Document Server: Handling Concurrent Saves

Asked 16 years, 4 months ago Modified 10 years, 10 months ago
Viewed 665 times

2

I'm implementing a document server. Currently, if two users open the same document, then modify it and save the changes, the document's state will be undefined (either the first user's changes are saved permanently, or the second's). This is entirely unsatisfactory. I considered two possibilities to solve this problem:

The first is to lock the document when it is opened by someone the first time, and unlock it when it is closed. But if the network connection to the server is suddenly interrupted, the document would stay in a forever-locked state. The obvious solution is to send regular pings to the server. If the server doesn't receive K pings in a row (K > 1) from a particular client, documents locked by this client are unlocked. If that client re-appears, documents are locked again, if someone hadn't already locked them. This could also help if the client application (running in web browser) is terminated unexpectedly, making it impossible to send a 'quitting, unlock my documents' signal to the server.

The second is to store multiple versions of the same document saved by different users. If changes to the

document are made in rapid succession, the system would offer either to merge versions or to select a preferred version. To optimize storage space, only document diffs should be kept (just like source control software).

What method should I choose, taking into consideration that the connection to the server might *sometimes* be slow and unresponsive? How should the parameters (ping interval, rapid succession interval) be determined?

P.S. Unfortunately, I can't store the documents in a database.

concurrency locking versioning

Share

Improve this question

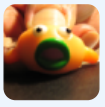
Follow

edited Feb 14, 2014 at 16:37

pschueller

4,427 ● 2 ● 28 ● 50

asked Aug 13, 2008 at 11:49



entropia

23 ● 1 ● 2

3 Answers

Sorted by: Highest score (default)

1

The first option you describe is essentially a pessimistic locking model whilst the second is an optimistic model. Which one to choose really comes down to a number of

factors but essentially boils down to how the business wants to work. For example, would it unduly inconvenience the users if a document they needed to edit was locked by another user? What happens if a document is locked and someone goes on holiday with their client connected? What is the likely contention for each document - i.e. how likely is it that the same document will be modified by two users at the same time?, how localised are the modifications likely to be within a single document? (If the same section is modified regularly then performing a merge may take longer than simply making the changes again).

Assuming the contention is relatively low and/or the size of each change is fairly small then I would probably opt for an optimistic model that resolves conflicts using an automatic or manual merge. A version number or a checksum of the document's contents can be used to determine if a merge is required.

Share Improve this answer

Follow

answered Aug 13, 2008 at 13:26

Wheelie

3,906 ● 2 ● 35 ● 39

0

My suggestion would be something like your first one. When the first user (Bob) opens the document, he acquires a lock so that other users can only read the current document. If the user saves the document while he is using it, he keeps the lock. Only when he exits the document, it is unlocked and other people can edit it.

If the second user (Kate) opens the document while Bob has the lock on it, Kate will get a message saying the document is uneditable but she can read it until it the lock has been released.

So what happens when Bob acquires the lock, maybe saves the document once or twice but then exits the application leaving the lock hanging?

As you said yourself, requiring the client with the lock to send pings at a certain frequency is probably the best option. If you don't get a ping from the client for a set amount of time, this effectively means his client is not responding anymore. If this is a web application you can use javascript for the pings. The document that was last saved releases its lock and Kate can now acquire it.

A ping can contain the name of the document that the client has a lock on, and the server can calculate when the last ping for that document was received.

Share Improve this answer

Follow

answered Aug 13, 2008 at 12:10

deadtime

1,200 ● 1 ● 12 ● 19

0

Currently documents are published by a limited group of people, each of them working on a separate subject. So, the inconvenience introduced by locks is minimized. People mostly extend existing documents and correct mistakes in them.

Speaking about the pessimistic model, the 'left client connected for N days' scenario could be avoided by setting lock expire date to, say, one day before lock start date. Because documents edited are by no means mission critical, and are modified by multiple users quite rarely, that could be enough.

Now consider the optimistic model. How should the differences be detected, if the documents have some regular (say, hierarchical) structure? If not? What are the chances of successful automatic merge in these cases?

The situation becomes more complicated, because some of the documents (edited by the 'admins' user group) contain important configuration information (document global index, user roles, etc.). To my mind, locks are more advantageous for precisely this kind of information, because it's not changed on everyday basis. So some hybrid solution might be acceptable.

What do you think?

answered Aug 13, 2008 at 14:07

entropia

23 ● 1 ● 2