Item 35
git_comments:

1. If a user specifies a too-restrictive or too-slow scanner, the client might time out and disconnect while the server side is still processing the request. We should abort aggressively in that case.
2. * * Filter which makes sleeps for a second between each row of a scan. * This can be useful for manual testing of bugs like HBASE-5973. For example: * <code> * create 't1', 'f1' * 1.upto(100) { |x| put 't1', 'r' + x.to_s, 'f1:q1', 'hi' } * import org.apache.hadoop.hbase.filter.TestFilter * scan 't1', { FILTER => TestFilter::SlowScanFilter.new(), CACHE => 50 } * </code>

git_commits:

1. **summary:** HBASE-5973. Add ability for potentially long-running IPC calls to abort if client disconnects. Contributed by Todd Lipcon.
   message: HBASE-5973. Add ability for potentially long-running IPC calls to abort if client disconnects. Contributed by Todd Lipcon. git-svn-id: https://svn.apache.org/repos/asf/hbase/branches/0.94@1336788 13f79535-47bb-0310-9956-ffa450edef68

github_issues:

github_issues_comments:

github_pulls:

github_pulls_comments:

github_pulls_reviews:

jira_issues:

1. **summary:** Add ability for potentially long-running IPC calls to abort if client disconnects
   description: We recently had a cluster issue where a user was submitting scanners with a very restrictive filter, and then calling next() with a high scanner caching value. The clients would generally time out the next() call and disconnect, but the IPC kept running looking to fill the requested number of rows. Since this was in the context of MR, the tasks making the calls would retry, and the retries wuld be more likely to time out due to contention with the previous still-running scanner next() call. Eventually, the system spiraled out of control. We should add a hook to the IPC system so that RPC calls can check if the client has already disconnected. In such a case, the next() call could abort processing, given any further work is wasted. I imagine coprocessor endpoints, etc, could make good use of this as well.
   label: test
2. **summary:** Add ability for potentially long-running IPC calls to abort if client disconnects
   description: We recently had a cluster issue where a user was submitting scanners with a very restrictive filter, and then calling next() with a high scanner caching value. The clients would generally time out the next() call and disconnect, but the IPC kept running looking to fill the requested number of rows. Since this was in the context of MR, the tasks making the calls would retry, and the retries wuld be more likely to time out due to contention with the previous still-running scanner next() call. Eventually, the system spiraled out of control. We should add a hook to the IPC system so that RPC calls can check if the client has already disconnected. In such a case, the next() call could abort processing, given any further work is wasted. I imagine coprocessor endpoints, etc, could make good use of this as well.
3. **summary:** Add ability for potentially long-running IPC calls to abort if client disconnects
   description: We recently had a cluster issue where a user was submitting scanners with a very restrictive filter, and then calling next() with a high scanner caching value. The clients would generally time out the next() call and disconnect, but the IPC kept running looking to fill the requested number of rows. Since this was in the context of MR, the tasks making the calls would retry, and the retries wuld be more likely to time out due to contention with the previous still-running scanner next() call. Eventually, the system spiraled out of control. We should add a hook to the IPC system so that RPC calls can check if the client has already disconnected. In such a case, the next() call could abort processing, given any further work is wasted. I imagine coprocessor endpoints, etc, could make good use of this as well.
   label: code-design
4. **summary:** Add ability for potentially long-running IPC calls to abort if client disconnects

description: We recently had a cluster issue where a user was submitting scanners with a very restrictive filter, and then calling next() with a high scanner caching value. The clients would generally time out the next() call and disconnect, but the IPC kept running looking to fill the requested number of rows. Since this was in the context of MR, the tasks making the calls would retry, and the retries wuld be more likely to time out due to contention with the previous still-running scanner next() call. Eventually, the system spiraled out of control. We should add a hook to the IPC system so that RPC calls can check if the client has already disconnected. In such a case, the next() call could abort processing, given any further work is wasted. I imagine coprocessor endpoints, etc, could make good use of this as well.
label: code-design

