



Hadoop Map/Reduce [MAPREDUCE-3184](#)

## Improve handling of fetch failures when a tasktracker is not responding on HTTP

### ▼ Details

Type:	↑ Improvement	Status:	CLOSED
Priority:	🔥 Major	Resolution:	Fixed
Affects Version/s:	0.20.205.0	Fix Version/s:	1.0.1
Component/s:	jobtracker		
Labels:	None		
Target Version/s:	1.0.1		
Hadoop Flags:	Reviewed		
Release Note:	▼ The TaskTracker now has a thread which monitors for a known Jetty bug in which the selector thread starts spinning and map output can no longer be served. If the bug is detected, the TaskTracker will shut itself down. This feature can be disabled by setting <code>mapred.tasktracker.jetty.cpu.check.enabled</code> to false.		

### ▼ Description

On a 100 node cluster, we had an issue where one of the TaskTrackers was hit by [MAPREDUCE-2386](#) and stopped responding to fetches. The behavior observed was the following:

- every reducer would try to fetch the same map task, and fail after ~13 minutes.
  - At that point, all reducers would report this failed fetch to the JT for the same task, and the task would be re-run.
  - Meanwhile, the reducers would move on to the next map task that ran on the TT, and hang for another 13 minutes.
- The job essentially made no progress for hours, as each map task that ran on the bad node was serially marked failed.

To combat this issue, we should introduce a second type of failed fetch notification, used when the TT does not respond at all (ie `SocketTimeoutException`, etc). These fetch failure notifications should count against the TT at large, rather than a single task. If more than half of the reducers report such an issue for a given TT, then all of the tasks from that TT should be re-run.

### ▼ Attachments

<a href="#">mr-3184.txt</a>	13 kB	18/Oct/11 00:17
-----------------------------	-------	-----------------

### ▼ Issue Links

is related to

🔥 [MAPREDUCE-5588](#) TaskTrackers get killed by JettyBugMonitor because of incredibly high cpu usage 🔥 RESOLVED

### ▼ Activity



- ▼ [Arun Murthy](#) added a comment - 14/Oct/11 00:20

Todd - is this with or without [MAPREDUCE-2524](#)?

- ▼ [Todd Lipcon](#) added a comment - 14/Oct/11 00:21

This is with [MAPREDUCE-2524](#).

- ▼ [Arun Murthy](#) added a comment - 14/Oct/11 00:34

Two ways around it:


1. An expedient way would be to add a HTTP GET to the node health script...

2. We could also rollup task failures in the JT against the TT.

Given where we are in the product lifecycle for MR1, I'd vote for 1. Seems reasonable for MR2 also...

Thoughts?


---

▼  [Todd Lipcon](#) added a comment - 14/Oct/11 00:46

Arun and I caught up on IRC. The summary is this: currently, #1 above will not work since it doesn't call through to `lostTaskTracker()` – hence, it will prevent scheduling new tasks on the node but won't reschedule the already-complete map output.

Now we should discuss whether there are important use cases for the health script where it would be problematic to make it call `lostTT` – any cases where admins want to use the health script to prevent new scheduling but not fail the stuff that's currently running?


---

▼  [Arun Murthy](#) added a comment - 14/Oct/11 01:36

Now we should discuss whether there are important use cases for the health script where it would be problematic to make it call `lostTT` – any cases where admins want to use the health script to prevent new scheduling but not fail the stuff that's currently running?

The point of the health-script was to quickly determine node faults. In that case, it seems very reasonable to assume that you lose tasks from a 'bad node' and re-run them. I don't see the point of trying to keep running tasks around when a node is deemed bad.

---

▼  [Todd Lipcon](#) added a comment - 14/Oct/11 21:42

I remember hearing at one point that another use case for the health script was to check for things like broken NFS mounts or missing shared libraries on a set of nodes. In those cases, it would make sense to not schedule new tasks, but doesn't make sense to lose the already-completed task outputs.


Another point here is that monitoring for timeouts on HTTP GET is insufficient – in the case that a TT is highly loaded, GETs would time out even though tasks are successfully retrieving map output. To correctly identify the scenario, we'd need to do one of the following:

- 1) impose a separate limit for number of concurrent `MapOutputServlet` invocations, which is a little lower than the limit for `tasktracker.http.threads`. If `MapOutputServlet` is requested when too many invocations are in progress, it would return an HTTP result code indicating that the TT is too busy – distinct from the timeout that it currently returns.
- 2) have smarter monitoring on the client side which looks at a metric like CPU usage to detect the "100% spinning" case.

Another option to consider would be to have the TT have a small thread which issues HTTP GETs to itself. If it times out, it can check how many Jetty threads are actually actively serving requests. If there are free Jetty threads, but can't serve output, the TT can just FATAL itself.

I think this last option may be preferable to the health-check approach, since it would ship with Hadoop without any extra configuration on the part of the user, and would be less prone to false detection issues.

---

▼  [Todd Lipcon](#) added a comment - 18/Oct/11 00:11


Having talked with Arun and Aaron about this a bit, we came up with a few candidate solutions, several of which are described above. However, several of the solutions would require semi-invasive changes to the JT or TT, or require semantic changes to the behavior of the health check script. As Arun put it, we don't want to introduce a generalized solution when the issue here is a very specific Jetty bug – the generalized solution might have other ill effects that would be hard to pin down, making the change hard to verify.

