MIT/LCS/TM-278

PROBABILISTIC ANALYSIS OF A NETWORK RESOURCE ALLOCATION ALGORITHM

Michael J. Fischer

Nancy Griffeth

Leonidas J. Guibas

Nancy A. Lynch

June 1985

Probabilistic Analysis of a Network Resource Allocation Algorithm

Nancy A. Lynch
Massachusetts Institute of Technology
Cambridge, Massachusetts

Nancy D. Griffeth
Georgia Institute of Technology
Atlanta, Georgia

Michael J. Fischer
Yale University
New Haven, Connecticut

Leonidas J. Guibas
Stanford University
Stanford, California,-and
DEC/SRC
Palo Alto, California

April, 1985

ABSTRACT

A distributed algorithm is presented, for allocating a large number of identical resources (such as airline tickets) to requests which can arrive anywhere in a distributed network. Resources, once allocated, are never returned. The algorithm searches sequentially, exhausting certain neighborhoods of the request origin before proceeding to search at greater distances. Choice of search direction is made nondeterministically.

Analysis of expected response time is simplified by assuming that the search direction is chosen probabilistically, that messages require constant time, that the network is a tree with all leaves at the same distance from the root, and that requests and resources occur only at leaves. It is shown that the response time is approximated by the number of messages of one that are sent during the execution of the algorithm, and that this number of messages is a nondescreasing funtion of the interarrival time for requests. Therefore, the worst case occurs when requests come in so far apart that they are processed sequentially.

The expected time for the sequential case of the algorithm is analyzed by standard techniques. This time is shown to be bounded by a constant, independent of the size of the network. It follows that the expected response time for the algorithm is bounded in the same way.

Keywords: Resource allocation, distributed algorithm, probabilistic algorithm

This work was supported in part by the Office of Naval Research under Contract N00014-82-K-0154, by the Office of Army Research under Contracts DAAG29-79-C-0155 and DAAG29-84-K-0058, by the Army Institute Research in Management Information and Computer Systems under Contract DAAK70-79-D-0087, by the National Science Foundation under Grants MCS-7924370, MCS-8116678, MCS-8200854, MCS-8306854, and DCR-8302391, and by the Defense Advanced Research Projects Agency (DARPA) under Grant N00014-83-K-0125.

List of Symbols:

$\in$	element of
$\|...\|$	absolute value of
$\varnothing$	empty set
$-$	minus
$<$	less than
$>$	greater than
$\leq$	less than or equal to
$\geq$	greater than or equal to
Σ	summation
$\cup$	union
$\{\}$	set
$\neq$	is not equal to
$:=$	assignment
$\rightarrow$	right arrow (function mapping, limit)
φ	Greek letter phi
$\mathcal{C}$	capital script C
$[...]$	brackets
$/$	division
∞	infinity
α	Greek letter alpha
x	multiplication sign
$\surd$	radical sign

1. Introduction

We consider the problem of allocating a number of identical resources to requests arriving at the sites of a distributed network. We assume that the network is configured as a tree. The nodes of the tree are processors and the edges are communication lines connecting the processors. Processes at a node may communicate only over the tree edges, with processes at other nodes. Resource allocation is managed by a collection of communicating resource allocation processes, one at each node. We will henceforth refer only to the node, identifying it with both the processor and the resource allocation process at the node.

From time to time, a request arrives at a node (potentially any node of the network) from the outside world. One of the resources should eventually be granted to the request, subject to the following conditions:

1. No resource is granted more than once. (Once granted, a resource is not returned. Thus, there is no legitimate reason to grant it more than once.)

2. At most one resource is granted to each request.

3. A node grants resources only to those requests which arrive at that node.

4. If the number of requests is no greater than the number of resources, then each request eventually receives a resource.

5. If the number of resources is no greater than the number of requests, then each resource is eventually granted to a request.

For convenience in describing allocation of specific resources to specific requests, we assume that each resource and request has a unique identifier.

The execution model for this distributed network is event-based. Two types of events may occur at a node: (1) a request may arrive from the outside world, and (2) a message may arrive from a neighbor in the tree. Each event triggers an indivisible step at the node. This step may include changing state, sending messages to other nodes, and granting resources to requests. (We ignore the time involved in this local processing when we measure the response time, considering only the communication time.) We assume that the communication lines are reliable, that is, each message is delivered exactly once. However, we do not make any assumptions about the order of message arrivals.

There are many interesting approaches to solving this resource allocation problem. In a centralized approach, all resources are controlled by a single central node. When a request arrives at a possibly different node, a "buyer" is commissioned, who travels, via messages, to the central node

to obtain a resource. The buyer then carries the resource back to the node where the request originated, so that the resource can be granted to the request at that location.

An alternative approach is to decentralize control of the resources, giving each node of the network control of some of them. In this approach, the buyers must search for the resources. An important choice to be made in designing an efficient search strategy is the choice between sending only one buyer to search for resources for each request and sending several buyers in parallel to search different parts of the tree. The former search strategy, which we call the sequential search strategy, avoids a number of problems arising from the parallel strategy, such as what to do about other buyers when one of them has found a resource. The next choice, if the sequential search strategy is used, is the choice of direction to search the tree. A good choice would involve guessing which nodes are most likely to have free resources when the buyer arrives at them.

Other strategies involve combining a decentralized search strategy with a dynamic resource redistribution strategy, letting resources search for requests (rather than vice versa), or giving nodes control of fractions of resources rather than whole resources.

One complexity measure which is useful for evaluating different strategies is the expected response time. This is a measure upon which any of the design choices could have a major impact. For example, the response time when using a centralized strategy must depend strongly on the network size. However, the decentralized strategies have the potential of depending on this size to a lesser extent.

In the first half of this paper, we present an algorithm for solving this resource allocation problem. Our algorithm is a decentralized solution in which each node controls some whole number of resources. A sequential search strategy is used, in which the direction to be searched is chosen nondeterministically. Certain neighborhoods of the node at which a request originates are exhausted before the search proceeds to more distant neighborhoods.

In order to gain some insight into the expected response time for our algorithm, we simulated its behavior, in some special cases. The nondeterministic choice of search direction was resolved by using a probabilistic choice, where the probabilities for the different directions depended on the initial placement of resources in those directions. We assumed an exponential distribution for time of arrival of requests, a uniform distribution for arrival location, and a normal probability distribution for message delivery time. We also assumed that all leaves of the network tree were at the same distance from the root, and that requests and resources occurred only at leaves. We first noted that expected response time was extremely good, with an upper bound that seemed to be independent of the size of the network. This was in marked contrast to a centralized algorithm. Next, we made a surprising observation: the expected response time appeared to be a nondecreasing function of the expected

interarrival time for requests. If true, this observation would imply that the worst case for the algorithm was actually the case where requests come in so far apart that they are processed one at a time. This observation contradicted our preliminary intuitions about the algorithm: we had thought that the worst cases would arise when there was greatest competition among requests searching for resources.

Using these observations as hints, we were able to carry out a substantial amount of analysis of the algorithm's behavior, and this analysis comprises the second half of this paper. Namely, we prove an upper bound on the expected response time for a special case in which, among other restrictions, all leaves of the network tree are at the same distance from the root, and requests and resources occur only at leaves. First, we show that the response time can be bounded in terms of the number of messages of one type that are sent during the execution of the algorithm. Then we show that this number of messages is a nondecreasing function of the interarrival time for requests. Therefore, the worst case occurs when requests come in so far apart that they are processed sequentially. We analyze the expected time for the sequential case, showing it to be bounded by a constant, independent of the size of the network. It follows that the expected response time for the algorithm is also bounded by a constant.

Although the expected response time for our algorithm is very good, we do not claim that it is optimal. In fact, there are some simple changes that one would expect to yield improvements. Unfortunately, with these changes, the algorithm can no longer be analyzed using the same techniques; thus, we are not really certain that they are improvements at all.

There are several contributions in this paper. First, we think that the algorithm itself is interesting. Second, we have identified an interesting criterion for the performance of a distributed algorithm: that the performance be independent of the size of the network. Satisfying this criterion seems to require an appropriate, decentralized style of programming. Third, the analysis is decomposed in an interesting way: a sequential version is analyzed using traditional methods, and the performance of the concurrent algorithm is shown to be bounded in terms of the sequential algorithm. It is likely that this kind of decomposition will prove to be useful for analysis of other distributed algorithms. For instance, a similar decomposition was used in the proof of correctness of a systolic stack [Guibas, and Liang (1982)].

The contents of the rest of the paper are as follows. Section 2 contains the algorithm, and Section 3 contains arguments for its correctness. Sections 4-6 contain the analysis of the algorithm. Section 4 proves the monotonicity result, which implies that the sequential case of the algorithm is worst. Section 5 analyzes the sequential case. Section 6 pulls together the results of Sections 4 and 5, thus giving a general upper bound. Finally, Section 7 describes some remaining questions.

2. The Algorithm

In this section, we present our algorithm. We begin with an informal decription, followed by a more formal presentation.

2.1. Informal Description

We assume that the network is a rooted tree.

Our algorithm is a decentralized algorithm with a sequential searching strategy. Requests send buyers to search for resources. When a buyer finds a resource, it "captures" it. Each captured resource travels back to the origin of this buyer (or possibly some other buyer, if there is interference between the processing of concurrent requests), so that the grant can occur where the request originated.

When a request or buyer arrives at any node, any free resource at the node is captured. If there are no free resources there, a buyer is sent to a neighboring node, determined as follows. Each node keeps track of the latest estimate it knows, for the number of resources remaining in each of its subtrees. Each node sends a message informing its parent of each new request which has originated within the child's subtree. The estimate which a node keeps for the number of resources remaining in a subtree, is calculated from the initial placement of resources in that subtree, the number of requests which are known to have originated within that subtree, and the number of buyers which the node has already sent into that subtree. In order to decide on the direction in which to send a buyer, a node uses the following rules. First, it never sends a buyer out of its subtree if it estimates that its subtree still contains a resource. Second, it only sends a buyer downward to a child if it estimates that the child's subtree contains a resource. Third, if there is a choice of child to which to send the buyer, the node makes a nondeterministic choice. (Later, we will constrain this decision to a probabilistic choice using a particular random choice function. This constraint will be important for the complexity analysis, but is not needed for the correctness of the algorithm.)

It is easy to see that any subtree which a node considers to contain no resources, actually contains no resources. Thus, no buyer is ever sent out of a subtree actually containing a resource. On the other hand, the perceived information about the availability of a resource in a child's subtree can be an overestimate, in case of interference among concurrent requests.

EXAMPLE

Suppose that request A enters at the node shown below, and its buyer travels upward until it reaches an ancestor that perceives the availability of a resource in one of its subtrees. Then the buyer travels downward toward that resource. Shortly before A's buyer reaches the resource, another request, B, arrives at the node shown. Suppose B's buyer reaches the resource and captures

it before A's buyer does. When (or before) A's buyer finally arrives at the resource's location, it will encounter the information that the resource is no longer there. Then A's buyer will be sent upward, backtracking in its search for a resource.

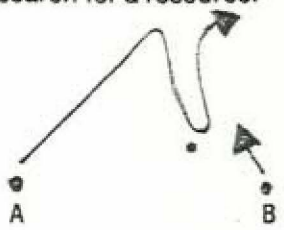

Figure 1

Although such interference can cause backtracking, the buyer will eventually find a resource if one exists. This is because no buyer ever leaves a subtree actually containing a resource.

