

# CSE 552 Midterm

Jeffrey Hightower

February 27, 2002

## 1 Coauthoring Historians

### 1.1 Understanding the Problem

The high-level problem to solve is we need a distributed total ordering on chapters where causal ordering is not important. The problem fits in the space shown in Table 1.

	No Causal Ordering	Causal Ordering
No Total Ordering	SRM	CBCAST
Total Ordering	Historians	ABCAST

Table 1: A characterization of the Coauthoring Historians problem.

The desire for *time-efficiency* means we cannot simply route new chapters on a static linear path through the group. Communication *fairness* implies that we cannot chose a single historian to act in a mediator role, for example, by using a star network topology where all new chapters go through the hub. The following is a list of the other relevant assumptions and requirements:

1. Group membership is of arbitrary size  $n$  but membership is known a priori by every participating historian.
2. Chapters are independent and the order of chapters does not matter, as long as all historians appended chapters respecting the same a total ordering. “Appending” a chapter to a historian’s book is a permanent operation.
3. Historians don’t fail and always will obey the protocol (neither fail-stop nor Byzantine failures).
4. Messages may be sent between any two historians (graph is fully connected) and messages are eventually delivered but may be reordered.

## 1.2 Message Ordering

We first build a FIFO abstraction for our channels. Let us start with the simplified case of a single historian  $A$  distributing chapters to the others. Given our assumptions, the complication to handle is messages from historian  $A$  being reordered in transit before reaching the other historians. This can be solved with standard sequence numbering.  $A$  initializes a counter  $c = 0$  and each time a new chapter is sent,  $c$  is incremented and included in the message. As soon as  $A$  complete a chapter it may append it immediately. Other historians maintain a copy of counter  $c$ . Receivers buffer chapter  $c = n$  until they have appended chapter  $c = n - 1$ . Whenever a chapter is appended the local copy of the counter is incremented.

Expanding the sequence numbering scheme to multiple authors is trivial, each historian  $H_i$  maintains a vector  $V_i[1 \dots n]$  of counters from other historians. It increments  $V_i[i]$  and includes this value vector in all chapters it sends. A recipient  $H_j$  will hold delivery of a message from  $H_i$  until it has delivered message  $V_j[i] - 1$ . This is sufficient to guarantee chapters passed between any given sender and receiver will be received in the same order sent. We can now modify our previous assumptions:

4. Messages may be sent between any two historians (graph is fully connected), messages are eventually delivered, *and messages arrive in the same order they are sent.*

## 1.3 Complete Algorithm

We can now define the complete Coauthoring Historians Algorithm. The basic idea is to use a known hash of the chapter numbers to elect a historian to act as “editor” for each new chapter. Historians suggest their completed chapters to the editor as candidates for chapter  $n$ . The editor approves one of them and informs all historians which one she chose. Upon hearing editorial approval, a historian can append the chapter.

Figure 1 illustrates the basic idea and the formal steps are shown in the algorithm below. Historian  $E_n$  is the editor for chapter  $n$ . Observe that lines 6-9 are the editing logic – they only run if this historian is the editor for the current round  $n$ . The last boundary case to handle is if a historian receives a *suggest*( $C', n + 1$ ) before it hears *approve*( $C, n$ ), it can simply buffer the later suggestion until the approval for chapter  $n$  arrives.

## 1.4 Analysis

This algorithm exhibits all the required properties. There are a constant number of mail delays to append a chapter independent of the number of historians. The algorithm is fair because rotating the editor distributes the communication responsibilities across all historians. Finally, this algorithm is starvation free assuming the hash evenly distributes the editing job amongst the historians and

