File Allocation for a Parallel Webserver

Rahul Simha
Department of Computer Science
College of William and Mary
Williamsburg, VA 23185
simha@cs.wm.edu

B. Narahari, Hyeong-Ah Choi, Li-Chuan Chen
Department of EE&CS
George Washington University
Washington, DC 20052
{narahari,choi,chen}@seas.gwu.edu

Abstract

This paper considers the problem of allocating files in a document tree among multiple processors in a parallel webserver. It is assumed that access patterns are characterized by branching probabilities for an access that starts at a node and progresses down the tree. A combinatorial optimization problem is formulated that includes load balancing and communication costs. The general problem is shown to be NP-complete, and a pseudo-polynomial time algorithm is outlined. In addition, two fast heuristic algorithms are presented and evaluated using simulation.

1 Introduction

While the world wide web [2] hardly needs introduction, the problems posed by its increased use certainly demand close attention. These problems have to do with handling network traffic, caching and replicating data across the network, and designing fast web servers. This paper focuses on one aspect of designing fast web servers – allocating files in a parallel webserver.

We examine the problem of allocating files in a parallel webserver, when statistical information about the access patterns is known. In particular, we consider a tree-structured hierarchy of files, with access probabilities for branching among these files, and ask the question: how should these files be allocated among processors in a distributed memory parallel machine? The probabilities for branching are a reflection of the historical frequency with which a file is accessed. We formulate a discrete optimization problem and show that the problem is in general NP-complete. We first provide a pseudo-polynomial time algorithm to find the optimal allocation. Two heuristics are then proposed and compared via simulation. The first is a simple heuristic based on level ordering of tree nodes; the second is more complex, similar to a branch and bound algorithm, and has better performance.

Our particular problem formulation is motivated by several key observations. First, traffic rates (i.e., access rates) are very high at popular sites [13]. Second, the total main memory available in a parallel processor is sufficient to store files that can satisfy the vast majority of requests (without needing to retrieve from disk); for example, Kwan et al [13] note that 60 MB is enough to satisfy 95 percent of all incoming requests at the National Center for Supercomputing Applications (NCSA) webserver. Third, it is possible (see [13]) to obtain statistics on how frequently each file is accessed using logging methods. Finally, consider the “cost” of a particular allocation. A typical request that scans several files will access the processors that hold these files, thereby causing interprocessor communication costs for the request. Along with these communication costs, the load on each processor depends on the allocation. Clearly, some files are accessed more frequently than others; if many of these files were inadvertently placed together on a single processor, the system load would be out of balance and, as a result, the degree of concurrency (and therefore the throughput) is limited. These two costs, communication and load, suggest diametrically opposite strategies: to minimize communication costs, one should concentrate the files in one place and yet, to balance the load, the files should be carefully distributed. Note that the two costs are analogous to system throughput and average response time for a request. In this paper, we formulate an optimization problem that captures this intuitive tradeoff.

While our problem formulation is theoretical and some of our results are based on simulation, there are practical consequences of our work. We believe we have shed some light on one aspect of the important problem of efficiently implementing information servers. Further work along these lines could include broader cost functions, those that include connection
set up costs, burstiness in traffic and dynamic file allocations. A strong theoretical understanding of the
problem would then be used to design allocation algorithms in practice.

Prior work on efficiency issues in the world wide web have focused on routing, caching and replication — see [3, 7, 10, 13] and references therein. Our work instead considers a single collection of files ordered hierarchically and studies the problem of allocating these files among processors in a parallel machine. Thus, our work is more closely related to the file allocation problem [4, 8, 9, 12] and the general facility location problem [14]. However, it is significantly different in that we consider a specific graph structure of access among the files and consider requests that access multiple files; the literature on file allocation (or, for that matter, location theory) typically allows one file per access. Our work is also somewhat related to mapping tree data structures onto disk blocks. Again, the problem we study is quite different in the cost function: the cost of communication sets our problem apart from this body of literature as well as the literature on task mapping problems.

The problem formulation that comes closest to the one presented here is perhaps the one in [15]. In [15], n document files are organized in distinct clusters and a mapping problem is defined by requiring each among p processors in a distributed system to hold about n/p files. A cluster diameter is defined as the largest distance that needs to be traveled between two files in a cluster. The problem is shown to be NP-complete and genetic algorithms are proposed and evaluated. Our problem is considerably different in many ways. First, we explicitly consider the dependence between access rates and load: files that are more frequently accessed cause higher load. In [15], load is taken to be the average number of files per processor. Second, our document structure more closely resembles website documents. Third, our communication cost structure is quite different: whereas they are interested in heterogeneous costs between processors (appropriate for a wide-area distributed system), we consider fixed costs in a closely-coupled system. In addition, our communication costs depend on the access pattern; a request that moves back and forth between processors in our regime incurs higher communication costs that a request with fewer interprocessor accesses.

