Theorem 5.1: The cascade-search terminates.

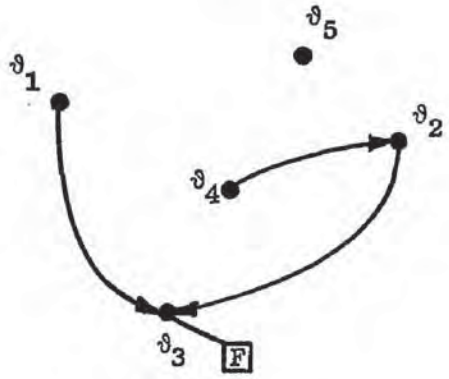
Proof: Since there is no interaction between the various files in the DFS, the theorem is proved for any one file F.

Let N be the number of nodes in the DFS. Let $a_{i,j}$ be a directed arc from node i to j if there exists an entry for F in the RUD at i, and it points to j. This is true for all $1 \leq i,j \leq N$, $i \neq j$. By definition arc $a_{i,j}$ has head i and tail j.

Let G be a simple directed graph, such that G = [V, A(t)], where V is a finite set of vertices of cardinality N, and A(t) is the finite set of arcs $a_{i,j}$ at time t. A path $p_{i,j}$ of G is a cascade-search sequence initiated at i and terminating at j. Figure 5.6 shows such a graph G.

From Axiom 5.1 it can be concluded that only one arc can leave any node. From Axiom 5.2 it can be concluded that any node that has no arcs leaving it represents a case where the file either resides at that node, or does not and the node does not know where the file is. Such a node is a point of termination for the algorithm. Since hosts may reclaim RUD entries at will (Axiom 5.3), points of termination may appear anywhere in the graph.

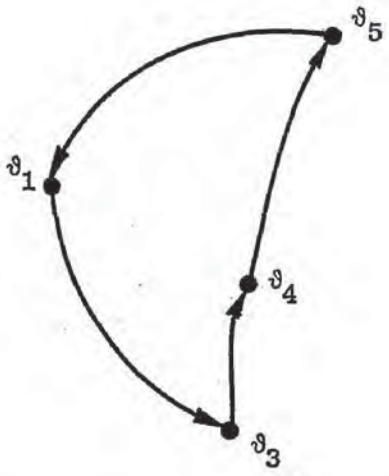
The theorem is proved by showing that G is always loop free, when files migrate, under the constraints of the four axioms. The theorem is proved by contradiction. We show that when a file moves, if a loop is created, then an axiom is violated.



$$V = \{\vartheta_1, \vartheta_2, \vartheta_3, \vartheta_4, \vartheta_5\}$$

$$A(t) = \{a_{1,3}, a_{4,2}, a_{2,3}\}$$

Figure 5.6. A GRAPH G = [V,A(t)].



$$V = \{\vartheta_1, \vartheta_2, \vartheta_3, \vartheta_4, \vartheta_5\}$$

$$A(t) = \{a_{3,4}, a_{4,5}, a_{5,1}\}$$

$$A(t+1) = \{a_{1,3}, a_{4,5}, a_{5,1}\}$$

Figure 5.7. A LOOP IS PRODUCED IF AXIOM 5.2 IS VIOLATED.

Consider a G with $|V| \geq 1$ and $|A(t)| \geq 1$, such that the cascade-search terminates. The file may or may not be present at the tail of any arc. Now, if the file moves it may do so to a node which does not already lie on the graph, in which case an RUD entry will be created by virtue of Axiom 5.4, and the cascade-search will terminate since there is no path continuation from the new node. On the other hand, the file may move to a node already on the graph. By virtue of Axiom 5.2, the RUD entry (should one have existed) at the new node of residence must be removed. If this entry is not removed, and the node from which the file moved was also in the graph, the addition of the arc will produce a loop. Figure 5.7 shows a file moving from node v_1 to v_3, and producing a loop if and only if v_3 does not remove arc $a_{3,4}$.

Since no loops are introduced when a file moves, the cascade-search terminates. RUD entries are created only if the file actually resides at that destination. Hence, in a correctly functioning system, there is no spurious way of introducing RUD entries, which may cause loops.

Q.E.D.

Corollary 5.1.1: If there exist duplicate copies of a file (with the same name) within the DFS, then the cascade-search still terminates.*

*Assume that the consistency problem has been solved for duplicate copies of modifiable files, or that the duplicate copies are read-only.

Proof: For the purpose of the proof, each of the duplicate copies can be thought of as a separate file, and since the proof of Theorem 5.1 assumed independence between files, the corollary follows.

Q.E.D.

5.2.4.3 Reliability of the Cascade-Search

It was proved that the cascade-search will always terminate in a finite time under the assumptions of a correctly functioning system (no random errors to change table entries). In such a correctly functioning environment, the protocol used for implementing the cascade-search could be one in which the initiator of the search listens for a response on a particular port, the identity of which is specified along with the request for the file. The initiator is unaware of the number of RUD pointers chained through by intermediary hosts, and will either receive the file from the host that has it (and possibly update its RUD entry for this file), or realize that the cascade-search has failed, or timeout. The latter signifies that the file was most probably not found using the cascade-search and a alternative approach should be taken.

If the search is performed as described above, then the host from which the request was initiated, has no idea how many other hosts were involved in searching for the file. This is of no consequence if the file is found. In considering the action taken if the cascade-search fails, the requesting host will have to broadcast the request to all hosts. If it knew which ones were visited by the cascade-search, and

thus did not have the file (at that time), then the broadcast request need only go to the remaining hosts, thus minimizing the extra network traffic required to find the file. Of course, one can construct an example where a race condition causes a file not to be found. The file being searched for could have moved to a host that was visited by the cascade-search and at that time knew nothing about the file. The broadcast to the remaining hosts will not find the file, unless one of those hosts has an RUD pointer to the file.

On the other hand, if the DFS was unreliable, then the cascade-search could fail because of the existence of a loop in which a request circulates indefinitely, or because a request is lost. Figure 5.8 shows how the creation of a spurious RUD entry at node v_5 causes there to be a loop. A(t) is the set of arcs that exist at time t, and A(t+1) the set at time t+1. A spurious entry replaced $a_{5,6}$ by $a_{5,8}$ at t+1. A loop in the cascade-search has been created if it is initiated from v_5, v_7 or v_8 for the file. If the requesting host knew which hosts were involved in the loop, it could attempt to break it.

The cascade-search need not be performed "recursively", but could also be performed "iteratively". That is, other hosts do not perform the search on behalf of the initiator, but instead each host on the search path returns the contents of its RUD entry to the initiator. Hence, the initiator iteratively interrogates each host on the search path until the search terminates. In the process of doing so, the initiator can keep track of each host visited, and thus can determine the existence of a loop. The loop can be broken by having all hosts

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.