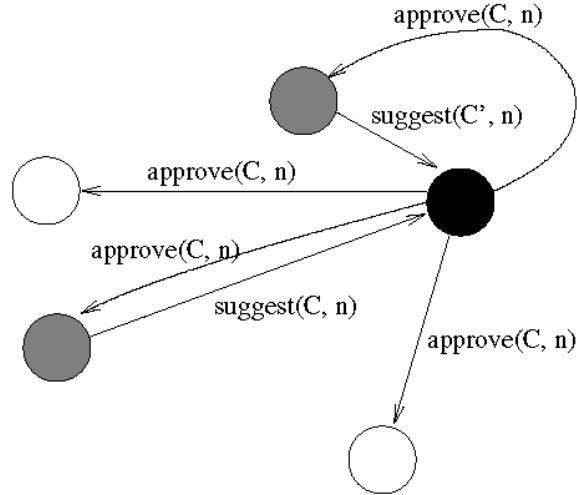


Figure 1: Illustration of the basic Coauthoring Historians algorithm. The two gray nodes are suggesting a chapter for chapter  $n$ . The black node is the editor.

---

**Require:**  $n \leftarrow 0$   
**Require:**  $lastapproved \leftarrow 0$

- 1: **loop**
- 2:    $E_n = hash(n)$
- 3:   **if** chapter  $C$  is complete and not yet appended **then**
- 4:     send  $suggest(C, n)$  to  $E_n$
- 5:   **end if**
- 6:   **if** receive  $suggest(C', n)$  and  $lastapproved < n$  **then**
- 7:     send  $approve(C', n)$  to all historians (including ourselves)
- 8:      $lastapproved \leftarrow n$
- 9:   **end if**
- 10:   **if** receive  $approve(\overline{C}, n)$  from  $E_n$  **then**
- 11:     append  $\overline{C}$  as chapter  $n$  {note: if  $C = \overline{C}$  then this writer got its chapter chosen, otherwise it will just try again next iteration}
- 12:      $n \leftarrow n + 1$
- 13:   **end if**
- 14: **end loop**

---

Algorithm 1: The algorithm used by each Coauthoring Historian.

---

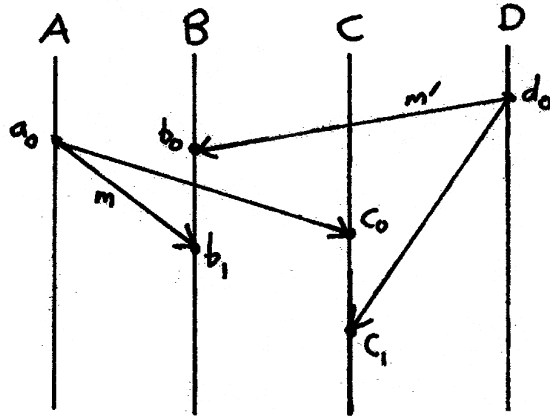


Figure 2: Illustration showing how CBCAST does not ensure totally ordered delivery.

an editor always chooses its own submission first if it has a chapter complete (i.e. 0 message delay to self between lines 4-6 in the algorithm and  $C = C'$ ).

## 2 Multicast Ordering

Figure 2 shows how causally ordered delivery does not ensure totally ordered delivery. Messages  $m$  and  $m'$  are delivered in different orders at process  $B$  and  $C$  so there is clearly no total ordering. However, because  $a_0$  is causally concurrent to  $d_0$ , we have correct causal delivery of messages.

The opposite case is shown in Figure 3. Here, totally ordered delivery does not ensure causally ordered delivery. Messages  $m$  and  $m'$  are delivered in the same order at all recipients ( $m'$  before  $m$ ) meeting the definition of total ordering. However,  $a_0 \rightarrow a_1$ , so causal ordering is not respected.

## 3 A DNS for Tomorrow

The original design goals of DNS were to:

1. replace `hosts.txt`
2. allow distributed maintenance
3. have no size limit for names
4. be an Internet standard
5. have tolerable performance

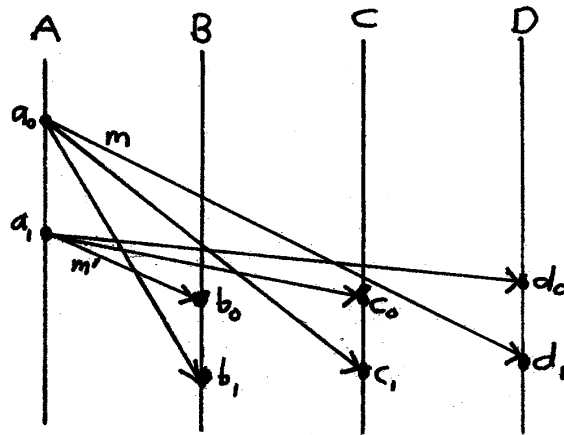


Figure 3: Illustration showing how ABCAST does not ensure causally ordered delivery.

