Item 22
git_comments:

git_commits:

1. **summary:** HBASE-10449 Wrong execution pool configuration in HConnectionManager
   message: HBASE-10449 Wrong execution pool configuration in HConnectionManager git-svn-id: https://svn.apache.org/repos/asf/hbase/trunk@1563878 13f79535-47bb-0310-9956-ffa450edef68

github_issues:

github_issues_comments:

github_pulls:

github_pulls_comments:

github_pulls_reviews:

jira_issues:

1. **summary:** Wrong execution pool configuration in HConnectionManager
   description: There is a confusion in the configuration of the pool. The attached patch fixes this. This may change the client performances, as we were using a single thread.
2. **summary:** Wrong execution pool configuration in HConnectionManager
   description: There is a confusion in the configuration of the pool. The attached patch fixes this. This may change the client performances, as we were using a single thread.
   label: code-design
3. **summary:** Wrong execution pool configuration in HConnectionManager
   description: There is a confusion in the configuration of the pool. The attached patch fixes this. This may change the client performances, as we were using a single thread.
   label: code-design
4. **summary:** Wrong execution pool configuration in HConnectionManager
   description: There is a confusion in the configuration of the pool. The attached patch fixes this. This may change the client performances, as we were using a single thread.
5. **summary:** Wrong execution pool configuration in HConnectionManager
   description: There is a confusion in the configuration of the pool. The attached patch fixes this. This may change the client performances, as we were using a single thread.
6. **summary:** Wrong execution pool configuration in HConnectionManager
   description: There is a confusion in the configuration of the pool. The attached patch fixes this. This may change the client performances, as we were using a single thread.
7. **summary:** Wrong execution pool configuration in HConnectionManager
   description: There is a confusion in the configuration of the pool. The attached patch fixes this. This may change the client performances, as we were using a single thread.
   label: code-design
8. **summary:** Wrong execution pool configuration in HConnectionManager
   description: There is a confusion in the configuration of the pool. The attached patch fixes this. This may change the client performances, as we were using a single thread.
9. **summary:** Wrong execution pool configuration in HConnectionManager
   description: There is a confusion in the configuration of the pool. The attached patch fixes this. This may change the client performances, as we were using a single thread.
10. **summary:** Wrong execution pool configuration in HConnectionManager
    description: There is a confusion in the configuration of the pool. The attached patch fixes this. This may change the client performances, as we were using a single thread.
    label: code-design
11. **summary:** Wrong execution pool configuration in HConnectionManager
    description: There is a confusion in the configuration of the pool. The attached patch fixes this. This may change the client performances, as we were using a single thread.
12. **summary:** Wrong execution pool configuration in HConnectionManager
    description: There is a confusion in the configuration of the pool. The attached patch fixes this. This may change the client performances, as we were using a single thread.
13. **summary:** Wrong execution pool configuration in HConnectionManager
    description: There is a confusion in the configuration of the pool. The attached patch fixes this. This may change the client performances, as we were using a single thread.
14. **summary:** Wrong execution pool configuration in HConnectionManager
    description: There is a confusion in the configuration of the pool. The attached patch fixes this. This may change the client performances, as we were using a single thread.
15. **summary:** Wrong execution pool configuration in HConnectionManager
    description: There is a confusion in the configuration of the pool. The attached patch fixes this. This may change the client performances, as we were using a single thread.

