Hadoop Common / HADOOP-312

Connections should not be cached

Details

Type:	Improvement	Status:	CLOSED
Priority:	Major	Resolution:	Fixed
Affects Version/s:	None	Fix Version/s:	0.6.0
Component/s:	ipc		
Labels:	None		

Description

Servers and clients (client include datanodes, tasktrackers, DFSClients & tasks) should not cache connections or maybe cache them for very short periods of time. Clients should set up & tear down connections to the servers everytime they need to contact the servers (including the heartbeats). If connection is cached, then reuse the existing connection for a few subsequent transactions until the connection expires. The heartbeat interval should be more so that many more clients (order of tens of thousands) can be accomodated within 1 heartbeat interval.

Attachments

control_conn_caching.patch	13 kB	06/Sep/06 20:25
no_conn_caching.patch	13 kB	01/Sep/06 19:08
no_conn_caching.patch	7 kB	29/Aug/06 20:05
no_conn_caching.patch	7 kB	29/Aug/06 20:00
no_conn_caching.patch	12 kB	22/Aug/06 12:45
no_conn_caching.patch	12 kB	19/Aug/06 18:15
no_conn_caching.patch	11 kB	18/Aug/06 05:45
no_connection_caching.patch	10 kB	07/Aug/06 18:40
no_connection_caching.patch	9 kB	26/Jul/06 09:30

Issue Links

is blocked by

HADOOP-362 tasks can get lost when reporting task completion to the JobTracker has an error — CLOSED

Activity

Devaraj Das created issue - 22/Jun/06 01:55

Devaraj Das added a comment - 30/Jun/06 01:06

The way I have designed this is to have the connection caching configurable. Given any IPC server (like the namenode or jobtracker) the caching can range from fully cached (which is the current behavior) to number of transactions per created connection. The configuration related to caching can be set in the client's hadoop config file like:
<property>
<name>ipc.connection.notcached.address</name>
<value>server1:port=num_transactions, server2:port=num_transactions</value>
<description>Defines the connections that should not be cached.
</description>
</property>
The above means that clients close the connections to server1:port after it has done num_transactions (a transaction is defined as

one request-response). Any number of such servers can be specified by a comma-separated list. Connections to servers not explicitly mentioned in the config is cached (though the server will disconnect clients that have not communicated with it for a specified period of time).
Doing this will limit the number of connections cached significantly. For example, one can always specify in the config that connections to the namenode should not be cached at all (num_transactions = 1) and so on.
The patch is ready but I can probably incorporate any inputs you may have before I submit that.

Devaraj Das added a comment - 01/Jul/06 01:55

Modified the datanode Heartbeat logic to allow the datanodes to choose a random heartbeat interval between specified limits (the range, after some calculations on the constants defined in FSconstants.java, is between 3 sec and 103 sec).

Devaraj Das made changes - 01/Jul/06 01:55

Field	Original Value	New Value
Attachment		config_connection_cache.patch [12336213]

Eric Baldeschwieler added a comment - 01/Jul/06 02:40

Could we break the datanode heartbeat thing you are doing into another bug? It seems unrelated to a redesign of our general IPC and this bug is not complete enough on the motivation of your work for me to understand why randomizing heartbeat intervals is a good idea.

Eric Baldeschwieler added a comment - 01/Jul/06 02:47

I'm very uncomfortable with this direction. Turning this into a per installation configuration challenge sounds like a very bad and unscalable idea.

-1

Devaraj Das added a comment - 04/Jul/06 12:40

Had a discussion with Eric/Owen on this subject and decided to do the following:

- Have time-based connection caching - i.e., on the client side, disconnect those connections that have been idle for a (configurable) period of time (default is 1 sec).
- Postpone tweaking the heartbeat logic until we see a problem there.

Devaraj Das added a comment - 13/Jul/06 19:50

I have a version of the patch for this but am not able to release it because of the problem described in Hadoop-362. I use randomwriter and sort as my regular test cases. Strangely enough, the problem described in Hadoop-362 surfaces only when I run the sort benchmark **with** the patch for this issue.