5. **summary:** Add ability for potentially long-running IPC calls to abort if client disconnects
description: We recently had a cluster issue where a user was submitting scanners with a very restrictive filter, and then calling next() with a high scanner caching value. The clients would generally time out the next() call and disconnect, but the IPC kept running looking to fill the requested number of rows. Since this was in the context of MR, the tasks making the calls would retry, and the retries wuld be more likely to time out due to contention with the previous still-running scanner next() call. Eventually, the system spiraled out of control. We should add a hook to the IPC system so that RPC calls can check if the client has already disconnected. In such a case, the next() call could abort processing, given any further work is wasted. I imagine coprocessor endpoints, etc, could make good use of this as well.
label: code-design
6. **summary:** Add ability for potentially long-running IPC calls to abort if client disconnects
description: We recently had a cluster issue where a user was submitting scanners with a very restrictive filter, and then calling next() with a high scanner caching value. The clients would generally time out the next() call and disconnect, but the IPC kept running looking to fill the requested number of rows. Since this was in the context of MR, the tasks making the calls would retry, and the retries wuld be more likely to time out due to contention with the previous still-running scanner next() call. Eventually, the system spiraled out of control. We should add a hook to the IPC system so that RPC calls can check if the client has already disconnected. In such a case, the next() call could abort processing, given any further work is wasted. I imagine coprocessor endpoints, etc, could make good use of this as well.
7. **summary:** Add ability for potentially long-running IPC calls to abort if client disconnects
description: We recently had a cluster issue where a user was submitting scanners with a very restrictive filter, and then calling next() with a high scanner caching value. The clients would generally time out the next() call and disconnect, but the IPC kept running looking to fill the requested number of rows. Since this was in the context of MR, the tasks making the calls would retry, and the retries wuld be more likely to time out due to contention with the previous still-running scanner next() call. Eventually, the system spiraled out of control. We should add a hook to the IPC system so that RPC calls can check if the client has already disconnected. In such a case, the next() call could abort processing, given any further work is wasted. I imagine coprocessor endpoints, etc, could make good use of this as well.
8. **summary:** Add ability for potentially long-running IPC calls to abort if client disconnects
description: We recently had a cluster issue where a user was submitting scanners with a very restrictive filter, and then calling next() with a high scanner caching value. The clients would generally time out the next() call and disconnect, but the IPC kept running looking to fill the requested number of rows. Since this was in the context of MR, the tasks making the calls would retry, and the retries wuld be more likely to time out due to contention with the previous still-running scanner next() call. Eventually, the system spiraled out of control. We should add a hook to the IPC system so that RPC calls can check if the client has already disconnected. In such a case, the next() call could abort processing, given any further work is wasted. I imagine coprocessor endpoints, etc, could make good use of this as well.
9. **summary:** Add ability for potentially long-running IPC calls to abort if client disconnects
description: We recently had a cluster issue where a user was submitting scanners with a very restrictive filter, and then calling next() with a high scanner caching value. The clients would generally time out the next() call and disconnect, but the IPC kept running looking to fill the requested number of rows. Since this was in the context of MR, the tasks making the calls would retry, and the retries wuld be more likely to time out due to contention with the previous still-running scanner next() call. Eventually, the system spiraled out of control. We should add a hook to the IPC system so that RPC calls can check if the client has already disconnected. In such a case, the next() call could abort processing, given any further work is wasted. I imagine coprocessor endpoints, etc, could make good use of this as well.
label: code-design
10. **summary:** Add ability for potentially long-running IPC calls to abort if client disconnects
description: We recently had a cluster issue where a user was submitting scanners with a very restrictive filter, and then calling next() with a high scanner caching value. The clients would generally time out the next() call and disconnect, but the IPC kept running looking to fill the requested number of rows. Since this

was in the context of MR, the tasks making the calls would retry, and the retries wuld be more likely to time out due to contention with the previous still-running scanner next() call. Eventually, the system spiraled out of control. We should add a hook to the IPC system so that RPC calls can check if the client has already disconnected. In such a case, the next() call could abort processing, given any further work is wasted. I imagine coprocessor endpoints, etc, could make good use of this as well.

11. **summary:** Add ability for potentially long-running IPC calls to abort if client disconnects
description: We recently had a cluster issue where a user was submitting scanners with a very restrictive filter, and then calling next() with a high scanner caching value. The clients would generally time out the next() call and disconnect, but the IPC kept running looking to fill the requested number of rows. Since this was in the context of MR, the tasks making the calls would retry, and the retries wuld be more likely to time out due to contention with the previous still-running scanner next() call. Eventually, the system spiraled out of control. We should add a hook to the IPC system so that RPC calls can check if the client has already disconnected. In such a case, the next() call could abort processing, given any further work is wasted. I imagine coprocessor endpoints, etc, could make good use of this as well.
12. **summary:** Add ability for potentially long-running IPC calls to abort if client disconnects
description: We recently had a cluster issue where a user was submitting scanners with a very restrictive filter, and then calling next() with a high scanner caching value. The clients would generally time out the next() call and disconnect, but the IPC kept running looking to fill the requested number of rows. Since this was in the context of MR, the tasks making the calls would retry, and the retries wuld be more likely to time out due to contention with the previous still-running scanner next() call. Eventually, the system spiraled out of control. We should add a hook to the IPC system so that RPC calls can check if the client has already disconnected. In such a case, the next() call could abort processing, given any further work is wasted. I imagine coprocessor endpoints, etc, could make good use of this as well.
13. **summary:** Add ability for potentially long-running IPC calls to abort if client disconnects
description: We recently had a cluster issue where a user was submitting scanners with a very restrictive filter, and then calling next() with a high scanner caching value. The clients would generally time out the next() call and disconnect, but the IPC kept running looking to fill the requested number of rows. Since this was in the context of MR, the tasks making the calls would retry, and the retries wuld be more likely to time out due to contention with the previous still-running scanner next() call. Eventually, the system spiraled out of control. We should add a hook to the IPC system so that RPC calls can check if the client has already disconnected. In such a case, the next() call could abort processing, given any further work is wasted. I imagine coprocessor endpoints, etc, could make good use of this as well.
14. **summary:** Add ability for potentially long-running IPC calls to abort if client disconnects
description: We recently had a cluster issue where a user was submitting scanners with a very restrictive filter, and then calling next() with a high scanner caching value. The clients would generally time out the next() call and disconnect, but the IPC kept running looking to fill the requested number of rows. Since this was in the context of MR, the tasks making the calls would retry, and the retries wuld be more likely to time out due to contention with the previous still-running scanner next() call. Eventually, the system spiraled out of control. We should add a hook to the IPC system so that RPC calls can check if the client has already disconnected. In such a case, the next() call could abort processing, given any further work is wasted. I imagine coprocessor endpoints, etc, could make good use of this as well.
15. **summary:** Add ability for potentially long-running IPC calls to abort if client disconnects
description: We recently had a cluster issue where a user was submitting scanners with a very restrictive filter, and then calling next() with a high scanner caching value. The clients would generally time out the next() call and disconnect, but the IPC kept running looking to fill the requested number of rows. Since this was in the context of MR, the tasks making the calls would retry, and the retries wuld be more likely to time out due to contention with the previous still-running scanner next() call. Eventually, the system spiraled out of control. We should add a hook to the IPC system so that RPC calls can check if the client has already disconnected. In such a case, the next() call could abort processing, given any further work is wasted. I imagine coprocessor endpoints, etc, could make good use of this as well.
16. **summary:** Add ability for potentially long-running IPC calls to abort if client disconnects
description: We recently had a cluster issue where a user was submitting scanners with a very restrictive filter, and then calling next() with a high scanner caching value. The clients would generally time out the next() call and disconnect, but the IPC kept running looking to fill the requested number of rows. Since this was in the context of MR, the tasks making the calls would retry, and the retries wuld be more likely to time out due to contention with the previous still-running scanner next() call. Eventually, the system spiraled out of control. We should add a hook to the IPC system so that RPC calls can check if the client has already disconnected. In such a case, the next() call could abort processing, given any further work is wasted. I imagine coprocessor endpoints, etc, could make good use of this as well.
17. **summary:** Add ability for potentially long-running IPC calls to abort if client disconnects

