Competitive Distributed File Allocation*

Baruch Awerbuch† Yair Bartal‡ Amos Fiat§

Abstract

This paper deals with the *file allocation* problem [BFR92] concerning the dynamic optimization of communication costs to access data in a distributed environment. We develop a dynamic file re-allocation strategy that adapts on-line to a sequence of read and write requests whose locations and relative frequencies are completely unpredictable. This is achieved by replicating the file in response to read requests and migrating the file in response to write requests, so as to be closer to processors that access it frequently.

We develop the first explicit centralized deterministic on-line strategy. Previous (deterministic) solutions were complicated and more expensive. Our solution has (asymptotically optimal) logarithmic competitive ratio. The paper also contains the first explicit deterministic *data migration* [BS89] algorithm achieving the best known competitive ratio for this problem.

Using somewhat different techniques, we also develop the first deterministic distributed file allocation algorithm (using only local information) with a poly-logarithmic competitive ratio against a globally-optimized optimal prescient strategy.

*A preliminary version of this paper appeared in the proceedings of the 25th Annual ACM Symposium on Theory of Computing, 1993, pp. 164 – 174.

†Department of Computer Science, John Hopkins University.

‡Department of Computer Science, Tel-Aviv University, Tel-Aviv 69978, Israel. Research supported in part by Ben Gurion Fellowship, the Ministry of Science and Arts. E-mail: yairb@math.tau.ac.il.

§Address correspondence to: Amos Fiat, Department of Computer Science, School of Mathematics, Tel-Aviv University, Tel-Aviv 69978, Israel. Supported by a grant from the Israeli Academy of Sciences. E-mail: fiat@math.tau.ac.il.

1 Introduction

1.1 The Problem

This paper is concerned with finding efficient strategies for replicating and migrating data in presence of dynamic patterns of reads and writes (see figure 1, details in Section 2). This is a well-studied problem in distributed systems [DF82].

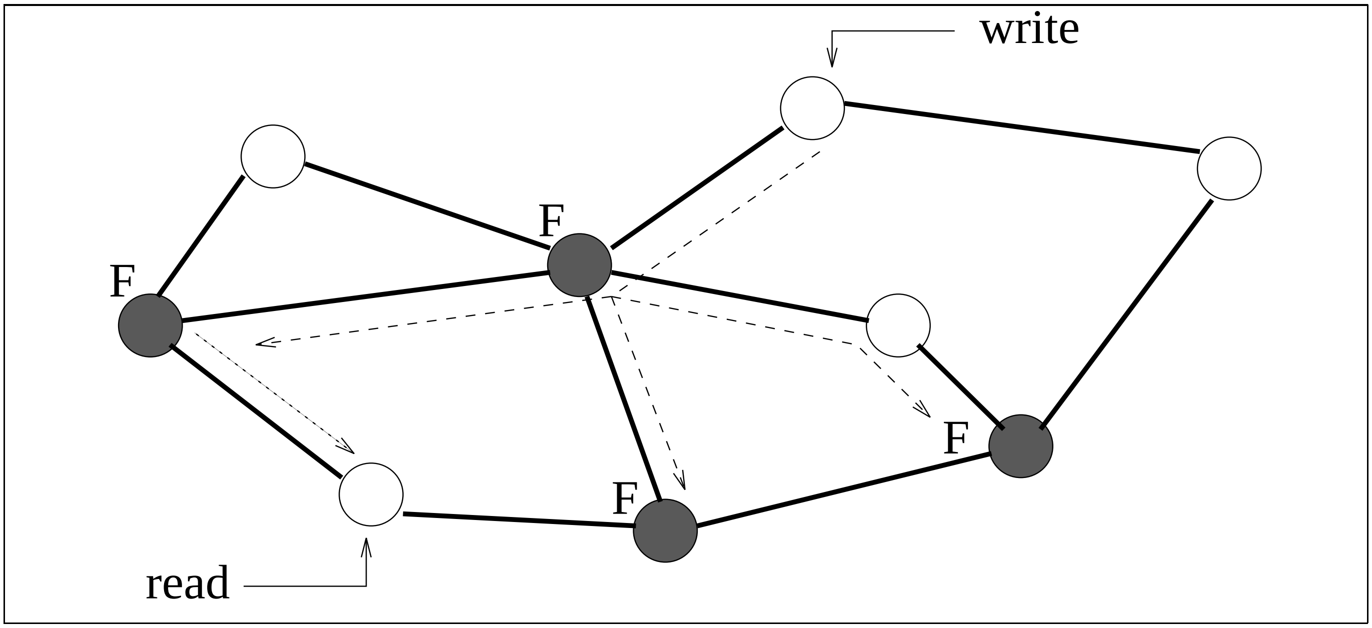

Figure 1: The file allocation problem. A subset of the network processors are holding copies of a file F. The file may be read or written by any of the processors.

In order to achieve maximum efficiency, we need to maximally exploit the locality of reference in the underlying distributed program, and incorporate it with the spatial locality of the underlying network architecture. More specifically, if many read requests to a specific file are issued in some vicinity, it is advisable to copy the relevant file to, or near, that vicinity. However, this should be balanced by the relatively greater cost of moving an entire file versus the smaller cost of transferring only the data being read. On the other hand, if the file is frequently written, then it seems advisable to remove all replica but one, and locate this replica close to the vicinity of the writes.

In this paper, we will avoid making any statistical assumptions about the locations or frequencies of read and write requests. The above conflicting heuristics must somehow be balanced in a dynamic fashion, so that the resulting algorithm automatically adapts to changing access patterns. The goal here is to find an "almost optimal" dynamic policy, rather than to learn the best "static" file assignment.

We will use, as a measure of performance, the "competitive ratio" which is defined as the ratio between the costs associated with an on-line algorithm versus the costs expended by an optimal dynamic policy (referred to as "adversary"), who has perfect knowledge of the future, to deal with the same sequence of events. To capture additional subtleties arising in distributed systems, such as the need for local control, the standard definition of competitiveness, as introduced by Sleator and Tarjan [ST85, KMRS88], needs to be refined. Below we briefly elaborate on this important issue.

1.2 Centralized & distributed models

Centralized model. In this "standard" setting, as introduced in [ST85, KMRS88], the on-line algorithm is fully aware of the global state of the system, which consists of the current configuration (i.e. the position of all file replicas) and the current input, but is unaware of the future sequence of input requests. An on-line algorithm is called c-competitive if for every sequence of requests its cost is at most c times the cost of a prescient adversary up to a constant additive term (if there is no additive term the algorithm is called strictly competitive).

In other words, we assume some "daemon" that keeps track of the migrating, replicating, and dying populations of files in the network. In particular, we assume that the daemon tells processors how to go about finding the closest current copy of every file, how to update all replicas after a write, etc. More importantly, the daemon coordinates the decisions made by different processors.

Distributed model. In a more realistic *distributed* setting, such a daemon does not exist. The decisions are made based solely on local information. It is thus up to the on-line algorithm to find out (at some cost) the relevant part of the global state necessary to take a decision; the only information that the algorithm has "for free" is the past local input. The additional handicap imposed on the on-line distributed algorithm is that it is evaluated against the off-line adversary that does *not* pay for the overhead of control needed to make an intelligent decision.

Consequently, the task of designing efficient on-line distributed algorithms is quite a challenging one and cannot be taken for granted. While certain techniques have been developed in the literature in order to competitively *locate* mobile objects based on local information [AP91, BFR92], it is hard (and sometimes provably impossible) to "localize" global-knowledge decision strategy without compromising the performance. Consequently, "good" global-knowledge on-line algorithms may prove to be absolutely useless in the distributed environment. For example, the method of

[BBK+90] for derandomizing on-line algorithms is based on obtaining global information and thus cannot be be efficiently implemented in a distributed setting.

1.3 Previous results.

The management of data in a multiprocessing environment has been extensively studied, both from the theoretical and the practical standpoints. The 1981 survey paper by Dowdy and Foster [DF82], dealing with the file allocation (or assignment) problem, cites close to a hundred references.

The file allocation problem that we deal with in this paper may be viewed as the combined solution to the two subproblems defined in [BS89]: *file migration* and *file replication*.

The file migration problem deals with a single file copy located at some processor in a network. Access requests (reads or writes) are generated in network processors, each such request is associated with the cost of the current distance between the file copy and the issuing processor. The file can also be moved to a new location, at a cost proportional to the size of the file times the distance between the source and target processor. The goal is to minimize the total cost as defined above.

The *file replication* problem defined in [BS89] deals with multiple file copies. The file replication problem is a finite game where copies of files can be made, but no copy is ever discarded. This means that when all network nodes contain a file copy then the game is over. Here, the goal is to minimize the cost of communication to the *nearest* file copy, where additional copies can be made at a cost proportional to the file size times the distance.

Our file allocation problem, defined formally below, tries to deal with multiple file copies and allows both reads and writes. A read request can go to any file copy, a write request needs to be transmitted to all file copies. Thus, it may be beneficial to reproduce several copies for a while, and then to reduce the number of copies later.

The file migration problem can be translated to the file allocation problem when only write requests are issued. The file replication problem can be translated to the file allocation problem when only read requests are issued.

For file migration, [BS89] give 3-competitive algorithms for special networks, the uniform network in which the distance between every pair of vertices is exactly one, and tree based networks. [BS89] also prove a lower bound of 3 for randomized algorithms against an adaptive on-line adversary and any non-trivial network.

For general networks, [WY93] give a randomized file migration algorithm with a competitive ratio of 3 against an adaptive on-line adversary, and a $(1+\phi)$-competitive algorithm against an oblivious adversary, where ϕ is the golden ratio. Using the results of [BBK$^+$90] this leads to a $3(1+\phi)$-competitive deterministic algorithm for the data migration problem on arbitrary networks. Optimal $2+1/(2D)$ randomized file migration algorithms against an oblivious adversary for uniform networks are given in [LRWY94] and for trees in [CLRW93]. Chrobak et. al. [CLRW93] also prove a deterministic lower bound strictly greater than 3 for some network topologies, specifically $85/27 \sim 3.148$.

In this paper we give an explicit file migration deterministic algorithm for arbitrary network topologies which is 7-competitive. Subsequent to our work, [BCI96] gave a deterministic ~ 4.086-competitive algorithm for the data migration problem. The algorithm used in [BCI96] is a small modification of the algorithm given here.

For file replication [BS89] gave a 2-competitive algorithm for uniform networks and for trees. [LRWY94] and [AK93] give optimal randomized algorithms for the same network topologies. The competitive ratio of these algorithms approach $e/(e-1)$ for large D. In [AK93] the problem is studied on rings. They give deterministic 4-competitive and randomized 3.16-competitive file replication algorithms. Our result provides the first deterministic file replication algorithm for arbitrary networks.

For the *file allocation* problem that deals with both read and write requests, [BFR92] give an optimal deterministic 3-competitive algorithm for the uniform network. [LRWY94] give an optimal deterministic 3-competitive and optimal randomized $(2+1/D)$-competitive algorithm for tree networks.

In [BFR92], randomized algorithms have been developed for the general network, with competitive ratios of $O(\min\{\log n, \log(Diam)\})$ for the centralized problem and $O(\log^4 n)$ for the distributed problem where $Diam$ denotes the diameter of the graph.

The construction of [BBK$^+$90] applied to the algorithm of [BFR92] can be used to obtain an $O(\log^2 n)$-competitive deterministic centralized on-line "algorithm". One of its disadvantages is the exponential computational cost, which makes it unattractive from the practical perspective. Since this algorithm is based on obtaining global information, it cannot be efficiently implemented in the distributed setting.

1.4 Summary of our results

In this paper, we develop the following results:

- File migration: we develop a simple deterministic 7-competitive data migration algorithm, called Move-To-Min (MTM).
- File allocation:
  - Centralized we develop a deterministic file allocation algorithm with an $O(\min\{\log n, \log(Diam)\})$ competitive ratio.
  - Distributed: we develop the first deterministic *distributed* file allocation algorithm with an $O(\log^4 n)$ competitive ratio.

1.5 Structure of the paper.

In Section 2 we give a somewhat more formal statement of the problem. Section 3 describes some basic tools. We start by presenting in section 4 the building blocks of our solution, which deal with the two special cases of the problem. The first is a simple potential function analysis of the greedy Steiner tree heuristic, and the second is a simple deterministic 7-competitive data migration algorithm. In section 5 we present a (deterministic) *centralized* file allocation algorithm which is $O(\log n)$-competitive, based on the basic ideas presented in section 4. This algorithm is optimally-competitive (up to a constant factor) for general network topologies. In section 7 we present a different (deterministic) *distributed* file allocation algorithm which is $O(\log^4 n)$-competitive.

2 Problem statement

Network model. The underlying network topology is modeled by a weighted undirected graph (also referred to as "weighted graph", or "metric space") where processors are represented by vertices, and edge weights represent transmission costs between corresponding pairs of vertices. The weighted graph need not obey the triangle inequality, but a natural metric space can be defined where the points are processors and the distance between two points is equal to the length of the shortest path between the processors in the weighted graph. The *diameter* of the graph, denoted $Diam$, is the maximal distance between a pair of vertices when the minimal non-zero edge weight has been normalized to 1.

Input/Output. The algorithm receives, as an input, the sequence of read and write requests issued at different network nodes (figure 1). As an output, it produces a sequence of *responses*, each response changing the *configuration*, namely the network nodes currently holding the replicas of the file (figure 2). The (centralized or distributed) algorithm may also maintain various data structures and change them in response to the requests.

The response is based on the information available to the algorithm. In the case of an *off-line* algorithm, this information consists of the past and the future inputs. In the case of *centralized on-line* algorithms, it consists of the past inputs. In the case of *distributed on-line* algorithms, it consists of the past *local* inputs and the part of the current global information explicitly communicated to the location at which the decision is made.

Data is assumed to be organized in indivisible blocks such as files (or pages). It can be accessed via communication links by paying a charge equal to the data transfer size times the distance traversed (in the above metric space). Individual words or records can be read or written remotely, but a file cannot be split among processors. Files may be replicated in various processors throughout the network, but copy consistency must be maintained. In other words, all file replicas must be identical and must reflect all write transactions performed until now. Each read must return the most updated value, and the value written must be propagated to all replicas.

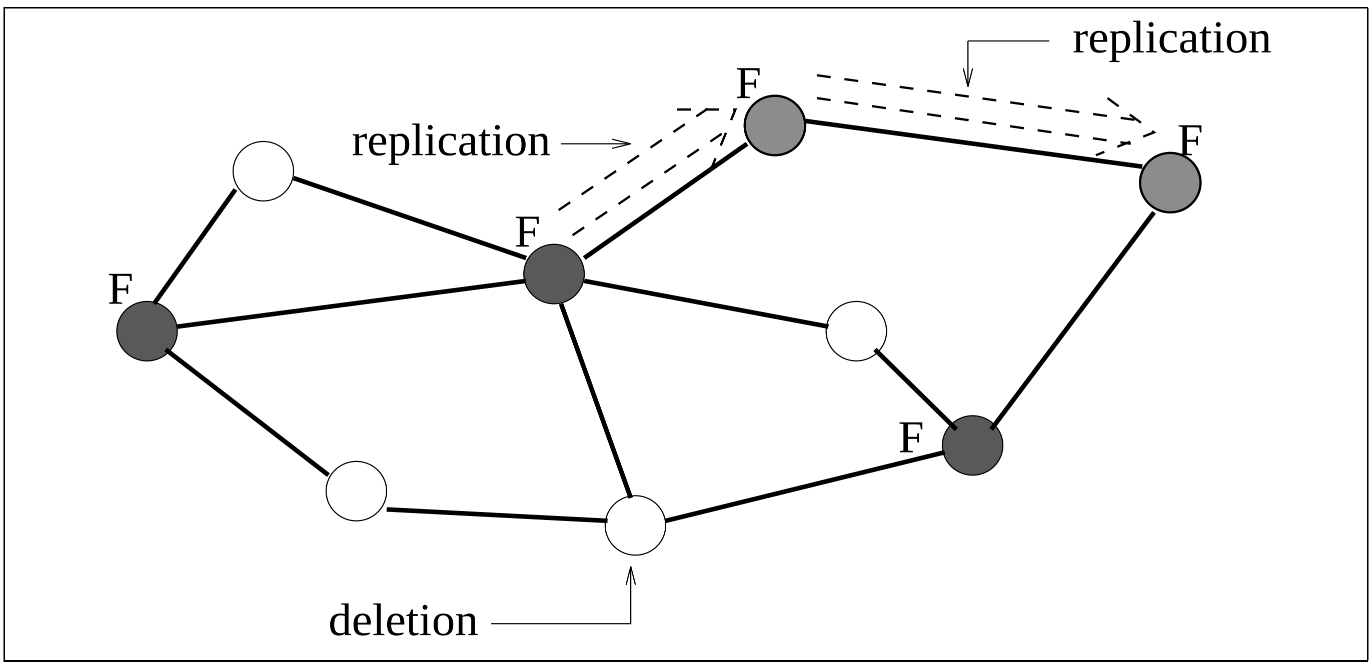

Figure 2: The file allocation operations. The file may be either replicated or deleted.

Costs. The total cost expended by a centralized (on-line or off-line) algorithm consists of the cost caused by the configuration changes, plus the cost of serving the input requests. The costs of *configuration changes* are:

- The cost of erasing a file replica is zero.
- The cost of replicating a file from one location to another is the cost of communication between these two locations, *times* the file size, D (which may be quite large).

The costs of a *request service* are as follows:

- The cost of serving a read request is the cost of communication from the location of the request to the closest replica. (The minimum cost is obtained by communicating over the shortest path.)
- The cost of serving a write request is the cost of communication from the location of the write request to all the replicas. (The minimum cost is obtained by communicating over the minimum-weight Steiner tree of the above locations.)

Without loss of generality, we can assumed that the cost of a read is the length of the shortest path to a file replica because a centralized algorithm will always use such a path. Similarly we have associated the write cost with the smallest Steiner tree because the centralized algorithm can compute it.

In the case of a distributed on-line algorithm one cannot assume that the shortest path is used to service a read or that the smallest Steiner tree is used to service a write. Additionally, one must also incorporate the the cost of control, i.e. maintenance of distributed data structures. Thus, the costs associated with a distributed file allocation algorithm are not only a function of the locations of the file replicas, but rather of the entire behavior of the algorithm.

We assume that a control message may be $O(\log n)$ bits long. When a control message is sent, the on-line algorithm pays the number of bits in the message, times the length of the path traversed by that message.

3 Preliminaries

In this section we describe some basic tools we use.

We note the all logarithms in this papers are base 2.

3.1 The Cover Problem

3.1.1 The Cover Problem Definition

The *on-line cover problem* [BFR92] is the problem of maintaining a covering of small number of small diameter subsets of a dynamically changing set in a weighted graph.

Let G be a weighted graph. Let Q be a subset of nodes of G.

For integers $r, s > 0$, a set $C = \{C_1, C_2, \ldots, C_s\}$ of mutually exclusive subsets of nodes, and a set of vertices $p_1, p_2, \ldots, p_s$, $p_i \in C_i$, is called an *r-cover* of Q iff for every i, $i = 1, 2, \ldots, s$, $Q \cap C_i \neq \emptyset$, and $Q \subset \cup_{i=1}^{s} C_i$, and for every C_i, $i = 1, 2, \ldots, s$, the distance between any node in C_i and p_i is at most r. r is called the *radius* of the cover.