Several optimizations are incorporated into the algorithm, as follows.

1. Buyers, unlike requests, need not be uniquely identified. Instead, each node keeps track of the number of buyers received and sent and the net flow of buyers over each of its incident edges. Captured resources then travel in such a way as to negate net flow of buyers, and because a buyer will eventually leave a subtree which does not contain a resource.

2. Buyers can travel "discontinuously". Assume node v sends a buyer to a child node w, thinking that there is an available resource in w's subtree. Assume that, soon thereafter, v receives a message from w, informing v of an arrival of a new request in w's subtree, and implying that v's previous supposition of an available resource was false. Then v knows that w will eventually send some buyer back up to v, at which time v should send the buyer in another direction. Since v knows this will eventually occur, v need not actually wait for the buyer to arrive from w; it can create a new buyer and send it in anticipation of the later return of the first buyer. Since the first buyer will not find any free resources in the subtree, this extra parallelism does no harm. In fact, with this optimization, it is no longer necessary for w to return the buyer at all, since v must ignore it when it returns to it in any case.

3. If each node knows how many resources were initially placed in each of its children's subtrees, then it is not even necessary for explicit buyers to be sent upward at all! All that is necessary is for nodes to send "ARRIVAL" messages upward to their parents, informing them of the arrival of new requests in their subtree. The parent is able to deduce the number of resources which the child would like to have sent down (i.e. the number of buyers emanating from the child's subtree), from the initial number of resources in the subtree, the number of arrivals in the subtree and the number of buyers already sent down into the subtree. We will say more in a moment about how this deduction is made.

If information about newly-arrived requests (in the form of "ARRIVAL" messages) only flows upward in the tree, there is no way that a child can deduce that its parent would like it to send a resource upward. Thus, it is still necessary to send explicit buyers downward. Let us designate these explicit downward buyer messages as "BUYER" messages. Thus, the algorithm only uses two kinds of messages to search for resources: "ARRIVAL" messages flow upward to inform parents about new requests, and "BUYER" messages flow downward to inform a child that its parent would like the child to send up a resource.

The precise deduction which a parent can make about the number of buyers emanating from a child's subtree is as follows.

Let a be the number of "ARRIVAL" messages which have been received by the parent from the child. Let b be the number of "BUYER" messages which have been sent by the parent to the child. Let p be the number of resources initially placed in the child's subtree. Then the number of buyers perceived as emanating from a child's subtree is max(a + b - p, 0). This number is called the estimate of "virtual buyers" emanating from the subtree.

That is, if the total number of "ARRIVAL" and "BUYER" messages indicated above is no greater than the initial placement, no buyers are perceived as emanating from the subtree. On the other hand, if this total is greater, then the excess is perceived to be the number of buyers.

Analogously, the child node deduces an estimate of the number of "virtual buyers" it has sent out of its subtree, as follows. Let a be the number of "ARRIVAL" messages which the child has sent to its parent. Let b be the number of "BUYER" messages which have been received by the child from its parent. Let p be the number of resources initially placed in the child's subtree. Then the number of buyers the child perceives that it has sent out of its subtree (also called the estimate of "virtual buyers" sent out of the subtree) = max(a + b - p, 0). Because of message delays, the child and the parent may differ on their estimates of the number of virtual buyers.

In order to make the actual grants to specific requests, it seems necessary that each specific identified resource "travel" to a point of request origin, in order to get properly paired with a request. This travel requires a third kind of message to be sent around, namely, a specific "captured resource". The algorithm which sends resources around is particularly simple - resources are just sent in such a way as to negate the net flow of buyers. This part of the algorithm executes concurrently with, but has no effect on, the searching part.

2.2. Formal Description

In this subsection, we present a program implementing the algorithm described above. A sketch of a correctness proof is presented in the next section. Primarily, the proof consists of showing the correctness of the invariant assertions made at various points in the program. The reader may wish to examine the proof while reading the program.

We assume that the network is described by a rooted tree T. For uniformity, let the root of T have an outgoing upward edge. (Messages sent along this edge will never be received by anyone.) We can then write a single program for all the nodes of T, including the root.

Let V denote the set of vertices of T. Let RESOURCES(v) denote the resources placed at vertex v, for each $v \in V$, and let PLACE(v) = |RESOURCES(v)| for all v. Let REQUESTS(v) denote the requests arriving at v. We assume that all the sets RESOURCES(v) and REQUESTS(v) are finite. Let PARENT(v), CHILDREN(v), DESCENDANTS(v) and NEIGHBORS(v) denote the designated vertices and sets of vertices, for vertex v.

The kinds of messages used are "ARRIVAL", "BUYER" and messages corresponding to specific captured resources.

Program for node v, $v \in V$:

In the program for node v, we use RESOURCES as a shorthand for RESOURCES(v), and similarly for the other notation above.

It is convenient to think of the state of v as consisting of "independent variables" and "dependent variables". The independent variables are just the usual kind of variables, which can be read and assigned to. The dependent variables are virtual variables whose values are defined in terms of the independent variables. These values can be read, but not modified. We can think of the reading of a dependent variable as shorthand for a read of several independent variables, together with a calculation of the function giving the dependency.

Independent Variables

REQUESTS, for the set of requests that originated at v,

ACTIVE, for the set of requests that originated at v, which are still unsatisfied,

FREE, for the set of resources in RESOURCES that have not yet been "captured" by requests,

GRANT, for a single captured resource on its way back to a request,

ARRIVALS(w), $w \in$ NEIGHBORS, for the number of "ARRIVAL" messages received from each of v's children, and sent to v's parent, respectively,

BUYERS(w), w ∈ NEIGHBORS, for the number of "BUYER" messages sent to v's children and received from v's parent, respectively,

NETGRANTS(w), w ∈ NEIGHBORS, for the net flow of captured resources out of v to each of its neighbors,

NEXT, a temporary variable which can hold a vertex.

Initialization of Independent Variables

REQUESTS = ACTIVE = Ø, FREE = RESOURCES, and all other variables are 0.

Dependent Variables and their Dependencies

CAPTURED, for the set of resources in RESOURCES which have been captured,

Dependency: CAPTURED = RESOURCES - FREE.

SATISFIED, for the set of requests in REQUESTS which have been satisfied,

Dependency: SATISFIED = REQUESTS - ACTIVE.

NETBUYERS(w), w ∈ NEIGHBORS, for the net flow of buyers and virtual buyers into v from each neighbor, (Recall the definition of "virtual buyers" from the last subsection.)

Dependency: If w = PARENT, then NETBUYERS(w) = min(PLACE(DESCENDANTS) - ARRIVALS(w), BUYERS(w)). If w ∈ CHILDREN, then NETBUYERS(w) = -min(PLACE(DESCENDANTS(w) - ARRIVALS(w), BUYERS(w)).

These two equations can be understood as follows. Consider, for example, the first equation, for w = PARENT. If PLACE(DESCENDANTS) - ARRIVALS(w) < BUYERS(w), it means that the placement originally given for v's subtree is not adequate for handling the requests (arrivals) which have originated in v's subtree, together with the "BUYER" messages sent down from w. Therefore, all the resources in v's subtree are allocated to requests, either within or outside of v's subtree. Whether the net flow of buyers should be regarded as into or outward from v's subtree then depends solely on the sign of PLACE(DESCENDANTS) - ARRIVALS(w), without regard to the number of "BUYER" messages received from w. That is, if PLACE(DESCENDANTS) $\leq$ ARRIVALS(w), then the sign is negative and the net flow of buyers is outward from v's subtree, while otherwise it is inward; in either case, its magnitude is equal to |PLACE(DESCENDANTS) - ARRIVALS(w)|. On the other hand, if PLACE(DESCENDANTS) - ARRIVALS(w) $\geq$ BUYERS(w), then the placement originally given for v's subtree is adequate for handling both the requests which have originated in v's subtree, together with the "BUYER" messages sent down from w. Therefore, the net flow of buyers is inward, and its amount is just equal to BUYERS(w), without regard to the other two values. The second equation is similar, with appropriate changes of sign.

Another way to understand the equations is as follows. Again, consider the first equation, for w = PARENT. Then NETBUYERS(w) = BUYERS(w) - VIRTBUYERS(w), where the latter quantity is the number of virtual buyers which v estimates it has sent to its parent. Using the expression which was derived in the preceding subsection for the number of virtual buyers, we see that NETBUYERS(w) = BUYERS(w) - max(ARRIVALS(w) + BUYERS(w) - PLACE(DESCENDANTS),0). This is equal to min(PLACE(DESCENDANTS) - ARRIVALS(w),BUYERS(w)), as needed. Again, the other calculation is similar.

The remaining dependent variables are:

NETBUYERS, for the total of all the NETBUYERS(w),

Dependency: NETBUYERS = $\Sigma_{w \in NEIGHBORS}$NETBUYERS(w).

NETFLOW, for the net flow of buyers into v,

Dependency: NETFLOW = |REQUESTS| + NETBUYERS.

NETGRANTS, for the net flow of grants out of v,

Dependency: NETGRANTS = $\Sigma_{w \in NEIGHBORS}$NETGRANTS(w).

The following code is executed in response to the receipt of any message or request, M. The first part of the code does initial processing of messages, updating estimates and sending any required "ARRIVAL" messages. After the first part of the code, there will be at most one excess request left at the node, and if there is such a request left at the node, then the node is able to service that request, either locally or by sending a buyer into a subtree. In the second part of the code, the node decides where it can service such an excess request, and it does so. (In case a buyer is sent down into a subtree, the subtree is chosen nondeterministically. Later, we will refine the algorithm to use a probabilistic choice at this point.) Finally, in the third part of the code, the node processes an excess captured resource, if it happens to have one. (It cannot have more than one.) The node can have a captured resource either because M was a captured resource message, or because (in the second part of the code) the node itself captured a local resource. The resource is granted to a local request if possible; otherwise, it is sent in such a way as to negate the net flow of buyers into the node along some edge. It is always possible to process such a captured resource in one of these ways.

```
/* The following invariants hold at the beginning of any node step.
   NETFLOW = |CAPTURED|.
   ARRIVALS(PARENT) = |REQUESTS| + Σ_{w ∈ CHILDREN} ARRIVALS(w)].
   GRANT = 0.
   NETGRANTS = |CAPTURED| - |SATISFIED|. */

(Part 1)

If M is a request then
  [REQUESTS := REQUESTS ∪ {M}
   ACTIVE := ACTIVE ∪ {M}
   send "ARRIVAL" to PARENT
   ARRIVALS(PARENT) := ARRIVALS(PARENT) + 1]

If M = "ARRIVAL" from w then
  [ARRIVALS(w) := ARRIVALS(w) + 1
   send "ARRIVAL" to PARENT
   ARRIVALS(PARENT) := ARRIVALS(PARENT) + 1]

If M = "BUYER" then BUYERS(PARENT) := BUYERS(PARENT) + 1

/* Now slightly revised invariants hold:
   |CAPTURED| ≤ NETFLOW ≤ |CAPTURED| + 1.
   ARRIVALS(PARENT) = |REQUESTS| + Σ_{w ∈ CHILDREN} ARRIVALS(w)].
   GRANT = 0.
   NETGRANTS = |CAPTURED| - |SATISFIED|. */

(Part 2)

/* Next, if there is an excess request, service it.  */

If NETFLOW = |CAPTURED| + 1 then

  if FREE ≠ ∅
    then
      [choose s ∈ FREE
       FREE := FREE - {s}
       GRANT := Ss]
    else
      [/* Send "BUYER" down into a subtree. */

       S := {s ∈ CHILDREN: PLACE(DESCENDANTS(s)) > ARRIVALS(s) + BUYERS(s)}

       /* S ≠ ∅. */

       choose NEXT ∈ S
       send "BUYER" to NEXT
       BUYERS(NEXT) := BUYERS(NEXT) + 1]

/* NETFLOW = |CAPTURED|. */
```

```
(Part 3)

/* Process M if M is a captured resource message. */

If M is a resource from w then
  [NETGRANTS(w) := NETGRANTS(w) - 1
   GRANT := M]

/* Send a captured resource, if you have one, toward a request origin. */

If GRANT ≠ 0 then

  /* NETGRANTS = |CAPTURED| - |SATISFIED| - 1. */

  [if ACTIVE ≠ ∅ then
     then
       [choose r ∈ ACTIVE
        ACTIVE := ACTIVE - {r}
        output (r,GRANT)]
     else
       [choose w ∈ NEIGHBORS with NETBUYERS(w) - NETGRANTS(w) > 0
        send GRANT to w
        NETGRANTS(w) := NETGRANTS(w) + 1]
    GRANT := 0]
```