16. **summary:** Wrong execution pool configuration in HConnectionManager
    description: There is a confusion in the configuration of the pool. The attached patch fixes this. This may change the client performances, as we were using a single thread.
17. **summary:** Wrong execution pool configuration in HConnectionManager
    description: There is a confusion in the configuration of the pool. The attached patch fixes this. This may change the client performances, as we were using a single thread.
18. **summary:** Wrong execution pool configuration in HConnectionManager
    description: There is a confusion in the configuration of the pool. The attached patch fixes this. This may change the client performances, as we were using a single thread.
    label: code-design
19. **summary:** Wrong execution pool configuration in HConnectionManager
    description: There is a confusion in the configuration of the pool. The attached patch fixes this. This may change the client performances, as we were using a single thread.
    label: code-design
20. **summary:** Wrong execution pool configuration in HConnectionManager
    description: There is a confusion in the configuration of the pool. The attached patch fixes this. This may change the client performances, as we were using a single thread.
21. **summary:** Wrong execution pool configuration in HConnectionManager
    description: There is a confusion in the configuration of the pool. The attached patch fixes this. This may change the client performances, as we were using a single thread.
    label: code-design
22. **summary:** Wrong execution pool configuration in HConnectionManager
    description: There is a confusion in the configuration of the pool. The attached patch fixes this. This may change the client performances, as we were using a single thread.
23. **summary:** Wrong execution pool configuration in HConnectionManager
    description: There is a confusion in the configuration of the pool. The attached patch fixes this. This may change the client performances, as we were using a single thread.
    label: code-design
24. **summary:** Wrong execution pool configuration in HConnectionManager
    description: There is a confusion in the configuration of the pool. The attached patch fixes this. This may change the client performances, as we were using a single thread.
25. **summary:** Wrong execution pool configuration in HConnectionManager
    description: There is a confusion in the configuration of the pool. The attached patch fixes this. This may change the client performances, as we were using a single thread.
26. **summary:** Wrong execution pool configuration in HConnectionManager
    description: There is a confusion in the configuration of the pool. The attached patch fixes this. This may change the client performances, as we were using a single thread.
27. **summary:** Wrong execution pool configuration in HConnectionManager
    description: There is a confusion in the configuration of the pool. The attached patch fixes this. This may change the client performances, as we were using a single thread.
    label: code-design
28. **summary:** Wrong execution pool configuration in HConnectionManager
    description: There is a confusion in the configuration of the pool. The attached patch fixes this. This may change the client performances, as we were using a single thread.

jira_issues_comments:

1. Ouch. Is it better to leave the default value of hbase.hconnection.threads.core at 0 and let your new default logic kick in?
2. **body:** It's something new (~3 months old, not in .94), so imho it's better to come back to the initial behavior, as the performances should be better. We can also play if safe, with a different patch for the .96 and for trunk. This makes things more complicated to understand & test however...
   label: code-design
3. **body:** {color:red}-1 overall{color}. Here are the results of testing the latest attachment http://issues.apache.org/jira/secure/attachment/12626350/HBASE-10449.v1.patch against trunk revision . ATTACHMENT ID: 12626350 {color:green}+1 @author{color}. The patch does not contain any @author tags. {color:red}-1 tests included{color}. The patch doesn't appear to include any new or modified tests. Please justify why no new tests are needed for this patch. Also please list what manual steps were performed to verify this patch. {color:green}+1 hadoop1.0{color}. The patch compiles against the hadoop 1.0 profile. {color:green}+1 hadoop1.1{color}. The patch compiles against the hadoop 1.1 profile. {color:green}+1 javadoc{color}. The javadoc tool did not generate any warning messages. {color:green}+1 javac{color}. The applied patch does not increase the total number of javac compiler warnings. {color:green}+1 findbugs{color}. The patch does not introduce any new Findbugs (version 1.3.9) warnings. {color:green}+1 release audit{color}. The applied patch does not increase the total number of release audit warnings. {color:green}+1 lineLengths{color}. The patch does not introduce lines longer than 100 {color:green}+1 site{color}. The mvn site goal succeeds with this patch. {color:red}-1 core tests{color}. The patch failed these unit tests: {color:red}-1 core zombie tests{color}. There are 1 zombie test(s): at org.apache.hadoop.hbase.TestAcidGuarantees.testGetAtomicity(TestAcidGuarantees.java:331) Test results: https://builds.apache.org/job/PreCommit-HBASE-Build/8568//testReport/ Findbugs warnings: https://builds.apache.org/job/PreCommit-HBASE-Build/8568//artifact/trunk/patchprocess/newPatchFindbugsWarningshbase-protocol.html Findbugs warnings: https://builds.apache.org/job/PreCommit-HBASE-