description: We recently had a cluster issue where a user was submitting scanners with a very restrictive filter, and then calling next() with a high scanner caching value. The clients would generally time out the next() call and disconnect, but the IPC kept running looking to fill the requested number of rows. Since this was in the context of MR, the tasks making the calls would retry, and the retries wuld be more likely to time out due to contention with the previous still-running scanner next() call. Eventually, the system spiraled out of control. We should add a hook to the IPC system so that RPC calls can check if the client has already disconnected. In such a case, the next() call could abort processing, given any further work is wasted. I imagine coprocessor endpoints, etc, could make good use of this as well.

18. **summary:** Add ability for potentially long-running IPC calls to abort if client disconnects
    description: We recently had a cluster issue where a user was submitting scanners with a very restrictive filter, and then calling next() with a high scanner caching value. The clients would generally time out the next() call and disconnect, but the IPC kept running looking to fill the requested number of rows. Since this was in the context of MR, the tasks making the calls would retry, and the retries wuld be more likely to time out due to contention with the previous still-running scanner next() call. Eventually, the system spiraled out of control. We should add a hook to the IPC system so that RPC calls can check if the client has already disconnected. In such a case, the next() call could abort processing, given any further work is wasted. I imagine coprocessor endpoints, etc, could make good use of this as well.
19. **summary:** Add ability for potentially long-running IPC calls to abort if client disconnects
    description: We recently had a cluster issue where a user was submitting scanners with a very restrictive filter, and then calling next() with a high scanner caching value. The clients would generally time out the next() call and disconnect, but the IPC kept running looking to fill the requested number of rows. Since this was in the context of MR, the tasks making the calls would retry, and the retries wuld be more likely to time out due to contention with the previous still-running scanner next() call. Eventually, the system spiraled out of control. We should add a hook to the IPC system so that RPC calls can check if the client has already disconnected. In such a case, the next() call could abort processing, given any further work is wasted. I imagine coprocessor endpoints, etc, could make good use of this as well.
20. **summary:** Add ability for potentially long-running IPC calls to abort if client disconnects
    description: We recently had a cluster issue where a user was submitting scanners with a very restrictive filter, and then calling next() with a high scanner caching value. The clients would generally time out the next() call and disconnect, but the IPC kept running looking to fill the requested number of rows. Since this was in the context of MR, the tasks making the calls would retry, and the retries wuld be more likely to time out due to contention with the previous still-running scanner next() call. Eventually, the system spiraled out of control. We should add a hook to the IPC system so that RPC calls can check if the client has already disconnected. In such a case, the next() call could abort processing, given any further work is wasted. I imagine coprocessor endpoints, etc, could make good use of this as well.
21. **summary:** Add ability for potentially long-running IPC calls to abort if client disconnects
    description: We recently had a cluster issue where a user was submitting scanners with a very restrictive filter, and then calling next() with a high scanner caching value. The clients would generally time out the next() call and disconnect, but the IPC kept running looking to fill the requested number of rows. Since this was in the context of MR, the tasks making the calls would retry, and the retries wuld be more likely to time out due to contention with the previous still-running scanner next() call. Eventually, the system spiraled out of control. We should add a hook to the IPC system so that RPC calls can check if the client has already disconnected. In such a case, the next() call could abort processing, given any further work is wasted. I imagine coprocessor endpoints, etc, could make good use of this as well.
22. **summary:** Add ability for potentially long-running IPC calls to abort if client disconnects
    description: We recently had a cluster issue where a user was submitting scanners with a very restrictive filter, and then calling next() with a high scanner caching value. The clients would generally time out the next() call and disconnect, but the IPC kept running looking to fill the requested number of rows. Since this was in the context of MR, the tasks making the calls would retry, and the retries wuld be more likely to time out due to contention with the previous still-running scanner next() call. Eventually, the system spiraled out of control. We should add a hook to the IPC system so that RPC calls can check if the client has already disconnected. In such a case, the next() call could abort processing, given any further work is wasted. I imagine coprocessor endpoints, etc, could make good use of this as well.
23. **summary:** Add ability for potentially long-running IPC calls to abort if client disconnects
    description: We recently had a cluster issue where a user was submitting scanners with a very restrictive filter, and then calling next() with a high scanner caching value. The clients would generally time out the next() call and disconnect, but the IPC kept running looking to fill the requested number of rows. Since this was in the context of MR, the tasks making the calls would retry, and the retries wuld be more likely to time out due to contention with the previous still-running scanner next() call. Eventually, the system spiraled out of control. We should add a hook to the IPC system so that RPC calls can check if the client has already