3. Correctness of the Algorithm

Theorem 1: The given algorithm solves the resource allocation problem.

Proof: We claim that the node program given above implements the strategy described informally in the previous section. We do not give a proof of this correspondence here. Rather, we argue correctness of the key assertions of the program and give informal arguments for the rest of the proof of correctness of the algorithm.

The first portion of the algorithm, the initial processing of the first three kinds of messages, simply sends the appropriate "ARRIVAL" messages and records the proper changes to the various sets and counters.

For any of the three kinds of messages, node v is finding out about a new request that needs to be processed. In some cases, v will need to do more to help process the request. If the message is an "ARRIVAL", and node v thinks that the corresponding request can be serviced in the sender's subtree, then v has no further work to do. If the message is a request or an "ARRIVAL", and if node v thinks that it is impossible to service that request in v's subtree, then the "ARRIVAL" message sent upward by v will be counted by v's parent in its estimate of virtual buyers emanating from v's subtree. Thus, after sending this "ARRIVAL" message upward, v will have no further work to do. Also, if the message is a "BUYER" and v thinks that it is impossible to service the request in v's subtree, then v need not do anything more. However, if v thinks that the new request can be serviced in its subtree, then it has some further work to do, in the second portion of the algorithm.

The second portion of the algorithm manages the disposition of any excess flow of requests into the node. We must first check that the number of excess requests after the initial processing of a single message can only be 0 or 1. That is, we must verify that |CAPTURED| $\leq$ NETFLOW $\leq$ |CAPTURED| + 1 between Parts 1 and 2 of the code.

A quick check of the cases shows that the only way this could fail to be true is if M = "ARRIVAL" from a child s, and the result of processing M causes NETBUYERS(s) to remain unchanged, while NETBUYERS(PARENT) decreases. In this case, we can deduce some relationships among the values of v's local variables at the beginning of the node step.

For every t $\in$ CHILDREN, it must be the case just before execution of Part 1 that

-NETBUYERS(t) = min(PLACE(DESCENDANTS(t)) - ARRIVALS(t), BUYERS(t)),

so that

-NETBUYERS(t) $\leq$ PLACE(DESCENDANTS(t)) - ARRIVALS(t).

That is,

NETBUYERS(t) $\geq$ ARRIVALS(t) - PLACE(DESCENDANTS(t)).

Since NETBUYERS(s) remains unchanged, then it must be the case that

-NETBUYERS(s) $\neq$ PLACE(DESCENDANTS(s)) - ARRIVALS(s).

(If they were equal, then PLACE(DESCENDANTS)(s)) - ARRIVALS(s) = min(PLACE(DESCENDANTS(s)) - ARRIVALS(s), BUYERS(s)), and an increase to ARRIVALS(s) would cause a change to the minimum, thereby changing NETBUYERS(s).)

Therefore, NETBUYERS(s) $\neq$ ARRIVALS(s) - PLACE(DESCENDANTS(s)), and so

NETBUYERS(s) > ARRIVALS(s) - PLACE(DESCENDANTS(s)).

Since NETBUYERS(PARENT) decreases, it means that NETBUYERS(PARENT) = PLACE(DESCENDANTS) - ARRIVALS(PARENT).

Now consider NETFLOW = |REQUESTS| + NETBUYERS. The right side is equal to

|REQUESTS| + NETBUYERS(PARENT) + NETBUYERS(s) + $\Sigma_{t \in CHILDREN, t \neq s}$NETBUYERS(w).

By previous results, this is, in turn, strictly greater than

|REQUESTS| + PLACE(DESCENDANTS) - ARRIVALS(PARENT)

+ ARRIVALS(s) - PLACE(DESCENDANTS(s))

+ $\Sigma_{t \in CHILDREN, t \neq s}$ ARRIVALS(t) - PLACE(DESCENDANTS(t)).

By the ARRIVAL invariant, this is equal to

PLACE(DESCENDANTS) · PLACE(DESCENDANTS(s)) · $\Sigma_{t \in CHILDREN,\ t \neq s}$ PLACE(DESCENDANTS(t)),

= PLACE(v).

Thus, NETFLOW > PLACE(v). However, the original invariant says that NETFLOW = |CAPTURED|, and |CAPTURED| is never permitted to be greater than PLACE(v), a contradiction.

We have thus shown that |CAPTURED| $\leq$ NETFLOW $\leq$ |CAPTURED| + 1 at the point where that claim is made. Thus, there is at most one excess request that requires disposition. In the case where there is an excess request, node v must service that request in its subtree. There are two possibilities: either v can service the request locally, or it cannot. If FREE $\neq \emptyset$, then a free resource is captured to service the excess request. If not, then a "BUYER" message must be sent down into some subtree. We must show that, in the event FREE = $\emptyset$, it is possible to send such a "BUYER" message. That is, we must check that S $\neq \emptyset$ at the place where that claim is made.

Assume not. We will make some deductions about the values of the variables at the point where that claim is made. At that point, we know that FREE = $\emptyset$, so that PLACE(v) = |CAPTURED|. We also know that

NETBUYERS(PARENT) $\leq$ PLACE(DESCENDANTS) · ARRIVALS(PARENT), by definition of NETBUYERS.

Then

NETFLOW = |REQUESTS| + NETBUYERS(PARENT) + $\Sigma_{s \in CHILDREN}$NETBUYERS(s),

$\leq$ |REQUESTS| + PLACE(DESCENDANTS) · ARRIVALS(PARENT) + $\Sigma_{s \in CHILDREN}$NETBUYERS(s).

Because S = $\emptyset$, it follows that

PLACE(DESCENDANTS(s)) · ARRIVALS(s) $\leq$ BUYERS(s) for each s $\in$ CHILDREN.

Therefore,

NETBUYERS(s) = -(PLACE(DESCENDANTS(s)) - ARRIVALS(s)).

Thus, the right-hand side of the next-to-last inequality is equal to

|REQUESTS| + PLACE(DESCENDANTS) · ARRIVALS(PARENT) · $\Sigma_{s \in CHILDREN}$(PLACE(DESCENDANTS(s)) · ARRIVALS(s)).

This expression is, in turn, equal to

$$\text{PLACE(DESCENDANTS)} \cdot \Sigma_{s \in \text{CHILDREN}} \text{PLACE(DESCENDANTS(s))},$$

$$= \text{PLACE(v)} = |\text{CAPTURED}|.$$

Thus, NETFLOW $\leq$ |CAPTURED|, a contradiction.

Thus, we have shown that it is always possible to service an excess request.

Next, we must show that NETFLOW = |CAPTURED| between Parts 2 and 3 of the code. This means that after servicing any excess request, there is no remaining request to be serviced. Previous to Part 2, |CAPTURED| $\leq$ NETFLOW $\leq$ |CAPTURED| + 1. If NETFLOW was equal to |CAPTURED| + 1, then the body of the conditional was executed. If the first case of the conditional held (i.e. the case for FREE $\neq \emptyset$), then |CAPTURED| is increased by 1, so the invariant is restored. Otherwise, a "BUYER" message was sent to a child, s, for which PLACE(DESCENDANTS(s)) - ARRIVALS(s) > BUYERS(s). This caused NETBUYERS(s) to increase by 1, thereby increasing the value of NETFLOW and restoring the invariant.

The third portion of the algorithm manages the travel of captured resources back to requests. First, note that there can be only one captured resource assigned to GRANT at any node in a single step, since the two assignments to GRANT cannot both be executed during a single step. If the message is a captured resource, then no progress is done until the clause contains the second grant. Otherwise, this clause is skipped. We must argue that such a neighbor exists in this case.

Assume not. Then NETBUYERS(s) $\leq$ NETGRANTS(s) for all s $\in$ NEIGHBORS. Now, NETFLOW = |CAPTURED|, so that |CAPTURED| = |REQUESTS| + NETBUYERS,

$\leq$ |REQUESTS| + NETGRANTS,

= |REQUESTS| + |CAPTURED| - |SATISFIED| - 1

= |ACTIVE| + |CAPTURED| - 1. Therefore, 1 $\leq$ |ACTIVE|, a contradiction.

Thus, we have checked that the key assertions hold and the code can be executed at all points. We have claimed (and tried to argue) that the algorithm follows the strategy of the preceding section, in setting up a flow of buyers from requests to resources. Eventually, the values of all the NETBUYERS(w) variables will stabilize, and the values taken on by corresponding NETBUYERS(w) variables at either end of a single edge will be negations of each other. (We use the fact that there are only finitely many requests here. Eventually, no further requests will arrive, so no additional "ARRIVAL" messages will be sent. There is a bound on how many "BUYER" messages will be sent downward along any edge. Therefore, there are only finitely many total "ARRIVAL" and "BUYER" messages which get sent, so that eventually, they will all be delivered.) Similarly, all the REQUESTS variables will eventually stabilize.

Finally, we must consider the travel of captured resources to request origins. Define a new variable, MESSAGES(w), at node v, where w $\in$ NEIGHBORS(v). Its value is defined to

be the number of captured resource messages which have been sent from w to v but have not yet arrived. (Of course, neither v nor w actually "knows" the value of this variable.) For any time, t, after the NETBUYERS(w) and REQUEST values have stabilized, any node v, and any w $\in$ NEIGHBORS(v), let A(v,w,t) be the value of NETBUYERS(w) - NETGRANTS(w) + MESSAGES(w) at v at time t. Note that A(v,w,t) = -A(w,v,t) in all cases. Let SUM(t) denote $\Sigma_{v,w}|A(v,w,t)|$. We claim that any event which involves the receipt of a captured resource message does not change SUM(t), while any event which involves the sending of a captured resource message decreases SUM(t). Therefore, captured resource messages will not be sent forever: they will eventually subside, at which time they must have found a matching request.