Build/8568//artifact/trunk/patchprocess/newPatchFindbugsWarningshbase-thrift.html Findbugs warnings: https://builds.apache.org/job/PreCommit-HBASE-Build/8568//artifact/trunk/patchprocess/newPatchFindbugsWarningshbase-client.html Findbugs warnings: https://builds.apache.org/job/PreCommit-HBASE-Build/8568//artifact/trunk/patchprocess/newPatchFindbugsWarningshbase-hadoop2-compat.html Findbugs warnings: https://builds.apache.org/job/PreCommit-HBASE-Build/8568//artifact/trunk/patchprocess/newPatchFindbugsWarningshbase-examples.html Findbugs warnings: https://builds.apache.org/job/PreCommit-HBASE-Build/8568//artifact/trunk/patchprocess/newPatchFindbugsWarningshbase-prefix-tree.html Findbugs warnings: https://builds.apache.org/job/PreCommit-HBASE-Build/8568//artifact/trunk/patchprocess/newPatchFindbugsWarningshbase-common.html Findbugs warnings: https://builds.apache.org/job/PreCommit-HBASE-Build/8568//artifact/trunk/patchprocess/newPatchFindbugsWarningshbase-server.html Findbugs warnings: https://builds.apache.org/job/PreCommit-HBASE-Build/8568//artifact/trunk/patchprocess/newPatchFindbugsWarningshbase-hadoop-compat.html Console output: https://builds.apache.org/job/PreCommit-HBASE-Build/8568//console This message is automatically generated.
label: code-design

4. TestAcidGuarantees is likely unrelated. I will run it locally a few times to be sure. [~ndimiduk], [~saint.ack@gmail.com], [~apurtell], what do you think? Should I commit the patch to trunk,.96 & .98 as it is, if TestAcidGuarantees is proven to be unrelated?
5. +1
6. Committed to trunk/.98/.96 (this way it will be in the next .98 RC) Stack; Nick, if you prefer something different for trunk / .96 I can revert/change the patch. But it seems simpler to have the same logic for all versions.
7. SUCCESS: Integrated in HBase-TRUNK #4875 (See [https://builds.apache.org/job/HBase-TRUNK/4875/]) HBASE-10449 Wrong execution pool configuration in HConnectionManager (nkeywal: rev 1563878) * /hbase/trunk/hbase-client/src/main/java/org/apache/hadoop/hbase/client/HConnectionManager.java
8. SUCCESS: Integrated in hbase-0.96 #277 (See [https://builds.apache.org/job/hbase-0.96/277/]) HBASE-10449 Wrong execution pool configuration in HConnectionManager (nkeywal: rev 1563879) * /hbase/branches/0.96/hbase-client/src/main/java/org/apache/hadoop/hbase/client/HConnectionManager.java
9. SUCCESS: Integrated in HBase-0.98-on-Hadoop-1.1 #113 (See [https://builds.apache.org/job/HBase-0.98-on-Hadoop-1.1/113/]) HBASE-10449 Wrong execution pool configuration in HConnectionManager (nkeywal: rev 1563880) * /hbase/branches/0.98/hbase-client/src/main/java/org/apache/hadoop/hbase/client/HConnectionManager.java
10. **body:** Thank [~nkeywal] To be clear, default was 8 * cores rather than agreed upon 256?
    label: code-design
11. [~nkeywal] I have no objection to the change in default as I have no evidence to argue for one value or another. I do find it strange that this default changes as part of a point release on 0.96. [~stack]: it appears the default for hbase.hconnection.threads.core on 0.96, 0.98, and trunk is now 256. An administrator can attain the 8 * cores behavior by setting this configuration value to 0.
12. [~ndimiduk] You have a point but a release note should be cover enough for a change few if any will notice IMO.