The next section describes our problem formulation. Sections 3 and 4 discuss complexity and heuristic algorithms. Simulation results will be presented in Section 5.

2 Problem Formulation

We consider a tree-structured file system, that roughly corresponds to the hierarchy of files often found under a web homepage. We believe that techniques used for the allocation of files in a tree-structured system might be extended to more general graph structures and access patterns.

The following definitions will be used in this paper:

- Let $T = (V, E)$ denote a connected directed tree graph, with root $r \in V$. In terms of our application, the root $r$ denotes the so-called 'home-page' of an information server, and each node represents a file in the system of files under this homepage. Since accesses typically start at the root and progress downwards into the tree, we will assume the tree consists of directed links that point towards the leaves. No assumption is made regarding branching factor or about any structural regularity within the tree.

- For each $v \in V$ define

$$L(v) = \{e = (v, u), e \in E\}$$

the set of all outgoing edges from node $v$. In our application, these are files that a user would access after file $v$.

- For each node $v$ that is not a leaf, we assume that probabilities of access are given for each child of $v$. That is, once a user has accessed $v$, a user will next access a child of $v$ with a certain probability. Let $\alpha(e)$ denote the probability that edge $e$ is accessed from $v$, where $e \in L(v)$. We will also use the notation $\alpha(v, u)$ for edge $(v, u)$. Naturally, we assume that

$$\forall v : \sum_{e \in L(v)} \alpha(e) = 1.$$ 

Note that the above assignment of probabilities can easily be modified to allow an access to complete at internal node $v$ and not proceed any further.

- Let $f(v)$ denote the (tree) parent of $v$.

- Let $P = \{1, \ldots, p\}$ be the set of processors.

- Let $h : V \to P$ denote a mapping or allocation of files to processors. We will use the notation $h^{-1}$ to mean the pre-image of $h$, i.e., for $i \in P$, $h^{-1}(i) \subseteq P$. 

17
We note that our model assumes that the probabilities $\alpha(e)$ will be computed based on statistics collected on the frequency of references to each file. For example, the number of times the home page, or root node $r$, is accessed $b$ denotes the total number of accesses. The number of times $b$, each file is accessed can now be used to calculate the probabilities $\alpha(e)$ and $\beta(v)$. It can also be noted that the computation cost of each file can also be incorporated into the problem model without affecting the complexity of the problem.

Given the definitions above, we next construct our objective function. Some additional definitions and notation will be needed. Define $\beta(v)$, the probability that node $v$ is accessed on a random access, as follows:

$$\beta(v) = \begin{cases} 1, & v = r \\ \beta(f(v))\alpha(f(v), v), & \text{otherwise} \end{cases}$$

This is simply the product of (conditional) probabilities from $r$ to $v$.

Next, we focus on the communication cost incurred by an assignment $h$. Note that any assignment $h$ maps files to processors. Thus, any access to the system (i.e., a traversal from the root to a leaf of the tree) is also a traversal of processors (those processors which contain the files accessed). Thus, communication costs are incurred whenever the access moves from one processor to another. Note that some processors may repeat in the sequence of processors; we count a communication cost of $c[j,l]$ for every ‘switch’ from processor $j$ to processor $l$. This notion corresponds to context switching in operating systems (in which case the cost $c[j,l]$ can simply defined to be unit cost). Accordingly, consider a vertex $v \in V$ and suppose $v_1 = r, v_2, \ldots, v_{k-1}, v_k = v$ is the path from the root $r$ to $v$ and define

$$\delta_i = \begin{cases} c[h(v_i), h(v_j)] & \text{if } h(v_i) \neq h(v_{i+1}) \text{ for } i = 1, \ldots, k - 1 \\ 0, & \text{otherwise} \end{cases}$$

Next, let

$$\delta(v) = \sum_{i=1}^{k-1} \delta_i.$$ 

Then, $\delta(v)$ is the number of times we switch processors in traversing the path from the root $r$ to file $v$. Now, we can define the expected communication cost as:

$$R(T, h) = \sum_{v \in V} \beta(v)\delta(v).$$

Note: The cost $c[j,l]$ between processors denotes the cost of switching from processor $j$ to processor $l$ and may include cost of context switching and the cost of message transmission from $j$ to $l$ thereby allowing us to model the architecture topology.