Each of the sets C_i is called a *cover set*. The vertices, $p_1, p_2, \ldots, p_s$, are called *covering nodes*.

Initially the set contains a single node $Q = \{q_0\}$ and the cover contains one covering set $C_1 = \{q_0\}$, and a single covering node $p_1 = q_0$.

The on-line cover problem for some fixed parameter r is the problem of maintenance of an r-cover for a dynamic set Q, where insertions into Q and deletions from Q are allowed (but Q is never allowed to be empty), where s changes with Q.

Define the optimal cost of an insertion to be the minimum distance between a node of Q and the inserted node, and the optimal cost for a deletion to be 0.

For every integer $k > 0$, [BFR92] give an algorithm for the on-line cover problem that maintains a $2(k-1)$-cover such that for every sequence of insertions and deletions, if the optimal cost for the sequence is UPD, the final value of s is at most $1 + \mathrm{UPD}/k$.

When all operations are insertions we can define a tree S over the set of nodes Q. For every insertion we view the inserted node as a destination and its closest vertex in Q as the origin of the insertion. The tree is defined by adding an edge between the origin and destination nodes for each insertion. In this case the covering of Q can be viewed as a *tree cover*, i.e., a partition of the dynamically growing subtree S in a graph into $O(T(S)/k)$ subtrees of diameter at most k each, where $k > 0$ is some integer.

3.1.2 The Hierarchical Cover Problem

In most of the cover problem applications we need to simultaneously solve r-cover problems with $r = O(2^i)$ for all $0 \leq i \leq \log(Diam)$.

We would like to minimize the final number of covering sets.

To obtain the hierarchy of covers define i-level cover algorithms for solving the $2(k-1)$-cover problem for Q for $k = 2^i$, i.e., give all level algorithms the entire sequence of insertions and deletions. Naively, this will result in $\log(Diam)$ factor in the number of covering sets.

We have stronger claims on the performance of the hierarchical algorithm.

Let the current number of covering sets maintained by the i-level cover algorithm be denoted $s(i)$. Let the total number of covering sets creations made by the i-level cover algorithm be denoted $c(i)$. That is, both $s(i)$ and $c(i)$ increase by 1 every time a set is created, but $s(i)$ decreases by 1 every time a set is removed. Thus $s(i) \leq c(i)$.

The following theorem is a slight modification of [BFR92]:

Theorem 3.1 *The following invariant is maintained by the hierarchical cover algorithm*

$$\begin{aligned}\sum_{i=0}^{\log(Diam)} 2^i \cdot (s(i)-1) &\leq \sum_{i=0}^{\log(Diam)} 2^i \cdot (c(i)-1) \\ &\leq O(\min\{\log n, \log(Diam)\}) \cdot \text{UPD}.\end{aligned}$$

The solution and analysis of the cover problem is described in the appendix.

3.2 The On-line Steiner Tree Problem

Let G be a weighted graph. An *on-line Steiner tree algorithm* receives a sequence of vertices $\sigma = v_1, v_2, \ldots, v_\ell$ of the graph G.

In response the Steiner tree algorithm computes subtrees $T_1, T_2, \ldots, T_\ell$ of G, such that for every i, $i = 1, 2, \ldots, \ell$, T_i spans all vertices v_j, $j = 1, 2, \ldots, i$ (and possibly other vertices as well). The subtree T_i must include T_{i-1} as a subgraph. The algorithm must compute T_i independently of vertices v_j, $j > i$.

The configuration of a Steiner tree algorithm is the tree it currently holds. The cost for changing from configuration T_i to configuration T_{i+1} is defined to be equal to the sum of weights of the edges added to T_i as to obtain T_{i+1}. This is also referred to as the distance between the configurations $\text{dist}(T_i, T_{i+1})$.

For a Steiner tree algorithm Alg, the cost over an input request sequence σ, denoted $\text{cost}_{\text{Alg}}(\sigma)$, is defined to be the sum of the individual request costs: $\sum_i \text{dist}(T_i, T_{i+1})$. It follows from the definition that this cost is equal to the weight of T_ℓ.

The cost of an optimal adversary is the weight of a minimum Steiner tree spanning all vertices in σ.

In the context of the on-line Steiner tree problem we are interested in strictly competitive algorithms, that is we do not allow a positive additive term in the definition of competitiveness. Thus, the word "strictly" is often omitted when discussing the on-line Steiner tree problem.

Notation. For a weighted graph G, $d(p,q)$ denotes the weight of the minimum weighted path between vertices p and q of G . Where Q is a subset of vertices and p is a vertex of G, $d(Q,p) = \min_{q \in Q}\{d(p,q)\}$.

The k-neighborhood of a vertex v is the set of all vertices u, s.t., $d(u,v) \leq k$. This set is denoted $N_v(k)$.

$T(Q)$ denotes the weight of a minimum Steiner tree spanning the vertices in Q. $T(Q)$ is also used to denote the Steiner tree itself, and the meaning should be clear from the context.

Where S is a tree, $T(S)$ simply denotes the weight of the tree.

The on-line Steiner tree problem is equivalent to a special case of the file allocation problem, where $D = 1$, only read requests are issued, and the algorithm is forced to replicate upon a read request.

Imaze and Waxman [IW91] define this problem and give upper and lower bounds for it. They show that the greedy on-line Steiner tree algorithm is $\lceil \log n \rceil$-competitive.

For completeness of the discussion, in Section 4.1 we give a potential function proof that shows the competitive ratio of the greedy Steiner tree algorithm is $O(\min\{\log n, \log(Diam)\})$. That potential function proof gives the intuition behind our more complex file allocation algorithm proof. Analysis of the greedy Steiner tree algorithm was also independently done by [AA92], [CV92] and [WY93] giving similar bounds.

4 The Building Blocks

This section presents the two basic ideas used in the construction of the deterministic file allocation algorithm in section 5.

4.1 More on the On-line Steiner Tree Problem

The on-line Steiner tree problem is equivalent to a special case of the file allocation problem, where $D = 1$, only read requests are issued, and the algorithm is forced to replicate upon a read request.

In the following paragraph we consider the greedy strategy for the Steiner tree problem and give a simple potential function proof for the competitiveness of the greedy on-line Steiner tree algorithm, and thus to read-only replication.

Although this is not essential for the proofs of competitiveness of our file-allocation algorithms, the intuition behind this potential function serves as a basis to the more complex potential function used to prove our deterministic file-allocation algorithm, and we believe that it gives more insight into the problem.

The Greedy Steiner Tree Algorithm. The greedy Steiner tree algorithm connects a new point to the closest point already in the tree. To this algorithm corresponds a simple replication algorithm (for the read-only case), namely replicate from the closest replica.

Theorem 4.1 *The greedy Steiner tree algorithm is strictly $O(\min\{\log n, \log(Diam)\})$ competitive for any weighted graph over n vertices.*

For the definition of the potential function we need a way to form a *tree cover* (Section 3.1), i.e., a partition of a subtree S in a graph to $O(T(S)/k)$ subtrees of radius at most k each, where k is some integer. As described in Section 3.1 the tree cover problem is a restriction of the *cover problem* of [BFR92]. In fact we use the hierarchical cover problem, defined in Section 3.1, to obtain a hierarchy of tree covers.

The analysis of the Steiner tree potential function Φ is somewhat different than the usual one-step analysis, where the change in the potential function is bounded after every request. We first let the adversary serve all the requests, and then define the potential function based on its final configuration. We show that $\Phi \geq 0$. Then we prove that its initial value $\Phi_0 \leq c\times$ (adversary cost), where c is the competitive ratio. Finally we show that when the on-line algorithm serves a request the potential function decreases by at least the service cost. It follows that the on-line cost is at most c times the off-line cost.

Proof of Theorem 4.1. Let A denote the adversary optimal Steiner tree, and B denote the greedy on-line subtree.

Define a hierarchical tree cover for the adversary tree A as in Section 3.1. That is for each level $0 \leq i \leq \log(Diam)$ we have a tree cover of $s(i) = O(T(A)/2^i)$ covering subtrees of radius at most $2(2^i - 1) < 2^{i+1}$.

Let $p_1^i, p_2^i, \ldots, p_{s(i)}^i$ be the covering nodes in the i-level tree cover.

Let $k = 2^{i+1}$. For every covering node such that the on-line subtree has not reached its k-neighborhood, the potential function holds a credit of $4k$.

The potential function Φ is the sum of these credits over all levels and covering nodes:

$$\Phi \quad = \quad 4 \sum_{i=0}^{\log(Diam)} 2^{i+1} \sum_{j=1}^{s(i)} \begin{cases} 1 & \text{if } N_{p_j^i}(2^{i+1}) \cap B = \emptyset \\ 0 & \text{otherwise} \end{cases}$$

We proceed with the analysis of the potential function Φ.

Clearly $\Phi \geq 0$. Using Theorem 3.1 we obtain

$$\Phi \;\leq\; 4 \sum_{i=0}^{\log(Diam)} 2^{i+1} \cdot (s(i) - 1) \;\leq\; O(\min\{\log n, \log(Diam)\}) \cdot T(A).$$

The first inequality follows since for every level i, the first covering node is the origin node v_1 which is contained in B, and therefore at most $s(i) - 1$ nodes may donate to the value of the potential function.

Therefore we only need to show the potential function decreases by at least the on-line cost expended for adding a new vertex v to its subtree upon a request.

This cost is the distance from the nearest node in the on-line subtree to the requested node v, $d(B, v)$.

Choose $k = 2^{i+1}$ so that $4k$ is the least power of two such that $4k \geq d(B, v)$.

Since $2k < d(B, v)$ there was no vertex in the on-line subtree in the $2k$-neighborhood of v before the request.

Since $v \in A$, there is a covering node p, in the i-level tree cover, at distance at most k from v.

Let b be the nearest node in the on-line subtree, B, to node p. Then,

$$d(p, b) \;\geq\; d(v, b) - d(v, p) \;>\; 2k - k \;=\; k.$$

Thus the k-neighborhood of p did not contain a node in B before the request, but contains a new node in the on-line subtree after the request for v. We therefore conclude that the potential function decreases by at least $4k \geq d(B, v)$. ∎

We remark that a somewhat improved bound can be obtained by first dealing with all requests of cost at most $T(A)/n$ for the on-line algorithm. Since there may be at most n such requests the total cost incurred by the on-line algorithm for such requests is at most $T(A)$.

To deal with the rest of the requests tree covers are needed only for levels i, such that $T(A)/n \leq 4 \cdot 2^{i+1} \leq 2 \cdot diam(A)$. Since the maximal cost for a request is bounded by $diam(A)$.

For each level we have $2^{i+1} \cdot (s(i) - 1) \leq O(T(A))$, so that the proof gives a competitive ratio proportional to the number of levels: $O(\log(n \cdot \frac{diam(A)}{T(A)}))$.

This result was independently established in [WY93].

4.2 File Migration

While the Steiner tree problem models a special case of the *read* only file allocation problem, the data migration problem relates to file allocation with only *write* requests.

The file migration problem consists of a single copy of a file held at some processor in a network of processors, and a sequence of file accesses issued at different processors. Each access to the file costs the distance between the requesting processor and the processor holding the file copy.

In what follows we give the first deterministic algorithm for file migration on arbitrary topologies.

The following theorem states that a solution to the file migration problem also applies to the file allocation problem with only write requests.

Theorem 4.2 *Given a file migration algorithm M with a competitive ratio of c, M has competitive ratio c even against a file allocation adversary so long as only write requests are issued.*

Proof.

We need to show that an adversary A that holds multiple copies can be translated to an adversary A' that holds only a single copy at no increase in cost.

Consider any configuration computed by A during the sequence, every file copy can be traced back via a sequence of replications to some file copy in the original configuration. Define A' to have the following sequence of configurations: Take an arbirary file copy p in the final configuration and let this single copy be the final configuration of A'. The previous configuration is the single copy from which the copy in p was generated.

Clearly, the cost of the write requests for A' is no more than the cost of these requests for A.

■

Algorithm **Move-To-Min** (MTM).

The algorithm divides the request sequence into phases. Each phase consists of D consecutive accesses to the file at processors $w_1, w_2, \ldots, w_D$. During a phase the algorithm does not move the copy of the file. At the end of a phase migrate the copy to a processor m in the network such that the sum of distances from m to the w_i's is minimized.

Theorem 4.3 *Algorithm Move-To-Min is 7-competitive for file migration in arbitrary network topologies.*

Proof. We analyze the performance of the algorithm in a phase. Let $a = a_0$ denote the position of the adversary copy at the beginning of a phase, and $a_i, 1 \leq i \leq D$ denote its position after the i'th request of the phase. Also let b denote the position of the on-line copy at the beginning of the phase.

During the phase MTM incurs a cost of $d(b, w_i)$ on the i'th request. At the end of the phase its cost for migrating the file is $D \cdot d(b, m)$. Thus its total cost for the phase is

$$\begin{aligned} \text{Cost}_{\mathsf{MTM}} &= \sum_i d(b, w_i) + D \cdot d(b, m) \\ &\leq \sum_i d(a, w_i) + D \cdot d(b, a) + D \cdot d(b, m). \end{aligned}$$

The adversary incurs a cost of $d(a_{i-1}, w_i)$ on the i'th request, plus a cost of $D \cdot d(a_{i-1}, a_i)$ for its migration after the request. Thus the adversary cost during the entire phase is

$$\text{Cost}_{\text{Adv}} = D \cdot \sum_i d(a_{i-1}, a_i) + \sum_i d(a_{i-1}, w_i).$$

Using the triangle inequality we get that for any $0 \leq \ell \leq D$:

$$\begin{aligned} \text{Cost}_{\text{Adv}} &= D \cdot \sum_i d(a_{i-1}, a_i) + \sum_i d(a_{i-1}, w_i) \\ &\geq \sum_i \{d(a_\ell, a_{i-1}) + d(a_{i-1}, w_i)\} \\ &\geq \sum_i d(a_\ell, w_i). \end{aligned}$$

Thus we obtain

$$D \cdot d(a_\ell, m) \;\leq\; \sum_i d(a_\ell, w_i) + \sum_i d(m, w_i) \;\leq\; 2 \sum_i d(a_\ell, w_i) \;\leq\; 2\text{Cost}_{\text{Adv}}.$$

We use the potential function $\Phi = 2D \cdot d(a,b)$. We show that over a phase it increases by at most 7 times the adversary cost and decreases by at least the on-line cost. Let $a' = a_D$.

$$\begin{aligned}
\Delta\Phi &= 2D \cdot (d(a',m) - d(a,b)) \\
&\leq D(2d(a',m) + d(a,m) - d(b,m) - d(a,b)) \\
&= 2D \cdot d(a',m) + D \cdot d(a,m) \\
&+ \sum_i d(a,w_i) - \sum_i d(a,w_i) - D \cdot d(b,m) - D \cdot d(a,b) \\
&\leq 7\mathrm{Cost}_{\mathrm{Adv}} - \mathrm{Cost}_{\mathsf{MTM}}.
\end{aligned}$$

■

5 Centralized File Allocation

In this section we present a deterministic $O(\min\{\log n, \log(Diam)\})$-competitive file allocation algorithm for arbitrary networks.

We assume that initially the algorithm holds a single copy of the file in the network. Otherwise we can *delete* all copies but one, incurring no additional cost. Set $\lambda = 4$.

The File Allocation Algorithm (FA).
The algorithm partitions the request sequence into phases. Each phase, except perhaps for the last one, contains exactly D successive *write* requests.

Within each phase, the algorithm deals only with *read* requests.

Dealing with read requests. At all times the algorithm maintains a partial list, L, of previous read requests in the current phase.

Upon receiving a new *read* request initiated by processor r, add r to L.

If L consists of more than D *read* requests, then consider the smallest k-neighborhood of r containing D requests.

If the algorithm does not hold a copy of the file at distance less than λk from r, *replicate* a copy of the file to r from the closest processor to r holding a copy, and remove the D requests from L.

Dealing with write requests. When a phase ends, let $w_1, w_2, \ldots w_D$ be the locations of the D *writes* initiated during the phase. Let m be a processor such that the sum of distances from m

to the w_i's (i.e., $\sum_i d(m, w_i)$) is minimized. Now, *replicate* a copy of the file to m from the closest processor to it holding a copy, and then *delete* all copies of the file except the one at m. The read requests list, L, is initialized to be empty. This procedure is called the *reorganization step.*

m can also be chosen among the w_i's to minimize the sum of distances to the other write locations, instead of minimizing over all vertices. (We show that choosing m this way increases the competitive ratio by at most a constant factor.)

Theorem 5.1 *Algorithm* FA *is* $O(\min\{\log n, \log(Diam)\})$*-competitive for file allocation on arbitrary network topologies.*

5.1 Competitive Analysis of FA.

This section describes the proof of Theorem 5.1 for the competitiveness of the file allocation algorithm FA.

We prove theorem 5.1 using a potential function argument. We analyze the change in the potential function Φ, in phases. For each operation within a phase we show that during the phase the potential function increases by at most $O(\log n)$ times the adversary cost and decreases by at least the on-line cost, which concludes the proof of the theorem.

The analysis is separated into five main parts:

- We begin by proving Φ is nonnegative in Lemma 5.2.
- Next we prove that the total sum of changes in Φ over all configuration changes made by the adversary during the phase is $O(\log n)$ times the sum of its configuration change costs. This is done in Lemma 5.3.
- Lemma 5.4 deals with the change in the potential function after both the adversary and the on-line algorithm serve a *read* request. We show that Φ increases by at most $O(1)$ times the adversary cost for the *read* and decreases by at least the on-line cost for that *read.*
- Lemma 5.5 deals with the change in the potential function due to a *replication* made by algorithm FA. We prove that Φ increases by at most $O(1)$ times the adversary cost for the D reads that caused that replication and decreases by at least the cost of the replication to the on-line algorithm.

- Finally, in Lemma 5.6 we analyze the change in the potential function at the end of a phase proving that it increases by at most $O(\log n)$ times the cost the adversary incurred during the phase for serving the D *write* requests invoked during the phase and decreases by at least the cost FA incurred for those *write* requests and for performing the reorganization step taken at the end of the phase.

Throughout the proof we use A to denote the adversary configuration, and B to denote the on-line configuration. $\overline{B}$ is the subtree spanning B, implied by the on-line replications. This is defined as follows: When FA replicates from $p \in B$ onto a vertex $q \notin B$, then the shortest path between p and q is added to $\overline{B}$.

In the proof we make use of an "intermediate adversary" defined below.

At the beginning of a phase the intermediate adversary holds file copies in all vertices of a minimum Steiner tree spanning the copies held by the original adversary and the single copy held by the on-line algorithm.

During a phase the intermediate adversary replicates to the same locations that the original adversary replicates to, if the intermediate adversary has no file copy in this vertex. (There may be vertices for which the intermediate adversary holds copies but the adversary does not). If the true adversary transmits the file through a specific set of vertices to the target, we consider it as thought there were replications to every vertex along this path. The intermediate adversary never discards any file copies.