13. FAILURE: Integrated in HBase-TRUNK-on-Hadoop-1.1 #76 (See [https://builds.apache.org/job/HBase-TRUNK-on-Hadoop-1.1/76/]) HBASE-10449 Wrong execution pool configuration in HConnectionManager (nkeywal: rev 1563878) * /hbase/trunk/hbase-client/src/main/java/org/apache/hadoop/hbase/client/HConnectionManager.java
14. FAILURE: Integrated in HBase-0.98 #123 (See [https://builds.apache.org/job/HBase-0.98/123/]) HBASE-10449 Wrong execution pool configuration in HConnectionManager (nkeywal: rev 1563880) * /hbase/branches/0.98/hbase-client/src/main/java/org/apache/hadoop/hbase/client/HConnectionManager.java
15. SUCCESS: Integrated in hbase-0.96-hadoop2 #191 (See [https://builds.apache.org/job/hbase-0.96-hadoop2/191/]) HBASE-10449 Wrong execution pool configuration in HConnectionManager (nkeywal: rev 1563879) * /hbase/branches/0.96/hbase-client/src/main/java/org/apache/hadoop/hbase/client/HConnectionManager.java
16. **body:** Note that we're not exactly changing a default. The code wanted to do: {panel}Create up to 'max' (default: 256) threads. Expires them if they are not used for 10 seconds, excepted for 'core' (default 0) of them. If there is more than 'max' tasks, queue them.{panel} Actually it was doing: {panel}Create a single thread, queue all the tasks for this thread.{panel} So the patch actually implements that was supposed to be implemented (or tries to implement it at least :-)). I Moreover, it's a regression from HBASE-9917, so actually 96.0 really uses 256 threads. It's a *96.1* issue only. But yes, it does have an impact on performances, and this impact can be good or bad. That's why I would like it to be in the .98 RC, and also why I think it's simpler to have the same defaults on all versions. Lastly, and unrelated, we didn't have a limit of the number of threads before the .96. I'm wondering if we don't have an impact if a server hangs. The client may ends up with all its connections stuck to this server, until it timeouts.
    label: code-design
17. Closing this issue after 0.99.0 release.
18. Back again [~nkeywal] I'm looking at failing tests and see thread dumps with pools of 256 threads per client instance. Where does 'Create a single thread, queue all the tasks for this thread.' come from? Our 'core' setting is same as our 'max' so we will keep spinning new threads until we hit max whether all of the other 255 are idle or not. That seems wrong (it is for sure a PITA looking at thread dumps of 256 threads doing nought. I opened new issue HBASE-14433 for discussion.
19. Sorry for the delay, I'm seeing this now only. Let me have a look.
20. > Where does 'Create a single thread, queue all the tasks for this thread.' come from? This is what HBASE-9917 actually implemented: with the ThreadPoolExecutor if the task queue is unbounded, it does not create new threads: From: http://docs.oracle.com/javase/7/docs/api/java/util/concurrent/ThreadPoolExecutor.html If fewer than corePoolSize threads are running, the Executor always prefers adding a new thread rather than queuing. If corePoolSize or more threads are running, the Executor always prefers queuing a request rather than adding a new thread. If a request cannot be queued, a new thread is created unless this would exceed maximumPoolSize, in which case, the task will be rejected. But having less than 256 threads is fine. This was just restoring the previous value.