Devaraj Das made changes - 21/Jul/06 15:54

Assignee	Devaraj Das [devaraj]

Devaraj Das made changes - 21/Jul/06 15:55

Link	This issue is blocked by ~~HADOOP-362~~ [~~HADOOP-362~~]

Devaraj Das made changes - 26/Jul/06 06:47

Attachment	config_connection_cache.patch [12336213]

Devaraj Das added a comment - 26/Jul/06 09:30

Attached is the patch. This has been tested independently but it should be commited only after hadoop-362's patch gets commited. During my testing a few days back, I noticed that hadoop-362 surfaces (everytime, almost consistently) with this patch.

Devaraj Das made changes - 26/Jul/06 09:30

Attachment	no_connection_caching.patch [12337537]

Doug Cutting made changes - 03/Aug/06 17:46

Workflow	no-reopen-closed [12374199]	no-reopen-closed, patch-avail [12377502]

Devaraj Das added a comment - 07/Aug/06 18:34

This patch implements the following:
1) Caching of client - server connections is made optional. Defaults to no-caching.
2) If no-caching is true, clients will disconnect idle connections to a server after a configured time. The idle time defaults to 1 second.

The performance hit in this case is that once in a while clients are not able to establish a connection to a server (if the server is too busy to accept incoming connections). I have seen this in the case of TaskTracker -> JobTracker protocol. It happens once in a while. When it happens, the JobTracker assumes that the TaskTracker is lost and then there is a whole set of reruns for the tasks that were running on this "lost" tasktracker. This slows down the overall progress of the job. Of course, this also happens in the case where the connections are cached but the difference is that the RPCs timeout as opposed to connect failing.

If the above doesn't happen, the performance figures with/without caching on a 370 node cluster is nearly the same.

Devaraj Das made changes - 07/Aug/06 18:34

Affects Version/s	0.4.0 [12311021]	
Status	Open [1]	Patch Available [10002]

Devaraj Das made changes - 07/Aug/06 18:40

Attachment		no_connection_caching.patch [12338304]

Yoram Arnon added a comment - 07/Aug/06 19:40

lost tasktrackers because the jobtracker is temporarily busy is a bad thing.
we should have some sort of retry mechanism, or a longer accept queue on the server, or some allowable number of lost heartbeats, or something - to overcome this.
As a rule, barring hardware failure and user-code bugs, any time a tasktracker is lost is a bug.

Devaraj Das added a comment - 07/Aug/06 20:05

I agree with this. In the current code, there is a timeout of 10 minutes and only when a TaskTracker is out of contact for this much amount of time does the JobTracker assume that the TaskTracker is dead. Unfortunately, even with this large timeout, sometimes an unfortunate TaskTracker cannot make it. Yes, the accept queue can be made longer but we will hit the problem sometime later when we have more clients. So, do you think, in addition to increasing the accept queue size, it makes sense to have a two-way heartbeat here? That is, if a server doesn't receive a heartbeat from a client and the expiry-timeout is about to expire, it schedules a heartbeat to the client and probably invokes a GETSTATUS or some such method on the client and if that method returns a valid response, it keeps the client alive for another expiry-timeout interval and this goes on... We can also look at other approaches - some of them are outlined in hadoop-362.
By the way, the patch for hadoop-181 should handle the lost tracker problem but this kind of a problem might turn up for any client-server interaction.

Yoram Arnon added a comment - 08/Aug/06 00:30

the current model, where a tasktracker sends a heartbeat every few seconds, and is declared dead only after 10 minutes is fine, as long as the rate of loss of heartbeats is low.
I'm only concerned, since the size of the accept queue is fairly small by default (~5), that the probability of missing a new connection will be significantly greater than the probability of losing a message in an open tcp stream (very low). Increasing the size of the queue, or implemeting some form of retries could help.

There's a tradeoff here between the overhead of many 'accept's per second and the overhead of 'select'ing on many sockets, many of which are idle. Let's compare performance with and without connection caching, and see where we get more lost heartbeats, and better jobtracker performance.