Thus, the read requests occurring during the phase cost the intermediate adversary at most as much as their cost for the original adversary.

Associated with the intermediate adversary is a tree, denoted $\overline{A}$, that includes all vertices in the intermediate adversary configuration, and the edges include the original edges at the beginning of a phase and edges to new vertices, these are the edges that were used by the true adversary to replicate the file.

The analysis described above applies to complete phases. To deal with the cost of the last phase one can add to the request sequence dummy write requests at some node in the adversary's configuration until the phase ends (there would be at most D such requests). The adversary may delete all file copies in other nodes before this sequence of requests begins thus avoiding an increase in it cost. On the other hand the cost of the on-line algorithm is at least the actual cost. Now that we have that the last phase is complete we can apply the analysis.

5.2 The Potential Function for FA.

The potential function for FA, Φ, is the sum of 3 components: Ψ, $-\Lambda$, and Θ, each of which is responsible for different parts of the analysis.

Basically, Ψ is the potential for the cost of on-line replications, Λ is the potential for the cost of read requests, and Θ is the potential for the cost of write requests.

Following the ideas of the proof of Theorem 4.1 for greedy Steiner tree, we use a *hierarchical tree cover* (Section 3.1) of the intermediate adversary subtree $\overline{A}$.

At the end of each phase the hierarchical tree cover is reconstructed according to the new intermediate adversary configuration. *I.e.*, we start with the processor holding the migrated single on-line file copy, and add all other processors in the intermediate adversary configuration one by one, later we maintain the cover dynamically to reflect the changes in $\overline{A}$.

For every level $0 \leq i \leq \log(Diam)$ we have a tree cover of $s(i)$ covering subtrees of radius at most $2(2^i - 1)$.

Let $p_1^i, p_2^i, \ldots, p_{s(i)}^i$ be the covering nodes in the i-level tree cover.

Given the tree cover we define a partition of the network into disjoint sets: $S_1^i, S_2^i, \ldots, S_{s(i)}^i$, such that every node v is contained in a single set S_j^i such that p_j^i is closest to v among all i-level covering processors (breaking ties arbitrarily).

Define the constants $\lambda_1^* = 2\frac{2\lambda+2}{2\lambda+1}$ and $\lambda^* = 3(2 + (2\lambda_1^*)^2(1 + \frac{1}{\lambda}))$.

We use the following notation in the potential function definition.

Notation. For some $k \geq 0$, and nodes p and q, let $d_k(p,q) = \min\{d(p,q), k\}$.

Let $x \ominus y = \max\{x - y, 0\}$.

The definition of Ψ is similar to the Steiner tree potential function definition.

Let $R(i) = 8(\lambda + 1)(2^i - 1)$. For every i-level covering processor, p_j^i, if the nearest on-line file copy is in its $R(i+1)$-neighborhood, we give a credit proportional to D times its distance to that copy, $D \cdot d(B, p_j^i)$. If the nearest on-line file copy is not in the $R(i+1)$-neighborhood of p_j^i we associate a credit of $D \cdot R(i+1)$ with it.

$$\Psi \quad = \quad \lambda^* \cdot D \sum_{i=0}^{\log(Diam)} \sum_{j=1}^{s(i)} d_{R(i+1)}(B, p_j^i).$$

We now define Λ. Let L be the list of read requests, issued during the current phase, used by the algorithm.

For every i-level covering processor, p^i_j, consider the read requests in L initiated at processors, $u \in S^i_j$. We associate with p^i_j a credit proportional to $[d_{R(i+1)}(B, p^i_j) \ominus \max\{(2(\lambda + 1)d(p^i_j, u), \frac{1}{2}R(i)\}]$.

$$\Lambda = \lambda^*_1 \sum_{i=0}^{\log(Diam)} \sum_{j=1}^{s(i)} \sum_{\text{read } u \in S^i_j \text{ in } L} \left[d_{R(i+1)}(B, p^i_j) \ominus \max\{2(\lambda+1)d(p^i_j, u), \tfrac{1}{2}R(i)\}\right].$$

Finally, let $\Theta = D \cdot T(\overline{B})$, the potential function is:

$$\Phi = \Psi - \Lambda + \Theta.$$

5.3 The Potential Function Analysis.

Lemma 5.2 *The potential function, Φ, is nonnegative.*

Proof. We prove that $\Lambda \leq \Psi$.

Consider a level $0 \leq i \leq \log(Diam)$, and a covering processor, $p = p^i_j$, $1 \leq j \leq s(i)$.

Let $u \in S^i_j$ be a location of read requests in L. Let

$$\Lambda(i,j,u) = d_{R(i+1)}(B, p^i_j) \ominus \max\{2(\lambda+1)d(p^i_j, u), \tfrac{1}{2}R(i)\}$$

We first show that there are at most D read requests in L, for which the above expression is positive.

Assume the opposite, then there are D requests in L, at processors $u_1, u_2, \ldots u_D$ (not necessarily distinct) according to their order of appearance, for which

$$\begin{aligned} 2(\lambda+1)d(p^i_j, u_t) &< d(B, p^i_j) \\ &\leq d(B, u_t) + d(p^i_j, u_t), \end{aligned}$$

for $1 \leq t \leq D$, and hence $(2\lambda+1)d(p^i_j, u_t) < d(B, u_t)$.

Consider the D'th such request at processor u_D. Let u_t, $1 \leq t < D$ be a location of one of the previous $D-1$ requests, then

$$\begin{aligned} d(B, u_D) + d(B, u_t) &> (2\lambda+1) \cdot (d(u_D, p^i_j) + d(p^i_j, u_t)) \\ &\geq (2\lambda+1) \cdot d(u_D, u_t). \end{aligned}$$

Therefore,

$$\begin{aligned} 2 \cdot d(B, u_D) &\geq d(B, u_D) + d(B, u_t) - d(u_D, u_t) \\ &> 2\lambda \cdot d(u_D, u_t). \end{aligned}$$

and hence, $d(B, u_D) > \lambda \cdot d(u_D, u_t)$.

It follows that

$$d(B, u_D) \;>\; \lambda \cdot \max_{1 \leq t \leq D} d(u_D, u_t).$$

and therefore algorithm FA would have replicated upon the request at u_D, and would have removed it from L, a contradiction.

Consider now one of the read requests for which $\Lambda(i, j, u)$ is positive, where u is the processor initiating the read, then $\Lambda(i, j, u) \leq d_{R(i+1)}(B, p_j^i)$.

Since there are at most D such components, and $\lambda^* \geq \lambda_1^*$, we get that $\Lambda \leq \Psi$. ■

Lemma 5.3 *The total change in Φ due to adversary replications and deletions during a phase is at most $O(\lambda \cdot \min\{\log n, \log(Diam)\})$ times the cost of these operations.*

Proof. Clearly if the adversary deletes copies, the potential function does not change since it's only a function of the intermediate adversary configuration $\overline{A}$, and the configuration of the on-line algorithm, B.

Thus the overall change in Φ due to adversary replications is composed of the total change in Ψ and Λ for these operations. Λ cannot decrease as a result of a replication made by the adversary.

Let $\Delta\Psi$ denote the total change in Ψ due to adversary replications. Let $\overline{A}_0$ denote the intermediate adversary tree at the beginning of the current phase.

For the i-level tree cover there can be at most $s(i) - 1$ covering processors, p_j^i, such that $d_{R(i+1)}(B, p_j^i) > 0$, since always the first covering processor coincides with the processor holding the single on-line copy at the beginning of the phase.

Using Theorem 3.1 we obtain

$$\begin{aligned} \Delta\Psi &\leq \lambda^* \cdot D \sum_{i=0}^{\log(Diam)} R(i+1) \cdot (s(i) - 1) \\ &\leq O(\lambda) \cdot D \sum_{i=0}^{\log(Diam)} 2^i \cdot (s(i) - 1) \\ &\leq O(\lambda \cdot \min\{\log n, \log(Diam)\}) \cdot D \cdot (T(\overline{A}) - T(\overline{A}_0)). \end{aligned}$$

Recall that the intermediate adversary defined in Section 5.1 does not discard file copies during a phase. At the beginning of a phase the intermediate adversary is holds file copies in all the vertices of a Steiner tree connecting the online copy and the adversary copies. Thus, the total cost of replication associated with the intermediate adversary during the phase is exactly $T(\overline{A})-T(\overline{A}_0)$. The cost of replications for the real adversary at least the cost of replications for the intermediate adversary. This completes the proof. ∎

Lemma 5.4 *For each read request, Φ increases by at most $O(\lambda)$ times the adversary cost for that request minus the cost* FA *incurs for the same request.*

Proof. Consider a read request initiated at processor r. The only potential component affected is Λ. The on-line cost for serving the request is $d(B,r)$.

Let i be largest such that $R(i) \leq d(B,r)$.

Let a be the processor nearest to r where the intermediate adversary keeps a copy of the file. There exists a covering processor q at the i-level tree cover at distance of at most $2(2^i-1)$ from a. Then

$$d(q,a) \;\leq\; 2(2^i-1) \;=\; \tfrac{1}{4(\lambda+1)}R(i) \;\leq\; \tfrac{1}{4(\lambda+1)}d(B,r).$$

Therefore,

$$d(q,r) \;\leq\; d(q,a)+d(a,r) \;\leq\; \tfrac{1}{4(\lambda+1)}d(B,r)+d(a,r).$$

It follows that if $p=p^i_j$ is the i-level covering processor such that $r \in S^i_j$, then

$$d(p,r) \;\leq\; d(q,r) \;\leq\; \tfrac{1}{4(\lambda+1)}d(B,r)+d(a,r).$$

Since $R(i) \leq d(B,r) \leq R(i+1)$ we have that the change in Λ obeys

$$\begin{aligned}
\Delta\Lambda \;&\geq\; \lambda_1^* \cdot \left[d(B,p) \ominus \max\{2(\lambda+1)\cdot d(p,r), \tfrac{1}{2}R(i)\}\right] \\
&\geq\; \lambda_1^* \cdot \left[d(B,r) - d(p,r) - \max\{2(\lambda+1)\cdot d(p,r), \tfrac{1}{2}R(i)\}\right] \\
&\geq\; \lambda_1^* \cdot \left[d(B,r) - ((\tfrac{1}{2}+\tfrac{1}{4(\lambda+1)})\cdot d(B,r) + (2\lambda+3)\cdot d(a,r))\right] \\
&\geq\; d(B,r) - \lambda_1^*(2\lambda+3)\cdot d(a,r).
\end{aligned}$$

Since the intermediate adversary never discards copies, the file copy held nearest to r by the adversary is at least as far as that of the intermediate adversary.

We therefore conclude that

$$\Delta\Phi \;=\; -\Delta\Lambda \;\leq\; \lambda_1^*(2\lambda+3)\cdot d(A,r) - d(B,r).$$

$\blacksquare$

For every replication made by algorithm FA, associate the D read requests that were removed from L after the replication.

Lemma 5.5 *For each replication* FA *performs,* Φ *increases by at most* $O(1)$ *times the adversary cost for the* D *read requests associated with the replication minus the cost incurred by* FA *for the replication.*

Proof. Let r be the processor FA replicated to. Let B be the on-line configuration prior to the replication. Let $\mathrm{Cost}_{\mathrm{FA}}^{\mathrm{R}}$ denote the on-line cost for the replication. We have

$$\mathrm{Cost}_{\mathrm{FA}}^{\mathrm{R}} \;=\; D \cdot d(B,r).$$

As a result of the replication Θ increases by exactly the replication cost: $\Delta\Theta = D \cdot d(B,r)$.

Let the locations of the D requests associated with the replication to r be $r_1, r_2, \ldots r_D = r$, then for every $1 \leq j \leq D$:

$$d(r_j, r) \;\leq\; \frac{1}{\lambda} d(B,r).$$

For any i-level covering processor p, the replication to r cannot increase $d_{R(i+1)}(B,p)$, and therefore cannot increase $\Psi - \Lambda$.

We would now like to bound the decrease that may be caused to Λ as a result of the removal of these D requests from L. Let r_t, $1 \leq t \leq D$, be one of these requests. Then for every i-level covering processor, $p = p_j^i$ such that r_t is included in S_j^i, the potential may decrease by at most

$$\lambda_1^* \cdot \left[d_{R(i+1)}(B,p) \ominus \max\{(2(\lambda+1)d(p,r_t), \tfrac{1}{2}R(i)\} \right] \;\leq\; \lambda_1^* \cdot R(i+1).$$

This decrease is positive if $R(i) \leq 2d(B,p)$ and $2(\lambda+1) \cdot d(p,r_t) \leq d(B,p)$.

It follows that $(2\lambda+1) \cdot d(p,r_t) \leq d(B,p) - d(p,r_t) \leq d(B,r_t)$ and hence

$$R(i) \;\leq\; 2d(B,p) \;\leq\; 2(d(B,r_t) + d(p,r_t)) \;\leq\; 2(1+\tfrac{1}{2\lambda+1}) \cdot d(B,r_t) \;=\; \lambda_1^* \cdot d(B,r_t).$$

Therefore, if ℓ is the largest such that $R(\ell) \leq \lambda_1^* \cdot d(B,r_t)$, then the total decrease in Λ due to the removal of r_t from L may be at most

$$\lambda_1^* \cdot \sum_{i,\mathrm{s.t.} R(i) \leq \lambda_1^* \cdot d(B,r_t)} R(i+1) \;\leq\; 2\lambda_1^* \cdot R(\ell+1).$$

Therefore

$$R(\ell+1) \;\leq\; 2\lambda_1^* \cdot d(B, r_t) \;\leq\; 2\lambda_1^* \cdot (d(B,r) + d(r, r_t)) \;\leq\; 2\lambda_1^* \cdot (1 + \tfrac{1}{\lambda}) \cdot d(B, r).$$

Summing over all D read requests,

$$\Delta\Lambda \;\geq\; -(2\lambda_1^*)^2(1 + \tfrac{1}{\lambda})D \cdot d(B, r).$$

We therefore obtain that

$$\begin{aligned} \mathrm{Cost}^{\mathrm{R}}_{\mathrm{FA}} - \Delta\Lambda + \Delta\Theta \;&\leq\; (2 + (2\lambda_1^*)^2(1 + \tfrac{1}{\lambda}))D \cdot d(B, r) \\ &=\; \tfrac{\lambda^*}{3} D \cdot d(B, r). \end{aligned} \tag{1}$$

We will now show that the change in Ψ compensates for this quantity.

Let i be largest such that $R(i) \leq d(B, r)$.

For $1 \leq j \leq D$, let a_j be the processor nearest to r_j where the intermediate adversary keeps a copy of the file.

Since the intermediate adversary never discards copies the file copy held nearest to r_j by the adversary, at the time of the request, was at least as far from r_j as a_j. Let $\mathrm{Cost}^{\mathrm{R}}_{\mathrm{Adv}}$ denote the adversary cost for serving the D requests associated with the replication. We have

$$\mathrm{Cost}^{\mathrm{R}}_{\mathrm{Adv}} \;\geq\; \sum_{j=1}^{D} d(a_j, r_j).$$

For every j, there exists a covering processor in the i-level tree cover at distance of at most $2(2^i - 1)$ from a_j. Let q_j denote that covering processor. Then

$$d(q_j, a_j) \;\leq\; 2(2^i - 1) \;=\; \tfrac{1}{4(\lambda+1)} R(i) \;\leq\; \tfrac{1}{4(\lambda+1)} d(B, r).$$

Therefore,

$$\begin{aligned} d(q_j, r) \;&\leq\; d(q_j, a_j) + d(a_j, r_j) + d(r_j, r) \\ &\leq\; \tfrac{1}{4(\lambda+1)} d(B, r) + d(a_j, r_j) + \tfrac{1}{\lambda} d(B, r) \\ &<\; \tfrac{1}{3} d(B, r) + d(a_j, r_j), \end{aligned}$$

for $\lambda \geq 4$.

Let p be the i-level covering processor minimizing $d(q_j, r)$ then

$$\begin{aligned} D \cdot d(p,r) &\leq \sum_{j=1}^{D} d(q_j, r) \\ &\leq \tfrac{1}{3} D \cdot d(B,r) + \sum_{j=1}^{D} d(a_j, r_j) \\ &\leq \tfrac{1}{3} D \cdot d(B,r) + \mathrm{Cost}^{\mathrm{R}}_{\mathrm{Adv}}. \end{aligned}$$

After the replication to processor r, the on-line copy nearest to p is at distance of at most $d(p,r)$.

Since $d(B,r) \leq R(i+1)$ we have that the change in Ψ obeys

$$\begin{aligned} \Delta\Psi &\leq \lambda^* \cdot D \cdot (d(p,r) - d_{R(i+1)}(B,p)) \\ &\leq \lambda^* \cdot D \cdot (2d(p,r) - d(B,r)) \\ &\leq \lambda^* \cdot (2\mathrm{Cost}^{\mathrm{R}}_{\mathrm{Adv}} + \tfrac{2}{3} D \cdot d(B,r) - D \cdot d(B,r)) \\ &= 2\lambda^* \cdot \mathrm{Cost}^{\mathrm{R}}_{\mathrm{Adv}} - \tfrac{\lambda^*}{3} D \cdot d(B,r). \end{aligned}$$

Using inequality 1 we obtain

$$\Delta\Phi = \Delta\Psi - \Delta\Lambda + \Delta\Theta \leq 2\lambda^* \cdot \mathrm{Cost}^{\mathrm{R}}_{\mathrm{Adv}} - \mathrm{Cost}^{\mathrm{R}}_{\mathrm{FA}}.$$

■

We are left with dealing with the on-line cost for serving write requests, and with analyzing the potential change at the end of a phase.

Three events take place associated with the *reorganization step* at the end of a phase:

- *End-of-phase replication*: A file copy is replicated onto processor m.
- *End-of-phase deletion*: All other file copies are deleted.
- *Intermediate adversary initialization*: The intermediate adversary's configuration is initialized.

The adversary cost associated with a phase is its cost for all requests issued in the phase and for all configuration changes made by the adversary during the phase.

Lemma 5.6 *The change in Φ due to the events associated with the* reorganization step *at the end of a phase is at most $O(\lambda \cdot \min\{\log n, \log(Diam)\})$ times the adversary cost associated with* *the phase minus the on-line cost spent for serving the write requests during the phase and for the reorganization step.*

Proof. We analyze the change in Φ separately for the end-of-phase replication, and separately for the end-of-phase deletion and intermediate adversary initialization.

The *end-of-phase replication* involves a replication of a new file copy at some processor m.

To analyze the change in Φ due to the replication to m, we analyze the change in Φ after an imaginary simulation of an additional D read requests at processor m, that forces FA to replicate to m, (viewing all D requests simultaneously and ignoring other requests still in L), and therefore reaching the required configuration.

Thus, we have added additional read requests to the request sequence (the "extended" sequence). For an optimal adversary, adding these requests to the request sequence cannot reduce the cost. In general, adding events to an online sequence of events can reduce the on-line cost (it is not the case here). What is important is that the request sequence added does not change the final configuration of FA at the end of the phase. At the beginning of a phase, the potential function is only a function of the current configurations of adversary and on-line algorithm.