disconnected. In such a case, the next() call could abort processing, given any further work is wasted. I imagine coprocessor endpoints, etc, could make good use of this as well.

24. **summary:** Add ability for potentially long-running IPC calls to abort if client disconnects
    description: We recently had a cluster issue where a user was submitting scanners with a very restrictive filter, and then calling next() with a high scanner caching value. The clients would generally time out the next() call and disconnect, but the IPC kept running looking to fill the requested number of rows. Since this was in the context of MR, the tasks making the calls would retry, and the retries wuld be more likely to time out due to contention with the previous still-running scanner next() call. Eventually, the system spiraled out of control. We should add a hook to the IPC system so that RPC calls can check if the client has already disconnected. In such a case, the next() call could abort processing, given any further work is wasted. I imagine coprocessor endpoints, etc, could make good use of this as well.
25. **summary:** Add ability for potentially long-running IPC calls to abort if client disconnects
    description: We recently had a cluster issue where a user was submitting scanners with a very restrictive filter, and then calling next() with a high scanner caching value. The clients would generally time out the next() call and disconnect, but the IPC kept running looking to fill the requested number of rows. Since this was in the context of MR, the tasks making the calls would retry, and the retries wuld be more likely to time out due to contention with the previous still-running scanner next() call. Eventually, the system spiraled out of control. We should add a hook to the IPC system so that RPC calls can check if the client has already disconnected. In such a case, the next() call could abort processing, given any further work is wasted. I imagine coprocessor endpoints, etc, could make good use of this as well.
26. **summary:** Add ability for potentially long-running IPC calls to abort if client disconnects
    description: We recently had a cluster issue where a user was submitting scanners with a very restrictive filter, and then calling next() with a high scanner caching value. The clients would generally time out the next() call and disconnect, but the IPC kept running looking to fill the requested number of rows. Since this was in the context of MR, the tasks making the calls would retry, and the retries wuld be more likely to time out due to contention with the previous still-running scanner next() call. Eventually, the system spiraled out of control. We should add a hook to the IPC system so that RPC calls can check if the client has already disconnected. In such a case, the next() call could abort processing, given any further work is wasted. I imagine coprocessor endpoints, etc, could make good use of this as well.
27. **summary:** Add ability for potentially long-running IPC calls to abort if client disconnects
    description: We recently had a cluster issue where a user was submitting scanners with a very restrictive filter, and then calling next() with a high scanner caching value. The clients would generally time out the next() call and disconnect, but the IPC kept running looking to fill the requested number of rows. Since this was in the context of MR, the tasks making the calls would retry, and the retries wuld be more likely to time out due to contention with the previous still-running scanner next() call. Eventually, the system spiraled out of control. We should add a hook to the IPC system so that RPC calls can check if the client has already disconnected. In such a case, the next() call could abort processing, given any further work is wasted. I imagine coprocessor endpoints, etc, could make good use of this as well.
28. **summary:** Add ability for potentially long-running IPC calls to abort if client disconnects
    description: We recently had a cluster issue where a user was submitting scanners with a very restrictive filter, and then calling next() with a high scanner caching value. The clients would generally time out the next() call and disconnect, but the IPC kept running looking to fill the requested number of rows. Since this was in the context of MR, the tasks making the calls would retry, and the retries wuld be more likely to time out due to contention with the previous still-running scanner next() call. Eventually, the system spiraled out of control. We should add a hook to the IPC system so that RPC calls can check if the client has already disconnected. In such a case, the next() call could abort processing, given any further work is wasted. I imagine coprocessor endpoints, etc, could make good use of this as well.
    label: test
29. **summary:** Add ability for potentially long-running IPC calls to abort if client disconnects
    description: We recently had a cluster issue where a user was submitting scanners with a very restrictive filter, and then calling next() with a high scanner caching value. The clients would generally time out the next() call and disconnect, but the IPC kept running looking to fill the requested number of rows. Since this was in the context of MR, the tasks making the calls would retry, and the retries wuld be more likely to time out due to contention with the previous still-running scanner next() call. Eventually, the system spiraled out of control. We should add a hook to the IPC system so that RPC calls can check if the client has already disconnected. In such a case, the next() call could abort processing, given any further work is wasted. I imagine coprocessor endpoints, etc, could make good use of this as well.
30. **summary:** Add ability for potentially long-running IPC calls to abort if client disconnects
    description: We recently had a cluster issue where a user was submitting scanners with a very restrictive filter, and then calling next() with a high scanner caching value. The clients would generally time out the