Devaraj Das added a comment - 18/Aug/06 05:45

Increasing the accept queue length and a simple retry mechanism helped very much. Two cases - (1) where idle connections are cached for a max of 1 sec at the client, and (2) where connections are fully cached.
The performance of the sort benchmark (total time it takes to complete the run) is, most of the times, better with (1). But with a few tasks failing here and there (in both cases), it's actually hard to conclusively say anything about performance in terms of the time it takes to run the benchmark. Made the accept queue length configurable (since that can be manually set on Linux systems as part of the configurable TCP/IP parameters) with the default being 128.

Devaraj Das made changes - 18/Aug/06 05:45

Attachment		no_conn_caching.patch [12339069]

Doug Cutting added a comment - 18/Aug/06 18:31

When connecting, we should first create an unconnected socket, then attempt to connect with an explicit timeout, and retry the connect if it fails. Otherwise the retries are subject to whaterver the default timeout is. Since, with this patch, we're connecting much more, the connect timeout is more significant and should not be left to the system default.

Doug Cutting made changes - 18/Aug/06 18:31

Status	Patch Available [10002]	Open [1]

Doug Cutting added a comment - 18/Aug/06 18:51

Also, all connection retries should be logged.

Devaraj Das added a comment - 19/Aug/06 18:15

Incorporated the comments by Doug.

Devaraj Das made changes - 19/Aug/06 18:15

Attachment		no_conn_caching.patch [12339170]

Doug Cutting added a comment - 21/Aug/06 19:15

Several of the logging statements look like:

if (LOG.isDebugEnabled())
LOG.info(getName() + ...);

But you should log and check the same level. So these should instead be:

if (LOG.isDebugEnabled())
LOG.debug(getName() + ...);

Also, when connecting, you catch Exception, then check exception types with 'instanceof'. It would be better to catch those particular exceptions. You're currently silently trapping other exceptions too.

Devaraj Das added a comment - 22/Aug/06 12:45

Sorry. My mistake for having overlooked the things you pointed out.

Devaraj Das made changes - 22/Aug/06 12:45

Attachment		no_conn_caching.patch [12339309]

Doug Cutting added a comment - 23/Aug/06 17:30

A few more issues:

The IS_CACHING_DISABLED field should not be all caps, since it is not a constant.

When connection caching is disabled then the connection threads should exit when there are no outstanding responses. Then the connections.remove(address) could also be moved to Connection.close(), removing some duplicate logic.

Finally, the check of 'cachingDisabled' could be moved to be inside of incrementRef and decrementRef, rather than around each call, changing these to something like:

private void incrementRef() {
if (cachingDisabled)
{ return; }

synchronized (this)
{ ... }

}

Devaraj Das added a comment - 23/Aug/06 18:12

Actually, I thought that I could save the overhead of destroying/creating threads every so often (could be as frequent as once every second). In the current patch, client connection thread will be created just once for every server and that is destroyed & re-created only when there is an error in receiving server response, etc.

Regarding the check for cachingDisabled, it's outside incrementRef/decrementRef since I wanted to avoid so-many-unnecessary method invocations when caching is disabled.

I agree with the IS_CACHING_DISABLED comment that it should not be all caps.

Makes sense ?

Devaraj Das added a comment - 23/Aug/06 18:24

Meant to say "Regarding the check for cachingDisabled, it's outside incrementRef/decrementRef since I wanted to avoid so-many-unnecessary method invocations when caching is enabled. "

Doug Cutting added a comment - 23/Aug/06 22:32

> I thought that I could save the overhead of destroying/creating threads [...]

Threads should be cheap to create, but keeping too many around can be expensive. Again, I think the life of the thread should be the same as the life of the connection.

> I wanted to avoid so-many-unnecessary method invocations when caching is disabled.

Method invocations that check a flag are plenty fast. This is not an inner loop. I'd rather have this code be easier to maintain than a few nanoseconds faster.

Devaraj Das added a comment - 24/Aug/06 05:17

>Threads should be cheap to create, but keeping too many
>around can be expensive. Again, I think the life of the thread
>should be the same as the life of the connection.

There are not many threads in the system. It is one per server (in the current version of hadoop, it is a single digit number, maybe, 2 or 3). If connection life is 1 sec and if the thread's life is same as the life of the connection, then a thread is destroyed/created ~once/second per server. I really wanted to avoid creating/destroying threads that often.