Thus, the change in the potential function for FA on the original sequence is the same as the change in the potential for FA on the extended sequence.

Let A,B, and $\overline{B}$ denote the appropriate configurations at the end of the phase, before the reorganization step takes place.

From lemma 5.5 we infer the following.

Claim 5.6.1 The change in the potential function following the end-of-phase replication is:

$$\Delta\Phi \;\; \leq \;\; O(1) \cdot D \cdot d(A, m) - D \cdot d(B, m).$$

After the end-of-phase deletion is made, FA leaves only one copy of the file at processor m. Hence, if $\overline{B}'$ denotes the tree implied by the on-line replications during the phase including the replication to m, then

$$\Delta\Theta \;\; = \;\; -D \cdot T(\overline{B}')$$

The intermediate adversary configuration is initialized to a minimal Steiner tree spanning the processors in the real adversary configuration and m.

From Lemma 5.3 we have that $\Psi - \Lambda$ is always bounded by $O(\lambda \cdot \min\{\log n, \log(Diam)\})$ times the weight of the intermediate adversary subtree. Therefore the initialization of its configuration increases the potential function by at most $O(\lambda \cdot \min\{\log n, \log(Diam)\}) \cdot D \cdot T(A \cup \{m\})$.

Thus we have that the total change in the potential function due to the reorganization step is:

$$\begin{aligned} \Delta\Phi &\leq O(\lambda \cdot \min\{\log n, \log(Diam)\}) \cdot D \cdot \{T(A \cup \{m\}) + d(A, m)\} \\ &\quad - D \cdot \{T(\overline{B}') + d(B, m)\} \end{aligned} \tag{2}$$

Let $\text{Cost}^{\text{W}}_{\text{FA}}$ denote the cost FA incurred for the D write requests of the phase, and $\text{Cost}^{\text{W}}_{\text{Adv}}$ denote the adversary cost for the same i requests plus its replications cost.

The following two claims conclude the lemma.

Claim 5.6.2 The cost the adversary incurs for the write requests of the phase and for its replications made during the phase, satisfies:

$$\text{Cost}^{\text{W}}_{\text{Adv}} \geq D \cdot \frac{1}{4}\{T(A \cup \{m\}) + d(A, m)\}.$$

Proof of Claim 5.6.2. Let $w_1, w_2, \ldots, w_D$ be the locations of the D write requests of the phase.

Let A_i be the adversary configuration after the request at w_i arrived, thus $A_D = A$.

By the triangle inequality we have:

$$\begin{aligned} \text{Cost}^{\text{W}}_{\text{Adv}} &\geq \sum_i T(A_{i-1} \cup \{w_i\}) + D \cdot \sum_i \text{dist}(A_{i-1}, A_i) \\ &\geq \sum_i T(A \cup \{w_i\}). \end{aligned}$$

The last sum is at least the sum of distances from one processor in A to the w_i's, and is therefore at least $\sum_i d(m, w_i)$.

We conclude that:

$$\begin{aligned} D \cdot \{T(A \cup \{m\}) + d(A, m)\} &\leq 2D \cdot T(A \cup \{m\}) \\ &\leq 2(\sum_i T(A \cup \{w_i\}) + \sum_i d(m, w_i)) \\ &\leq 4\sum_i T(A \cup \{w_i\}) \leq 4\text{Cost}^{\text{W}}_{\text{Adv}}. \end{aligned}$$

■

Claim 5.6.3 The cost for FA for the write requests of the phase satisfies:

$$\text{Cost}^{\text{W}}_{\text{FA}} \leq D \cdot T(\overline{B}') + \text{Cost}^{\text{W}}_{\text{Adv}}.$$

Proof of Claim 5.6.3. Let $w_1, w_2, \ldots, w_D$ be the locations of the D write requests of the phase.

Let $\overline{B}_i$ be FA's configuration after the request at w_i arrived, and $\overline{B}_D = \overline{B}$. Since the on-line subtree is monotonically growing during a phase, we have:

$$\begin{aligned} \text{Cost}_{\text{FA}}^{\text{W}} &= \sum_i T(\overline{B}_{i-1} \cup \{w_i\}) \\ &\leq \sum_i T(\overline{B} \cup \{w_i\}) \\ &\leq D \cdot T(\overline{B}') + \sum_i d(m, w_i) \\ &\leq D \cdot T(\overline{B}') + \text{Cost}_{\text{Adv}}^{\text{W}}. \end{aligned}$$

Since $\sum_i d(m, w_i) \leq \text{Cost}_{\text{Adv}}^{\text{W}}$, as proved in claim 5.6.2. ∎

Now, using claims 5.6.2 and 5.6.3 in equation 2 we get the lemma. ∎

Note that the proof of Lemma 5.6 holds even if the end-of-phase replication is made to a processor p such that $\sum_i d(p, w_i) \leq \gamma \cdot \sum_i d(m, w_i)$, increasing the competitive ratio by at most a constant factor.

The following lemma implies that choosing p amongst $w_1, w_2, \ldots, w_D$ has this property.

Lemma 5.7 *Let $w_1, w_2, \ldots, w_D$ be D arbitrary points in a metric space. Then,*

$$\min_i \sum_j d(w_i, w_j) \leq (2 - \frac{2}{D}) \min_m \sum_j d(m, w_j).$$

Proof. The proof of the lemma is established using the triangle inequality and symmetry:

$$\begin{aligned} D \cdot \min_i \sum_j d(w_i, w_j) &\leq \sum_i \sum_j d(w_i, w_j) \\ &\leq \sum_i \sum_{j \neq i} (d(w_i, m) + d(m, w_j)) \\ &\leq (2D - 2) \sum_i d(m, w_i). \end{aligned}$$

∎

6 Preliminaries for the Distributed Algorithm

6.1 Graph Decompositions

The hierarchical network decomposition [AP90] defines the notion of clusters, partitions, and a hierarchy of partitions.

All these definitions are a function of some parameter k. (Usually $k = \log n$.)

A *cluster* $\mathcal{C}$ is a set of vertices.

A *partition* is a set of (possibly overlapping) clusters.

Partitions are classified into one of $\log(Diam)$ levels, denoted respectively $\mathcal{P}_0, \mathcal{P}_1, \ldots, \mathcal{P}_{\log(Diam)}$.

The collection of these partitions is called the *hierarchical network decomposition*, denoted by $\mathcal{P}$.

A vertex may belong to several clusters and a cluster may belong to many partitions.

For every level i and every cluster $\mathcal{C} \in \mathcal{P}_i$ we arbitrarily choose a vertex $v \in \mathcal{C}$ called the i-level cluster leader and denote it by $L_i(\mathcal{C})$.

The diameter of a cluster $\mathcal{C}$, denoted by $diam(\mathcal{C})$, is defined to be the maximal distance between any two vertices in $\mathcal{C}$.

The stretch of the i'th level partition P_i is defined to be minimal upper bound S such that for every cluster $\mathcal{C} \in \mathcal{P}_i$, $diam(\mathcal{C}) \leq (2^i - 1) \cdot S$. Similarly, the stretch of a collection of partitions, $\mathcal{P}$, is defined to be the maximal stretch of any partition in $\mathcal{P}$.

The degree of an i'th level partition $\mathcal{P}_i$ is defined to be the maximal number of mutually overlapping clusters in $\mathcal{P}_i$ (i.e., clusters with nonempty intersection). Similarly, we define the degree of a collection of partitions, $\mathcal{P}$, to be the maximal degree of any partition in $\mathcal{P}$. We use the notations $stretch(\mathcal{P}_i)$, $deg(\mathcal{P}_i)$, $stretch(\mathcal{P})$ and $deg(\mathcal{P})$ in the obvious manner.

As a function of some parameter k, the level i partition, $\mathcal{P}_i$, $0 \leq i \leq \log(Diam)$, has the following properties:

- Every $2^i - 1$ neighborhood is properly contained in some cluster, $\mathcal{C} \in \mathcal{P}_i$.
- A vertex participates in at most $2 \cdot n^{1/k} k$ clusters $\mathcal{C} \in \mathcal{P}_i$, i.e., $deg(\mathcal{P}_i) \leq 2 \cdot n^{1/k} k$.
- The stretch of a cluster $\mathcal{C} \in \mathcal{P}_i$ is at most $2k - 1$, *i.e.*, $stretch(\mathcal{P}_i) \leq 2k - 1$.

It follows from the above properties that $\mathcal{P}_0$ consists of n disjoint clusters, each of which is a single vertex.

For the case $k = \log n$ we have $deg(\mathcal{P}) \leq 4 \log n$ and $stretch(\mathcal{P}) \leq 2 \log n$.

6.2 Distributed Data Tracking

6.2.1 Data Tracking.

The *data tracking* mechanism of [BFR92] is a generalization of the mobile user tracking mechanism of [AP91].

In a network over a set P of n processors, the data tracking problem allows to maintain a subset Q of processors holding copies of the file with the following operations on Q:

Insert(u,v), initiated at $u \in Q$, inserts v to the set Q.

Delete(v), initiated at v, removes v from the set Q.

Find(u), initiated at u, returns the address of a processor $v \in Q$.

Definition. A distributed on-line data tracking algorithm serves sequences of Insert, Delete and Find operations initiated at processors of the network. The cost of a distributed on-line data tracking algorithm for a sequence of operations is the total cost of messages it sends to conduct those operations.

Definition. The *approximation factor* for an on-line data tracking algorithm, α, is the maximum over all Find operations, of the ratio $d(u,v)/d(u,Q)$, where u is the node initiating the Find, and $v \in Q$ is the returned node.

Definition. The optimal cost of Insert(u,v) is the cost of transmitting the file from u to v ; i.e., $D \cdot d(u,v)$.

The optimal cost of Delete(v) is 0.

The optimal cost of Find(u) is the cost of sending a message from u to the closest processor in Q; i.e., $d(u,Q)$

[BFR92] present a distributed on-line data tracking algorithm, named TRACK, dealing with arbitrary sequences of Insert, Delete and Find operations, such that the following theorem holds.

Theorem 6.1 *For every n-processor network, for every sequence of operations σ,*

1. *TRACK's total cost for conducting Insert and Delete in σ is $O(\min\{\log^2 n, \log n \log(Diam)\}/\log^2 D)$ times the total optimal cost of those operations.*

2. *TRACK's cost on each Find in σ is $O(\log^2 n/\log^2 D)$ times the optimal cost of that Find.*

3. *TRACK's approximation factor is $O(\log n/\log D)$.*

(Where the value of D is truncated to $[2, n]$).

The data tracking competitive ratio over request sequences of of updates and searches is denoted C_{TRACK}. We have that $C_{\mathsf{TRACK}} = O(\log^2 n/\log^2 D)$.

The data tracking approximation factor is denoted by α.

The memory needed for the algorithm is at most $O(\log^2(Diam))$ per processor.

If no memory considerations are made then the approximation factor can be reduced to $\alpha = O(1)$.

6.2.2 Scanning Mechanism.

Another simple tool is a distributed data structure that enables scanning through a subset of processors.

The scanning mechanism is usually used together with the data tracking mechanism, as to enable performing tracking operations over all processors, such as a Delete-All operation.

In a network over a set P of n processors, let Q be a subset of processors in P. Initially Q includes a single processor v_0.

Given a sequence of insertions for processors $v_1, v_2, \ldots, v_t$, we maintain a distributed data structure.

The processors $v_1, v_2, \ldots, v_t$ are connected in a tree structure $\mathcal{T}$, by using an adjacency list at each processor in $\mathcal{T}$.

When an insertion for v_i arrives: i.e., Insert(v_j,v_i) , $j < i$ is made, we add v_i to the tree $\mathcal{T}$, by leaving a pointer in v_j to v_i, we also leave a back-tracking pointer in v_i to v_j.

This procedure enables scanning through all processors in the tree structure starting at every one of the processors in $\mathcal{T}$, at a cost equal to the weight of $\mathcal{T}$.

The above description requires a considerable amount $(\Theta(n))$ of memory per processor, for keeping the list of its children. In the next section we describe how this can be reduced to $O(\log(Diam))$.

6.2.3 List Manipulation.

We now turn to the question of memory needs of our distributed algorithms. In our algorithms processors, p, maintain lists of pointers $L(p)$ to other processors.

Assume that the address of a processor appears in the lists of at most x processors.

Keeping every list at the processor maintaining it requires $\Theta(n)$ pointers per processor.

To overcome this difficulty, for every processor we translate the list $L(p)$, into $\log(Diam)$ lists. List $L_i(p)$, $0 \leq i \leq \log(Diam)$, is the list of processors at distance between 2^i and $2^{i+1} - 1$ from p.

Now, instead of keeping the entire list in p, pointers are kept to only one of the processors in each of the $\log(Diam)$ lists, and within a list $L_i(p)$, each processor in the list keeps a pointer to the next. The distance between any two processors u and v in $L_i(p)$ is at most $d(u,p) + d(p,v) \leq 2(2^{i+1} - 1)$, and therefore every operation on the list $L_i(p)$, is proportional to 2^i.

It follows that every processor needs only $\log(Diam)$ pointers for maintaining the information for its own list, and since it may appear in the lists of at most x processors it may need x more pointers. Therefore the total memory requirements is reduced to $O(x + \log(Diam))$ pointers per processors.

The operations provided by this data structures, for the list $L(p)$, are:

- Insert(v) — insert the address of processor v into $L(p)$, this is done by updating the pointers in the required level. The cost associated is $O(d(p,v))$.
- Delete(v) — delete the address of processor v into $L(p)$, this is done by updating the pointers in the required level. The cost associated is $O(d(p,v))$.
- Find — Find a processor in $L(p)$, by using the pointer to a processor in the nonempty list $L_i(p)$ with minimal i. The cost associated is $O(d(p, L(p)))$.
- Scan — Scan the entire list, at a cost proportional to the sum of communication costs between p and every processor in $L(p)$.

7 The Distributed Algorithm

In this section we give a distributed deterministic polylog(n)-competitive file allocation algorithm for arbitrary networks.

Considering the deterministic file allocation algorithm FA presented in Section 5 we see that the algorithm maintains a list L of previous read requests issued at the different processors in the network. In the distributed setting, this information is not available, and decisions must be made only upon local information.

In what follows, we give a distributed file allocation algorithm (DFA), that using the graph-decompositions described in Section 6.1 gathers local information about the read requests in sequence and operate similarly to algorithm FA.

We adapt the proof of competitiveness given in Section 5 to prove that the increase in the competitive ratio obtained is within polylogarithmic factors.

Distributed Data Structures

Algorithm DFA uses several distributed data structures:

- The hierarchical graph decomposition (Section 6.1). The network is decomposed in $\log(Diam)$ levels partitions $\mathcal{P} = \{\mathcal{P}_0, \mathcal{P}_1, \ldots, \mathcal{P}_{\log(Diam)}\}$, each of which partitions the graph into clusters, such that every (2^i-1)-neighborhood is contained in an i-level cluster, and $stretch(\mathcal{P}) \leq 2\log n$ and $deg(\mathcal{P}) \leq 4\log n$.

  Each cluster $\mathcal{C}$ is assigned a cluster leader $L_i(\mathcal{C})$. We let every node know of all leaders of clusters in all levels that contain that node. There are at most $deg(\mathcal{P})\log(Diam)$ such clusters.

- Data tracking (Section 6.2.1) and scanning mechanism (Section 6.2.2), for the set Q of processors holding copies of the file. Let $C_{\mathsf{TRACK}} = O(\log^2 n/\log^2 D)$ be the data tracking competitive ratio, and let α be the approximation factor of the data tracking mechanism.

- The list manipulation data structure described in Section 6.2.3 for efficiently keeping lists at the processors.

We assume that initially the algorithm holds a single copy of the file in the network. Let $\rho = 9$.

The Distributed File Allocation Algorithm (DFA).
The algorithm partitions the request sequence into phases. Each phase, except perhaps for the last one, contains exactly D consecutive *write* requests.

Within each phase, the algorithm deals only with *read* requests.

Dealing with read requests. For every level, i, and every cluster $\mathcal{C} \in \mathcal{P}_i$, the algorithm holds a read requests counter in the cluster leader $L_i(\mathcal{C})$, denoted $c_i(\mathcal{C})$.

Every processor keeps a list of processors reading the file from it during the phase.

Initially all counters, $c_i(\mathcal{C})$, are set to zero.

Upon receiving a new *read* request initiated by processor r, we search for a copy of the file using the distributed data tracking mechanism's Find(r) operation.

Let s be the processor at which a copy of the file has been found. s updates the list of processors reading it to include r. Let $h = d(s,r)$ be the distance of that copy to r.

Let ℓ be the largest integer such that for $k = 2^\ell - 1$, $k \leq h/(\rho\alpha \cdot stretch(\mathcal{P}))$.

For every cluster $\mathcal{C}$ in the ℓ-level partition $\mathcal{P}_\ell$ such that $r \in \mathcal{C}$, inform cluster leader $L_\ell(\mathcal{C})$ to increase the reads counter $c_\ell(\mathcal{C})$ by 1.

If the counter of any of the clusters reached D, then initialize that counter to 0. Then, *replicate* a copy of the file to r from processor s, holding a copy of the file.

Update the distributed data structures, by initiating an Insert(s,r) data tracking operation, and performing the same insertion to the tree scanning mechanism, $\mathcal{T}$.

Dealing with write requests. Algorithm DFA deals with writes the same as algorithm FA does.

A list of the locations of the write requests issued during the phase (with repetitions), is kept at the processor initially holding a copy of the file, denoted z.

The algorithm maintains the tree scanning mechanism, $\mathcal{T}$, spanning the set of copies it holds by adding to it the edges along which it replicates.

When a *write* request arrives at w, DFA initiates a Find(w) operation to find a processor holding a copy of the file, then it traverses the tree $\mathcal{T}$, updating the data in all processors holding a copy of the file, adds the location w of the writing processor to the list at the processor, z, initially holding a copy of the file.

Once the write counter at z reaches the file size D, the phase ends, and we make a "*reorganization step*", as follows:

Let the D locations where the file has been written be $w_1, w_2, \ldots, w_D$. Find the "central" location, $m = w_i$, with the property that the sum of distances from m every one of the other $D-1$ write locations is minimized.

The central location, m, is computed by processor z, that knows about all the D write locations.

Then z initiates a replication onto m, by sending a message to m to perform a Find(m)

operation and then replicate a copy from the found location, s. Update the data tracking structure by initiating Insert(s,m).

Now, DFA traverses the tree $\mathcal{T}$ again, for every processor in $\mathcal{T}$, communicate with every processor in the list of processors reading from it to inform all cluster leaders defined in the read procedure above to initialize their reads counters. Then DFA deletes all the replicas , except for the last one at m.

For each deleted replica kept at processor v, update the distributed data structure by initiating a Delete(v) data tracking operation, and eventually initialize the tree scanning mechanism $\mathcal{T}$ to the single node m.

Recall that α is the data tracking approximation factor and that $\alpha = O(1)$ if no memory constraints are enforced.

Theorem 7.1 *The competitive ratio of algorithm* DFA *for file allocation on arbitrary network topologies is* $O(\min\{\log n, \log(Diam)\} \cdot stretch(\mathcal{P}) \cdot \max\{deg(\mathcal{P}), \alpha \cdot C_{\mathsf{TRACK}}\})$.

