t \) of \( T_t \) are the vertices at levels \( \log_2 n - \lceil \log_2 t \rceil \). If \( s: V \rightarrow \mathbb{R}^+ \) is such that \( s(v) = 0 \) for all \( v \in V \) at levels below \( \lceil \log_2 t \rceil + 1 \), then \( s_t: V_t \rightarrow \mathbb{R}^+ \) can be defined from \( s \) by simply ignoring the missing vertices. Then it is easy to see that \( \text{expcost}(T)(s) = \text{expcost}(T_t)(s_t) + t \cdot (\log_2 n - \lceil \log_2 t \rceil) \).

(Superscripts distinguish the trees under consideration.) We then have the following.

Theorem 4.1.2. If \( t \gg 1 \), then \( \text{minexpcost} = \text{minexpcost}_t + t \cdot (\log_2 n - \lceil \log_2 t \rceil) \).

Proof. (2) follows from Theorem 4.1.1 and the remarks.

(3) follows because any placement for \( T_t \) can be augmented to a placement for \( T \) by placing zeros on the additional nodes, thereby incurring the stated cost.

Thus, in the remainder of this section, we assume that the maximum level in \( T \) is at most \( \lceil \log_2 t \rceil + 1 \) (that is, \( n < 2t \)). The reader can then use Theorem 4.1.2 to infer corresponding results about cases where \( n \geq 2t \).

Example 4.1.1. The case where \( t = 1 \) is somewhat peculiar. The reader can verify that for all \( T \) with maximum level number at least 2, the following pictures all represent optimal placements.
This is the only value of \( t \) for which an optimal placement can have nonzero values below level \( \lceil \log t \rceil + 1 \).

**Example 4.1.2.** If \( t = 2^k \) and \( n > t \), then the placement with one resource on each of the \( t \) vertices at level \( k+1 \) is optimal: nothing is placed below this level, and an optimal placement for the whole tree results from an optimal placement within levels up to \( \log t + 1 \). But clearly putting one resource across all leaves of \( T_t \) is optimal, as seen in Example 2.4.2.

### 4.2. Algorithms for Finding Optimal Placements

In this section, we prove two sharper versions of Theorem 2.2.4 for the special case of this section, and use them as bases for two algorithms for finding optimal placements.

**Theorem 4.2.1.** Let \( t \in \mathbb{N} \), \( u \in \mathbb{R}^k \), and \( k \in \mathbb{N} \), \( k \leq \log_2 n - 1 \). Then \( \text{minexpcost}_k(u) = \text{expflow}_k(u) + 2 \cdot \min \{ \text{minexpcost}_{k+1}(u') : u' \in \{0,1, \ldots, \frac{1}{2}, \frac{3}{2} \} \} \).

**Proof.** \( s \) is clear. We show \( z \). Write \( f \) for \( \text{minexpcost}_{k+1} \). Consider any \( u_1, u_2 \in \mathbb{R}^k \) with \( u_1 + u_2 \leq u \) and \( f(u_1) + f(u_2) \) minimal. We will produce \( u' \in \{0,1, \ldots, \frac{1}{2}, \frac{3}{2} \} \) with \( 2f(u') \leq f(u_1) + f(u_2) \).

By Theorem 2.2.5, there exists \( r \in \mathbb{N} \) such that \( r \) is the smallest element of \( \mathbb{N} \) with \( f(r) \leq f(s) \) for all \( s \geq r \). We consider two cases.

**Case 1.** \( u \geq 2r \).

Choose \( u' = r \). Then \( u' \in \{0,1, \ldots, \frac{1}{2}, \frac{3}{2} \} \) and \( 2f(u') \leq f(u_1) + f(u_2) \).

**Case 2.** \( u < 2r \).

Then \( u_1 + u_2 = u \). Choose \( u' = \frac{u}{2} \). Then \( 2f(u') = 2f(\frac{u_1 + u_2}{2}) \leq f(u_1) + f(u_2) \) by convexity of \( f \).

An appeal to Theorem 2.2.4 completes the proof.