Observe that if our only goal was to minimize communication costs, we would place all files on a single processor. This extreme allocation, however, would cause severe load imbalance. Thus, a tradeoff is captured by also defining a load cost $L(T, h)$ which we take to be the maximum processor load, or the access probability of the processor that is most frequently accessed:

$$L(T, h) = \max_{v \in P} \sum_{v \in h^{-1}(i)} \beta(v).$$

We are now in a position to formally state our problem of interest. Since we have two diametrically opposed costs, we could state the problem in two ways:

1. Given load constraint $\lambda$, minimize $R(T, h)$ such that $L(T, h) \leq \lambda$, i.e., find an allocation $h^*$ such that $R(T, h^*) \leq R(T, h)$ for any $h$ satisfying $L(T, h) \leq \lambda$.

2. Given communication cost constraint $\rho$, minimize $L(T, h)$, i.e., find an allocation $h^*$ such that $L(T, h^*) \leq L(T, h)$ for every $h$ satisfying $R(T, h) \leq \rho$.

Alternatively, we could define a single objective function as a weighted sum of the two quantities, load and communication. In this paper, we will address the first version of the problem.

3 Complexity

This section shows the NP-completeness of the problem and then outlines a pseudo-polynomial time algorithm to find the optimal allocation.

The decision version of our optimization problem can be stated as:

- **Tree-Allocation**: Given graph $T = (V,E)$, processors $P = \{1, \ldots, p\}$, access probabilities $\alpha(e), e \in E$ and numbers $\lambda, \rho$, is there an allocation $h$ such that $R(T, h) \leq \rho$ and $L(T, h) \leq \lambda$.

**Theorem 3.1** Tree-Allocation is NP-complete even when number of processors $p = 2$.

The details of the proof are given in [16]. The tree allocation problem is seen to be NP-complete in the weak sense, i.e., there may exist a pseudo-polynomial time algorithm to find an optimal allocation for any
given (fixed) value of \( p \). We next outline a pseudo-polynomial time algorithm, Allocate-Tree, which uses dynamic programming to find an optimal allocation for a given value of \( p \). The details of the algorithm are included in [16]. The algorithm provides a dynamic programming formulation to compute the variable \( R[v, j, \lambda] \) where

- \( \lambda \) is a load capacity vector \( \lambda_1, \lambda_2, ..., \lambda_p \). Each \( \lambda_i \) denotes the current load capacity of processor \( i \). \( \lambda \) denotes the amount of remaining load we can currently assign to processor \( i \) without exceeding the load limit \( \lambda_{\text{max}} \).

- Let \( R[v, j, \lambda] \) denote the minimum communication cost for any allocation applied to the subtree rooted at \( v \), assuming that the parent of \( v \) is assigned to processor \( j \) and load capacity vector \( \lambda \) is available.

- Let \( \lambda_{\text{max}} \) be the the \( p \)-vector of load capacity constraints whose entries are all equal to \( \lambda_{\text{max}} \).

An optimal assignment for \( R[v, j, \lambda] \) is determined by searching for all assignments of the children and siblings of \( v \).

The Allocate-Tree algorithm operates in a bottom-up manner, visiting the nodes of the binary representation of \( T \) in postorder. For each node \( v \) that is visited, we compute the minimum communication cost, \( R[v, j, \lambda] \), for \( 1 \leq j \leq p \) and all admissible load capacity vectors \( \lambda \). The final value returned is \( R[v_0, 0, \lambda_{\text{max}}] \). For further details and a proof of the following theorem we refer the reader to [16].

**Theorem 3.2** Algorithm Allocate-Tree finds an optimal allocation of files to processors in pseudo polynomial time \( O(p^2d^p n^{2p+1}) \), for any fixed \( p \geq 1 \).

Note that the algorithm’s complexity is essentially \( O(n^{2p}) \), unacceptably slow by practical standards. Accordingly, we next consider some heuristics for computing an assignment of files to processors.