next() call and disconnect, but the IPC kept running looking to fill the requested number of rows. Since this was in the context of MR, the tasks making the calls would retry, and the retries wuld be more likely to time out due to contention with the previous still-running scanner next() call. Eventually, the system spiraled out of control. We should add a hook to the IPC system so that RPC calls can check if the client has already disconnected. In such a case, the next() call could abort processing, given any further work is wasted. I imagine coprocessor endpoints, etc, could make good use of this as well.
label: code-design

31. **summary:** Add ability for potentially long-running IPC calls to abort if client disconnects
description: We recently had a cluster issue where a user was submitting scanners with a very restrictive filter, and then calling next() with a high scanner caching value. The clients would generally time out the next() call and disconnect, but the IPC kept running looking to fill the requested number of rows. Since this was in the context of MR, the tasks making the calls would retry, and the retries wuld be more likely to time out due to contention with the previous still-running scanner next() call. Eventually, the system spiraled out of control. We should add a hook to the IPC system so that RPC calls can check if the client has already disconnected. In such a case, the next() call could abort processing, given any further work is wasted. I imagine coprocessor endpoints, etc, could make good use of this as well.

jira_issues_comments:

1. **body:** Attached patch implements the suggested idea, and hooks it up for scanner.next(). I spent 2.5 hours trying to write a test case for it, but we have so many layers of byzantine caching going on above the IPC sockets that I couldn't figure out how to make a client IPC connection actually hard-disconnect. So I tested it from the shell. here's the manual test plan: 1) create a table with 100 or so rows 2) issue following from shell: {code} HBase Shell; enter 'help<RETURN>' for list of supported commands. Type "exit<RETURN>" to leave the HBase Shell Version 0.95-SNAPSHOT, r5c65cc4a19fbc00876a365b10e98142238dc9a97, Wed May 9 13:06:25 PDT 2012 hbase(main):001:0> import org.apache.hadoop.hbase.filter.TestFilter => Java::OrgApacheHadoopHbaseFilter::TestFilter hbase(main):002:0> scan 't1', { FILTER => TestFilter::SlowScanFilter.new(), CACHE => 50 } ROW COLUMN+CELL {code} (shell will hang here) On the server side, you should see: {code} 12/05/09 15:03:29 INFO filter.TestFilter: Handler thread Thread[IPC Server handler 0 on 58364,5,main] sleeping in filter... 12/05/09 15:03:30 INFO filter.TestFilter: Handler thread Thread[IPC Server handler 0 on 58364,5,main] sleeping in filter... 12/05/09 15:03:31 INFO filter.TestFilter: Handler thread Thread[IPC Server handler 0 on 58364,5,main] sleeping in filter... 12/05/09 15:03:32 INFO filter.TestFilter: Handler thread Thread[IPC Server handler 0 on 58364,5,main] sleeping in filter... {code} Now ^C the shell. You should see on the server: {code} 12/05/09 15:03:33 ERROR regionserver.RegionServer: org.apache.hadoop.hbase.ipc.CallerDisconnectedException: Aborting call scan(null, scannerId: 4581116627867187291 numberOfRows: 50 closeScanner: false), rpc version=1, client version=1, methodsFingerPrint=-944626147 from 127.0.0.1:55648 after 5009 ms, since caller disconnected at org.apache.hadoop.hbase.ipc.HBaseServer$Call.throwExceptionIfCallerDisconnected(HBaseServer.java:417) at org.apache.hadoop.hbase.regionserver.HRegion$RegionScannerImpl.nextInternal(HRegion.java:3433) at org.apache.hadoop.hbase.regionserver.HRegion$RegionScannerImpl.next(HRegion.java:3391) at org.apache.hadoop.hbase.regionserver.HRegion$RegionScannerImpl.next(HRegion.java:3415) at org.apache.hadoop.hbase.regionserver.RegionServer.scan(RegionServer.java:828) at sun.reflect.GeneratedMethodAccessor26.invoke(Unknown Source) at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:25) at java.lang.reflect.Method.invoke(Method.java:597) at org.apache.hadoop.hbase.ipc.WritableRpcEngine$Server.call(WritableRpcEngine.java:358) at org.apache.hadoop.hbase.ipc.HBaseServer$Handler.run(HBaseServer.java:1387) 12/05/09 15:03:33 WARN ipc.HBaseServer: IPC Server Responder, call scan(null, scannerId: 4581116627867187291 numberOfRows: 50 closeScanner: false), rpc version=1, client version=1, methodsFingerPrint=-944626147 from 127.0.0.1:55648: output error 12/05/09 15:03:33 WARN ipc.HBaseServer: IPC Server handler 0 on 58364 caught a ClosedChannelException, this means that the server was processing a request but the client went away. The error message was: null {code} We could probably improve the messaging slightly, but this is at least an improvement in that the thread doesn't continue to get hung up indefinitely.
label: test
2. Nice feature. {code} + public boolean isCallerConnected() { {code} Is the above method used anywhere ? Can throwExceptionIfCallerDisconnected() be added to Delayable ? There is only one implementation of Delayable: {code} protected class Call implements Delayable { src/main/java/org/apache/hadoop/hbase/ipc/HBaseServer.java {code}