**Theorem 4.2.2.** For any \( t \in \mathbb{N} \), there exists \( s \) such that \( s(v) = s(v') \) for all pairs of vertices \( v, v' \) at the same level, which is optimal for \( t \).

**Proof.** \( s \) can be found by a recursive algorithm based on Theorem 4.2.1.

We proceed as before to analyze the cost of finding the placement of Theorem 4.2.2. During the algorithm, one must calculate \( \text{expflow}_0(t), \text{expflow}_1(t), \ldots, \text{expflow}_{\log_2 n}(t) \). In addition, one must calculate \( \text{expflow}_k(i) \) for all \( i \in \mathbb{N} \), \( i \leq \frac{t}{2} \), \( \text{expflow}_k(\frac{t}{2^i}) \) for all \( i \in \mathbb{N} \), \( i \leq 5 \) for all \( i \in \mathbb{N} \). This is a total of \( O(t \log n) \) \( \text{expflow} \) computations.

Note that the bound of Theorem 4.2.3 represents an improvement over the bound in Theorem 2.3.2 applied to the special case of this section; the placements produced by the two algorithms have somewhat different properties, however. The remainder of this subsection deals with integral placements, the situation considered in Theorem 2.3.2.

**Theorem 4.2.4.** Let \( t, u, k \in \mathbb{N} \), \( k \leq \log_2 n - 1 \).

Then \( \text{minexpcost}_k(u) = \text{expflow}_k(u) + \min \{ \text{minexpcost}_{k+1}(u'_1) : u'_1, u'_2 \in \mathbb{N}, u'_1 + u'_2 \leq u, \text{ and } \left| u_2 - u_1 \right| = 1 \} \).

**Proof.** Again, we show \( s \). Write \( f \) for \( \text{minexpcost}_{k+1} \). By Theorem 4.2.1, there is a value \( u' \in \{0,1, \ldots, \frac{1}{2}, \frac{3}{2} \} \) such that \( \text{minexpcost}_k(u) = \text{expflow}_k(u) + 2f(u') \). If \( u' \in \{0,1, \ldots, \frac{1}{2}, \frac{3}{2} \} \), then we simply take \( u'_1 = u'_2 = u' \). If \( u' = \frac{\sqrt{3}}{2} \), then we take \( u'_1 = \frac{\sqrt{3}}{2} \) and \( u'_2 = \frac{3}{2} \). In this case, piecewise linearity of \( f \) guarantees the required properties.

**Theorem 4.2.5.** For any \( t \in \mathbb{N} \), there exists \( s : V \times N \) which is optimal for \( t \). Moreover, \( s \) has the additional properties:

(a) If \( e_1 \) and \( e_2 \) are two edges with \( \text{high}(e_1) = \text{high}(e_2) \), then \( |\text{total}(e_1, s) - \text{total}(e_2, s)| \leq 1 \),

(b) If \( e \) is any edge with \( \text{high}(e) \) at level \( k \), then \( \text{total}(e, s) \leq \frac{\sqrt{3}}{2^k} \).
Proof. Theorem 4.2.4 leads naturally to a recursive algorithm yielding a placement in $V + N$ and obviously satisfying (a). To see that (b) is also satisfied, assume the contrary, and let $e$ be a highest edge in the tree with $\text{total}(e, s) > \left\lceil \frac{t}{2^{k+1}} \right\rceil$, where $\text{high}(e)$ is at level $k$.

Since $\text{total}(e, s) \in N$, we have $\text{total}(e, s) \geq \left\lceil \frac{t}{2^{k+1}} \right\rceil$. If $e$ is not rooted(T), let $e'$ be the edge with $\text{high}(e') = \text{high}(e)$, and let $e''$ be the edge immediately above $e$ in $T$. Then

$$\text{total}(e'', s) \leq \left\lceil \frac{t}{2^{k-1}} \right\rceil - \left\lceil \frac{t}{2^{k+1}} \right\rceil + 1.$$ 

Therefore,

$$\text{total}(e', s) \leq \text{total}(e'', s) - \text{total}(e, s) < \left(\frac{t}{2^{k-1}}\right) - \left(\frac{t}{2^{k+1}}\right) + 1.$$

But then

$$|\text{total}(e, s) - \text{total}(e', s)| > 1,$$

contradicting property (a).