First, consider an event involving the receipt of a captured resource, by v, from w. The only term in the sum which is affected is A(v,w,t). The receipt of the messages causes v's values of MESSAGES(w) and NETGRANTS(w) both to decrease by 1, so that A(v,w,t) is unchanged. Therefore, SUM(t) is unchanged. Second, consider an event involving the sending of a captured resource, by v, to w. The only terms in the sum which are affected are A(v,w,t) and A(w,v,t). At time t just prior to the sending event, it must be that v's value of NETBUYERS(w) - NETGRANTS(w) > 0, which implies that A(v,w,t) > 0. The result of sending the message is to increase NETGRANTS(w), which means that A(v,w,t) gets decreased by 1. Therefore, |A(v,w,t)| gets decreased by 1. Thus, also, |A(w,v,t)| gets decreased by 1, so that SUM(t) gets decreased by 2.∎

4. Monotonicity Analysis

The rest of the paper is devoted to an analysis of the time requirements of the algorithm. Specifically, we measure the sum of the times between requests and their corresponding grants. For the purpose of carrying out the analysis, certain restrictions will be made. These restrictions will be introduced as needed.

We begin with some basic definitions. Next, we introduce two restrictions which are needed throughout the analysis. Then we define and categorize the complexity measures of interest. We then prove a basic combinatorial result, and use it to prove the monotonicity of the number of "BUYER" messages as a function of interarrival time. Finally, we show that the expected running time of the algorithm is bounded by the expected time for the sequential case of the algorithm.

4.1. Definitions

Let N denote the set of natural numbers, including 0. Let R^+ denote the set of nonnegative reals. If f is a numerical function with domain V, then extend f to subsets of V by $f(W) = \Sigma_{v \in W} f(v)$.

Let T be a rooted tree. We write $vertices_T$, $internal_T$, and $leaves_T$ to denote the indicated sets of vertices of T. Let $root_T$ denote the root. If v $\in$ $vertices_T$, we write $desc_T(v)$ for the set of vertices of T which are descendants of v (including v itself), $parent_T(v)$ for v's parent in T, $children_T(v)$ for v's children, and $neighbors_T(v)$ for $children_T(v) \cup \{parent_T(v)\}$.

If v is a vertex of T, let $height_T(v)$ denote the maximum distance from v to a leaf in its subtree. If e is an edge in T, then define $height_T(e)$ to be the same as $height_T(v)$, where v is e's upper endpoint. Let $height_T$ denote $height_T(root_T)$.

A *placement* for T is a function p: $vertices_T \rightarrow N$, representing the number of resources at each vertex. We write *total(p)* for $p(vertices_T)$, the total number of resources in the entire tree. We say that p is *nonnull* provided total(p) > 0.

A *weighted tree*, T, is an undirected, rooted tree with an associated probability density function, φ_T, on the leaves of T, such that $\varphi_T(v) > 0$ for all leaves v. (This assumption is made for technical reasons, so that we can normalize probability functions without danger of dividing by 0.) If T is a weighted tree, $v \in internal_T$, and S is a nonempty subset of $children_T(v)$, then let $random_{T,S}$ denote the probability function which returns $s \in S$ with probability $\varphi_T(desc_T(s))/\varphi_T(desc_T(S))$. Thus, $random_{T,S}$ returns s with probability proportional to the sum of the probability function values for the descendants of s.

4.2. Initial Restrictions

For the remainder of Section 4, we assume that the following two restrictions hold.

Restriction 1:

T is a weighted tree, and the nondeterministic choice step in Part (2) of the algorithm uses a call to $random_{T,S}$.

Restriction 2:

Delivery time for messages is always exactly 1.

Restriction 1 describes a particular method of choosing among alternative search directions. This method does not use all the information available during execution, but only the "static" probability distribution information available at the beginning of execution. One might expect a more adaptive choice method to work better; however, we do not know how to analyze such strategies.

Restriction 2 has the effect of restricting the executions under consideration; for example, all messages between any two nodes are pipelined - they arrive in the order in which they are sent. While we would like to understand the behavior of the algorithm in the presence of variable message delivery times, such analysis appears to be more difficult.

4.3. Cost Measures and Preliminary Results

A *request pattern*, r, is a finite sequence of elements of $vertices_T \times R^+$ whose second components are monotone nondecreasing. A request pattern represents the sequence of requests that occur, their locations and times. If r is a request sequence, then *length(r)* denotes its length.

A *choice sequence*, c, for $v \in internal_T$ is an infinite sequence of elements of $children_T(v)$, with infinitely many occurrences of each child. If $\mathcal{C} = \{c_v\}$ is a collection of choice sequences, one for each $v \in internal_T$, then $\mathcal{C}$ can be used in place of probabilistic choices in an execution of the algorithm, as follows. Each internal node, v, makes choices among its children by choosing the first unused element of c_v satisfying the inequality $PLACE(desc_T(s)) > ARRIVALS(s) + BUYERS(s)$. That is, v chooses a child, s, for which v thinks there are still remaining resources in s's subtree.

Let p be a placement for T. Let r be a request pattern, and $\mathcal{C} = \{c_v\}$ a collection of choice sequences, one for each $v \in internal_T$. Then $cost_{T,p}(r,\mathcal{C})$ is defined to be the total time from requests to corresponding grants, if requests arrive according to r and $\mathcal{C}$ is used in place of probabilistic choices. (With suitable conventions for handling events which happen at the same time, the execution, and hence the cost, is uniquely defined for fixed r and $\mathcal{C}$.)

The cost measure defined above can be broken up into two pieces, as follows. Let $searchcost_{T,p}(r,\mathcal{C})$ be the total of the times from requests to corresponding captures of resources, if r and $\mathcal{C}$ are used as above. Let $returncost_{T,p}(r,\mathcal{C})$ be the total of the times from captures to corresponding grants of resources.

Now we incorporate a probabilistic construction of $\mathcal{C}$ into the cost measure. If r is a request pattern, let $cost_{T,p}(r)$ denote the expected value of $cost_{T,p}(r,\mathcal{C})$, where $\mathcal{C}$ is constructed using φ_T. (That is, for each $v \in internal_T$, the sequence c_v is constructed by successive choices from among $children_T(v)$, choosing s with probability $\varphi_T(desc_T(s))/\varphi_T(desc_T(S))$, where $S = children_T(v)$. Among the sequences thereby generated are some for which it is not the case that each child occurs infinitely often. However, these sequences form a set of measure 0, so that we can ignore them in calculating the expected cost measure.) We claim that $cost_{T,p}(r)$ is exactly the expected total time from requests to grants, provided the algorithm is run in the normal way, using probabilistic choices. That is, the two strategies of constructing choice sequences independently of the algorithm and carrying out the probabilistic choices on-line give identical results.

Let f denote an arbitrary probability density function whose domain consists of positive reals. Extend the domain of the function $cost_{T,p}$ to the set of such functions by defining $cost_{T,p}(f)$ to be the expected value of $cost_{T,p}(r)$, where r is of length total(p), with its successive locations chosen independently using the distribution φ_T, and its successive interarrival times chosen using f. That is, at the time the algorithm begins, and at the time of each request, the probability that the next arrival occurs exactly t units later is f(t). We will be primarily interested in this cost, $cost_{T,p}(f)$.

We define $searchcost_{T,p}(r)$, $searchcost_{T,p}(f)$, etc., analogously to our earlier definitions.

The following claim is true for all domains for which the definitions are valid.

Lemma 2: $cost_{T,p} = searchcost_{T,p} + returncost_{T,p}$.

Proof: Straightforward.∎

Next, we will relate the given cost measures to the total numbers of various kinds of messages sent during the execution of the algorithm. Note that during the execution of an algorithm, the estimates of "BUYER" and virtual buyer messages sent along an edge can be different at the two ends of the edge. However, after the entire execution of the algorithm is completed, the discrepancy disappears, so that the following definitions are unambiguous. Let $bnum_{T,p}(r,\mathcal{C})$ denote the total number of "BUYER" messages sent on all edges during the execution of the algorithm on r using $\mathcal{C}$. Let $vbnum_{T,p}(r,\mathcal{C})$ denote the total number of virtual buyer messages sent on all edges during the execution of the algorithm on r using $\mathcal{C}$. Let $gnum_{T,p}(r,\mathcal{C})$ denote the total number of captured resource messages sent on all edges during the execution of the algorithm on r using $\mathcal{C}$. As before, define $bnum_{T,p}(r)$, $bnum_{T,p}(f)$, etc.

Because of the fact that message delivery time is assumed to be exactly 1, there are some relationships between the measures describing time costs and the measures describing numbers of messages. The following lemma describes a set of relationships among the various measures. Note that all the statements are true over all possible domains of definition.

Lemma 3: (a) $\text{searchcost}_{T,p} \leq \text{bnum}_{T,p} + \text{vbnum}_{T,p}$.

(b) $\text{returncost}_{T,p} = \text{gnum}_{T,p}$.

(c) $\text{gnum}_{T,p} \leq \text{bnum}_{T,p} + \text{vbnum}_{T,p}$.

(d) $\text{cost}_{T,p} \leq 2[\text{bnum}_{T,p} + \text{vbnum}_{T,p}]$.

Proof: (a) This inequality is true because buyers continue to make progress up and down the edges of the tree; all time used by the algorithm is occupied by the transmission of appropriate buyer and virtual buyer messages. The reason that we have an inequality rather than an equation here is that buyers are permitted to travel "discontinuously", as described in Section 3.

(b) This equation is true because captured resources travel continuously via captured resource messages.

(c) We must show that each captured resource message always moves in such a way as to "negate" a buyer or virtual buyer message. This is a bit tricky to argue, because of the discrepancies between estimates at opposite ends of an edge.

A captured resource only moves over an edge if the net flow of buyers into the node on that edge, as estimated at the near endpoint, is positive. By moving over that edge, the captured resource negates an incoming buyer or virtual buyer along that edge, as estimated at the near endpoint. Because of the assumption that all messages take exactly time 1, by the time the captured resource reaches the far endpoint, the negated buyer or virtual buyer is also counted in the estimate of outgoing buyers and virtual buyers at the opposite endpoint. The arrival of the captured resource at the far endpoint can thus be regarded as negating an outgoing buyer or virtual buyer at the far endpoint as well.

(d) Straightforward by Lemma 2 and (a)-(c).∎

Now, we introduce an additional restriction, to remain in force for the remainder of Section 4.

Restriction 3:

T has all leaves at the same distance from the root, and r and p are nonzero only at leaves.

As usual, the following lemma is intended to hold for all valid domains of definition.

Lemma 4: $bnum_{T,p} = vbnum_{T,p}$.

Proof: We sketch the argument for fixed r and $\mathbb{C}$. For a particular edge e, let a_e denote the number of request arrivals below e, b_e denote the number of "BUYER" messages sent downward along e, and p_e denote the number of resources placed below e, in the execution for r and $\mathbb{C}$. Since all resources get matched to requests, we must have $a_e + b_e \geq p_e$, so that the number of virtual buyers sent over edge e is exactly $\max(a_e + b_e - p_e, 0) = a_e + b_e - p_e$.