4 Heuristic Algorithms

We now present two heuristics to minimize communication costs while keeping the load roughly balanced. In each case, we order the nodes in \( V \) according to some criterion. For example, in the level-based heuristic, we order nodes according to the depth from the root (nodes within a level are ordered arbitrarily). Then, nodes are placed on processors in this order while keeping the load balanced. To achieve this, a simple packing approach is used, resembling the first-fit algorithm in bin packing. Define

\[
\lambda^* = \frac{1}{p} \sum_v \beta(v).
\]

Here, \( \lambda^* \) is a lower bound on the best possible load. Given an order of nodes \( v_1, \ldots, v_n \), we assign these to the first \( p - 1 \) processors such that no processor load exceeds \( \lambda^* \); the remaining nodes are assigned to processor \( p - 1 \). We expect that the load so obtained will be reasonably close to optimal (certainly no worse than twice optimal, as shown for first-fit packing).

4.1 A Level-Based Heuristic

We first consider a simple level-based heuristic. Nodes are first level-ordered, i.e., an order \( v_1, \ldots, v_n \) is created so that if \( v_i \) is at a higher level than \( v_j \) (the distance of \( v_i \) from the root is less than that of \( v_j \)), then \( i < j \). Then, the nodes are scanned in level order and added to the processors such that the load bound \( \lambda^* \) is maintained. This is possible for the first \( p - 1 \) processors; the remaining nodes are assigned (even if they exceed \( \lambda^* \) in load) to processor \( p \). Note that the level ordering can be done during the execution of the algorithm using a queue data structure. Thus, since each node is scanned once, the algorithm takes \( O(V) \) steps.

4.2 A Greedy Heuristic

Instead of describing the order of nodes created, we present the greedy algorithm directly. Essentially, the algorithm explores the tree in a breadth-first manner, adding high-probability nodes whenever possible.

To describe the algorithm, the following notation will prove useful. For any subset of nodes \( F \subseteq V \), define the neighbors of \( F \) as

\[
N(F) = \{ u : (v, u) \in E, v \in F \}.
\]

Next, let

\[
m(F) = \arg\max_{v \in V} \beta(v),
\]

i.e., \( \beta(m(F)) = \max_{v \in V} \beta(v) \). At each step of the algorithm, we do one of the following: we expand the current set of nodes (to be assigned to processor \( k \) in the for-loop) or we start a new set (when the load limit is reached). The current set of nodes, \( F \), is examined to see which outgoing edge from \( F \) has the highest probability. If the load permits, the node at the end of the edge is added to \( F \); otherwise, a new \( F \) is started. Thus, the intuition is that nodes on paths
with high probability should be grouped together as far as possible. This way, a request will avoid traversing processors while going down a high-probability path. A more complete description is given in [16].

The complexity of the algorithm is $O(V \log V)$, as the following argument shows. Note that the nodes in $V$ are scanned once. For each node scanned we compute $m(A)$, which takes $O(\log V)$ using a (max) heap (the heap is initially constructed in $O(V)$ time).

### 5 Simulation results

We have developed a simulation to compare the level-based and greedy heuristics. In each run, a random tree was generated with random branching probabilities, and the two algorithms were run. The random generation of the tree can be described as follows. The structure of the tree was specified by branching parameters $b_l, b_h$; for each node, a random number of children was generated by selected a number in the range $[b_l, b_h]$ uniformly. Then, branching probabilities were randomly assigned from a real-valued uniform(0,1) generator and normalized to obtain a probability distribution. An average over a number of runs was taken for each parameter set. Some of our simulation results are summarized in the following tables.

Tables 1 and 2 show the communication costs obtained with each heuristic, together with the percentage difference. Values are given for various numbers of processors. Table 1 corresponds to trees with low branching factors (1 or 2), resembling a binary tree, whereas Table 2 show results for trees with a higher branching factor. The results show that up to 40% difference can exist between the two algorithms, with the greedy heuristic almost always performing better. Thus, if communication costs are significant, the greedy algorithm is to be preferred; otherwise the level-based heuristic is both faster and simpler to implement.

### 6 Conclusions

This paper considered one aspect of reducing congestion in webservers — that of allocating the files at a website among processors in a parallel machine to achieve load balance and minimize communication costs. We formulated a discrete optimization problem that captured the intuitive tradeoffs between the two costs and showed that the general problem is NP-Complete. A pseudo-polynomial time algorithm was outlined. The paper then presented two heuristics, a simple level assignment and a more complex greedy algorithm, and compared their performance through simulation. The greedy algorithm was seen to be significantly better (in reduced communication cost) in most cases. Future work might include a broader class of document structures, both in graph category as well as in file types and sizes. In addition the cost function might be expanded to account for system costs such as connection setup costs, I/O and caching. In addition, the problem of dynamic reallocation of files, on the server, can be investigated.

### References