3. **body:** bq. Is the above method used anywhere ? No, good point. It was used in an earlier rev but I took it out. bq. Can throwExceptionIfCallerDisconnected() be added to Delayable ? There is only one implementation of Delayable: It doesn't make sense inside an interface called Delayable. We could get rid of Delayable entirely (I considered that) but decided to keep it separate since it's a nice general interface with no tying to IPC currently.
   label: code-design
4. **body:** I agree with the above analysis. {code} +public interface RpcCallContext extends Delayable { {code} The new interface is tied to IPC. How about naming this interface DelayableRpcCall ?
   label: code-design
5. **body:** The only things that make me nervous here are: - does this hurt performance for the average case? we're now accessing a synchronized variable inside the scanner loop where we weren't before. I think it's in the noise, though, given it's uncontended. - is the layering violation too ugly? reaching out to IPC frm within RegionScanner is not exactly clean... should we introduce an interface? Any thoughts on the above?
   label: code-design
6. bq. The new interface is tied to IPC. How about naming this interface DelayableRpcCall ? Well, if all RpcCalls are delayable, then why not just make it RpcCall? But I think the point is that there are some things which are Delayable which aren't RpcCalls, in theory. I don' want to get bogged down in that discussion, though, so if you feel strongly, I'll change it. Otherwise let's leave it as is in the patch.
7. btw (and sorry for the many separate comments): I was thinking that RpcCallContext is also a useful interface should we want to add tracing, etc. I've been thinking more and more that cross-machine tracing is worth doing a basic implementation of (see cloudtrace in accumulo)
8. RpcCallContext is an interface. I feel letting RegionScanner access RpcCallContext is Okay.
9. **body:** Attached patch removes the unused method that Ted noticed. I checked for a performance degradation by running the following in shell: 4.times { count 'usertable', INTERVAL => 100000, CACHE => 1000 } on a 4-million row table generated by ycsb, after major compaction. I measured the elapsed User CPU on the server, and the wall clock on the client. with patch: client side: 149.8 wall clock server side: 1m52.03 user table without patch: client side: 153.3 wall clock server side: 1m51.4 user time These results seem to indicate there's no appreciable performance difference - just noise. Makes sense since ThreadLocal lookups and uncontended synchronization are both very cheap.
   label: code-design
10. -1 overall. Here are the results of testing the latest attachment http://issues.apache.org/jira/secure/attachment/12526306/hbase-5973.txt against trunk revision . +1 @author. The patch does not contain any @author tags. +1 tests included. The patch appears to include 6 new or modified tests. +1 hadoop23. The patch compiles against the hadoop 0.23.x profile. +1 javadoc. The javadoc tool did not generate any warning messages. +1 javac. The applied patch does not increase the total number of javac compiler warnings. +1 findbugs. The patch does not introduce any new Findbugs (version 1.3.9) warnings. +1 release audit. The applied patch does not increase the total number of release audit warnings. -1 core tests. The patch failed these unit tests: org.apache.hadoop.hbase.TestDrainingServer Test results: https://builds.apache.org/job/PreCommit-HBASE-Build/1828//testReport/ Findbugs warnings: https://builds.apache.org/job/PreCommit-HBASE-Build/1828//artifact/trunk/patchprocess/newPatchFindbugsWarnings.html Console output: https://builds.apache.org/job/PreCommit-HBASE-Build/1828//console This message is automatically generated.
11. Ted: I noticed you marked this "Reviewed". Is that a +1? :)
12. +1 from me. Minor comments below: For RpcCallContext.java and CallerDisconnectedException.java: {code} + * Copyright 2010 The Apache Software Foundation {code} Please remove year. Please provide patches for 0.92 branch, etc.
13. Attached patch fixes the copyright. I'll commit this to the branches where it applies, and leave it open. Dave Wang is working on backports for some earlier branches as well.
14. Attached patch is for 0.94. Had to resolve a trivial conflict since nextInternal() doesn't have a Metric parameter in 0.94.
15. -1 overall. Here are the results of testing the latest attachment http://issues.apache.org/jira/secure/attachment/12526370/hbase-5973-0.94.txt against trunk revision . +1 @author. The patch does not contain any @author tags. +1 tests included. The patch appears to include 6 new or modified tests. -1 patch. The patch command could not apply the patch. Console output: https://builds.apache.org/job/PreCommit-HBASE-Build/1831//console This message is automatically generated.
16. Woops, I forgot to svn add two files in my previous 0.94 commit/upload. Fixed in SVN, here's the proper patch.
17. Here's the patch for 0.92. Again pretty easy backport.