Now consider all the edges at any particular height h in the tree. Since all resources and requests are at the leaves, and the branches are all of equal length, it is clear that

$$\Sigma_{height_T(e) = h} a_e = total(p) = \Sigma_{height_T(e) = h} p_e.$$

Therefore,

$$\Sigma_{height_T(e) = h} b_e = \Sigma_{height_T(e) = h} (a_e + b_e - p_e).$$

That is, the numbers of buyers and virtual buyers sent over edges of height h are equal. Since this is true for all h, the result follows.∎

Theorem 5: $cost_{T,p} \leq 4(bnum_{T,p})$.

Proof: Immediate from Lemmas 3 and Restriction 3.∎

Thus, in order to obtain an upper bound on $cost_{T,p}(f)$, it suffices to prove a bound for $bnum_{T,p}(f)$.

4.4. A Combinatorial Result

This subsection contains a key combinatorial result which will be used in the subsequent analysis. We model the behavior of the algorithm at a single node v. The children of v are modelled as a set of *bins* for resources. (Here, we do not concern ourselves about the tree structure beyond the children.) Let c be a choice sequence for v. Each bin s is initialized to contain a number p(s) of resources.

The arrival of messages at v is described by a *script*, S. A script is a finite sequence of symbols, each of which is either a bin number s or an "X". A bin number represents the arrival of an "ARRIVAL" message from the specified child. The symbol X represents the arrival of a "BUYER" message from v's parent.

The processing of script S on c and p, is as follows. The elements of S are processed sequentially.

If S(i) is:

```
s ∈ bins, then
    if bin s is nonempty,
    then subtract 1 from the number of resources in s
    else if some bin is nonempty then
       [SELECT the first unused element of c describing a nonempty bin, t;
        subtract 1 from the number of resources in t]

X,  then
    if some bin is nonempty then
       [SELECT the first unused element of c describing a nonempty bin, t;
        subtract 1 from the number of resources in t]
```

Define *SELECT(S,c,p,i)* to be the number of times bin i is SELECTed during the course of processing S on c and p. (Note that a bin is only said to be SELECTed when the choice sequence is used to choose it, and not when it is explicitly chosen by the script. Define *choice(S,c,p,j)* to be equal to k provided that when S(j) is processed on c and p, the kth element of c is used to select a bin. (If no element of c is used, then *choice(S,c,p,j)* is undefined.) It follows that SELECT(S,c,p,i) = |{j: c(choice(S,c,p,j)) = i}|,

For any script S, let *binsequence(S)* denote the subsequence of S consisting of bin numbers. Script S is said to *dominate* script S' provided that: (a) T = T', where T = binsequence(S) and T' = binsequence(S'), (b) the total number of X's in S is at least as great as the total number of X's in S', and (c) for each i, the number of X's in S preceding T(i) is at least as great as the number of X's in S' preceding T'(i). The main result of this section is that, if S dominates S', then SELECT(S,c,p,i) $\geq$ SELECT(S',c,p,i) for all c, p and i.

We say that an interchange of two consecutive elements of a script S is *legal* provided that the first element of the pair is an X. We say that a script S' is *reachable* from a script S if S can be transformed to S' by a series of legal interchanges. Note that S dominates all scripts S' reachable from S; moreover, if S dominates S', then S' can be augmented with some suffix of X's, to a script which is reachable from S.

Lemma 6: For any scripts S and S' such that S' is reachable from S, and for any choice sequence c, placement function p and bin i,

SELECT(S,c,p,i) $\geq$ SELECT(S',c,p,i).

Proof: We prove this lemma by showing that if S' is reachable from S by a single legal interchange, then the inequality holds. The lemma follows by induction on the number of legal interchanges.

Fix S, c, and p. Assume that S' is obtained from S by interchanging S(j) = X with S(j + 1). If S(j + 1) = X, then S = S', so the result is obvious. So assume S(j + 1) = s $\in$ bins. There are three cases.

Case 1: Bin s is empty after processing S(1)...S(j-1) on c and p.

Then choices using c are made for both S and S' at both steps j and j + 1. Thus, choice(S,c,p,j) = choice(S',c,p,j) and choice(S,c,p,j + 1) = choice(S',c,p,j + 1). The number of resources remaining in each bin after step j + 1 is the same for S and for S', and therefore processing continues identically for S and S' after that point. Thus, SELECT(S,c,p,i) = SELECT(S',c,p,i).

Case 2: Bin s contains more than one resource after processing S(1)...S(j-1) on c and p, or else c(choice(S,c,p,j)) is not bin s. (That is, the bin selected by the choice made at step j is not bin s.)

Then the effect of the pair of steps j and j + 1 is the same for both S and S': a resource is removed from bin s and a resource is removed from bin c(choice(S,c,p,j)). (When processing S, the choice from c occurs first, while when processing S', the explicit removal from s occurs first, but the net effect is the same.) Subsequent processing of the two scripts will be identical, and once again, SELECT(S,c,p,i) = SELECT(S',c,p,i).

Case 3: Bin s contains exactly one resource after processing S(1)...S(j-1), and c(choice(S,c,p,j)) = s. (That is, the bin selected by the choice made at step j is s.)

In this case, the processing of S uses choices from c at both steps j and j + 1, because the choice of s at step j removes the last resource from bin s, and so a choice must also be made at step j + 1. The processing of S' does not need a choice at step j, although it is forced to choose by the X at step j + 1. Thus, in both cases, step j removes the last resource from bin s, while step j + 1 makes a choice using c. Then choice(S,c,p,j + 1) = choice(S',c,p,j + 1); that is, the same entry in c is used at step j + 1 in both cases. The combined effect of steps j and j + 1 on the bins is the same for the two scripts. Subsequent processing is again identical, so SELECT(S,c,p,i) = SELECT(S',c,p,i) for bin i ≠ s, and SELECT(S,c,p,s) = SELECT(S',c,p,s) + 1 > SELECT(S',c,p,s).∎

We can now state the main result of this section.

Corollary 7: For any scripts S and S' such that S dominates S', and for any choice sequence c, placement function p and bin i,

SELECT(S,c,p,i) ≥ SELECT(S',c,p,i).

Proof: Let T be an augmentation of S' by a suffix of X's, such that T is reachable from S. Then Lemma 6 implies that SELECT(S,c,p,i) ≥ SELECT(T,c,p,i). But the latter term is obviously at least as great as SELECT(S',c,p,i).∎

4.5. Expansions

In this subsection, we show that the number of "BUYER" messages sent is a monotone nondecreasing function of the interarrival time of the arrival distribution. We do this by comparing particular pairs of executions.

For n ∈ N, let [n] denote {1,...,n}. If a ∈ R and $r = (v_i,t_i)_{i \in [k]}$, is a request pattern, then *ar*, the *expansion* of r by a, is the request pattern $(v_i,at_i)_{i \in [k]}$. That is, ar represents the request pattern in

which the successive requests occur at the same locations as in r, but in which the times are expanded by the constant factor a.

We will compare executions for request pattern r and request pattern ar, using the same choice sequence. We require a technical restriction, just to avoid the complications of having to consider multiple events occurring at the same node at the same time, in either execution. A request pattern, r, is said to be *a-isolated* provided that no two requests occur in r at the same time, and provided that the following holds. If t_1 and t_2 are two times at which requests arrive in r, where $t_1 \neq t_2$, and if k is an integer, then the following are true: (a) $t_1 - t_2 \neq 2k$, and (b) $t_1 - t_2 \neq (2/a)k$.

The next lemma is crucial to our analysis. Its truth was first observed empirically, and then proved analytically. It says that the number of "BUYER" messages sent during an execution cannot increase if the request pattern is expanded by a constant which is greater than or equal to 1.

Lemma 8: If $a \geq 1$, and r is a-isolated, then $bnum_{T,p}(r,C) \leq bnum_{T,p}(ar,C)$.

Proof: Fix T, p and C. Let bsent(r,e,t) denote the number of "BUYER" messages sent along edge e, in the execution for r (using T, p and C), up to and including time t. Let brec(r,v,t) denote the number of "BUYER" messages received by vertex v, in the execution for r, up to and including time t. Let arec(r,e,t) denote the number of "ARRIVAL" messages received along edge e, in the execution for r, up to and including time t. We will show the following:

Claim:

$bsent(r,e,t + height_T(e)) \leq bsent(ar,e,at + height_T(e))$ for all r, e, and t.

This is a stronger claim than required for the lemma, since it shows an inequality not only for the total number of "BUYER" messages, but for the number along each edge, up to corresponding times.

Fact 1: $arec(r,e,t + height_T(e)) = arec(ar,e,at + height_T(e))$.

This is so because the number of requests arriving in request sequence r by time t is the same as the number arriving in request sequence ar by time at, and messages just flow up the tree at a steady rate.

The rest of the proof proceeds by induction on $height_T(e)$, starting with $height_T(e) = height_T$, and working downward towards the leaves.

Base: $height_T(e) = height_T$

In this case, e's upper endpoint is $root_T$. The actions of $root_T$ are completely determined by the "ARRIVAL" messages it receives, which are the same at corresponding times in the two executions, by Fact 1. The Claim follows.

Inductive step: $height_T(e) < height_T$

Let v be the upper endpoint of edge e, and the lower endpoint of edge e'.

Fact 2: $brec(r,v,t + height_T(v)) \leq brec(ar,v,at + height_T(v))$.

This is so because $brec(r,v,t + height_T(v)) = bsent(r,e',t + height_T(v) - 1)$ because all messages take exactly time 1, $= bsent(r,e',t - 2 + height_T(e'))$, $\leq bsent(ar,e',a(t-2) + height_T(e'))$ by inductive hypothesis, $= bsent(ar,e',a(t-2) + 1 + height_T(v))$, $= brec(ar,v,a(t-2) + 2 + height_T(v)) \leq brec(ar,v,at + height_T(v))$, since $a(t-2) + 2 \leq at$.

Now let us consider the situation from v's viewpoint. Node v is comparing two executions, the first for r and the second for ar. All v sees is its incoming "ARRIVAL" and "BUYER" messages, and v uses the same choice sequence in both cases. At corresponding times in the two executions (i.e. $t + height_T(v)$ in the first execution vs. $at + height_T(v)$ in the second execution), the same number of "ARRIVAL" messages have been received along each edge (by Fact 1), and an inequality holds for the number of "BUYER" messages which have been received (by Fact 2). We will show the needed inequality for the number of "BUYER" messages sent by v along each edge, up to corresponding times.

Fix any time t. We compare the first execution up to time $t + height_T(v)$ with the second execution up to time $at + height_T(v)$. We claim that this situation is modelled by the combinatorial problem presented in the preceding subsection. First, we represent v's inputs in each of the two executions by a script, i.e. a sequence of X's and "bins" the latter of which are identified with children of v. An X models the arrival of a "BUYER", while $s \in$ bins models the receipt of a "ARRIVAL" message from s. (The fact that r is a-isolated means that no two of v's inputs occur at the same time in the same execution, so a unique sequence can be obtained in each case.) Let S and S' be the scripts for the first and second executions, respectively (up to the indicated times). The claims in the preceding paragraph imply that S' dominates S.