21. Thanks [~nkeywal] Our queue is unbounded then so we do not create new threads once we hit core ? Rather, we just queue? Can we make queue size zero ? I suppose I should test....
22. As I understand the doc, if we do that we create maxThreads and then reject all the tasks. Not really useful. But the patch in HBASE-14433 seems ok: - we create up to core threads (Runtime.getRuntime().availableProcessors()). If we have 10 tasks in parallel we still have Runtime.getRuntime().availableProcessors() threads. - the expire quite quickly (because we do allowCoreThreadTimeOut(true);) May be we should set maxThreads to coreThreads as well and increase HConstants.DEFAULT_HBASE_CLIENT_MAX_TOTAL_TASKS. But I'm +1 with HBASE-14433 as it is now.
23. Thanks [~nkeywal] Let me commit HBASE-14433. Lets go with less threads till we do the test that proves we need more. Thanks for the review boss.
24. **body:** Actually I'm having two doubts: - the core threads should already have this timeout, no. We should not see 256 threads, because they should expire already - IIRC, this thread pool is used when connecting to the various regionserver, and they block until they have an answer. So with 4 core threads (for example), it means that if we do a multi we contact 4 servers simultaneously at most. The threads are not really using CPUs, they're waiting (old i/o style). BUt may be it has changed?
    label: code-design
25. **body:** What's happening for the expire is: - we have a 60s timeout with 256 seconds. - let's imagine we have 1 query per second. We will still have 60 threads, because each new request will create a new thread until we reach coreSize. As the timeout is 60s, the oldest threads will expire after 60s. I haven't double-checked, but I believe that the threads are needed because of the old i/o pattern. So we do need a max in the x00 range (it's like this since 0.90 at least. In theory, it's good for small cluster (100 nodes), but not as good if the cluster is composed of thousands of nodes) I did actually spent some time on this a year ago, in HBASE-11590. @stack, what do you think of the approach? I can finish the work I started there. But I will need a review. There are also some ideas/hacks in http://stackoverflow.com/questions/19528304/how-to-get-the-threadpoolexecutor-to-increase-threads-to-max-before-queueing/19528305#19528305 I haven't reviewed them yet.
    label: code-design
26. Thanks [~nkeywal] bq. We should not see 256 threads, because they should expire already Maybe they spin up inside the keepalive time of 60 seconds. bq. We will still have 60 threads, because each new request will create a new thread until we reach coreSize Well, I was thinking that we'd go to core size -- say # of cores -- and then if one request a second, we'd just stay at core size because there would be a free thread when the request-per-second came in (assuming request took a good deal < a second). Let me look at HBASE-11590. What I saw was each client with hundreds -- up to 256 on one -- threads all in WAITING like follows: {code} "hconnection-0x3065a6a9-shared--pool13-t247" daemon prio=10 tid=0x00007f31c1ab2000 nid=0x7718 waiting on condition [0x00007f2f9ecec000] java.lang.Thread.State: TIMED_WAITING (parking) at sun.misc.Unsafe.park(Native Method) - parking to wait for <0x00000007f841b388> (a java.util.concurrent.locks.AbstractQueuedSynchronizer$ConditionObject) at java.util.concurrent.locks.LockSupport.parkNanos(LockSupport.java:226) at java.util.concurrent.locks.AbstractQueuedSynchronizer$ConditionObject.awaitNanos(AbstractQueuedSynchronizer.java:2082) at java.util.concurrent.LinkedBlockingQueue.poll(LinkedBlockingQueue.java:467) at java.util.concurrent.ThreadPoolExecutor.getTask(ThreadPoolExecutor.java:1068) at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1130) at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:615) at java.lang.Thread.run(Thread.java:744) {code} ... usually in TestReplicasClient. Here is example: https://builds.apache.org/view/H-L/view/HBase/job/PreCommit-HBASE-Build/15581/consoleText See zombies on the end. I also have second thoughts on HBASE-114433. I am going to change it so we set config for tests only. We need to do more work before can set the core threads down from max is what I am thinking. Thanks [~nkeywal] I'll look at HBASE-11590. Didn't we have a mock server somewhere such that we could standup a client with no friction and watch it in operation? I thought we'd make such a beast....
27. **body:** > I was thinking that we'd go to core size – say # of cores – and then if one request a second, we'd just stay at core size because there would be a free thread when the request-per-second came in (assuming request took a good deal < a second). I expect that if we have more than coreSize calls in timeout (256 vs 60 seconds in our case) then we always have coreSize threads. > Didn't we have a mock server somewhere such that we could standup a client with no friction and watch it in operation? I thought we'd make such a beast.... Yep, you built one, we used it when we looked at the perf issues in the client (the protobuf nightmare if you remember ;:-)).
    label: code-design
28. bq. I expect that if we have more than coreSize calls in timeout (256 vs 60 seconds in our case) then we always have coreSize threads. Say again. I'm not following [~nkeywal] Thanks. bq. ...the protobuf nightmare if you remember Yes. Smile. Need to revive it for here and for doing client timeouts....
29. **body:** The algo for the ThreadPoolExecutor is: onNewTask(){ if (currentSize < coreSize) createNewThread() else reuseThread() } And there is a timeout for each thread. So if we do a coreSize of 2, a time of 20s, and a query every 15s, we have: 0s query1: create thread1, poolSize=1 15s query2: create thread2, poolSize=2 20s close thread1, poolSize=1 30s query3: create thread3, poolSize=2 35s: close thread2, poolSize=1 45s: query4: create thread4, poolSize=2 And so on. So even if we have 1 query each 15s, we have 2 threads in the pool nearly all the time. > Yes. Smile. Need to revive it for here and for doing client timeouts.... I found the code in TestClientNoCluster#run , ready to be reused! I think we need to go for a hack like in Stackoverflow or for a different implementation for TPE like HBASE-11590...
    label: code-design
30. That makes sense. What happens if query happens if query every second: i.e. so there are periods when we have more queries than coreSize? Do the > coreSize query go in queue or do we make new threads to handle them? If latter, good, if former bad. Let me look at other issue.
31. It's the former: in this case, the queries are queued. A new thread will be created only when the queue is full. Then, if we reach maxThreads and the queue is full the new tasks are rejected. In our case the queue is nearly unbounded, so we stay with corePoolSize.
32. Ok. Not what we want. Lets look at alternative...