18. Committed to trunk, 94, 92. Dave is working on the 0.90 patch.
19. -1 overall. Here are the results of testing the latest attachment http://issues.apache.org/jira/secure/attachment/12526374/hbase-5973-0.92.txt against trunk revision . +1 @author. The patch does not contain any @author tags. +1 tests included. The patch appears to include 6 new or modified tests. -1 patch. The patch command could not apply the patch. Console output: https://builds.apache.org/job/PreCommit-HBASE-Build/1832//console This message is automatically generated.
20. Integrated in HBase-TRUNK #2860 (See [https://builds.apache.org/job/HBase-TRUNK/2860/]) HBASE-5973. Add ability for potentially long-running IPC calls to abort if client disconnects. Contributed by Todd Lipcon. (Revision 1336787) Result = FAILURE todd : Files : * /hbase/trunk/src/main/java/org/apache/hadoop/hbase/ipc/CallerDisconnectedException.java * /hbase/trunk/src/main/java/org/apache/hadoop/hbase/ipc/HBaseServer.java * /hbase/trunk/src/main/java/org/apache/hadoop/hbase/ipc/RpcCallContext.java * /hbase/trunk/src/main/java/org/apache/hadoop/hbase/ipc/RpcServer.java * /hbase/trunk/src/main/java/org/apache/hadoop/hbase/regionserver/HRegion.java * /hbase/trunk/src/main/ruby/hbase/table.rb * /hbase/trunk/src/main/ruby/shell/commands/scan.rb * /hbase/trunk/src/test/java/org/apache/hadoop/hbase/filter/TestFilter.java * /hbase/trunk/src/test/java/org/apache/hadoop/hbase/ipc/TestDelayedRpc.java
21. Integrated in HBase-0.94 #186 (See [https://builds.apache.org/job/HBase-0.94/186/]) Amend HBASE-5973. Add two missing files from previous commit (Revision 1336797) HBASE-5973. Add ability for potentially long-running IPC calls to abort if client disconnects. Contributed by Todd Lipcon. (Revision 1336788) Result = FAILURE todd : Files : * /hbase/branches/0.94/src/main/java/org/apache/hadoop/hbase/ipc/CallerDisconnectedException.java * /hbase/branches/0.94/src/main/java/org/apache/hadoop/hbase/ipc/RpcCallContext.java todd : Files : * /hbase/branches/0.94/src/main/java/org/apache/hadoop/hbase/ipc/HBaseServer.java * /hbase/branches/0.94/src/main/java/org/apache/hadoop/hbase/ipc/RpcServer.java * /hbase/branches/0.94/src/main/java/org/apache/hadoop/hbase/regionserver/HRegion.java * /hbase/branches/0.94/src/main/ruby/hbase/table.rb * /hbase/branches/0.94/src/main/ruby/shell/commands/scan.rb * /hbase/branches/0.94/src/test/java/org/apache/hadoop/hbase/filter/TestFilter.java * /hbase/branches/0.94/src/test/java/org/apache/hadoop/hbase/ipc/TestDelayedRpc.java
22. Integrated in HBase-0.92 #403 (See [https://builds.apache.org/job/HBase-0.92/403/]) HBASE-5973. Add ability for potentially long-running IPC calls to abort if client disconnects. Contributed by Todd Lipcon. (Revision 1336798) Result = FAILURE todd : Files : * /hbase/branches/0.92/CHANGES.txt * /hbase/branches/0.92/src/main/java/org/apache/hadoop/hbase/ipc/CallerDisconnectedException.java * /hbase/branches/0.92/src/main/java/org/apache/hadoop/hbase/ipc/HBaseServer.java * /hbase/branches/0.92/src/main/java/org/apache/hadoop/hbase/ipc/RpcCallContext.java * /hbase/branches/0.92/src/main/java/org/apache/hadoop/hbase/ipc/RpcServer.java * /hbase/branches/0.92/src/main/java/org/apache/hadoop/hbase/regionserver/HRegion.java * /hbase/branches/0.92/src/main/ruby/hbase/table.rb * /hbase/branches/0.92/src/main/ruby/shell/commands/scan.rb * /hbase/branches/0.92/src/test/java/org/apache/hadoop/hbase/filter/TestFilter.java * /hbase/branches/0.92/src/test/java/org/apache/hadoop/hbase/ipc/TestDelayedRpc.java
23. Might Todd. When we call this: {code} private boolean nextInternal(int limit, String metric) throws IOException { + RpcCallContext rpcCall = HBaseServer.getCurrentCall(); while (true) { + if (rpcCall != null) { + // If a user specifies a too-restrictive or too-slow scanner, the + // client might time out and disconnect while the server side + // is still processing the request. We should abort aggressively + // in that case. + rpcCall.throwExceptionIfCallerDisconnected(); + } {code} ... if connection is closed when we check, the exception does not come out here and abort this current nextInternal invocation? Rather, it comes out on the stuck handler? Does it interrupt the ongoing call, the one w/o a client? Pardon dumb question. Just trying to understand how this fix works.
24. Sorry, Stack.. I don't follow your question. If getCurrentCall() is non-null, that implies that we are within one of the handler threads. So, throwing the exception here will abort the next() call in that same handler.
25. @Todd You answered my question (that and looking at code). I see how this works now and how it can be a kill-switch particularly for the case where heavy filtering has us passing on lots of rows. +1 on the patch.
26. Integrated in HBase-TRUNK-security #198 (See [https://builds.apache.org/job/HBase-TRUNK-security/198/]) HBASE-5973. Add ability for potentially long-running IPC calls to abort if client disconnects. Contributed by Todd Lipcon. (Revision 1336787) Result = FAILURE todd : Files : * /hbase/trunk/src/main/java/org/apache/hadoop/hbase/ipc/CallerDisconnectedException.java * /hbase/trunk/src/main/java/org/apache/hadoop/hbase/ipc/HBaseServer.java *