We claim that the processing described for the combinatorial problem models the processing carried out at v during execution of the algorithm. In particular, a SELECT of a bin s, if it occurs, models the sending of a "BUYER" message to s and associated reduction of v's estimate of the number of resources remaining in s's subtree. With the given correspondence between the combinatorial problem and the executions, the conclusion of Corollary 7 translates immediately into the Claim.∎

Lemma 9: If $a \geq 1$, and r is a-isolated, then $bnum_{T,p}(r) \leq bnum_{T,p}(ar)$.

Proof: By Lemma 8, taking expectations.∎

Define $bnum_{T,p}(a,f)$ to be the expected value of $bnum_{T,p}(ar)$, where r is chosen according to φ_T and f. The next theorem states that the expected number of "BUYER" messages is a monotone nondecreasing function of the interarrival time of the request distribution.

Theorem 10: (a) If $a \geq 1$, then $bnum_{T,p}(f) \leq bnum_{T,p}(a,f)$. (b) If $0 < a \leq b$, then $bnum_{T,p}(a,f) \leq bnum_{T,p}(b,f)$.

Proof: (a) If a request sequence is chosen according to φ_T and f, then with probability 1, it will be a-isolated. The result then follows from Lemma 9, by taking expectations over r.

(b) Let g be the probability density function defined by g(at) = f(t). Since $b/a \geq 1$, we can apply Part (a) to b/a and g, obtaining $bnum_{T,p}(g) \leq bnum_{T,p}(b/a,g)$. But $bnum_{T,p}(g) = bnum_{T,p}(a,f)$ and $bnum_{T,p}(b/a,g) = bnum_{T,p}(b,f)$, yielding the result.∎

4.6. Summary of Monotonicity Analysis

In this section, we have made the following restrictions, repeated here for convenience.

Restriction 1:

T is a weighted tree, and the nondeterministic choice step in Part (2) of the algorithm uses a call to $random_{T,S}$.

Restriction 2:

Delivery time for messages is always exactly 1.

Restriction 3:

T has all leaves at the same distance from the root, and r and p are nonzero only at leaves.

The major results we have proved in this section are that the expected response time is closely approximated by the expected number of "BUYER" messages (Theorem 5) and that the expected number of "BUYER" messages is a monotonic function of the interarrival time (Theorem 10). We can combine these two results, obtaining the following:

Theorem 11: $cost_{T,p}(f) \leq 4 \lim_{a \to \infty} bnum_{T,p}(a,f)$.

Proof: Consider what happens to the value of $bnum_{T,p}(a,f)$ as a increases. This value is monotonically nondecreasing, by Theorem 10. Also, it is bounded above, because no execution causes more "BUYER" messages to be sent on any edge than the number of resources initially placed below that edge. Therefore, the limit exists. The result follows immediately from Theorems 5 and 10.∎

That is, the expected cost of the algorithm for any probability function, f, is bounded in terms of the limiting case of the algorithm, as the interarrival time approaches infinity. But note that as the interarrival time approaches infinity, the algorithm gravitates towards a purely sequential algorithm - one in which each request gets satisfied before the next one arrives. This kind of sequential algorithm is amenable to analysis of a more traditional kind, the subject of the next section of this paper.

It seems quite surprising that the sequential case is the worst case. Our initial expectation was that cases where considerable interference between requests occur would be the worst case. The monotonicity theorem indicates that that is not so. Of course, we have made a few assumptions in this section, most significantly the equal lengths of branches. It is quite likely that the sequential case will not be the worst case for an algorithm using more general tree topologies. The analysis in this more general case so far seems quite intractable, however.

5. Sequential Analysis

In this section, we analyze the sequential case of the algorithm. In the next section, we combine the results into an upper bound for the entire algorithm. Once again, we allow arbitrary weighted trees T, and allow r and p to be nonzero anywhere.

5.1. A Simplified Problem and Algorithm

The sequential case of the algorithm offers considerable simplification over the concurrent cases. There is no interference at all, since each request arrives after previous requests have been satisfied. This means that all the estimates of remaining resources are completely accurate. In fact, the result is equivalent to that of an algorithm in which all information is known globally.

The behavior of the algorithm in the sequential case can be modelled by repeated calls to the following sequential program, FIND. The program takes a weighted tree, a nonnull placement, and a vertex (the vertex at which a request occurs) as input, and returns a vertex (the vertex at which the resource to be granted is located).

FIND(T,p,v)

```
Case
  p(v) > 0                              : return v
  p(desc_T(v)) = 0                    : return FIND(T,p,parent_T(v))
  p(v) = 0 and p(desc_T(v)) > 0:
    [S := {w ∈ children_T(v): p(desc_T(w)) > 0}
     return FIND(T,p,random_T,S)]
endcase
```

Thus, a request is satisfied, as before, in the smallest containing subtree which contains a resource; where there is a choice, the probability function is used.

Lemma 12: If p is nonnull and $v \in vertices_T$, then FIND(T,p,v) eventually halts and returns a vertex, v, with $p(v) > 0$.

Proof: Straightforward.∎

We next want to prove a lemma which will be useful in the later analysis. The content of the lemma is as follows. Let $random_T$ denote the probability function which returns $s \in leaves_T$ with probability $\varphi_T(s)$. Assume T is a weighted tree and p is a placement which is nonzero only at leaves. Consider the following two experiments.

(1) Call FIND(T,p,$root_T$), and

(2) Call FIND(T,p,$random_T$).

We claim that the "results" of these two experiments are the same. That is, for each $w \in vertices_T$, the probability that experiment (1) returns w is exactly the same as the probability that experiment (2) returns w. It will follow from the next lemma that this is so.

Some notation is helpful here. The result of FIND on a particular set of arguments can be expressed as a probability distribution of vertices. Let $\alpha_{T,p,v}$ denote the probability distribution of results for FIND(T,p,v). That is, FIND(T,p,v) returns w with probability $\alpha_{T,p,v}(w)$.

Lemma 13: Let T be a weighted tree, p a nonnull placement for T. Assume that p is nonzero only at leaves. Then the following are true.

(a) If $v \in internal_T$, then $\alpha_{T,p,v} = \Sigma_{w \in children_T(v)} [[\varphi_T(desc_T(w))/\varphi_T(desc_T(v))] \alpha_{T,p,w}]$.

(b) If $v \in vertices_T$, then $\alpha_{T,p,v} = \Sigma_{w \in desc_T(v)} [[\varphi_T(w)/\varphi_T(desc_T(v))] \alpha_{T,p,w}]$.

Proof: In the proof, we assume T and p are fixed and write α_v in place of $\alpha_{T,p,v}$, etc.

(a) We consider cases.

Case 1: $p(desc_T(v)) = 0$

Then since the algorithm immediately calls FIND on $parent_T(v)$, we see that $\alpha_v = \alpha_{parent_T(v)}$. Similarly, for all children, w, of v, we have $\alpha_w = \alpha_{parent_T(v)}$. Since

$\Sigma_{w \in children_T(v)} [\varphi_T(desc_T(w))/\varphi_T(desc_T(v))] = 1$, the result follows.

Case 2: $p(desc_T(v)) > 0$

Since we are assuming that $v \in internal_T$, we know that $p(v) = 0$. Let S =

$\{w \in children_T(v): p(desc_T(w)) > 0\}$. Then $S \neq \emptyset$. The third case in the algorithm holds, and we have that

$\alpha_v = \Sigma_{w \in S} [[\varphi(desc_T(w))/\varphi_T(desc_T(S))] \alpha_w]$. Now,

$\Sigma_{w \in children_T(v)} [[\varphi_T(desc_T(w))/\varphi_T(desc_T(v))] \alpha_w]$

$= \Sigma_{w \in S} [[\varphi_T(\text{desc}_T(w))/\varphi_T(\text{desc}_T(v))]\ \alpha_w]$

$+ \Sigma_{w \in \text{children}_T(v)-S} [[\varphi_T(\text{desc}_T(w))/\varphi_T(\text{desc}_T(v))]\ \alpha_w].$

By the remark above, this sum is equal to

$\alpha_v\ [\varphi_T(\text{desc}_T(S))/\varphi_T(\text{desc}_T(v))]$

$+ \Sigma_{w \in \text{children}_T(v)-S} [[\varphi_T(\text{desc}_T(w))/\varphi_T(\text{desc}_T(v))]\ \alpha_w].$

If $w \in \text{children}_T(v) - S$, we know that $p(\text{desc}_T(w)) = 0$, so that $\alpha_w = \alpha_{\text{parent}_T(w)} = \alpha_v$. So the sum above is equal to

$\alpha_v\ [\varphi_T(\text{desc}_T(S))/\varphi_T(\text{desc}_T(v))] + \alpha_v\ \Sigma_{w \in \text{children}_T(v)-S} [\varphi_T(\text{desc}_T(w))/\varphi_T(\text{desc}_T(v))],$

$= \alpha_v\ [\varphi_T(\text{desc}_T(S))/\varphi_T(\text{desc}_T(v))] + \alpha_v\ [(\varphi_T(\text{desc}_T(v)) - \varphi_T(\text{desc}_T(S)))/\ \varphi_T(\text{desc}_T(v))],$

$= \alpha_v.$

(b) We proceed by induction on the height of v.

Base: $v \in \text{leaves}_T$

Then the only w in $\text{desc}_T(v)$ is v itself, so the sum on the right is just $[\varphi_T(v)/\varphi_T(v)]\ \alpha_v$,

$= \alpha_v$, as needed.

Inductive step: $v \in \text{internal}_T$

Then $\alpha_v = \Sigma_{w \in \text{children}_T(v)} [[\varphi_T(\text{desc}_T(w))/\varphi_T(\text{desc}_T(v))]\ \alpha_w]$, by part (a),

$= \Sigma_{w \in \text{children}_T(v)} [[\varphi_T(\text{desc}_T(w))/\varphi_T(\text{desc}_T(v))]\ \Sigma_{s \in \text{desc}_T(w)} [[\varphi_T(s)/\varphi_T(\text{desc}_T(w))]\ \alpha_s]]$, by inductive hypothesis,

$= \Sigma_{w \in \text{children}_T(v)}\ \Sigma_{s \in \text{desc}_T(w)} [[\varphi_T(s)/\varphi_T(\text{desc}_T(v))]\ \alpha_s],$

$= \Sigma_{s \in \text{desc}_T(v)} [[\varphi_T(s)/\varphi_T(\text{desc}_T(v))]\ \alpha_s],$

as needed.∎

Part (b) of this lemma, with $v = \text{root}_T$, proves equivalence of the two experiments described prior to the lemma.

We can restrict attention to "request sequences" in place of request patterns, in the sequential case of the algorithm. Assume that T is a weighted tree, and p is a placement for T. A *request sequence*, r, is a sequence of elements of $vertices_T$, representing a sequence of request arrival locations. Similarly, a *resource sequence*, r, is a sequence of elements of $vertices_T$, representing a sequence of resource locations. In either case, let *length(r)* denote the number of elements in a sequence. A resource sequence, s, is *compatible* with a placement, p, provided that $|s^{-1}(v)| \leq p(v)$ for each $v \in vertices_T$. (That is, the resource sequence grants at most the number of resources placed at each vertex.) If r is a request sequence and p is a placement with $total(p) \geq length(r)$, then a *matching* of r and p is a pair m = (r,s), where s is a resource sequence compatible with p and length(r) = length(s). A matching describes the successive locations of resources which are used to satisfy a sequence of requests.

