CS294-1 On-line Computation & Network Algorithms **Spring 1997**

Lecture 12: March 4

Lecturer: Yair Bartal *Scribe: Andrew Swan*

This lecture covered the *file allocation* problem, a generalization of the file migration problem due to Bartal, Fiat and Rabani [BFR].

12.1 The File Replication Problem

Recall that in the file (page) migration problem, a file is stored at a single node in a network with a distance between any two points p and q of $d(p,q)$. The problem input consists of a sequences of requests for the file, each from a node p in the network. At each request, if the file is stored at q, then the page is read by p at a cost of $d(p,q)$. An algorithm then may decide to *migrate* the page from q to some other node q', at a cost of $Dd(q,q')$.

In the file replication problem, copies of the file may be stored at more than one node in the network. Unlike the migration problem, requests to read file are treated differently from requests to write files. For a read request at node r, the cost is $d(r,Q)$, where Q is the set of nodes where a copy is stored, and $d(r,Q) = \min_{q \in Q} d(r,q)$. For a write request at node w, the cost is $T(Q \cup \{w\})$, where $T(S)$ is the total weight of the minimum Steiner tree (defined below) that spans the nodes in set S. After any request, a file may be replicated at node p' with cost $Dd(p',Q)$. Replicated copies of a file may be deleted at no cost, although the at least one copy of the file must be kept.

12.1.1 Steiner Trees

The cost of writing pages in the file replication problem is defined in terms of a Steiner tree that spans the writing node and all nodes containing copies of the file. A Steiner tree is a tree that spans some set of nodes and may include other nodes as well.

Computing a minimum weight Steiner tree is NP-complete, but there are effective approximation techniques. In particular, a minimum spanning tree that spans S has a cost that is not more than twice the cost of a corresponding Steiner tree.

12.2 A 3-competitive algorithm for uniform networks

A simple special case of the file replication problem is when the underlying network has uniform distances between every pair of nodes. That is, a read always costs 0 if the reading node already has the file and 1 otherwise. The cost of any write request is equal to the number of copies of the file when the request is made in processors other than the requesting processor. The cost of replication is D and deletes cost zero.

12.2.1 The Algorithm

The following algorithm for the file allocation problem on a uniform network is a variant of the Count algorithm presented in Lecture 10 for the file migration problem. Each processor p maintains a counter c_p. Each processor then goes through the following stages:

1. while $c_p < D$, increment c_p every time p initiates a read or p initiates a write while the network as a whole is in the *waiting* state (defined below).
2. replicate the file to p.
3. while $c_p > 0$, decrement c whenever a node q ($q \neq$p) initiates a write request.
4. (*waiting* state) If p has the last copy of the file in the network, remain in this state until the file is replicated to another node. Note that at most one node can be in this state at any time so the network as a whole is said to be in the waiting state if there is a single node in the waiting state.
5. Delete the page from p.
6. Repeat from 1

Note that writes are ordinarily "charged" to the nodes where the file is located. However, when a node p enters state 4, it is ready to delete its copy of the file but cannot since it has the last copy. Instead, when the algorithm enters this state, writes are charged to the node where the request originates. This serves to accelerate the replication of the file to another node, thus allowing p to remove its copy.

Note that if this algorithm is presented with a sequence consisting only of write requests, only one copy of the file will be maintained and the algorithm corresponds to the algorithm for the migration problem discussed in Lecture 10.

12.2.2 Analysis

Theorem 12.1 *This algorithm is 3-competitive*

In the analysis of this algorithm, every cost incurred is "charged" to a single processor; requests at processor p that increase c_p when p is in the first step of the algorithm, the replication cost at step 2, and requests that decrease c_p when p is in the third step are all counted as being incurred by p. Every time p completes a cycle through the algorithm, it has incurred a total cost of $3D$ — D requests at a cost of one while in step 1, a cost of D to replicate the page at step 2, and D requests at a cost of one while in step 3.

For the purposes of analyzing this algorithm, we consider an adversary which can replicate the file for free but incurs a cost of D when a copy of the file is removed. On a set of requests which were charged to p and constituted one pass through the algorithm, the adversary must incur a cost of at least D. There are 3 cases: p may never obtain a copy of the file and thus pay a cost of one for each of the D requests in step 1. If p does have a copy of the file, it must either delete its copy at some point at a cost of D, or pay a cost of one for each of the D requests in step 3. Thus, the worst case ratio of the costs incurred by the algorithm and the adversary is 3 and this algorithm is 3-competitive.