Corollary 7.2 *Algorithm* DFA *is* $O(\max\{\log^3 n, \alpha \cdot \log^4 n / \log^2 D\})$*-competitive for file allocation on arbitrary network topologies.*

Competitive Analysis of DFA**.**

The proof of Theorem 7.1 for the competitiveness of the distributed file allocation algorithm DFA, is a modification of the proof of Theorem 5.1 for algorithm FA, in Section 5.

We prove theorem 7.1 using a potential function argument. We analyze the change in the potential function Φ, in phases. For each operation within a phase we show that during the phase the potential function increases by at most polylog(n) times the adversary cost and decreases by at least the on-line cost.

Throughout the proof we use A to denote the adversary configuration, and B to denote the on-line configuration. $\overline{B}$ denotes the subtree spanning B, implied by the on-line replications, this is the same underlying tree as for the structure maintained by the distributed tree scanning mechanism, $\mathcal{T}$.

We define an intermediate adversary as in the proof of Theorem 5.1.

At the beginning of a phase the intermediate adversary holds a minimum Steiner tree spanning the copies held by the original adversary and the single copy held by the on-line algorithm.

During a phase the intermediate adversary replicates to the same locations that the original adversary replicates to, but never discards any file copies. Thus, the read requests occurring during the phase cost the intermediate adversary at most as much as their cost for the original adversary.

Associated with the intermediate adversary is a tree, denoted $\overline{A}$, that includes all vertices in the intermediate adversary configuration, and the edges include the original edges at the beginning of a phase and edges to new vertices, these are the edges that were used by the true adversary to replicate the file.

The Potential Function for DFA.

As for algorithm FA, the potential function for DFA, Φ, is composed of 3 main potential components, each responsible of different parts of the analysis.

Basically, Ψ is the potential for the cost of on-line replications, Λ is the potential for the cost of read requests, and Θ is the potential for the cost of write requests.

We use a *hierarchical tree cover* (Section 3.1) of the intermediate adversary subtree $\overline{A}$.

As for the centralized algorithm, the hierarchical tree cover is reconstructed at the end of a phase according to the new intermediate adversary configuration. *I.e.*, we start with the processor holding the migrated single on-line file copy, and add all other processors in the intermediate adversary configuration one by one, later we maintain the cover dynamically to reflect the changes in $\overline{A}$.

For every level $0 \leq i \leq \log(Diam)$ we have a tree cover of $s(i)$ covering subtrees of radius at most $2(2^i - 1)$.

Let $p_1^i, p_2^i, \ldots, p_{s(i)}^i$ be the covering nodes in the i-level tree cover.

Let $\beta = \rho \cdot stretch(\mathcal{P}) \cdot (\alpha \cdot C_{\mathsf{TRACK}} + deg(\mathcal{P}))$.

Let $\rho_1^* = \frac{\rho}{\rho-1}(\alpha \cdot C_{\mathsf{TRACK}} + deg(\mathcal{P}))$, and let $\rho^* = 2\rho_1^* + 6\alpha \cdot C_{\mathsf{TRACK}}$.

The definition of Ψ is similar to the one in Section 5.

Let $0 \leq \ell \leq \log(Diam)$. Let $R(\ell) = \rho \cdot stretch(\mathcal{P}) \cdot (2^\ell - 1)$. Let $i(\ell) = \max\{\ell - 2, 0\}$. For every ℓ, an $i(\ell)$-level covering processor, $p_j^{i(\ell)}$, if the nearest on-line file copy is in its $R(\ell+1)$-neighborhood, we give a credit proportional to D times its distance to that copy, $D \cdot d(B, p_j^{i(\ell)})$. If the nearest on-line file copy is not in the $R(\ell+1)$-neighborhood of $p_j^{i(\ell)}$ then we associate a credit of $D \cdot R(\ell+1)$ with it.

$$\Psi = \rho^* \cdot D \sum_{\ell=0}^{\log(Diam)} \sum_{j=1}^{s(i(\ell))} d_{R(\ell+1)}(B, p_j^{i(\ell)}).$$

For every level ℓ and cluster $\mathcal{C} \in \mathcal{P}_\ell$, let $e_\ell(\mathcal{C})$ be the number of read requests, issued during the current phase, that have been dealt with in $\mathcal{C}$; i.e., requests that have increased the counter $c_\ell(\mathcal{C})$ during the execution of the algorithm in the current phase.

For a node p, let $\mathcal{C}_\ell(p)$ denote an ℓ-level cluster containing p's $(2^\ell - 1)$-neighborhood.

For every ℓ, and $i(\ell)$-level covering processor, $p_j^{i(\ell)}$, we give a credit proportional to $e_\ell(\mathcal{C}_\ell(p_j^{i(\ell)})) \cdot d_{R(\ell+1)}(B, p_j^{i(\ell)})$.

$$\Lambda = \rho_1^* \sum_{\ell=0}^{\log(Diam)} \sum_{j=1}^{s(i(\ell))} e_\ell(\mathcal{C}_\ell(p_j^{i(\ell)})) \cdot d_{R(\ell+1)}(B, p_j^{i(\ell)}).$$

Finally, let $\Theta = C_{\mathsf{TRACK}} \cdot D \cdot T(\overline{B})$, the potential function is:

$$\Phi = \Psi - \Lambda + \Theta.$$

The Potential Function Analysis.

Lemma 7.3 *The potential function, Φ, is nonnegative.*

Proof. We prove that $\Lambda \leq \Psi$.

Consider some level ℓ and cluster $\mathcal{C} \in \mathcal{P}_\ell$.

It follows from the definition of the algorithm that whenever a read request, initiated at some processor, r, increases the counter $c_\ell(\mathcal{C})$ for the cluster, then $r \in \mathcal{C}$, and if $c_\ell(\mathcal{C})$ reaches a count of D then a file is replicated onto r, and hence onto $\mathcal{C}$. The counter is set to 0 and will not be increased again until the next phase arrives.

Therefore, we have

$$e_\ell(\mathcal{C}) \leq c_\ell(\mathcal{C}) \leq D.$$

Since $\rho^* \geq \rho_1^*$ we infer that $\Lambda \leq \Psi$. ∎

Lemma 7.4 *The total change in Φ due to adversary replications and deletions during a phase is at most $O(\beta \cdot \min\{\log n, \log(Diam)\})$ times the cost of these operations.*

Proof. The proof of this lemma is exactly the same as the proof of the the matching Lemma 5.3 for algorithm FA in Section 5. ■

Lemma 7.5 *For each read request, Φ increases by at most $O(\beta)$ times the adversary cost for that request minus the cost* DFA *incurs for the same request.*

Proof. Consider a read request initiated at processor r.

Let s be the processor at which a copy of the file has been found by DFA. Let $h = d(s, r)$ be the distance of that copy to r.

Let ℓ be largest such that $2^\ell - 1 \leq h/(\rho\alpha \cdot stretch(\mathcal{P}))$.

Since the file copy found is at most further then the closest to r by the data tracking approximation factor we have: $h \leq \alpha \cdot d(B, r)$, and hence,

$$\rho \cdot stretch(\mathcal{P})(2^\ell - 1) \quad \leq \quad d(B, r) \quad \leq \quad \rho\alpha \cdot stretch(\mathcal{P})(2^{\ell+1} - 1),$$

that is $R(\ell) \leq d(B, r) \leq \alpha \cdot R(\ell + 1)$.

The on-line cost for serving the read is composed of two components: One is the cost of the data tracking Find operation, which is bounded by $C_{\mathsf{TRACK}} \cdot d(B, r)$.

The second is the cost of communicating with all ℓ-level cluster leaders of clusters containing r. There are at most $deg(\mathcal{P})$ such clusters, $\mathcal{C}$, and communicating with each costs at most $diam(\mathcal{C}) \leq (2^\ell - 1) \cdot stretch(\mathcal{P})$.

Let $\mathrm{Cost}^{\mathrm{r}}_{\mathrm{DFA}}$ denote the total on-line cost for serving the read. We have

$$\begin{aligned} \mathrm{Cost}^{\mathrm{r}}_{\mathrm{DFA}} &\leq C_{\mathsf{TRACK}} \cdot d(B, r) + deg(\mathcal{P}) \cdot R(\ell) \\ &\leq (\alpha \cdot C_{\mathsf{TRACK}} + deg(\mathcal{P})) \cdot d_{R(\ell+1)}(B, r) \\ &\leq \beta \cdot (2^{\ell+1} - 1), \end{aligned} \tag{3}$$

using $R(\ell) \leq d(B, r) \leq \alpha \cdot R(\ell + 1)$.

If the intermediate adversary does not hold a copy of the file at distance of at most $(2^\ell - 1)/2$ from r then the real adversary does not hold such a copy either and therefore must expend a cost of $\mathrm{Cost}^{\mathrm{r}}_{\mathrm{Adv}} \geq \max\{(2^\ell - 1)/2, 1\}$ upon the request.

A read request cannot cause an increase in the potential function, and thus

$$\Delta\Phi \quad \leq \quad 0 \quad = \quad 2\beta \cdot (2 \cdot \tfrac{2^\ell - 1}{2} + 1) - \beta \cdot 2^{\ell+1} \quad \leq \quad 6\beta \cdot \mathrm{Cost}^{\mathrm{r}}_{\mathrm{Adv}} - \mathrm{Cost}^{\mathrm{r}}_{\mathrm{DFA}}.$$

Otherwise, the intermediate adversary holds a file copy at a processor a at distance less than $(2^\ell - 1)/2$ from r. Let $i(\ell) = \max\{\ell - 2, 0\}$, then there exists a covering processor p at the $i(\ell)$-level tree cover at distance of at most $\max\{2(2^{\ell-2} - 1), 0\} \leq (2^\ell - 1)/2$ from a. Therefore,

$$d(p,r) \;\leq\; d(p,a) + d(a,r) \;\leq\; (2^\ell - 1)/2 + (2^\ell - 1)/2 \;\leq\; (2^\ell - 1).$$

Consider now the ℓ-level cluster $\mathcal{C}_\ell(p)$ containing p's $(2^\ell - 1)$-neighborhood then since $r \in N_p(2^\ell - 1)$, then it is also contained in $\mathcal{C}_\ell(p)$ and therefore $e_\ell(\mathcal{C}_\ell(p))$ is increased by 1 upon the request at r.

It follows that the change in Λ, in response to the request at r, obeys

$$\begin{aligned}
\Delta\Lambda &\geq \rho_1^* \cdot d_{R(\ell+1)}(B,p) \\
&\geq \rho_1^* \cdot (d_{R(\ell+1)}(B,r) - d(p,r)) \\
&\geq \rho_1^* \cdot (d_{R(\ell+1)}(B,r) - \tfrac{1}{\rho} R(\ell)) \\
&\geq \rho_1^*(1 - \tfrac{1}{\rho}) \cdot d_{R(\ell+1)}(B,r) \\
&= (\alpha \cdot C_{\mathsf{TRACK}} + diam(\mathcal{P})) \cdot d_{R(\ell+1)}(B,r),
\end{aligned}$$

using $R(\ell) \leq d(B,r)$.

Since Λ is the only potential component affected by a read request, we obtain, using inequality 3,

$$\Delta\Phi \;\leq\; -\Delta\Lambda \;\leq\; -\mathrm{Cost}^{\mathrm{r}}_{\mathrm{DFA}}.$$

■

For every replication made by algorithm DFA, associate the D read requests that have increased the counter that have reached D at the cluster to which the replication is made.

Lemma 7.6 *For each replication* DFA *performs,* Φ *increases by at most* $O(\beta)$ *times the adversary cost for the* D *read requests associated with the replication minus the cost incurred by* DFA *for the replication.*

Proof. Let r be the processor DFA replicated to. Let B be the on-line configuration prior to the replication.

Let s be the processor at which a copy of the file has been found by DFA. Let $h = d(s,r)$ be the distance of that copy to r.

Let ℓ be the largest integer such that $2^\ell - 1 \leq h/(\rho\alpha \cdot stretch(\mathcal{P}))$.

Since s is further than the copy closest to r by a factor of at most α we have: $h \leq \alpha \cdot d(B,r)$, and hence,

$$\rho \cdot stretch(\mathcal{P})(2^{\ell}-1) \;\; \leq \;\; d(B,r) \;\; \leq \;\; \rho\alpha \cdot stretch(\mathcal{P})(2^{\ell+1}-1),$$

that is $R(\ell) \leq d(B,r) \leq \alpha \cdot R(\ell+1)$.

The on-line cost for the replication is the (amortized) cost of the data tracking update operations, which is bounded by $C_{\mathsf{TRACK}} \cdot D \cdot h$. Let this cost be denoted $\mathrm{Cost}^{\mathrm{R}}_{\mathrm{DFA}}$.

As a result of the replication Θ increases by exactly D times the length of the replication path: $\Delta\Theta = D \cdot h$.

Since $h \leq \alpha \cdot R(\ell+1)$ we obtain

$$\mathrm{Cost}^{\mathrm{R}}_{\mathrm{DFA}} + \Delta\Theta \;\; \leq \;\; 2\alpha \cdot C_{\mathsf{TRACK}} \cdot D \cdot d_{R(\ell+1)}(B,r). \tag{4}$$

Let $\mathcal{C}$ be the ℓ-level cluster for which the read counter reached D, when DFA decided to replicate to r.

Let the locations of the D requests associated with the replication to r be $r_1, r_2, \ldots r_D = r$. All requests were counted by the leader of cluster $\mathcal{C}$, and therefore were issued by processors in $\mathcal{C}$. Thus for every $1 \leq j \leq D$:

$$d(r_j,r) \;\; \leq \;\; diam(\mathcal{C}) \;\; \leq \;\; (2^{\ell}-1) \cdot stretch(\mathcal{P}) \;\; \leq \;\; \tfrac{1}{\rho} d_{R(\ell+1)}(B,r),$$

since $R(\ell) \leq d(B,r)$.

Consider the decrease that may be caused to Λ as a result of initializing the counter $c_\ell(\mathcal{C})$ to 0. Λ decreases only for processors, p, (if exist) in the $i(\ell)$-level cover, such that $\mathcal{C} = \mathcal{C}_\ell(p)$. Therefore

$$d(p,r) \;\; \leq \;\; diam(\mathcal{C}) \;\; \leq \;\; \tfrac{1}{\rho} d_{R(\ell+1)}(B,r).$$

$$d_{R(\ell+1)}(B,p) \;\; \geq \;\; d_{R(\ell+1)}(B,r) - d(p,r) \;\; \geq \;\; (\rho-1) \cdot d(p,r).$$

Let $\Psi(p) = \rho^* \cdot D \cdot d_{R(\ell+1)}(B,p)$ and $\Lambda(p) = \rho^*_1 \cdot e_\ell(\mathcal{C}_\ell(p)) \cdot d_{R(\ell+1)}(B,p)$.

Since $e_\ell(\mathcal{C}) \leq c_\ell(\mathcal{C}) \leq D$, and since a file copy is replicated onto r, we obtain that

$$\begin{aligned}
\Delta\Psi(p) - \Delta\Lambda(p) \;\; &\leq \;\; \rho^* \cdot D \cdot (d(p,r) - d_{R(\ell+1)}(B,p)) + \rho^*_1 \cdot D \cdot d_{R(\ell+1)}(B,p) \\
&\leq \;\; (\rho^*_1 - (1 - \tfrac{1}{\rho-1})\rho^*) \cdot d_{R(\ell+1)}(B,p) \\
&\leq \;\; 0,
\end{aligned} \tag{5}$$

since $\rho^* \geq 2\rho^*_1 \geq \rho^*_1/(1-\frac{1}{\rho-1})$.

For other i-level covering processors, p, clearly $d_{R(i+1)}(B,p)$ does not increase, and hence $\Delta\Psi(p) - \Delta\Lambda(p) \geq 0$.

We now turn to show that for some processor the change in $\Psi - \Lambda$ compensates for the increase in inequality 4.

For $1 \leq j \leq D$, let a_j be the processor nearest to r_j where the intermediate adversary keeps a copy of the file.

Since the intermediate adversary never discards copies, the file copy held nearest to r_j by the adversary at the time of the request, was at least as far from r_j as a_j.

Consider the case that there exists $1 \leq j \leq D$, such that

$$d(a_j, r_j) \;\leq\; \tfrac{1}{\rho} d_{R(\ell+1)}(B,r).$$

Then there exists a covering processor p at the $i(\ell)$-level tree cover at distance of at most $2^\ell - 1$ from a_j. Then

$$d(p, a_j) \;\leq\; 2^\ell - 1 \;\leq\; \tfrac{1}{\rho} R(\ell) \;\leq\; \tfrac{1}{\rho} d_{R(\ell+1)}(B,r).$$

Therefore,

$$\begin{aligned} d(p,r) &\leq d(p,a_j) + d(a_j,r_j) + d(r_j,r) \\ &\leq \tfrac{1}{\rho} d_{R(\ell+1)}(B,r) + \tfrac{1}{\rho} d_{R(\ell+1)}(B,r) + \tfrac{1}{\rho} d_{R(\ell+1)}(B,r) \\ &= \tfrac{1}{3} d_{R(\ell+1)}(B,r), \end{aligned}$$

for $\rho = 9$.

After the replication to processor r, the on-line copy nearest to p is at distance of at most $d(p,r)$. Therefore,

$$\begin{aligned} \Delta\Psi(p) - \Delta\Lambda(p) &\leq \rho^* \cdot D \cdot (d_{R(\ell+1)}(p,r) - d_{R(\ell+1)}(B,p)) + \rho_1^* \cdot D \cdot d_{R(\ell+1)}(B,p) \\ &\leq \rho^* \cdot D \cdot d_{R(\ell+1)}(p,r) + (\rho^* - \rho_1^*) \cdot D \cdot (d(p,r) - d_{R(\ell+1)}(B,r)) \\ &\leq (2\rho^* - \rho_1^*) \cdot \tfrac{1}{3} D \cdot d_{R(\ell+1)}(B,r) - (\rho^* - \rho_1^*) \cdot D \cdot d_{R(\ell+1)}(B,r)) \\ &= -\tfrac{\rho^* - 2\rho_1^*}{3} D \cdot d_{R(\ell+1)}(B,r). \end{aligned}$$

Since $\rho^* = 2\rho_1^* + 6\alpha \cdot C_{\mathsf{TRACK}}$, combining with inequalities 4 and 5 we conclude that

$$\Delta\Phi \;=\; \Delta\Psi - \Delta\Lambda + \Delta\Theta \;\leq\; -\mathrm{Cost}^{\mathrm{R}}_{\mathrm{DFA}}.$$

Consider now the case that for every $1 \leq j \leq D$,

$$d(a_j, r_j) \;\geq\; \tfrac{1}{\rho} d_{R(\ell+1)}(B,r).$$

Then we charge the adversary the cost of $d_{R(\ell+1)}(a_j, r_j)$. This cost is charged only once in each level, since after the replication, for every ℓ level cluster $\mathcal{C}'$ containing r_j, the distance between an on-line file copy and any processor in $\mathcal{C}'$ is at most

$$diam(\mathcal{C}') + d(r_j, r) \;\; \leq \;\; diam(\mathcal{C}') + diam(\mathcal{C}) \;\; \leq \;\; 2(2^{\ell} - 1) \cdot stretch(\mathcal{P}),$$

and hence less than $\rho\alpha \cdot (2^{\ell} - 1) \cdot stretch(\mathcal{P})$, so that a replication will not be initiated by that cluster's leader during this phase.

