<http://mail-archives.apache.org/mod_mbox/lucene-dev/200110.mbox/%3C4BC270C6AB8AD411AD0B00B0D0493DF0EE7C5E@mail.grandcentral.com%3E>

From DCutting@grandcentral.com Wed Oct 10 21:01:56 2001

Return-Path: <DCutting@grandcentral.com>

Mailing-List: contact lucene-dev-help@jakarta.apache.org; run by ezmlm

Delivered-To: mailing list lucene-dev@jakarta.apache.org

Received: (qmail 86000 invoked from network); 10 Oct 2001 21:01:56 -0000

Received: from unknown (HELO mta.12.com) (65.198.8.41)

by daedalus.apache.org with SMTP; 10 Oct 2001 21:01:56 -0000

Received: (qmail 1453 invoked from network); 10 Oct 2001 20:59:18 -0000

Received: from unknown (HELO riker.grandcentral.com) (10.102.15.55)

by mta.12.com with SMTP; 10 Oct 2001 20:59:18 -0000

Received: by mail.grandcentral.com with Internet Mail Service (5.5.2653.19)

id <42Y1HPXT>; Wed, 10 Oct 2001 13:51:24 -0700

Message-ID: <4BC270C6AB8AD411AD0B00B0D0493DF0EE7C5E@mail.grandcentral.com>

From: Doug Cutting <DCutting@grandcentral.com>

To: "'lucene-dev@jakarta.apache.org'" <lucene-dev@jakarta.apache.org>

Subject: RE: multithreading in SegmentsReader

Date: Wed, 10 Oct 2001 13:51:24 -0700

MIME-Version: 1.0

X-Mailer: Internet Mail Service (5.5.2653.19)

Content-Type: text/plain;

charset="iso-8859-1"

X-Spam-Rating: daedalus.apache.org 1.6.2 0/1000/N

Your analysis looks good to me.

I think it would be simpler, if a bit less optimized, to just make

SegmentsReader.numDocs() and SegmentsReader.delete() synchronized methods.

Does that sound like a reasonable fix to you?

Thanks for spotting this.

As for closing, your analysis also sounds correct: if an index is closed

from one thread while another thread is reading it, then the other thread

may experience errors. I think the a good solution here is simply to not

close an IndexReader when there may be other threads that may be using it.

FSDirectory closes files with finalize methods, so all files will be closed

when the IndexReader is garbage collected. The primary client of the

close() method is the index merging code, which requires prompt closing of

files to keep the number of file descriptors low, and knows that other

threads cannot be accessing them.

So I think changing the documentation and examples will suffice here. The

documentation for IndexReader.close() method should note that it should only

be called if no other threads are accessing the index, and the search

examples should not bother to call close, letting the garbage collector

close files. Is that acceptable?

Doug

> -----Original Message-----

> From: Dmitry Serebrennikov [mailto:dmitrys@earthlink.net]

> Sent: Wednesday, October 10, 2001 1:36 PM

> To: lucene-dev@jakarta.apache.org

> Subject: multithreading in SegmentsReader

>

>

> This fixes a potential race condition in SegmentsReader where

> a call to

> numDocs() could return -1 if a document is being deleted at the same

> time by another thread.

>

> (Caviat: I don't actually have a test case that shows this

> problem, nor

> was I able to verify this fix. Found it by reading the code and it

> seemed easy enough and important enough to put in a fix.)

>

> Also, I think there may be a larger problem with this class

> that happens

> when close() is called concurrently with any of the methods

> that iterate

> over the sub-readers and delegate to them. It looks like the second

> thread may get an IOException if it calls on a reader that has been

> closed. Not sure how to fix this without killing performance.

> Should we

> just document this as a "feature" and say that its application's

> responsibility to ensure that all queries are finished before calling

> close()?

>

> ===================================================================

> RCS file:

> /home/cvspublic/jakarta-lucene/src/java/org/apache/lucene/inde

> x/SegmentsReader.java,v

> retrieving revision 1.1.1.1

> diff -w -u -r1.1.1.1 SegmentsReader.java

> --- SegmentsReader.java 2001/09/18 16:29:54 1.1.1.1

> +++ SegmentsReader.java 2001/10/10 20:10:10

> @@ -78,7 +78,18 @@

> }

>

> public final int numDocs() {

> - if (numDocs == -1) { // check cache

> + // Synchro: two goals -- prevent duplicate cache

> revalidation (minor)

> + // and prevent invalidation of the cache between checking it and

> returning

> + // the value.

> +

> + // Atomically copy it. Even if it changes between the

> + // if and the return, we still return a number that was valid

> + // when we entered this method.

> + final int tmpNumDocs = numDocs;

> + if (tmpNumDocs != -1) return tmpNumDocs; // check cache

> +

> + synchronized(this) {

> + if (numDocs == -1) {

> int n = 0; // cache

> miss--recompute

> for (int i = 0; i < readers.length; i++)

> n += readers[i].numDocs(); // sum from readers

> @@ -86,6 +97,7 @@

> }

> return numDocs;

> }

> + }

>

> public final int maxDoc() {

> return maxDoc;

> @@ -102,10 +114,13 @@

> }

>

> public final void delete(int n) throws IOException {

> + // Synchro: synchronize with the cache re-validation

> logic in numDocs

> + synchronized(this) {

> numDocs = -1; // invalidate cache

> int i = readerIndex(n); // find segment num

> readers[i].delete(n - starts[i]); // dispatch

> to segment

> reader

> }

> + }

>