6. be independent of network topology and other naming systems

Here are several suggestions about how to redesign DNS to address modern issues.

- A trivial redesign is to eliminate all the fancy support for types of names as nobody uses this anyway. The real uses of DNS are for naming machines and the natural extension to naming mailboxes – simply the users who reside “@” a particular machine.
- We have seen how the majority of the load at the root servers is caused by bad names propagating up the hierarchy. Homework 1 proposed a solution where clients download the entire list of Level-2 NS records (or, more precisely, the differences) periodically. Given the number of Level-2 domains and the rate of growth, performance could be increased because only the rare missing update has to pass beyond the client. We could achieve a similar outcome in a manner that is much lighter weight to clients by exploiting name structure. To do this, we let local nameservers download the Level-2 list and generate a lightweight Level 0-2 “name parser”. Clients then get this parser from their local nameservers. The client must use this parser identify and drop obvious errors because the local nameserver will not accept names which they cannot parse. DNS was designed to make best-effort with any name it was given. The parsing approach represents a fundamental shift because the leaves of the tree (clients) must screen out bogus names. If they don’t, DNS simply ignores the request.
- Inverse lookups cause a large proportion of the failed lookups in today’s

DNS, so get rid of them and add what is really needed. I claim that most of this traffic is because inverse lookup is being used by many applications as a poor substitute for real authentication. The best solution, albeit a dramatic shift, is to eliminate inverse lookup capability completely and instead integrate real cryptographic authentication mechanisms to enable trust that a name to address mapping is correct.

- We observe that Level-2 is usually the highest point in the tree that any real administrative separation occurs in practice – defining a company, school, organization, or concern. Although one might argue that the country codes can be useful since they define administrative boundaries, NET, COM, ORG, etc seem to be arbitrary semantic constructs. Their inclusion seems to run contrary to the original DNS goal of *not* encoding any semantics in the names. The redesign is basically to truncate the hierarchy at what would currently be Level-2 in the tree instead of continuing the hierarchy all the way up to the TLDs and then the 13 mission-critical root servers (we have seen how killing at least seven of them would bring down all of DNS).

We will instead build a distributed database laterally at what was Level-2 and coordinate them with a distributed consensus protocol. Each organization that desires a TLD then manages its own nameserver can introduce its own names (e.g. `website.pepsi` or simply `pepsi`). Looking up a name (“pepsi” in this case) will retrieve an address and also a signing certificate. A trusted 3rd party, for example the Patents and Trademarks office, would act as the authoritative source of public keys and also could sign the name certificates. We now have the problem of distributing keys to clients, but this can be done using periodic (diff) client downloads because, unlike name TTLs, keys change very rarely. Another advantage is the 3rd party would not have to manage only keys and not all the databases that Verisign does today. As an added bonus, this redesign might eliminate the ridiculous (in my opinion) business of buying, selling, speculating, and trading in domain names.

This entire improvement is sort of analogous to the way routing is handled in today’s Internet. Most administrative domains run OSPF inside, but to do so over the wide area would be intractable. The backbone routers coordinate using BGP, which, although it has plenty of its own problems, is much more suited to the wide area.

A parting comment on DNS: Although mail must still be precisely addressed and the need for authentication is real, I believe that we are moving to a world where DNS names are less and less important for the biggest distributed application: the web. Thanks to Google, semantic naming of resources is now viable and even becoming the standard. There seems to be no reason in theory why Google could not work directly on IP addresses, providing The Web to people without the naming abstraction provided by DNS, assuming the availability of authentication information about the results (e.g. I want to be sure that this

page Google gave me of the NY Times *is* provided by the real NY Times). Furthermore, there is quite a bit of work into lookup and discovery services such as Jini and SDS providing very Google-like semantic naming to non-humans systems using the web.