Sameer Paranjpye added a comment - 25/Aug/06 18:14

> There are not many threads in the system.

Perhaps, but thread creation is quite cheap. I see two reasons for not keeping threads around.

a) The code will be simpler if all objects associated with a server (connection and thread in this case) are created/destroyed together.
b) The point of not caching connections is to avoid lingering unused resources, that applies to threads as much as it does to connections. If there is a use case where connections are created/destroyed once/second/server then the appropriate course to avoid this thrash would be to enable connection caching, no?

Devaraj Das added a comment - 29/Aug/06 20:00

Incorporated the comments.

Devaraj Das made changes - 29/Aug/06 20:00

Attachment	no_conn_caching.patch [12339798]

Devaraj Das added a comment - 29/Aug/06 20:05

Oops, forgot to remove an unused variable. Attached is the modified patch.

Devaraj Das made changes - 29/Aug/06 20:05

Attachment	no_conn_caching.patch [12339799]

Doug Cutting added a comment - 29/Aug/06 20:54

This disables connection caching by default (and hence, everywhere, since no one yet overrides this new option). If we really don't think we need connection caching, then the client could be radically simpler: no per-connection threads, etc.

I thought the original plan was not so much to disable all connection caching, but rather limit it to hosts that have been contacted recently. But I see no provision in the current patch for timing out idle connections.

The incrementRef & decrementRef methods are each only called once. In both cases the check for isCachingDisabled can be skipped. So these can simply become inUse++ and inUse--.

Let's not add a separate config option for the connect timeout, but rather just use the normal io timeout.

Finallly, the socket close and open should not be within a synchronized(connections) block, as, that way, a slow server will block access to all servers.

Devaraj Das added a comment - 01/Sep/06 19:08

This patch should address most of the issues. One issue - that of duplicate code to do with connections.remove(address) is still there. I couldn't find a nice way to remove the duplication of code. I tested this on small clusters (like 80 nodes). Haven't got a chance to test it out on bigger clusters, but in any case, I thought I would get it reviewed...

Devaraj Das made changes - 01/Sep/06 19:08

Attachment	no_conn_caching.patch [12340078]

Devaraj Das added a comment - 01/Sep/06 19:17

By the way, this patch has all the functionality that was planned for originally, less thread-caching.

Devaraj Das added a comment - 01/Sep/06 19:58

+ if (noException) {
+ synchronized (connections)
{ + connections.remove(address); + }

+ }
The above should be if (!noException)
{ ... }

Doug Cutting added a comment - 01/Sep/06 20:06

If CLOSE_CONNECTION is a constant, then it should be declared 'static final'. But I think it can be removed altogether, along with shouldCloseConnection. Why not have the ConnectionCuller simply set running=false, so that Connection.run() exits naturally and removes itself from the connection cache?

The culler doesn't appear to remove the connection from the cache anyway. It should, so that new requests are not handed to a connection that is exiting. And to further safeguard against that, we can check, before calling connections.remove(address) that connections.get(address) == this, rather than keeping a noException boolean.

ConnectionCuller's constructor should set its thread name.

MAX_RETRIES should be named 'maxRetries' and should get its value from a config parameter named ipc.client.connect.max.retries.

Do we need a separate nocache config parameter, and code to support it? Shouldn't this be the same as simply setting maxidletime to 0? If so, let's get rid of isCachingDisabled. This would give only one mode of operation, greatly reducing the number of possible bugs.

The culler should sleep for a minimum time (say 1 second), so it doesn't enter a tight loop.

ipc.server.system.somaxconn should be named something like ipc.server.listen.queue.size

---

Devaraj Das added a comment - 03/Sep/06 15:11

> If CLOSE_CONNECTION is a constant, then it should be declared 'static final'. But I think it can be
> removed altogether, along with shouldCloseConnection. Why not have the ConnectionCuller
> simply set running=false, so that Connection.run() exits naturally and removes itself from the
> connection cache?
If you are referring to the running boolean field attached to the Client class, the setting that to false will make all the response receiver threads exit. So instead, we can have another boolean field attached to the Connection class to signify that the corresponding response receiver thread should exit.