/hbase/trunk/src/main/java/org/apache/hadoop/hbase/ipc/RpcCallContext.java * /hbase/trunk/src/main/java/org/apache/hadoop/hbase/ipc/RpcServer.java * /hbase/trunk/src/main/java/org/apache/hadoop/hbase/regionserver/HRegion.java * /hbase/trunk/src/main/ruby/hbase/table.rb * /hbase/trunk/src/main/ruby/shell/commands/scan.rb * /hbase/trunk/src/test/java/org/apache/hadoop/hbase/filter/TestFilter.java * /hbase/trunk/src/test/java/org/apache/hadoop/hbase/ipc/TestDelayedRpc.java

27. Passed unit tests locally.
28. -1 overall. Here are the results of testing the latest attachment http://issues.apache.org/jira/secure/attachment/12526533/HBASE-5973-0.90.txt against trunk revision . +1 @author. The patch does not contain any @author tags. +1 tests included. The patch appears to include 3 new or modified tests. -1 patch. The patch command could not apply the patch. Console output: https://builds.apache.org/job/PreCommit-HBASE-Build/1847//console This message is automatically generated.
29. +1. Patch looks good to me. I'll commit to the 0.90 branch later today if there are no objections to backporting this.
30. @David: Can you publish the results from running test suite on 0.90 ? Such as cases tested, total time, etc. Thanks
31. +1 on backport to 0.90.
32. **body:** Sure. This is what I saw on my local box: Tests in error: Tests run: 723, Failures: 0, Errors: 1, Skipped: 9 [INFO] Total time: 1:31:50.066s Only test that failed was: Running org.apache.hadoop.hbase.master.TestRestartCluster Tests run: 2, Failures: 0, Errors: 1, Skipped: 0, Time elapsed: 45.891 sec <<< FAILURE! Test seems flaky as it passed just fine when run individually. **label:** test
33. Committed to 0.90 as well. Thanks, Dave, for the backport.
34. Integrated in HBase-0.92-security #107 (See [https://builds.apache.org/job/HBase-0.92-security/107/]) HBASE-5973. Add ability for potentially long-running IPC calls to abort if client disconnects. Contributed by Todd Lipcon. (Revision 1336798) Result = FAILURE todd : Files : * /hbase/branches/0.92/CHANGES.txt * /hbase/branches/0.92/src/main/java/org/apache/hadoop/hbase/ipc/CallerDisconnectedException.java * /hbase/branches/0.92/src/main/java/org/apache/hadoop/hbase/ipc/HBaseServer.java * /hbase/branches/0.92/src/main/java/org/apache/hadoop/hbase/ipc/RpcCallContext.java * /hbase/branches/0.92/src/main/java/org/apache/hadoop/hbase/ipc/RpcServer.java * /hbase/branches/0.92/src/main/java/org/apache/hadoop/hbase/regionserver/HRegion.java * /hbase/branches/0.92/src/main/ruby/hbase/table.rb * /hbase/branches/0.92/src/main/ruby/shell/commands/scan.rb * /hbase/branches/0.92/src/test/java/org/apache/hadoop/hbase/filter/TestFilter.java * /hbase/branches/0.92/src/test/java/org/apache/hadoop/hbase/ipc/TestDelayedRpc.java
35. Integrated in HBase-0.94-security #27 (See [https://builds.apache.org/job/HBase-0.94-security/27/]) Amend HBASE-5973. Add two missing files from previous commit (Revision 1336797) HBASE-5973. Add ability for potentially long-running IPC calls to abort if client disconnects. Contributed by Todd Lipcon. (Revision 1336788) Result = SUCCESS todd : Files : * /hbase/branches/0.94/src/main/java/org/apache/hadoop/hbase/ipc/CallerDisconnectedException.java * /hbase/branches/0.94/src/main/java/org/apache/hadoop/hbase/ipc/RpcCallContext.java todd : Files : * /hbase/branches/0.94/src/main/java/org/apache/hadoop/hbase/ipc/HBaseServer.java * /hbase/branches/0.94/src/main/java/org/apache/hadoop/hbase/ipc/RpcServer.java * /hbase/branches/0.94/src/main/java/org/apache/hadoop/hbase/regionserver/HRegion.java * /hbase/branches/0.94/src/main/ruby/hbase/table.rb * /hbase/branches/0.94/src/main/ruby/shell/commands/scan.rb * /hbase/branches/0.94/src/test/java/org/apache/hadoop/hbase/filter/TestFilter.java * /hbase/branches/0.94/src/test/java/org/apache/hadoop/hbase/ipc/TestDelayedRpc.java