## 4 Paxos Perscrutinations

### 4.1 Desperate Leader

There is no way for a desperate leader to guarantee that her proposed value is chosen without modifying the protocol. She could statistically increase her chances by rapidly starting one or more new rounds in the hope that agents will hear the interrogations for the new rounds and squelch previous rounds without any value being chose. However, the leader cannot guarantee this will work because she cannot predict the behavior of the network. It is possible that some number of these new round interrogations will be lost and another leader will concurrently propose a different value that will reach a majority of agents and be chosen.

### 4.2 Group Membership

The complications in using Paxos for group membership changes are the change in what constitutes a majority. Agents must broadcast group membership changes they have accepted so others are informed about the new membership.

If adding  $n$  members to a parliament of size  $p$ , for the addition to be chosen, the proposal must be accepted by a supermajority of the expanded membership. A supermajority has the property that any two supermajorities have at least  $n + 1$  members in common. This supermajority is then a proper majority for the new parliament of size  $p^+$  where  $p^+ = p + n$ .

When  $n$  members are leaving a parliament of size  $p$ , for the subtraction to be chosen, the proposal must be accepted by a majority of  $p$ . A submajority (any two submajorities have  $1 - n$  members in common) is **not** appropriate because it is possible that some of those accepting the subtraction are the ones leaving the parliament and we would end up with a case where a majority of  $p^-$ ,  $p^- = p - n$ , have not accepted the change, thus violating the definition of chosen in the new  $p^-$  parliament.

These conditions are sufficient to guarantee that a leader who returns and still believes the parliament is of size  $p$  will never falsely believe a something has been chosen for this decree given the true parliament is now of size  $p^+$  or  $p^-$ . The addition case is obvious. In the subtraction case, this tardy leader will simply not hear back from the  $n$  members who have left and will consider them failed until it learns about the membership change. Furthermore, if it does hear interrogation replies from what it considers a majority, at least one of these it hears from will have accepted the membership subtraction so the leader will pessimistically be forced to propose the subtraction.

	Agent 1	Agent 2	Agent 3	Agent 4	Agent 5
Round 1	-	-	no	no	<i>no</i> (1)

Table 2: A Byzantine agent can affect correctness.

### 4.3 Byzantine Faults

A single Byzantine agent can potentially affect correctness. Suppose there are five agents who report to the leader as shown in Table 2. Agent 5 is Byzantine because it should have reported “1”, but instead it reported “no”. The leader has heard from a majority. Due to the Byzantine agent, the leader mistakenly believes that it is free to propose any value it wants. Correctness could be violated if Agents 1 and 2 had previously accepted “1”, but the leader now proposes “2” and “2” is accepted by a majority.

Because of the way “chosen” is defined, a single Byzantine leader sending different values to different agents in a single round will not affect correctness. However, because a leader can act Byzantine, it could violate correctness across multiple rounds. For example, the leader starts Round 1 and sends value “1” to agents. Then, it starts Round 2 and, despite hearing an interrogation response from the majority indicating “1” was chosen in Round 1, propose “2” and get that chosen in Round 2, thus violating correctness.

### 4.4 Constrained Proposals

Given this 4 round history, the leader can propose any value it wishes in Round 5. Although it appears as if “2” may have been chosen if we look back at Round 2 in isolation, that must not be the case because we observe that “3” was proposed in Round 3 so Agent 2 must not have accepted “2” in Round 2.

### 4.5 Bogus History

There are two things wrong with this history:

- In Round 1 there were two different values proposed by the same leader because Agent 1 accepted “4” and Agent 3 accepted “3”.
- The leader should not have proposed “1” after Round 4 because “5” had already been chosen in Round 3. The leader could have correctly proposed “5” in Round 4 in this history (perhaps only accepted by Agent 2) but it should have proposed “5” again in Round 5.