So, instead, the approach we will take is to apply a very specific fix for this very specific Jetty issue: start a thread inside the TT which monitors for a spinning Jetty selector thread, and if detected, shut down the TT. This will cause any reducers to immediately start receiving "Connection refused" errors and recover from the situation rapidly. Existing monitoring scripts will easily notice the failed TT so that the admin can restart.

Clearly, this fix is hack-ish, but it's a hack localized to the scope of the TT, and a new thread in the TT at that. Thus it has little chance of causing regressions with regard to other shuffle heuristics, etc.

I will upload a patch momentarily.

---

▼  [Todd Lipcon](#) added a comment - 18/Oct/11 00:17


Here is a patch implementing the approach described above.

It includes a unit test which shows that the new code can identify a spinning thread.

I also tested the new code by setting the abort threshold to 50% and pounding a tasktracker with an HTTP benchmark tool. This resulted in the TT aborting as expected when CPU usage of the selector thread crossed 50%.

If administrators find that this triggers on false positives, the feature can be entirely disabled by setting `mapred.tasktracker.jetty.cpu.check.enabled` to false, or the threshold can be configured with `mapred.tasktracker.jetty.cpu.threshold.fatal` (default 90%)

---

▼  Eli Collins added a comment - 20/Oct/11 01:47

I thought about some other workarounds (eg bumping up `org.mortbay.io.nio.BUSY_PAUSE`) but after thiking about it more I think what you have here is a reasonable approach. Given where we are in MR1's life I agree a targeted approach makes more sense.


The code looks good. The only thing I'm wondering is whether we should disable the detection by default. The semantics of `getThreadCpuTime` aren't entirely clear (eg does it always return user or system time?) and platform (jdk/OS) specific (eg does IO time ever get counted?). Also according to the JDK docs "thread CPU measurement could be expensive in some Java virtual machine implementations." Eg see the following issues on some versions of Sun Java on Linux:

1. <http://download.oracle.com/javase/6/docs/api/java/lang/management/ThreadMXBean.html>
2. [http://bugs.sun.com/bugdatabase/view\\_bug.do?bug\\_id=6888526](http://bugs.sun.com/bugdatabase/view_bug.do?bug_id=6888526)
3. [http://bugs.sun.com/bugdatabase/view\\_bug.do?bug\\_id=6491083](http://bugs.sun.com/bugdatabase/view_bug.do?bug_id=6491083)

Btw I think it's possible for a subsequent call to `System#nanoTime` to return a smaller value, though this shouldn't cause a false positive in the detection routine.

Nit: I'd up the info on `JettyBugMonitor` line 80 a warning since it's logged once and perhaps disabling a feature the user thinks they have.

---

▼  Todd Lipcon added a comment - 25/Oct/11 17:31

eg does it always return user or system time

it returns the sum of the two. The docs aren't well written ("If the implementation distinguishes between user mode time and system mode time, the returned CPU time is the amount of time that the thread has executed in user mode or system mode") but looking at the source the intention is clear.

Worst case, if an implementation returns only user or only system time, then the CPU usage will be under-estimated, which is OK. As long as we don't over-estimate, it won't cause a false shutdown.


thread CPU measurement could be expensive in some Java virtual machine implementations

Looking at those sun bugs, I think "expensive" here is a relative term. For example, the bug says "the submitted test case takes almost 4 seconds to do 100k CPU measurements" - 40us per call. Given we make the call once every 15 seconds, I'm not too concerned. I think the "may be expensive" warning is just to warn people not to sprinkle these calls throughout their performance-sensitive code to do metrics, etc.

Btw I think it's possible for a subsequent call to `System#nanoTime` to return a smaller valu

I don't think so - `nanotime` is "time since a fixed but arbitrary point in the past". On Linux it's implemented with `clock_gettime(CLOCK_MONOTONIC)`


---

▼  Eli Collins added a comment - 27/Oct/11 01:24

You've sold me on enabling it by default. Agree a 8x-20x difference in particular JVMs should be in the noise for our use. +1 for the patch as is.


`CLOCK_MONOTONIC` can backwards due to kernel bugs (most of those were sorted out but there were versions of RHEL5 that had this issue) and in some hypervisors with buggy time keeping code. Not an issue here.

---

▼  Todd Lipcon added a comment - 27/Oct/11 03:54


Committed to 20-security. I changed the "info" to "warn" as you suggested on commit. Thanks for the review.

---

▼  [Matthew Foley](#) added a comment - 12/Feb/12 10:34

This patch has been tested at user sites and is believed stable. Nathan Roberts requested that I include it in 1.0.1, as its absence is causing ops problems with 1.0.0.

---



▼  [Matthew Foley](#) added a comment - 18/Mar/12 06:18

Closed upon release of 1.0.1.

---

▼ **People**

---

Assignee:  [Todd Lipcon](#)  
Reporter:  [Todd Lipcon](#)  
Votes: **0** [Vote for this issue](#)  
Watchers: **11** [Start watching this issue](#)

▼ **Dates**

---

Created: 14/Oct/11 00:01  
Updated: 11/Mar/14 17:34  
Resolved: 27/Oct/11 03:53