> The culler doesn't appear to remove the connection from the cache anyway. It should, so that new
> requests are not handed to a connection that is exiting.
It does remove I think. Note there is a i.remove() in the run method of the ConnectionCuller thread.

> Do we need a separate nocache config parameter, and code to support it? Shouldn't this be the
> same as simply setting maxidletime to 0? If so, let's get rid of isCachingDisabled. This would give
> only one mode of operation, greatly reducing the number of possible bugs.
So this patch will make the Hadoop clients operate in two modes - one with full caching of connections, and another idle-time-based caching (connections are culled when they are idle for some period of time). I am not sure I understood what you mean by one mode of operation. Are you saying that we should remove "if (isCachingDisabled)" and instead have the corresponding logic work under "if (maxIdleTime == 0)" ?

Also one more thing - currently, I do a check for isCachingDisabled in a few places and then execute relevant code. But some of the code, although not required for the mode where we don't do idle-time-based caching, doesn't really harm the execution of that mode (like incrementRef) except adding some unecessary code. Does it remain that way or do I remove checks for isCachingDisabled from those "safe" places ?

Devaraj Das added a comment - 03/Sep/06 15:53

> If you are referring to the running boolean field attached to the Client class, the setting that to false
> will make all the response receiver threads exit.

Meant to say: "If you are referring ... **all** the response receiver threads (connected to the different servers) exit, not just that thread which was idle".

Doug Cutting added a comment - 05/Sep/06 21:21

You're right, Client.running is inapproprate for this use, and the culler does in fact remove connections (not sure how I missed that).

Re modes: right now there are three modes, each with some separate logic: full-caching, timed caching, and no caching. It would simplify the logic (and hence ease testing and increase reliability) if there were only one mode. Otherwise, we should add unit tests for all modes. But the timed cache seems fairly universal. Setting the timeout to Integer.MAX_VALUE gives full-caching, while setting it to zero could cause most connections to terminate after a single request (although, if some get reused, that should not cause problems). The default should be a fairly short timeout, so that some kinds of requests will reuse cached connections and some will create new connections per request. This way the default configuration will test all of the code. I do not like having modes that might be useful but are not regularly used, since they'll not be tested and may stop working and then generate bugs.

> Are you saying that we should remove "if (isCachingDisabled)" and instead have the corresponding logic work under "if (maxIdleTime == 0)" ?

I don't think we should have either. MaxIdleTime==0 is simply a very short idle time. The culler thread should have a minimum sleep time so that it doesn't busywait.

Devaraj Das added a comment - 06/Sep/06 20:25

Thanks Doug. This patch should address your comments. I have set the time between culler runs to 1000 msec. Please modify it to a more meaningful value if you feel that's not appropriate (& if you feel the patch is in a form that can be committed 🙂).

Devaraj Das made changes - 06/Sep/06 20:25

Attachment		control_conn_caching.patch [12340292]

Doug Cutting added a comment - 06/Sep/06 22:49

I just committed this. Thanks, Devaraj!

Doug Cutting made changes - 06/Sep/06 22:49

Fix Version/s		0.6.0 [12312025]
Resolution		Fixed [1]
Status	Open [1]	Resolved [5]

Doug Cutting made changes - 08/Sep/06 21:19

Status	Resolved [5]	Closed [6]

Transition	Time In Source Status	Execution Times

Devaraj Das made transition - 07/Aug/06 18:34 OPEN → PATCH AVAILABLE	46d 16h 38m	1
Doug Cutting made transition - 18/Aug/06 18:31 PATCH AVAILABLE → OPEN	10d 23h 56m	1
Doug Cutting made transition - 06/Sep/06 22:49 OPEN → RESOLVED	19d 4h 18m	1
Doug Cutting made transition - 08/Sep/06 21:19 RESOLVED → CLOSED	1d 22h 29m	1

People

Assignee:

Devaraj Das

Reporter:

Devaraj Das

Votes:

0 Vote for this issue

Watchers:

0 Start watching this issue

Dates

Created:

22/Jun/06 01:55

Updated:

08/Sep/06 21:19

Resolved:

06/Sep/06 22:49