12.3 The On-Line Steiner Tree Problem

Given a weighted graph $G = (V, E, w)$, the input for an instance of the On-line Steiner Tree problem is a sequence of vertices $v_1, v_2, \ldots, v_t$. After the ith request, an algorithm must compute T_i which is a Steiner Tree for $v_1, v_2, \ldots, v_i$. Furthermore, for all $i > 0$, $T_{i+1} \supseteq T_i$. The total cost the algorithm pays is the sum of the weights in T_t. In other words, the algorithm must incrementally construct a Steiner Tree with the goal of constructing a Steiner Tree that is optimal after the last request.

The On-Line Steiner Tree Problem is similar to a limited case of the file replication problem in which the request sequence consists entirely of read requests and $D = 1$. Clearly, any intelligent algorithm for the file replication problem with only read requests will never delete extra copies of the file. Furthermore, if $D = 1$ then the cost of replication is the same as the cost of a non-replicating read so the algorithm that replicates the file at every step is within a factor of two from optimal.

12.4 A randomized reduction from File Replication to Steiner tree

The algorithm SB for the file replication problem is defined as follows. Let A be an algorithm for the on-line Steiner tree problem. SB maintains a simulated "instance" of A. If the file is initially at node p_0, SB starts A with $v_1 = p_0$. Upon a read request from node r, SB serves the request normally. Then, with probability $1/D$, SB "feeds" its instance of A with a request at node r and replicates the file to each new node added to the Steiner tree maintained by A. Upon a write request from node w. Then, with probability $1/\alpha D$, SB deletes all copies except the one closest to w, migrates the file to w, and then "re-starts" A with $v_1 = w$.

Theorem 12.2 *If A is strictly c-competitive for the on-line Steiner tree problem (against an adaptive on-line adversary) then the competitive ratio of SB (using A) is linearly proportional to c.*

The ratio of the competitive ratios is a function of α which is minimized for $\alpha = \sqrt{3}$ in which case the competitive ratio for SB is $(2 + \sqrt{3})c$ against an adaptive on-line adversary.

The proof of the theorem above uses the following construct.

12.5 The Natural Potential Function

Theorem 12.3 *If an algorithm ALG is c-competitive against an adaptive on-line adversary then there exists a "natural" potential function for it.*

If m_n denotes the memory of ALG, A denotes the configuration of the adversary at some point, and τ denotes the remaining requests the will be generated by the adversary given the current configuration, then the natural potential function is:

$$\Upsilon(m_n, A) = \sup_{\mathrm{ADV}} \{ \mathrm{E}(\mathrm{ALG}(m_n, \tau)) - c \cdot \mathrm{ADV}(A, \tau) \}$$

Considering the file replication problem again, let the potential function be:

$$\Phi = (\alpha + 2)D \cdot \Upsilon + \alpha D \cdot T(\hat{B})$$

where $\hat{B}$ is the Steiner tree spanned by A.

We shall demonstrate the potential function argument by describing the analysis for a read request. For a read request at node r, let B be the set of nodes at which a copy of the file exists and let $\Theta = T(\hat{B})$. Then:

$$\begin{aligned} \mathrm{E(SB)} &= d(r,B) + \frac{1}{D} D\mathrm{E}(\mathrm{A}(r)) \leq 2\mathrm{E}(\mathrm{A}(r)) \\ \mathrm{E}(\Delta\Upsilon) &\leq c \cdot \mathrm{ADV}_{\mathrm{St}}(r) - \mathrm{E}(\mathrm{A}(r)) \end{aligned}$$

where $\mathrm{ADV}_{\mathrm{St}}$ is an adversary that maintains a Steiner tree spanning the locations of requests on which the algorithm decided to perform a replication, and the locations of copies of the real adversary. The cost of such an adversary is at most the cost of the real file allocation adversary for serving these requests.

$$\begin{aligned} \mathrm{E}(\Delta\Phi) &= (1 - \frac{1}{D})0 + \frac{1}{D}((\alpha+2)D\mathrm{E}(\Delta\Upsilon) + \alpha D\mathrm{E}(\Delta\Theta)) \\ &= (\alpha+2)\mathrm{E}(\Delta\Upsilon) + \alpha\mathrm{E}(\Delta\Theta) \\ &= (\alpha+2)(c \cdot \mathrm{ADV}(r) - \mathrm{E}(\mathrm{A}(r))) + \alpha\mathrm{E}(\mathrm{A}(r)) \\ &= (\alpha+2)c \cdot \mathrm{ADV}(r) - 2\mathrm{E}(\mathrm{A}(r)) \\ &\leq (\alpha+2)c \cdot \mathrm{ADV}(r) - \mathrm{E(SB)} \end{aligned}$$

By dealing similarly with write requests and with adversary configuration changes, we can conclude that SB is $(\alpha+2)c$-competitive.

References

[BFR92] Y. Bartal, A. Fiat, and Y. Rabani. Competitive algorithms for distributed data management. In *Proc. 24th ACM Symp. on Theory of Computing*, pages 39–50, 1992.