Since $d(B, r) \geq R(\ell)$, it follows that the total cost charged for the adversary associated this way with the request at r_j, is at most

$$\begin{aligned} \sum_{\ell, \text{s.t.} \frac{1}{\rho} R(\ell) \leq d(a_j, r_j)} R(\ell + 1) \;\; &\leq \;\; 2 \cdot R(m+1) \\ &\leq \;\; 4\rho \cdot d(a_j, r_j), \end{aligned}$$

where m is the largest such that $\frac{1}{\rho} R(m) \leq d(a_j, r_j)$.

We therefore define the adversary cost associated with level ℓ to be

$$\begin{aligned} \text{Cost}^{\ell}_{\text{Adv}} \;\; &= \;\; \sum_{j=1}^{D} d_{R(\ell+1)}(a_j, r_j) \\ &\geq \;\; \tfrac{1}{\rho} D \cdot d_{R(\ell+1)}(B, r). \end{aligned}$$

Therefore, from inequalities 4 and 5 we obtain

$$\begin{aligned} \Delta\Phi + \text{Cost}^{\text{R}}_{\text{DFA}} \;\; &\leq \;\; \Delta\Psi - \Delta\Lambda + \Delta\Theta + \text{Cost}^{\text{R}}_{\text{DFA}} \\ &\leq \;\; 2\alpha \cdot C_{\mathsf{TRACK}} \cdot D \cdot d_{R(\ell+1)}(B, r) \\ &\leq \;\; 2\rho\alpha \cdot C_{\mathsf{TRACK}} \cdot \text{Cost}^{\ell}_{\text{Adv}}. \end{aligned}$$

■

We are left with dealing with the on-line cost for serving write requests, and with analyzing the potential change at the end of a phase.

Three events take place associated with the *reorganization step* at the end of a phase:

- *End-of-phase replication*: A file copy is replicated onto processor m.
- *End-of-phase deletion*: All other file copies are deleted.
- *Intermediate adversary initialization*: The intermediate adversary's configuration is initialized.

In addition, yet another operation is performed:

- *End-of-phase initialization*: All cluster leaders' counters accessed during the phase are reset to 0.

The cost of this operation is at most the cost expended by the algorithm for serving the read requests and therefore can be ignored (at the expense of a factor of two in the competitive ratio).

The adversary cost associated with a phase is its cost for all requests issued in the phase and for all configuration changes made by the adversary during the phase.

Lemma 7.7 *The change in Φ due to the events associated with the* reorganization step *at the end of a phase is at most $O(\beta \cdot \min\{\log n, \log(Diam)\})$ times the adversary cost associated with the phase minus the on-line cost spent for serving the write requests during the phase and for the reorganization step.*

Proof. The proof of this lemma follows the steps of the proof of the the matching Lemma 5.6 for algorithm FA in Section 5, with modifications needed to reflect the distributed overhead for serving the requests.

We have the following claims:

Let $\mathrm{Cost}^{\mathrm{M}}_{\mathrm{DFA}}$ be the cost incurred by DFA for the replication onto m.

Let $\overline{B}'$ denote the tree implied by the on-line replications during the phase including the replication to m.

Claim 7.7.1 The total change in the potential function due to the reorganization step is:

$$\begin{aligned}\Delta\Phi \quad \leq \quad & O(\beta \cdot \min\{\log n, \log(Diam)\}) \cdot D \cdot \{T(A \cup \{m\}) + d(A, m)\} \\ & -\{C_{\mathsf{TRACK}} D \cdot T(\overline{B}') + \mathrm{Cost}^{\mathrm{M}}_{\mathrm{DFA}}\}.\end{aligned}$$

Let $\mathrm{Cost}^{\mathrm{W}}_{\mathrm{DFA}}$ denote the cost DFA incurred for serving the D write requests of the phase, and $\mathrm{Cost}^{\mathrm{W}}_{\mathrm{Adv}}$ denote the adversary cost for the same requests plus its replications cost.

Claim 7.7.2 The cost the adversary incurs for the write requests of the phase and for its replications made during the phase, satisfies:

$$\mathrm{Cost}^{\mathrm{W}}_{\mathrm{Adv}} \quad \geq \quad D \cdot \frac{1}{6}\{T(A \cup \{m\}) + d(A, m)\}.$$

Claim 7.7.3 The cost for DFA for the write requests of the phase satisfies:

$$\mathrm{Cost}^{\mathrm{W}}_{\mathrm{DFA}} \quad \leq \quad C_{\mathsf{TRACK}} \cdot (D \cdot T(\overline{B}') + \mathrm{Cost}^{\mathrm{W}}_{\mathrm{Adv}}).$$

The rest of proof takes after the proof of Lemma 5.6. ■

8 Open Problems

In [BFR92] a lower bound of $\Omega(\log n)$ was proved for the file allocation problem in the global-view setting. While this paper shows this ratio to be tight up to a constant factor, the upper bound achieved here in the distributed setting is $O(\log^4 n)$. Thus the first obvious open problem is determining the exact competitive ratio of the distributed file allocation problem. We remark that analysis presented here is aimed at obtaining asymptotic bounds and does not optimize the additional constant factors, and thus it may be interesting to design algorithms that have better competitive ratios or better practical behavior.

Let the competitive ratio of the on-line Steiner tree problem on a specific metric space be c_n. We conjecture that there exists a deterministic (global-control) $O(c_n)$-competitive algorithm for the file allocation problem.

In [BFR92], a similar theorem was proved using randomization algorithms against adaptive adversaries (and thus proving an $O(c_n^2)$ bound on the deterministic competitive ratio via [BBK+90]). An $O(c_n^2)$ bound can also be directly obtained from a result of [BCI96].

Finally, the problem of devising competitive algorithms for the constrained file allocation problem [BFR92] on arbitrary network topologies is still open.

9 Acknowledgements

We wish to thank Jeffery Westbrook for introducing us to this subject and for his great help in answering our questions.

We also wish to thank the two referees for the excellent job they did in reading the paper and their very useful remarks.

References

[AA92] N. Alon and Y. Azar. On-line Steiner trees in the Euclidean plane. Discrete and Computation Geometry, Vol 10, 1993, pp. 113–121. Also in *Proc. 8th ACM Symp. on Computational Geometry*, pp. 337–343, 1992.

[AK93] S. Albers and H. Koga. New On-line algorithms for the page replication problem. In *J. of Algorithms*, 27(1):75-96, April 1998. Also *Proc. of the 4th Scandinavian Workshop on*

Algorithmic Theory, vol. 824 of *Lecture Notes in Computer Science*, 25–36, Aarhus, Denmark, July 1994.

[AP90] Baruch Awerbuch and David Peleg. Routing with Polynomial Communication-Space Trade-Off, *SIAM J. on Discrete Mathematics* **5**, (1992), 151–162. Also appeared as Sparse partitions. newblock In *Proc. 31st IEEE Symp. on Foundations of Computer Science*, pp. 503–513, 1990.

[AP91] Baruch Awerbuch and David Peleg. On-line tracking of mobile users. *J. ACM*, **42**, (1995), 1021–1058.

[BBK+90] S. Ben-David, A. Borodin, R.M. Karp, G. Tardos, and A. Wigderson. On the power of randomization in on-line algorithms. In Algorithmica, vol. 11, pp. 2–14, 1994. Also in *Proc.of the 22nd Ann. ACM Symp. on Theory of Computing*, pp. 379–386, 1990.

[BCI96] Y. Bartal, M. Charikar and P. Indyk. On page migration and other relaxed task systems. In *Proc. of the 8th Ann. ACM-SIAM Symp. on Discrete Algorithms*, pp. 43–52. January 1997.

[BFR92] Yair Bartal, Amos Fiat, and Yuval Rabani. JCSS, Vol. 51, no. 3, pp. 341–58, 1995. Also in *Proc. 24th ACM Symp. on Theory of Computing*, pp. 39–50, 1992.

[BS89] D.L. Black and D.D. Sleator. Competitive algorithms for replication and migration problems. Technical Report CMU-CS-89-201, Carnegie-Mellon, 1989.

[CLRW93] M. Chrobak, L. Larmore, N. Reingold, and J. Westbrook. Optimal multiprocessor migration algorithms using work functions. *Lecture Notes in Computer Science*, vol. 762, pages 406-415, Hong Kong, 1993, Springer-Verlag.

[CV92] B. Chandra and S. Vishwanathan. Constructing Reliable communication networks of small weight on-line. Journal of Algorithms, vol. 18, no. 1, pp. 159–175, 1995.

[DF82] D. Dowdy and D. Foster. Comparative models of the file assignment problem. Computing Surveys, 14(2), June 1982.

[IW91] M. Imaze and B.M. Waxman. Dynamic Steiner tree problem. *SIAM Journal on Discrete Mathematics*, 4(3):369–384, August 1991.

[KMRS88] Karlin, Manasse, Rudolpoh, and Sleator. Competitive snoopy caching. In *Algorithmica*, 3(1):79–119, 1988. Also in *Proc. 27th IEEE Ann. Symp. on Foundations of Computer Science*, pages 244-254, October 1986.

[LRWY94] C. Lund, N. Reingold, J. Westbrook, and D. Yan. On-line distributed data management. In *Proc. of European Symp. on Algorithms*, pages 202-214, 1994.

[MMS88] M.S. Manasse, L.A. McGeoch, and D.D. Sleator. Competitive algorithms for server problems. Journal of Algorithms, vol. 11., pp. 208–230, 1990. Also in *Proc. 20th ACM Symp. on Theory of Computing*, pp. 322–333, May 1988.

[ST85] D.D. Sleator and R.E. Tarjan. Amortized efficiency of list update and paging rules. *Comm. of the ACM*, 28(2):202–208, 1985. Also in *Proc. 6 ACM Symp. on Theory of Computing*, pp. 488–492, 1984.

[Wes92] J. Westbrook. Randomized algorithms for multiprocessor page migration. In *Proc. DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 7:135–150, 1992.

[WY93] J. Westbrook. and D.K. Yan. Greedy Algorithms for on-Line Steiner tree and generalized Steiner problems. In *Proc. of the 3rd Workshop in Algorithms and Data Structures*, Also *Lecture Notes in Computer Science*, vol. 709, pp. 622–633, Montréal, Canada, 1993, Springer-Verlag.

A The Cover Problem

We complete our discussion by showing a solution to the cover problem (Section 3.1). The algorithm presented appeared in [BFR92]. We present here a more accurate analysis and provide a bound which depends only on n. We repeat the formulation of the cover problem.

The Cover Problem

Given a weighted graph G, and a dynamically changing set Q, we would like to construct an r-cover for Q. That is dynamically define a set of mutually exclusive subsets $C = \{C_1, C_2, \ldots, C_s\}$, and a choice of nodes $p_1, p_2, \ldots, p_s$, $p_i \in C_i$, so that for every i, $i = 1, 2, \ldots, s$, $Q \cap C_i \neq \emptyset$, and $Q \subset \cup_{i=1}^{s} C_i$, and for every C_i, $i = 1, 2, \ldots, s$, the distance between any node in C_i and p_i is at most r.

Initially the set contains a single node $Q = \{q_0\}$ and the cover contains one covering set $C_1 = \{q_0\}$, and a single covering node $p_1 = q_0$.

The on-line cover problem for some fixed parameter r is the problem of maintenance of an r-cover for a dynamic set Q, where insertions into Q and deletions from Q are allowed (but Q is never allowed to be empty), where s changes with Q.

Define the optimal cost of an insertion to be the minimum distance between a node of Q and the inserted node, and the optimal cost for a deletion is 0. The optimal cost for a sequence of inserts and deletes is the sum of optimal costs of the operations.

We now turn to describe the cover algorithm and the proof of Theorem 3.1.

The Basic Cover Algorithm — Unweighted Case

We describe an on-line algorithm for maintaining a $2(k-1)$-cover for any integer $k > 0$.

Assume at first, that all weights in the weighted graph G are 1, and that all insertions are to nodes adjacent in this graph to nodes already in Q. Therefore, the cost charged for an insertion is 1.

Each cover set is represented by a directed tree. The root is the corresponding covering node, and all edges point towards the root. A node contained in a cover set is marked as such. A node in Q is marked as such.

The cover algorithm works as follows. Every node p is associated with a counter c_p. Initially q_0's counter is 0. For all other nodes the value is undefined.

Let the current cover sets be $C = \{C_1, C_2, \ldots, C_s\}$, let $\mathcal{C} = \cup_{i=1}^{s} C_i$, and let the covering nodes be $p_1, p_2, \ldots, p_s$. The algorithm maintains the following invariants:

1. For every $p \in \mathcal{C}$, $0 \le c_p \le 2k-2$.
2. For every i, $1 \le i \le s$, for every $p \in C_i$, c_p is an upper bound on $d(p, p_i)$.
3. For every i, $1 \le i \le s$, all nodes in C_i form a directed tree rooted at p_i. This tree is a subtree of G.
4. For every i, $1 \le i \le s$, every path from p_i to a leaf in the tree representing C_i contains at least one node of Q.
5. ($k\times$ the number of cover sets created) + (the number of nodes p such that $c_p \ge k$) $\le$ (the optimal cost for the sequence).

These invariants ensure the correctness of the algorithm. Invariants 1,2 and 4 imply that a node in Q is within distance at most $2k-2$ from a covering node.

We now describe the cover algorithm, by defining how it handles insertions and deletions.

Let p be an inserted node. Let $q \in Q$ be the node adjacent to p. If p is already in $\mathcal{C}$, it is simply marked as being in Q. Otherwise, the following update procedure is performed. First, c_p is set to $c_q + 1$, and p is added to q's tree by an edge pointing from p to q.

Now, if $c_p = 2k-1$, a scan-back procedure is conducted, starting at p. The counter of each node scanned is decreased by k, and then the scan moves to its parent in the tree. The scan-back stops once a node b with $c_b < k$ is encountered (the root has a counter 0, so the process must stop). This preserves invariant 1: $0 \le c_p \le 2k-2$.

Let t be the node scanned just before b. If the new value of c_t is 0, then a new cover set is created with t as the covering node. This is done by detaching t and all successors of t from b's tree.

Let p be a deleted node. p is marked as not in Q. For all $q \in \mathcal{C} \setminus Q$ that have no successors in Q, remove q and all successors. If q is the root of its tree then the appropriate cover set is removed.

This procedure preserves invariant 4.

The Basic Cover Problem — Analysis

The procedure described above clearly preserves properties 1 and 4, as mentioned above.

We turn to prove the other properties of the cover algorithm.

We show that the cover algorithm preserves a somewhat stronger property than invariant 2:

2'. For every i, $1 \leq i \leq s$, for every $p \in C_i$, c_p is an upper bound on $d(p, p_i)$. If $c_p \leq k$ then the length of the path from p_i to p equals c_p.

The proof that the invariant holds is by induction on the insertion steps in the algorithm, using the fact that when a node is inserted it is first assigned its predecessor's counter plus one, and the fact that if a counter is ever decreased then it decreased from some value greater than k to some value smaller than k.

It is clearly true at the beginning where there is only one node in the tree with zero counter. Consider a new insertion of node p from q. For a node x, let c'_x denote the value of its counter after the insertion. If $c'_p \leq 2k-2$ then since $c'_p = c_q + 1$ the invariant follows from the induction hypothesis. Assume $q \in C_i$, since the distance to the covering node p_i increases by at most 1. we have $d(p_i, p) \leq d(p_i, q) + 1 \leq c_q + 1 = c'_p$.

If $c'_p = 2k - 1$ then let b and t be as in the procedure above. If $c'_t > 0$ then since when t was inserted from b, we had that t's counter was greater than b's by 1, and before the insertion of p, $c_b < k$ while $c_t > k$, it follows that c_b was once decreased by k, and therefore $c'_t = c_t - k = c_b + 1$.

If $c_t = 0$ then the invariant holds since before the insertion all counters of nodes in the path from t to p were greater than k, and the length of the paths from the covering node to these nodes only decreased as a result of the creation of a new cover set. The only counters decreased are those along the path from t to p. Obviously each counter is equal to the length of the path from the new covering node t to that node. It follows that the procedure preserves invariant 2'.

To prove invariant 5, define $\Phi =$ the number of nodes p, such that $c_p \geq k$. Obviously when a new node is inserted $\Delta\Phi \leq 1$. A node deletion may only reduce the value of Φ. If a new cover set is created, that is $c_t = 0$, it follows that the length of the path from t to p is $k-1$. The counter of every node along the path has decreased from a value of at least k to a value below k. Since there are k such nodes we get $\Delta\Phi \leq -k$. As $\Phi \geq 0$ the claim follows.

The Cover Algorithm — Weighted Case

Arbitrary distances can be translated into integral distances while changing costs by no more than a constant factor. To deal with arbitrary integer distances between nodes, imagine that along an edge between two nodes there are virtual nodes that divide the distance into segments of length 1.

If an insertion of node p is performed, let u_p be the length of the shortest path from the nearest node q in Q to p. The insertion is done by simulating insertions for all virtual nodes along the shortest path from the nearest node q in Q to p in that order. If one of the virtual nodes is a covering node then let p be a covering node. This may only decrease distances between a covering node and the actual nodes in it's cover set. Following this, delete all virtual nodes.

The Hierarchical Cover Algorithm

The hierarchical cover algorithm (Section 3.1) is defined by simultaneously running cover algorithms for $\log(Diam)$ levels. For level $0 \leq i \leq \log(Diam)$ we run a $2(k-1)$-cover algorithm for $k = 2^i$.

Let the number of covering sets maintained by the i-level cover algorithm be denoted $s(i)$. Let the total number of covering sets creations made by the i-level cover algorithm be denoted $c(i)$. Then $s(i) \leq c(i)$.

We recall Theorem 3.1 stated in Section 3.1.

Theorem 3.1 The following invariant is maintained by the hierarchical cover algorithm

$$\begin{aligned}\sum_{i=0}^{\log(Diam)} 2^i \cdot (s(i)-1) &\leq \sum_{i=0}^{\log(Diam)} 2^i \cdot (c(i)-1) \\ &\leq O(\min\{\log n, \log(Diam)\}) \cdot \mathrm{UPD}.\end{aligned}$$

The theorem follows by replacing property 5 of the cover algorithm by the following stronger claim.

Let the optimal cost for insertions of nodes v, such that $u_v \geq (2^i-1)/2n$ be denoted UPD_i.

5'. For every level i,

$$(c(i)-1)\cdot(2^i-1)+\sum_{p;u_p\geq\frac{(k-1)}{2n},c_p\geq k} u_p \;\leq\; 2\cdot\mathrm{UPD}_i.$$

Thus the optimal cost, u_v, of an insertion of node v is accounted for only in levels i such that $2^i-1\leq 2n\cdot u_v$, and therefore in at most $O(\min\{\log n,\log(Diam)\})$ levels.