Once again, we analyze the cost of determining an optimal placement with the properties of Theorem 4.2.5. One must calculate $\expflow(t)$, and also $\expflow(i, t)$ for all $i \leq \log(n)$, $i \leq \left\lceil \frac{t}{2^{k+1}} \right\rceil$, $\expflow_{\log(n)}(i)$ for all $i \leq \log(n)$, $i \leq \left\lceil \frac{t}{2^{k+1}} \right\rceil$. This is a total of $O(t)$ expflow computations, (since we are assuming that $n$ is $O(t)$).

Theorem 4.2.6. There is an algorithm using $O(t^2)$ arithmetic operations, which for any tree $T$ and for any $t \in N$, determines an $s : V + N$ which is optimal for $t$, and which satisfies conditions (a) and (b) of Theorem 4.2.5.

4.3. A Fast Algorithm for Determining Optimal Placements

The results of Section 2.4 can be used to prune the algorithm's search space still further, leading to a much faster algorithm.

Theorem 4.3.1. Let $t, u, k \in N$,

$$k \leq \log(n) - 1, \quad \left\lceil \frac{t}{2^k} \right\rceil \leq u \leq \left\lceil \frac{t}{2^k} \right\rceil.$$ 

Then

$$\minexpcost_k(u) = \expflow_k(u)$$

$$+ \min\{\minexpcost_{k+1}(u_1) + \minexpcost_{k+1}(u_2)| u_1, u_2 \leq \left\lceil \frac{t}{2^{k+1}} \right\rceil \text{ and } u_1 + u_2 \leq s\}.$$ 

Proof. Again, we show $x$. Write $f$ for $\minexpcost_{k+1}$. By Theorem 4.2.4, there is a pair $u_1, u_2 \in N$ such that $u_1 + u_2 \leq u$, $|u_2 - u_1| \leq 1$ and $\minexpcost_{k+1}(u) = \expflow_{k+1}(u) + f(u_1) + f(u_2)$. Of all such pairs, choose one minimizing $u - (u_1 + u_2)$ and assume $u_1 \leq u_2$. We must check that $u_1, u_2 \leq \left\lceil \frac{t}{2^{k+1}} \right\rceil$, and obviously satisfying (a). To see that (b) is also satisfied, assume the contrary, and let $e$ be a highest edge in the tree with $\text{total}(e, s) > \left\lceil \frac{t}{2^{k+1}} \right\rceil$, where $\text{high}(e)$ is at level $k$.

If $u_2 > \left\lceil \frac{t}{2^{k+1}} \right\rceil$, then $u_2 > \left\lceil \frac{t}{2^{k+1}} \right\rceil + 1$, so $u_1 > \left\lceil \frac{t}{2^{k+1}} \right\rceil - 1$. Then $u_2 \geq u_1 + u_2 > \left\lceil \frac{t}{2^{k+1}} \right\rceil + 1 > \left\lceil \frac{t}{2^k} \right\rceil$, a contradiction. Thus, $u_2$ (and therefore $u_1$) satisfies the conditions used in choosing $u_1, u_2$. We must check

$$\minexpcost_{k+1}(u_1) = \expflow_{k+1}(u_1) + f(u_1) + f(u_2).$$

By Theorem 4.2.4, there is a pair $u_1, u_2 \in N$ such that $u_1 + u_2 \leq u$, $|u_2 - u_1| \leq 1$ and $\minexpcost_{k+1}(u) = \expflow_{k+1}(u) + f(u_1) + f(u_2)$. Of all such pairs, choose one minimizing $u - (u_1 + u_2)$ and assume $u_1 \leq u_2$. We must check that $u_1, u_2 \leq \left\lceil \frac{t}{2^{k+1}} \right\rceil$, and obviously satisfying (a). To see that (b) is also satisfied, assume the contrary, and let $e$ be a highest edge in the tree with $\text{total}(e, s) > \left\lceil \frac{t}{2^{k+1}} \right\rceil$, where $\text{high}(e)$ is at level $k$.