Next, we define a probabilistic program which takes as inputs a request sequence, r, and a placement, p, with $total(p) \geq length(r)$, and returns a matching of r and p. The procedure simply uses FIND repeatedly.

MATCH(T,p,r)

```
For i = 1 to length(r) do
  [s(i) := FIND(T,p,r(i))
   p(s(i)) := p(s(i)) - 1]
```

Theorem 14: Let r be a request sequence, p a placement with $total(p) \geq length(r)$. Then MATCH(T,p,r) will eventually halt and return a matching of r and p.

Proof: Straightforward.∎

This algorithm is designed to behave exactly as the sequential case of our general algorithm.

5.2. Cost Measures

Let $dist_T(u,v)$ denote the tree distance between u and v. If m = (r,s) is a matching, then $seqcost_{T,p}(m) = \Sigma_i dist_T(r(i),s(i))$. Thus, the "sequential cost" is just the sum of the tree distances between successive requests and their corresponding resources.

If r is a request sequence with $length(r) \leq total(p)$, then define $seqcost_{T,p}(r)$ to be the expected value of $seqcost_{T,p}(m)$, where m is constructed using MATCH(T,p,r). Let $seqcost_{T,p}$ denote the expected value of $seqcost_{T,p}(r)$, where r is of length total(p), with its successive locations chosen independently according to φ_T.

In the remainder of this section, we analyze $seqcost_{T,p}$.

5.3. A Useful Recurrence

In this subsection, we present a solution to a system of recurrence equations. This solution will be useful in later subsections.

Let $c \in R^+$. Define $G_c: N \times R^+ \rightarrow R^+$ by the equations:

$G_c(0,t) = 0$, and $G_c(k,t) = \max\{G_c(k\text{-}1,t_1) + G_c(k\text{-}1,t_2): t_1 + t_2 \leq t\} + ck\sqrt{t}$, for $k \geq 1$.

Lemma 15: For all $k \geq 0$, the following are true:

(a) The function mapping t to $G_c(k,t)$ is concave downward and monotone nondecreasing.

(b) If $k \geq 1$, then $G_c(k,t) = 2G_c(k\text{-}1,t/2) + ck\sqrt{t}$.

Proof: We proceed by induction on k. The base, $k = 0$, is trivial. For the inductive step, let $k \geq 1$. If $t_1 + t_2 \leq t$, then $G_c(k\text{-}1,t_1) + G_c(k\text{-}1,t_2) \leq 2G_c(k\text{-}1,(t_1 + t_2)/2)$, since the inductive hypothesis states that the function mapping t to $G_c(k\text{-}1,t)$ is concave. This is in turn $\leq 2G_c(k\text{-}1,t/2)$, by monotonicity. Therefore, $G_c(k,t) = 2G_c(k\text{-}1,t/2) + ck\sqrt{t}$, showing (b). Since each term is concave and monotone, the sum is also, showing (a).∎

Theorem 16: $G_c(k,t) \leq (3\sqrt{2} + 4)c\sqrt{t}\,2^k$.

Proof: By Lemma 15, $G_c(k,t) = 0$ if $k = 0$, and $= 2G_c(k\text{-}1,t/2) + ck\sqrt{t}$ if $k \geq 1$. Expanding this recurrence, we see that $G_c(k,t)$

$= c[\Sigma_{i=0,\ldots,k-1} 2^i(k\text{-}i)\sqrt{t/2^i}]$ for all $k \geq 0$,

$= c\sqrt{t}[\Sigma_{i=0,\ldots,k-1}(k\text{-}i)\sqrt{2^i}]$.

Let $x = 1/\sqrt{2}$, $n = 2^k$. Then

$G_c(k,t) = (c\sqrt{t}\sqrt{n}/\sqrt{2})[\Sigma_{i=0,\ldots,k-1}(k\text{-}i)x^{k-i-1}]$,

$= (c\sqrt{t}\sqrt{n}/\sqrt{2})[(1 + kx^{k+1} - (k+1)x^k)/(1\text{-}x)^2]$,

$= (c\sqrt{t}\sqrt{n})/(\sqrt{2}(1 - 1/\sqrt{2})^2[1 + kx^{k+1} - (k+1)x^k]$,

$= (c\sqrt{t}\sqrt{n})(3\sqrt{2} + 4)[1 + kx^{k+1} - (k+1)x^k]$,

$= (c\sqrt{t}\sqrt{n})(3\sqrt{2} + 4)[1 + (kx - k - 1)x^k]$,

$\leq (c\sqrt{t}\sqrt{n})(3\sqrt{2} + 4)$, since $kx - k - 1 < 0$,

$= (c\sqrt{t}\,2^k)(3\sqrt{2} + 4)$.∎

5.4. Recursive Analysis

Now, we require Restriction 3 and a new assumption, Restriction 4. These are to remain in force for the remainder of Section 5.

Restriction 3:

T has all leaves at the same distance from the root, and r and p are nonzero only at leaves.

Restriction 4:

T is a complete binary tree.

If T is a weighted tree, then a *weighted subtree*, T', of T consists of a subtree of T together with a probability function, $\varphi_{T'}$, given by $\varphi_{T'}(v) = \varphi_T(v)/\varphi_T(\text{vertices}_{T'})$. That is, the weights of the subtrees are just normalized restrictions of the weights of T. If T is a weighted binary tree, let $left_T$ and $right_T$ denote the designated weighted subtrees of T.

If T has height at least 1, then let T_1 and T_2 denote left_T and right_T, respectively. Let p_1 and p_2 denote $p|T_1$ and $p|T_2$, respectively. If r is a request sequence, let $overflow_{T,p}(r)$ denote $|\|r^{-1}(\text{vertices}_{T_1})| - p(\text{vertices}_{T_1})|$, the difference between the number of requests that arrive in the left subtree and the number of resources placed there. Let $overflow_{T,p}$ denote the expected value of $\text{overflow}_{T,p}(r)$, where r is a sequence of length total(p) chosen using φ_T.

The following is a key lemma which provides a recursive breakdown for the sequential cost. It says that the expected cost of matching breaks down into costs of matching within the two subtrees, plus a charge for the requests that overflow into the opposite subtree.

Lemma 17: Assume $\text{height}_T \geq 1$. Then $\text{seqcost}_{T,p} \leq \text{seqcost}_{T_1, p_1} + \text{seqcost}_{T_2,p_2} + 2\,\text{height}_T\,\text{overflow}_{T,p}$.

Proof: For any particular request sequence, r, there is some particular number, $\text{overflow}_{T,p}(r)$, of requests that do not get satisfied within their own subtree, but rather overflow into the opposite subtree to find a resource. To be specific, assume that it is the left subtree from which any excess requests overflow. Let r_1 be the subsequence of r consisting of requests arriving in T_1, truncated to length $p(T_1)$. Let r_2 be the subsequence of r consisting of requests arriving in T_2. Recall that $\text{seqcost}_{T,p}$ is the expectation of the search cost for enough requests to exhaust all resources present.

Before the time at which the left subtree overflows, the algorithm MATCH(T,p,r) runs exactly like $\text{MATCH}(T_1,p_1,r_1)$ within the left subtree. Requests originating within T_1 become matched to exactly the same resources in both executions.

We now consider the right subtree. Requests which originate within the right subtree are handled in the algorithm MATCH(T,p,r) exactly as they are in the algorithm for T_2 and p_2. However, there are also overflow requests from the left subtree, which enter T_2 at its root rather than at its leaves. By Lemma 13, whenever such a request arrives, its probability of being matched to any particular resource is exactly the same as if the

request had entered at a random leaf of T_2. All the requests remain independent, and these additional requests are just enough to exhaust the resources in T_2.

Now assume that the request sequences, r, are chosen at random according to φ_T. They result in subsequences r_1 and r_2 which are chosen at random according to φ_{T_1} and φ_{T_2}, respectively.

We claim that $seqcost_{T,p}$, the expected cost taken over all r, is bounded by

$seqcost_{T_1,p_1}$, the expected cost taken over random sequences in the left subtree,

+ $seqcost_{T_2,p_2}$, the expected cost taken over random sequences in the right subtree,

+ $2\,height_T\,overflow_{T,p}$.

The third term allows for the expected overflow of requests, and assigns them the maximum cost, $2\,height_T$.

Consider the first term. (The second term is analogous.) The first term allows for the expected cost incurred by an execution of MATCH on a random sequence of requests within T_1.

In case T_1 has its resources exhausted by requests originating within that subtree, this term measures exactly the expected cost for the matching of the first requests in T_1 to all the resources in T_1. This term ignores the cost incurred by any excess requests originating within T_1 which do not get matched within T_1. However, that is not a problem, since those costs are counted by the third term.

In case T_1 does not have its resources exhausted by requests originating within T_1, this term is actually greater than needed to measure the expected cost of matching all requests originating in T_1; in fact, it is enough to measure this cost of matching these requests, interspersed with enough other random requests (arriving at the leaves of T_1) to use up all the resources in T_1. We have already argued that these requests behave as if they were interspersed with other random requests, because requests arriving at the root match in the same way as if they arrived at random leaves. In this case, the first term does not account for the cost of matching those requests which enter T_1 at its root. However, that is not a problem since that cost is covered by the third term.∎

5.5. d-Fairness

We need to make another restriction on the algorithm, for the purpose of analysis. In particular, it is reasonable that the behavior of the algorithm should be best when the resources are distributed in the tree in some relationship with the probability distribution governing arrival of requests. (The paper [Fischer, Griffeth, Guibas and Lynch (1981)] considers optimal placements of resources in a tree network.)

For $d \in R^+$, we say that a placement, p, is *d-fair* provided that the following is true. Let $u,v \in$ vertices$_T$, where $u \in$ desc$_T$(v). Let $\varphi_1 = \varphi_T$(desc$_T$(v)) and $\varphi_2 = \varphi_T$(desc$_T$(u)). Then $|p(\text{desc}_T(u)) - (\varphi_2/\varphi_1)p(\text{desc}_T(v))| \leq d\sqrt{(\varphi_2/\varphi_1)p(\text{desc}_T(v))}$. That is, for each subtree, the number of resources placed in each of its subtrees is approximately proportional to the probability of arrivals in that subtree, and the difference is proportional to the "standard deviation". For any T, if t and d are sufficiently large, then techniques in [Fischer, Griffeth, Guibas and Lynch (1981)] can be used to produce d-fair placements of t resources in T.

From now on in Section 5, we assume the following.

Restriction 5:

p is d-fair (for some arbitrary but fixed d).

The next lemma says that restrictions of d-fair placements are also d-fair.

Lemma 18: Let p be a d-fair placement for T. Let T' be any subtree of T, and p' = p|vertices$_{T'}$, the restriction of p to vertices$_{T'}$. Then p' is d-fair for T'.

Proof: Let u, v $\in$ vertices$_{T'}$, with $u \in$ desc$_{T'}$(v). Let $\varphi_1 = \varphi_T$(desc$_T$(v)), $\varphi_2 = \varphi_T$(desc$_T$(u)), $\varphi'_1 = \varphi_{T'}$(desc$_{T'}$(v)), and $\varphi'_2 = \varphi_{T'}$(desc$_{T'}$(u)). Then $\varphi'_1 = \varphi_1/\varphi_T$(vertices$_{T'}$) and $\varphi'_2 = \varphi_2/\varphi_T$(vertices$_{T'}$), by definition. Therefore, $\varphi'(2)/\varphi'(1) = \varphi_2/\varphi_1$.