Summing up over all levels we get $\sum_{i=1}^{\log(Diam)}(c(i)-1)\cdot 2^i$ is bounded above by $O(\min\{\log n,\log(Diam)\})$ times the optimal cost, giving the bound in Theorem 3.1.

To prove invariant 5', let $k=2^i$, and define

$$\Phi=\sum_{p;u_p\geq\frac{(k-1)}{2n},c_p\geq k} u_p.$$

Obviously when a new node p such that $u_p\leq(k-1)/2n$ is inserted $\Delta\Phi\leq u_p$. A node deletion may only reduce the value of Φ. Let t be as defined in the cover algorithm. If a new cover set is created, that is $c_t=0$, let P be the path from t to p. It follows that the length of the path P is $k-1$. Since $\sum_{p\in P;u_p<\frac{(k-1)}{2n}}u_p\leq(k-1)/2$, we have that $\sum_{p\in P;u_p\geq\frac{(k-1)}{2n}}u_p\geq(k-1)/2$. The counter of every node along the path has decreased from a value of at least k to a value below k. We therefore get that

$$\Delta\Phi\;\leq\;-\sum_{p\in P;u_p\geq\frac{(k-1)}{2n}}u_p\leq-(k-1)/2.$$

As $\Phi\geq 0$ the claim follows.

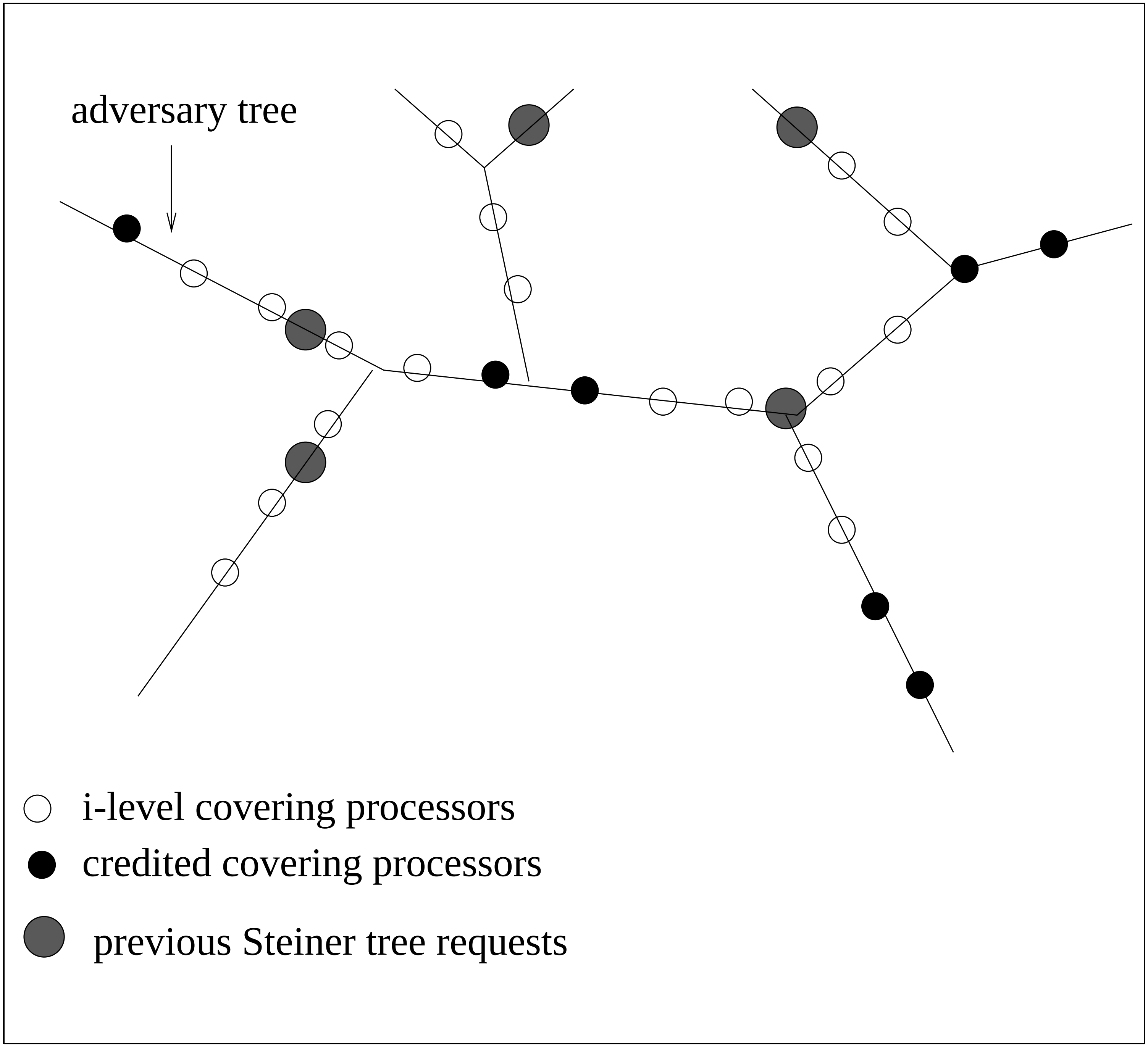

Figure 3: The greedy Steiner tree potential function.

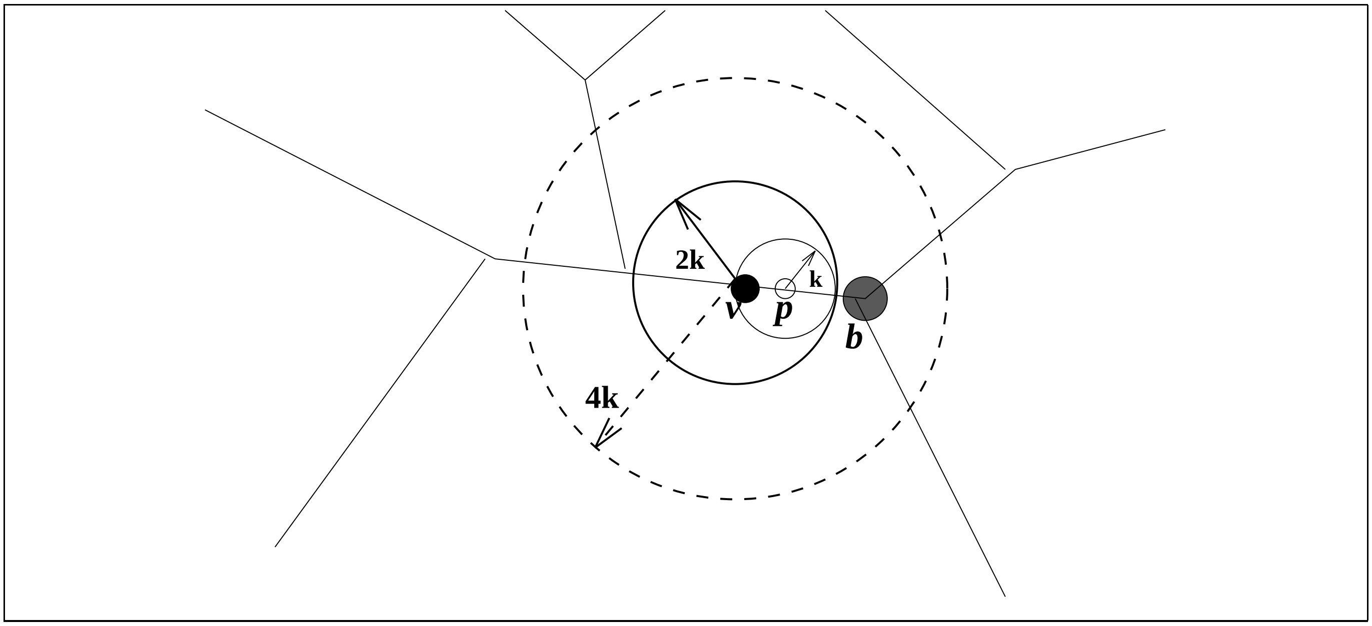

Figure 4: The newly requested node v enters the k-neighborhood of covering node p.

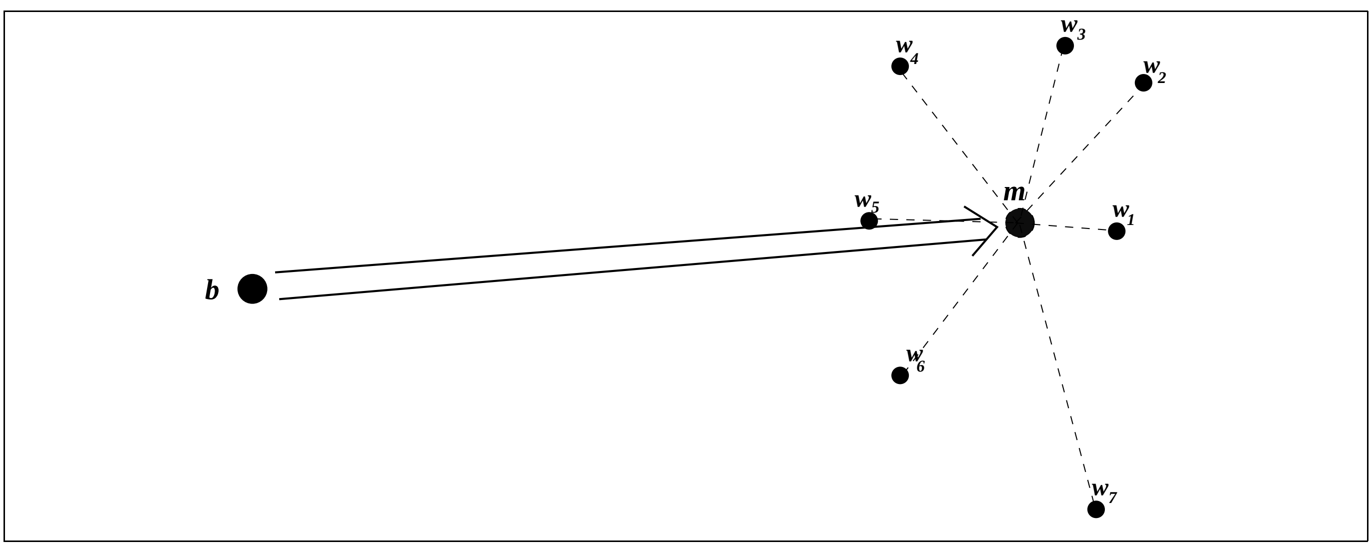

Figure 5: Algorithm Move-To-Min (MTM). In the example D=7. m minimizes the sum of distances to the D access locations.

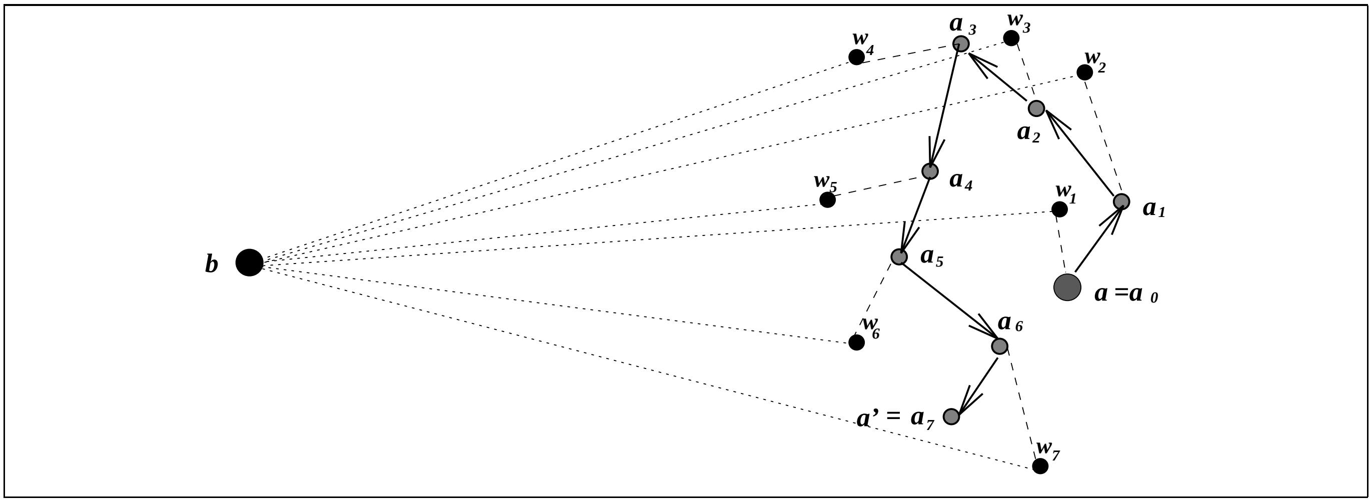

Figure 6: The costs incurred by the on-line algorithm and the adversary over a phase.

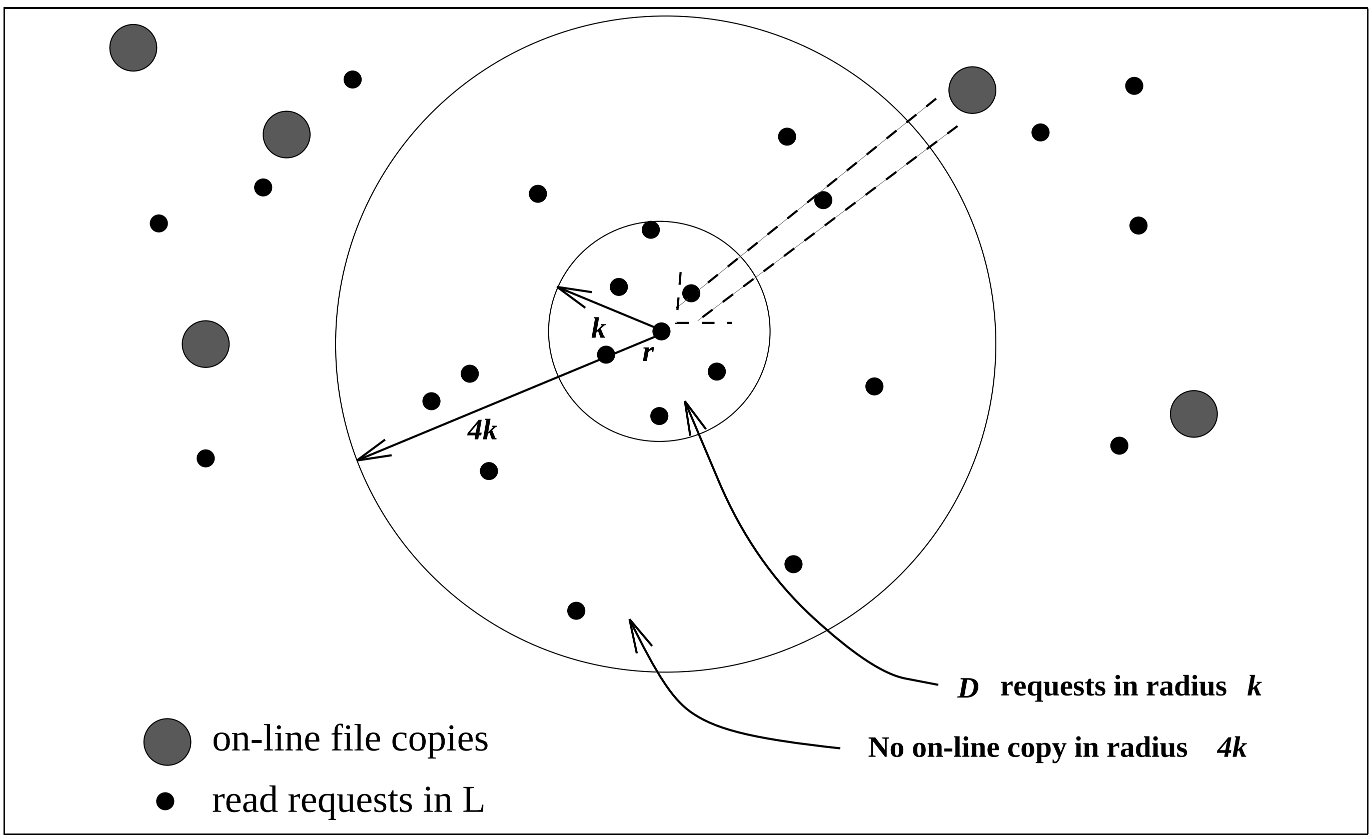

Figure 7: The file allocation algorithm (FA): dealing with read requests. r is the last read request. ($\lambda = 4$.)

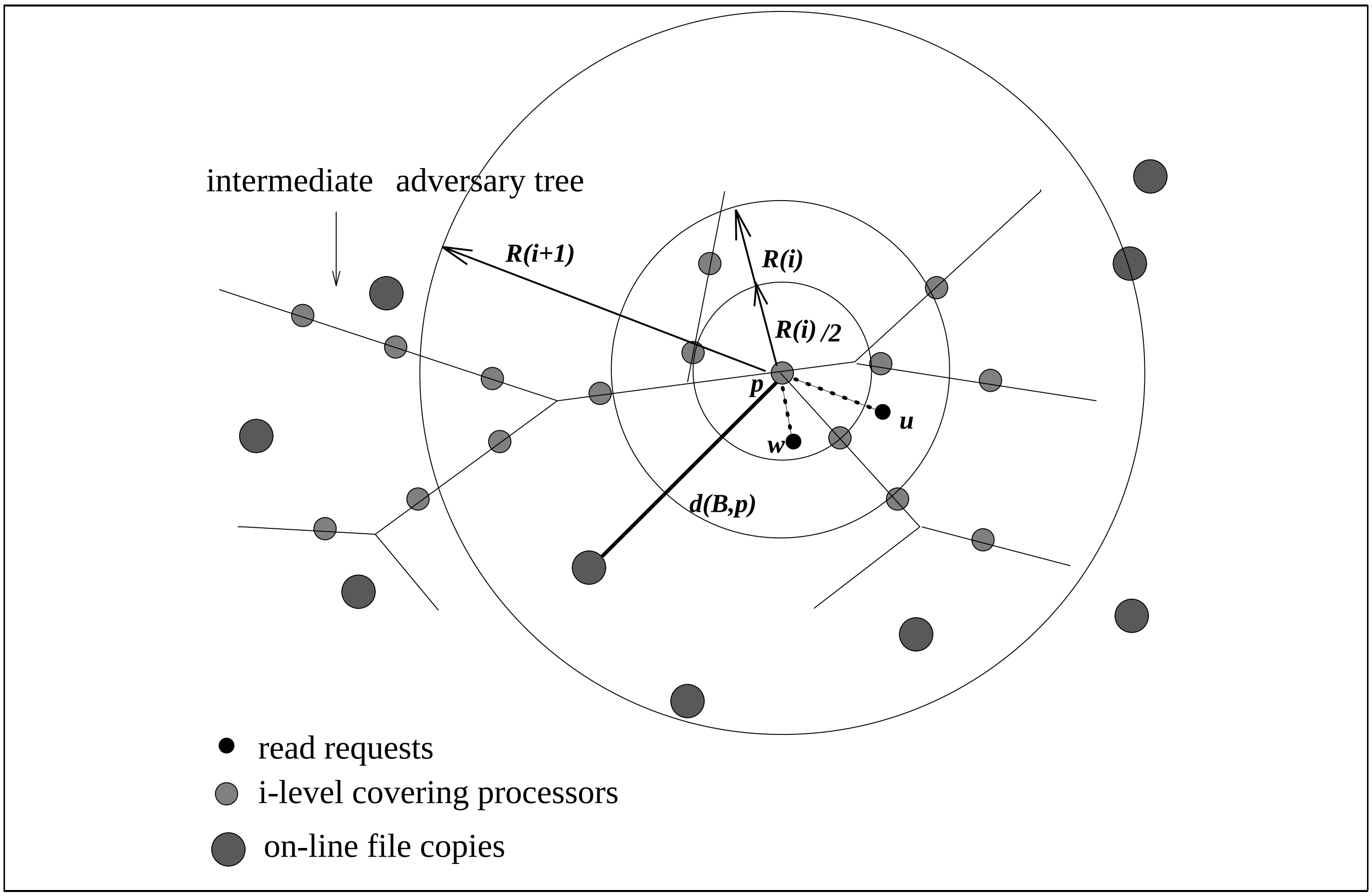

Figure 8: The potential function for FA. Here we view the contribution of one covering processor to Ψ $(D \cdot d(B,p))$, and the contribution of p and of a pair of requests u, w to Λ $(2d(B,p) - 2(\lambda+1)(d(p,u) + d(p,w)))$.