If $u_2 > \left\lceil \frac{t}{2^{k+1}} \right\rceil$, then $u_2 > \left\lceil \frac{t}{2^{k+1}} \right\rceil + 1$, so $u_1 > \left\lceil \frac{t}{2^{k+1}} \right\rceil - 1$. Then $u_2 \geq u_1 + u_2 > \left\lceil \frac{t}{2^{k+1}} \right\rceil + 1 > \left\lceil \frac{t}{2^k} \right\rceil$, a contradiction. Thus, $u_2$ (and therefore $u_1$) satisfies the conditions used in choosing $u_1, u_2$. We must check

$$\minexpcost_{k+1}(u_1) = \expflow_{k+1}(u_1) + f(u_1) + f(u_2).$$
Theorem 4.3.3. There is an algorithm using $O(t \log n)$ arithmetic operations which for any tree $T$ and for any $t \in \mathbb{N}$, determines a fair whole placement $s$ which is optimal for $t$.

4.4. Example

Some of the optimal placements discovered by our algorithms are rather unexpected. For example, the following represents an optimal placement of 11 resources in a balanced binary tree with uniform probability distribution:

![Diagram of a balanced binary tree with 11 resources placed optimally.]

Acknowledgements

The authors thank Carl Spruill, Charles Blair and Mike Paterson for contributing their ideas and suggestions for some of the results in this paper.

References

DISTRIBUTION LIST
Office of Naval Research Contract N00014-80-C-0221
Michael J. Fischer, Principal Investigator

Defense Documentation Center
Cameron Station
Alexandria, VA 22314
(1 copy)

Office of Naval Research
800 North Quincy Street
Arlington, VA 22217

Dr. R. B. Grafton, Scientific
Officer (1 copy)
Information Systems Program (437)
(2 copies)
Code 200 (1 copy)
Code 455 (1 copy)
Code 458 (1 copy)

Office of Naval Research
Branch Office, Pasadena
1030 East Green Street
Pasadena, CA 91106
(1 copy)

Naval Research Laboratory
Technical Information Division
Code 2627
Washington, D.C. 20375
(6 copies)

Office of Naval Research
Resident Representative
University of Washington, JD-27
422 University District Building
1107 NE 45th Street
(1 copy)

Dr. A. L. Slafkosky
Scientific Advisor
Commandant of the Marine Corps
Code RD-1
Washington, D.C. 20380
(1 copy)

Naval Ocean Systems Center
Advanced Software Technology Division
Code 5200
San Diego, CA 92152
(1 copy)

Mr. E. H. Gleissner
Naval Ship Research and
Development Center
Computation and Mathematics Department
Bethesda, MD 20084
(1 copy)

Captain Grace M. Hooper (008)
Naval Data Automation Command
Washington Navy Yard
Building 166
Washington, D.C. 20374
(1 copy)

Defense Advanced Research Projects Agency
ATTN: Program Management/MIS
1400 Wilson Boulevard
Arlington, VA 22209
(3 copies)

Professor Nancy A. Lynch
School of Information and Computer Science
Georgia Institute of Technology
Atlanta, GA 30332
(1 copy)

Professor Nancy Griffeth
School of Information and Computer Science
Georgia Institute of Technology
Atlanta, GA 30332
(1 copy)

Dr. Leo J. Guibas
Xerox Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, CA 94304
(1 copy)

Professor Philip Enslow
School of Information and Computer Science
Georgia Institute of Technology
Atlanta, GA 30332
(1 copy)

Office of Naval Research
Branch Office, Chicago
536 South Clark Street
Chicago, IL 60605
(1 copy)

Dr. John Cherniavsky
Program Director
Theoretical Computer Science
National Science Foundation
Washington, D.C. 20550
(2 copies)

Department of the Army
U.S. Army Research Office
P.O. Box 12211
Research Triangle Park, NC 27709
(1 copy)