Note that p'(desc$_{T'}$(u)) = p(desc$_T$(u)), and p'(desc$_{T'}$(v)) = p(desc$_T$(v)). Thus, $|p'(\text{desc}_{T'}(u)) - (\varphi'_2/\varphi'_1)\, p'(\text{desc}_{T'}(v))|$

$= |p(\text{desc}_T(u)) - (\varphi_2/\varphi_1)\, p(\text{desc}_T(v))|$,

$\leq d\sqrt{(\varphi_2/\varphi_1)p(\text{desc}_T(v))}$ since p is d-fair,

$= d\sqrt{(\varphi'_2/\varphi'_1)p'(\text{desc}_{T'}(v))}$, as needed.∎

The final lemma of this subsection bounds the expected overflow for d-fair placements.

Lemma 19: $\text{overflow}_{T,p} \leq (6 + d)\sqrt{\varphi_T(\text{vertices}_{T_1})\,\text{total}(p)}$.

Proof: $\|r^{-1}(\text{vertices}_{T_1})| - p(\text{vertices}_{T_1})| \leq \|r^{-1}(\text{vertices}_{T_1})| - \varphi_T(\text{vertices}_{T_1})\,\text{total}(p)| + |\varphi_T(\text{vertices}_{T_1})\,\text{total}(p)| - p(\text{vertices}_{T_1})|$.

The expected value of the first of these quantities, taken over r, is bounded by $6\sqrt{\varphi_T(\text{vertices}_{T_1})\,\text{total}(p)}$, using Lemma 3.1.5 of [Fischer, Griffeth, Guibas and Lynch (1981)].

The second quantity is bounded by $d\sqrt{\varphi_T(\text{vertices}_{T_1})\,\text{total}(p)}$, since p is d-fair.∎

5.6. Sequential Analysis

Let $size_T$ denote the number of vertices in T. We now give the main result of Section 5, that $seqcost_{T,p}$ is $O(\sqrt{\ } size_T\ total(p))$. This says, for example, that if total(p) is proportional to $size_T$, then the total cost is proportional to total(p). This implies that the average cost per request is just a constant, independent of the size of the network.

In order to prove this theorem, we require the following restrictions, repeated here for convenience.

Restriction 3:

T has all leaves at the same distance from the root, and r and p are nonzero only at leaves.

Restriction 4:

T is a complete binary tree.

Theorem 20: $seqcost_{T,p}$ is $O(\sqrt{\ } size_T\ total(p))$.

(More specifically, $seqcost_{T,p} \leq (3\sqrt{2} + 4)((6+d)\sqrt{2})\sqrt{2^{height_T}}\ total(p)$.)

Proof: By Theorem 16, it suffices to show that $seqcost_{T,p} \leq G_c(height_T, total(p))$, where $c = (6+d)\sqrt{2}$.

We proceed by induction on $height_T$.

Base: $height_T = 0$

Then T has a single vertex, and $seqcost_{T,p} = 0$. The inequality is immediate.

Inductive Step: $height_T \geq 1$

Then $seqcost_{T,p} \leq seqcost_{T_1,p_1} + seqcost_{T_2,p_2} + 2\ height_T\ overflow_{T,p}$, by Lemma 17,

$\leq seqcost_{T_1,p_1} + seqcost_{T_2,p_2} + 2\ height_T\ (6+d)\sqrt{\varphi_T(vertices_{T_1})}\ total(p)$, by Lemma 19,

A similar inequality holds for T_2 in place of T_1 within the square root. Since at least one of $\varphi_T(vertices_{T_1})$, $\varphi_T(vertices_{T_2})$ is no more than 1/2, it follows that $seqcost_{T,p} \leq$

$seqcost_{T_1,p_1} + seqcost_{T_2,p_2} + 2\ height_T\ (6+d)\sqrt{(1/2)}\ total(p)$,

$= seqcost_{T_1,p_1} + seqcost_{T_2,p_2} + (6+d)\sqrt{2}\ height_T\ \sqrt{\ } total(p)$.

By Lemma 18, we can apply the inductive hypothesis, which implies that the right hand side of this inequality is at most equal to

$$G_c(height_T - 1, total(p_1)) + G_c(height_T - 1, total(p_2)) + (6+d)\sqrt{2\, height_T}\,\sqrt{total(p)}.$$

The definition of G_c implies that this latter expression is at most equal to

$G_c(height_T, total(p))$, as needed.∎

6. The Final Analysis

In this section, we combine the monotonicity analysis and the sequential analysis, to obtain an upper bound for the expected cost for the algorithm.

6.1. Relationship Between the Costs

Now we require the following restrictions.

Restriction 1:

T is a weighted tree, and the nondeterministic choice step in Part (2) of the algorithm uses a call to $random_{T,S}$.

Restriction 2:

Delivery time for messages is always exactly 1.

With these restrictions, there is a close relationship between the costs of our general algorithm and the cost of the sequential algorithm MATCH.

Lemma 21: $seqcost_{T,p} = \lim_{a \to \infty}(bnum_{T,p}(a,f) + vbnum_{T,p}(a,f))$.

Proof: There is an absolute upper bound on the time for our algorithm to satisfy a single request, in the absence of concurrent requests. Thus, as a increases, the probability that there are any concurrent requests approaches 0.

Therefore, the limiting case of the general algorithm behaves like MATCH. There is no backtracking, so the total search time just reduces to the sum of the distances from requests to the resources which satisfy them. This sum is just the total number of buyer and virtual buyer messages.∎

Now let us add one more restriction:

Restriction 3:

T has all leaves at the same distance from the root, and r and p are nonzero only at leaves.

With this added restriction, we can prove a variant of the preceding lemma.

Lemma 22: $\text{seqcost}_{T,p} = 2 \lim_{a \to \infty}(\text{bnum}_{T,p}(a,f))$.

Proof: By Lemmas 21 and 4.∎

This immediately implies the following bound on the cost of the general algorithm.

Lemma 23: $\text{cost}_{T,p}(f) \leq 2\,\text{seqcost}_{T,p}$.

Proof: By Theorem 11 and Lemma 22.∎

6.2. The Main Theorem

Now we are ready to present the main result, the upper bound for the expected cost for the general algorithm In order to apply the results of both the monotonicity analysis and the sequential analysis, we must assume the restrictions made for both cases. More specifically, we assume all of Restrictions 1-5:

Restriction 1:

T is a weighted tree, and the nondeterministic choice step in Part (2) of the algorithm uses a call to $\text{random}_{T,S}$.

Restriction 2:

Delivery time for messages is always exactly 1.

Restriction 3:

T has all leaves at the same distance from the root, and r and p are nonzero only at leaves.

Restriction 4:

T is a complete binary tree.

Restriction 5:

p is d-fair (for some arbitrary but fixed d).

Theorem 24: Let f be a probability function. Then $\text{cost}_{T,p}(f)$ is $O(\sqrt{\text{size}_T}\ \text{total}(p))$.

(More specifically, $\text{cost}_{T,p}(f) \leq 2(3\sqrt{2} + 4)((6+d)\sqrt{2})\sqrt{2^{\text{height}_T}}\ \text{total}(p)$.)

Proof: By Lemma 23 and Theorem 20.∎

In particular, provided that total(p) is proportional to size_T, the expected average time taken by this algorithm to satisfy a single request is constant, independent of the size of the network.

Remark: It is possible to prove a variant of Theorem 24, for the case in which the placement p is chosen at random using φ_T, (just as the request locations are chosen), rather than being d-fair. We sketch the ideas briefly.

First, we must extend the cost definitions to include expectations taken over placements of a particular length. Thus, we define $cost_{T,t}(f)$ to be the expected value of $cost_{T,p}(f)$ for p with total(p) = t. Analogous definitions are made for $seqcost_{T,t}$ and $overflow_{T,t}$. Lemma 23 then implies that $cost_{T,t}(f) \leq 2\ seqcost_{T,t}$. It is also easy to see that $overflow_{T,t} \leq d \sqrt{\ } \varphi_T(vertices_{T_1})\ t$, for some constant d.

Next, we prove a consequence of Lemma 17 which says that $seqcost_{T,t} \leq Exp_{(t_1,t_2) \text{ with } t_1 + t_2 = t} (seqcost_{T_1, t_1} + seqcost_{T_2,t_2}) + 2\ height_T\ overflow_{T,t}$. (Here, the expectation is taken over pairs which are obtained by using φ_T to assign resources to T_1 or T_2.) This obviously implies that $seqcost_{T,t} \leq \max_{(t_1,t_2) \text{ with } t_1 + t_2 = t} (seqcost_{T_1, t_1} + seqcost_{T_2,t_2}) + 2\ height_T\ overflow_{T,t}$.

Now we prove a variant of Theorem 20 which says that $seqcost_{T,t}$ is $O(\sqrt{\ } size_T\ t)$. More specifically, we show that $seqcost_{T,t} \leq G_c(height_T,t)$, where $c = d \sqrt{\ } 2$. This is easily done by induction as before, using the new lemmas just described.

Combining these results, we see that $cost_{T,t}(f)$ is $O(\sqrt{\ } size_T\ t)$.

7. Remaining Questions

There are several directions for remaining research.

First, we would like to extend the analysis of the general algorithm. We would like to loosen our restrictions on tree shape, message delivery time, and locations for resources and requests. If we do this, is it possible to carry out an analysis similar to the one in this paper? In particular, can the concurrent cases of the algorithm still be bounded in some way in terms of the sequential case?

We would also like to extend the analysis of the sequential algorithm, MATCH. Here, we would like to loosen restrictions on tree shape and on locations for resources and requests.

There are some apparent improvements in the algorithm, for example adjusting the probabilities for the choice among children in response to knowledge of the number of resources remaining in each subtree. While this seems like an improvement, the resulting algorithm seems harder to analyze (since the recursive decomposition doesn't appear to work). Can any simple modifications be shown to be improvements?

We would like to compare the performance of this algorithm to that of alternative algorithms which solve the same problem. We have already observed that this algorithm performs much beiter than the centralized algorithm, which locates all resources at the root. How does it compare to algorithms

which allow requests to search for resources in parallel rather than sequentially? What about algorithms which rebalance resources? Are there other interesting ideas for algorithms for this problem?

Finally, the general analysis strategy is quite attractive. Proving a monotonicity result which bounds the concurrent cases of an algorithm in terms of the sequential case, and then analyzing the sequential case by traditional techniques, appears quite tractable. The use of this strategy for our algorithm appears to depend on many special properties of the algorithm and on restrictions on the execution. Is the strategy more generally useful? For what type of algorithms can it be used?

8. References

Fischer, M., Griffeth, N., Guibas, L., and Lynch, N. (1981), Optimal placement of identical resources in a distributed network, *in* "Proc. 2nd International Conference on Distributed Computing". Also, to appear in *Infor. and Control*.

Guibas, L.J., and Liang, F. M. (1982), Systolic stacks, queues, and counters", *in* "Proc. 2nd MIT VLSI Conference".