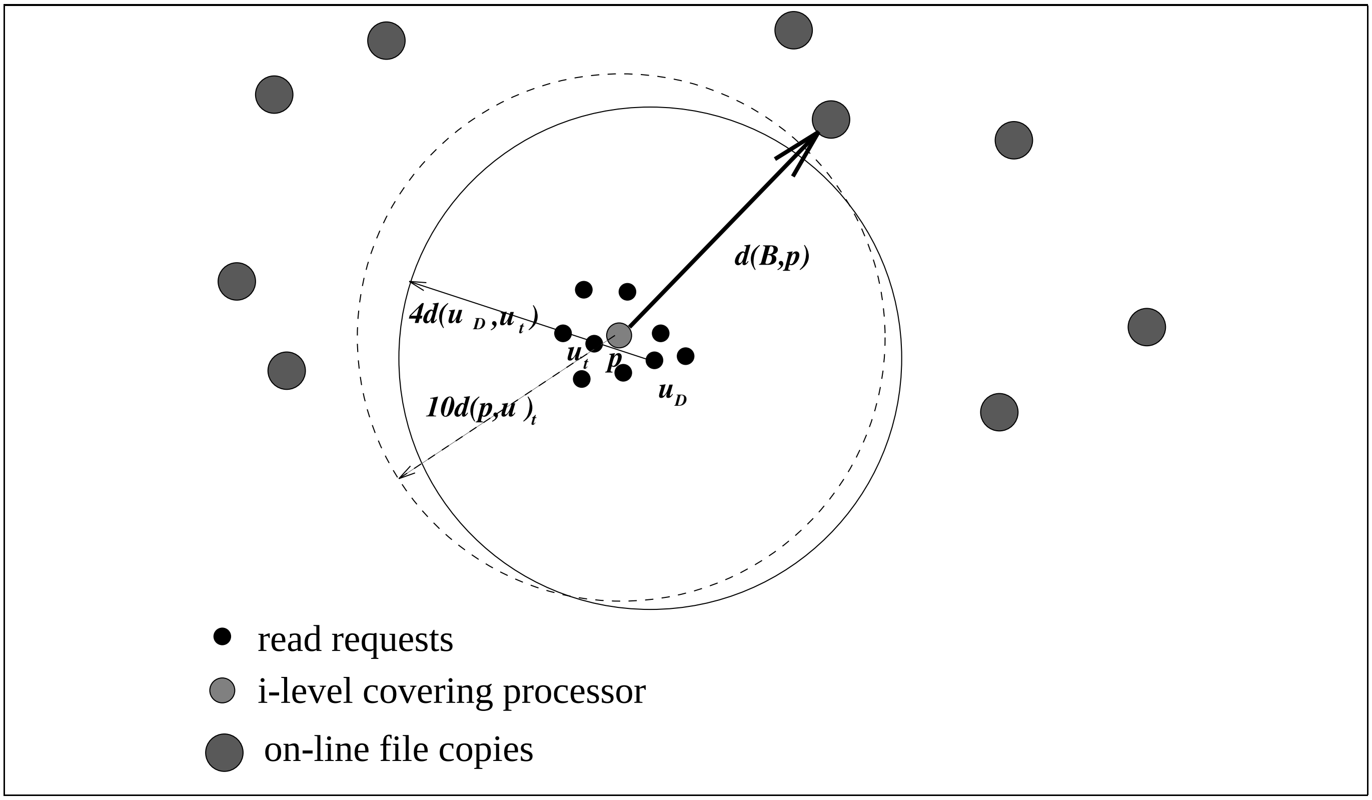

Figure 9: Demonstrating $2(\lambda+1)d(p,u_t) < d(B,p)$ and $\lambda \cdot d(u_t,u_D) < d(B,u_D)$. ($\lambda = 4$.)

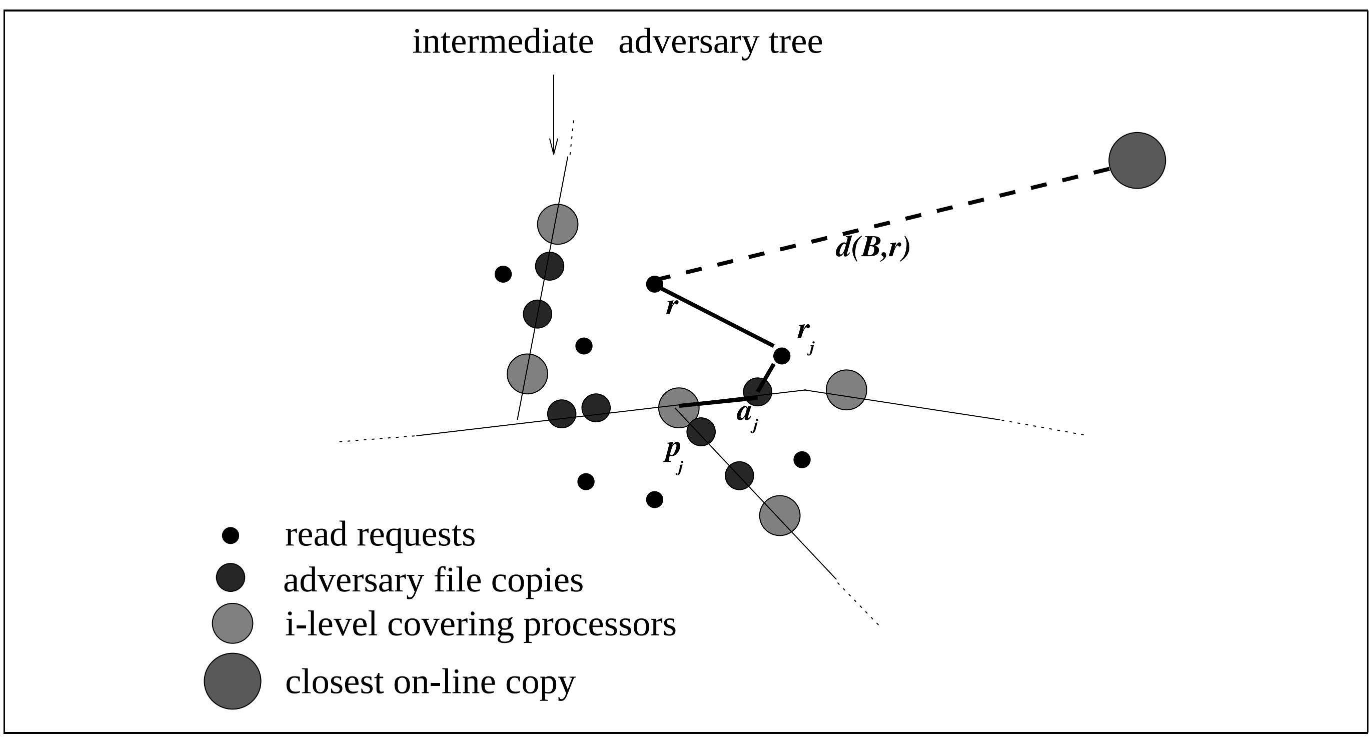

Figure 10: The adversary costs for the read requests associated with the phase, and the appropriate covering processors.

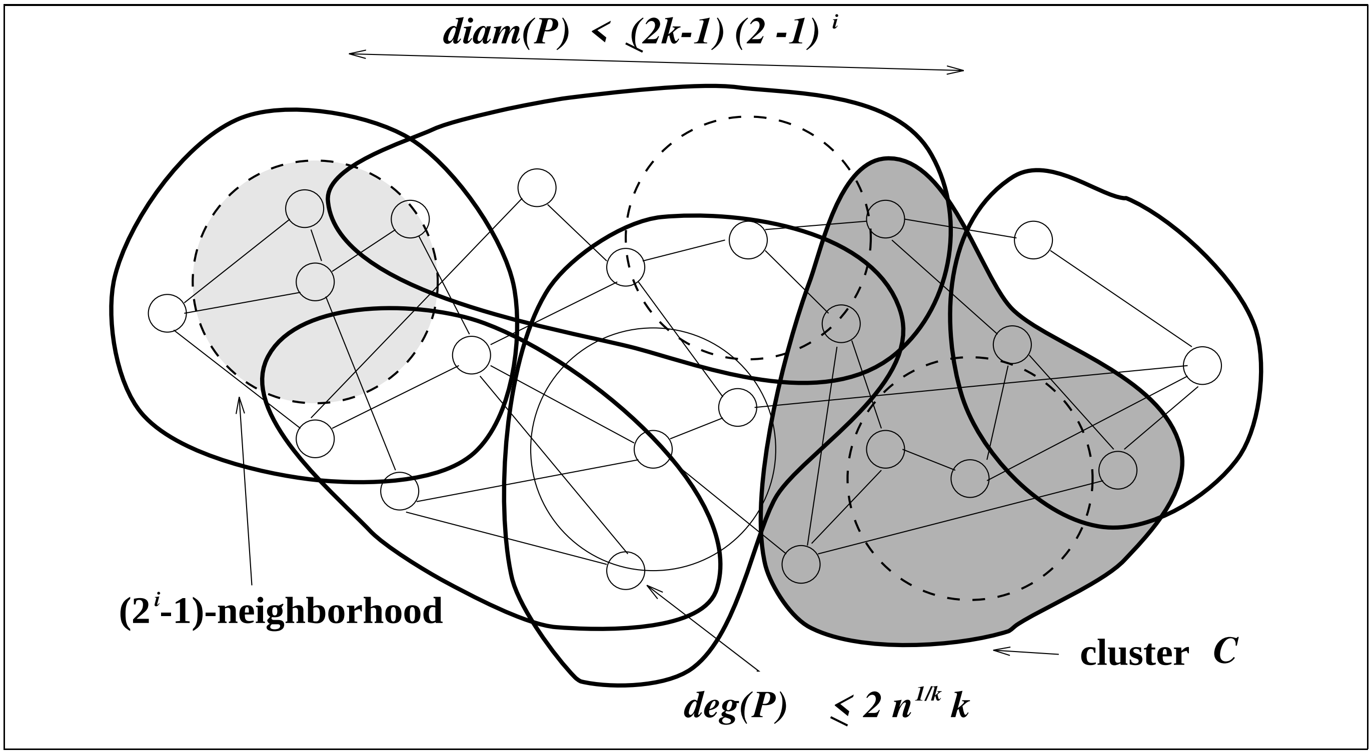

Figure 11: The i-level sparse graph partition and its properties.

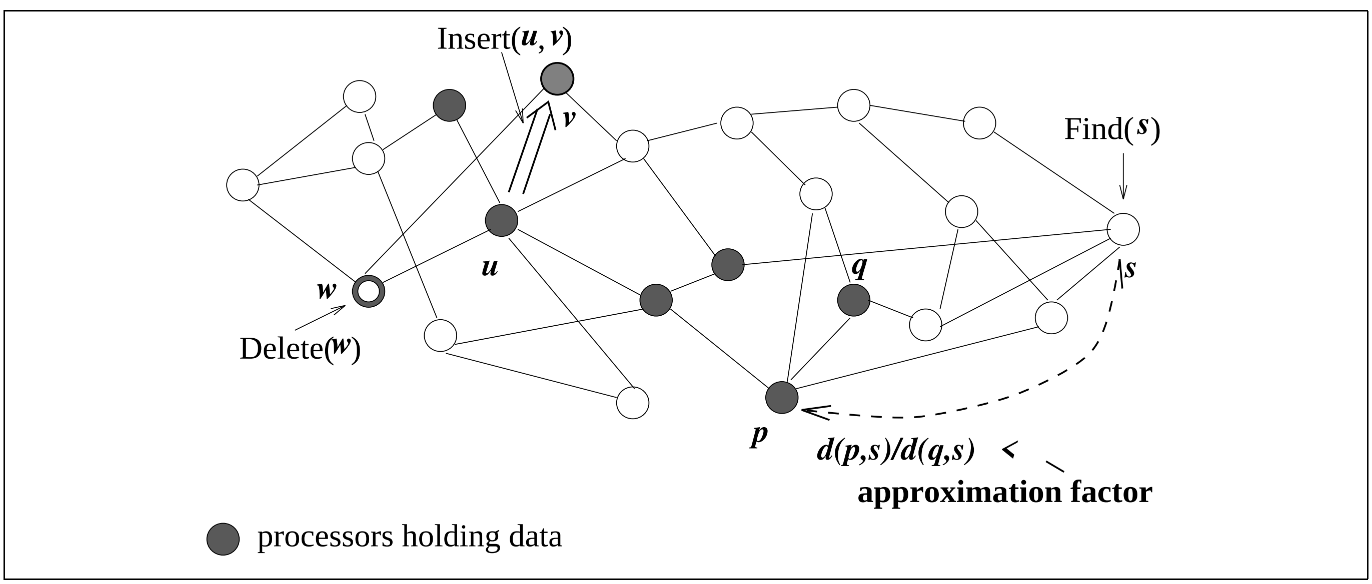

Figure 12: The operation of the data tracking mechanism.

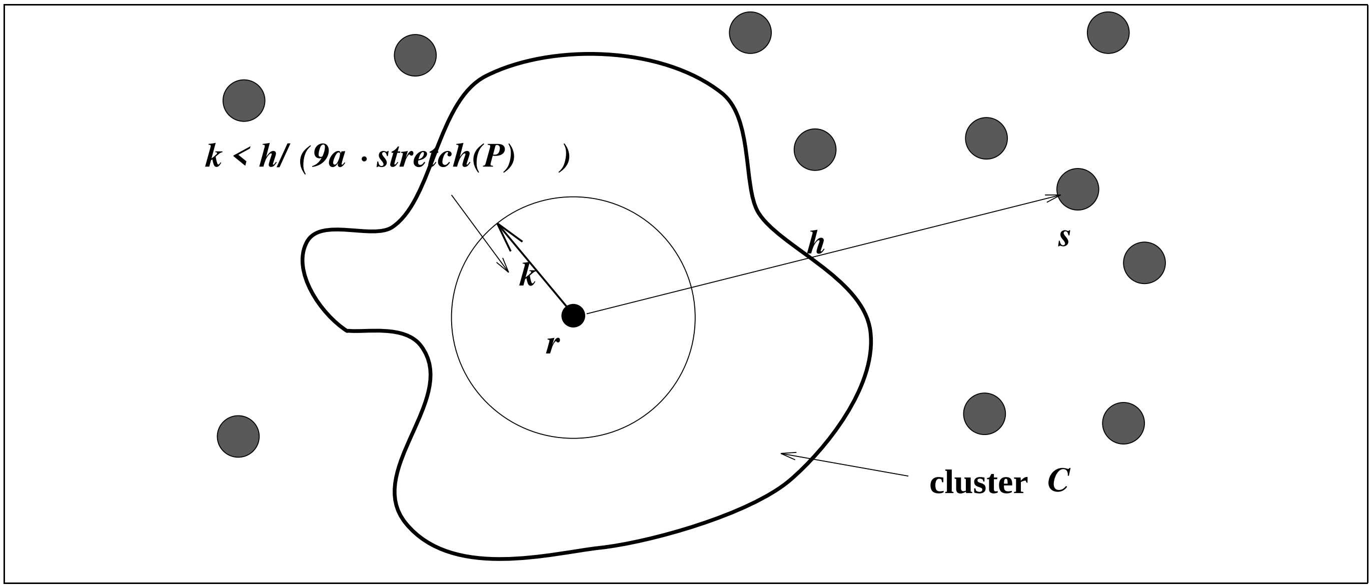

Figure 13: The distributed file allocation algorithm. Replication to r is made if there were D read requests in the interior of $\mathcal{C}$ during the current phase.

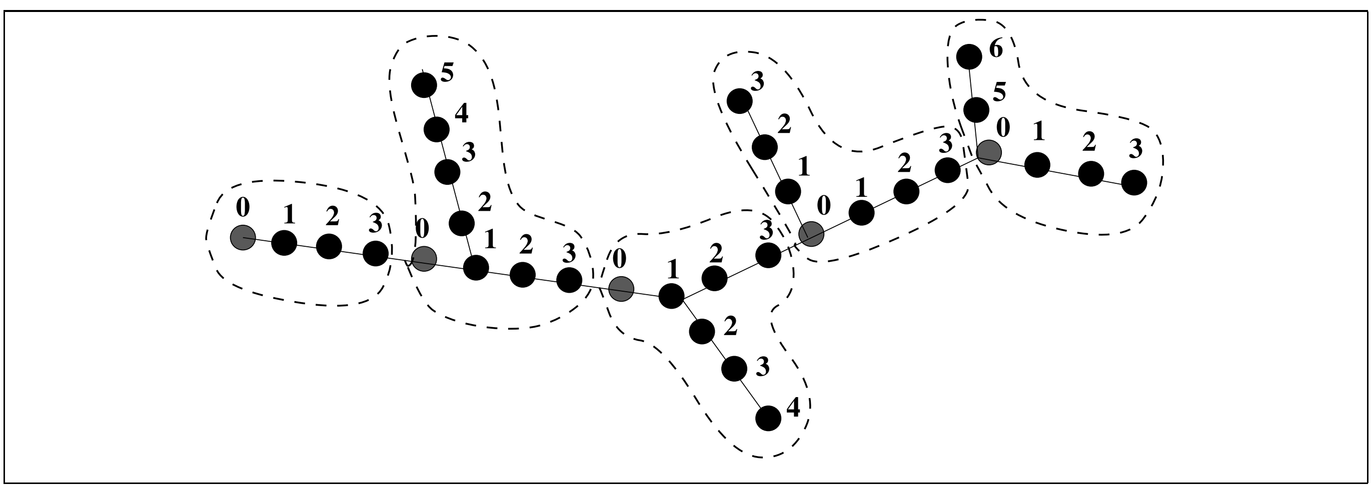

Figure 14: The cover problem algorithm for $k = 4$, and cover radius $2k - 2 = 6$.