

Cloudera Runtime 7.2.0

Troubleshooting Apache Kudu

Date published: 2020-02-28

Date modified: 2020-06-16

CLOUDERA

<https://docs.cloudera.com/>

Legal Notice

© Cloudera Inc. 2025. All rights reserved.

The documentation is and contains Cloudera proprietary information protected by copyright and other intellectual property rights. No license under copyright or any other intellectual property right is granted herein.

Unless otherwise noted, scripts and sample code are licensed under the Apache License, Version 2.0.

Copyright information for Cloudera software may be found within the documentation accompanying each component in a particular release.

Cloudera software includes software from various open source or other third party projects, and may be released under the Apache Software License 2.0 (“ASLv2”), the Affero General Public License version 3 (AGPLv3), or other license terms. Other software included may be released under the terms of alternative open source licenses. Please review the license and notice files accompanying the software for additional licensing information.

Please visit the Cloudera software product page for more information on Cloudera software. For more information on Cloudera support services, please visit either the Support or Sales page. Feel free to contact us directly to discuss your specific needs.

Cloudera reserves the right to change any products at any time, and without notice. Cloudera assumes no responsibility nor liability arising from the use of products, except as expressly agreed to in writing by Cloudera.

Cloudera, Cloudera Altus, HUE, Impala, Cloudera Impala, and other Cloudera marks are registered or unregistered trademarks in the United States and other countries. All other trademarks are the property of their respective owners.

Disclaimer: EXCEPT AS EXPRESSLY PROVIDED IN A WRITTEN AGREEMENT WITH CLOUDERA, CLOUDERA DOES NOT MAKE NOR GIVE ANY REPRESENTATION, WARRANTY, NOR COVENANT OF ANY KIND, WHETHER EXPRESS OR IMPLIED, IN CONNECTION WITH CLOUDERA TECHNOLOGY OR RELATED SUPPORT PROVIDED IN CONNECTION THEREWITH. CLOUDERA DOES NOT WARRANT THAT CLOUDERA PRODUCTS NOR SOFTWARE WILL OPERATE UNINTERRUPTED NOR THAT IT WILL BE FREE FROM DEFECTS NOR ERRORS, THAT IT WILL PROTECT YOUR DATA FROM LOSS, CORRUPTION NOR UNAVAILABILITY, NOR THAT IT WILL MEET ALL OF CUSTOMER’S BUSINESS REQUIREMENTS. WITHOUT LIMITING THE FOREGOING, AND TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, CLOUDERA EXPRESSLY DISCLAIMS ANY AND ALL IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, QUALITY, NON-INFRINGEMENT, TITLE, AND FITNESS FOR A PARTICULAR PURPOSE AND ANY REPRESENTATION, WARRANTY, OR COVENANT BASED ON COURSE OF DEALING OR USAGE IN TRADE.

Contents

Troubleshooting Apache Kudu.....	4
Issues starting or restarting the master or the tablet server.....	4
Errors during hole punching test.....	4
Already present: FS layout already exists.....	5
NTP clock synchronization.....	5
Disk space usage.....	9
Reporting Kudu crashes using breakpad.....	10
Troubleshooting performance issues.....	10
Kudu tracing.....	11
Memory limits.....	12
Block cache size.....	13
Heap sampling.....	13
Slow name resolution and nscd.....	14
Usability issues.....	14
ClassNotFoundException: com.cloudera.kudu.hive.KuduStorageHandler.....	14
Runtime error: Could not create thread: Resource temporarily unavailable (error 11).....	14
Tombstoned or STOPPED tablet replicas.....	15
Corruption: checksum error on CFile block.....	15
Generating a table list.....	15
Spark tuning.....	16
Symbolizing stack traces.....	16

Troubleshooting Apache Kudu

The basic Apache Kudu troubleshooting information is covered here. For more details, see the official Kudu documentation for troubleshooting.

Issues starting or restarting the master or the tablet server

You may face issues while starting or restarting the master or the tablet server in case there are errors in the hole punching tests, or if the FS layout already exists, or if the master and tablet server's clocks are not synchronized using NTP.

Errors during hole punching test

Kudu requires hole punching capabilities in order to be efficient. Hole punching support depends upon your operation system kernel version and local filesystem implementation.

- RHEL or CentOS 6.4 or later, patched to kernel version of 2.6.32-358 or later. Unpatched RHEL or CentOS 6.4 does not include a kernel with support for hole punching.
- Ubuntu 14.04 includes version 3.13 of the Linux kernel, which supports hole punching.
- Newer versions of the ext4 and xfs filesystems support hole punching. Older versions that do not support hole punching will cause Kudu to emit an error message such as the following and to fail to start:

```
Error during hole punch test. The log block manager requires a
filesystem with hole punching support such as ext4 or xfs. On el6,
kernel version 2.6.32-358 or newer is required. To run without hole
punching (at the cost of some efficiency and scalability), reconfigure
Kudu with --block_manager=file. Refer to the Kudu documentation for more
details. Raw error message follows.
```

**Note:**

ext4 mountpoints may actually be backed by ext2 or ext3 formatted devices, which do not support hole punching. The hole punching test will fail when run on such filesystems. There are several different ways to determine whether an ext4 mountpoint is backed by an ext2, ext3, or ext4 formatted device. See the [Stack Exchange post](#) for more details.

Without hole punching support, the log block manager is unsafe to use. It won't ever delete blocks, and will consume ever more space on disk.

If you can't use hole punching in your environment, you can still try Kudu. Enable the file block manager instead of the log block manager by adding the `--block_manager=file` flag to the commands you use to start the master and tablet servers. The file block manager does not scale as well as the log block manager.

**Attention:**

The file block manager is known to scale and perform poorly, and should only be used for small-scale evaluation and development, and only on systems where hole punching is unavailable.

The file block manager uses one file per block. As multiple blocks are written for each rowset, the number of blocks can be very high, especially for actively written tablets. This can cause performance issues compared to the log block manager even with a small amount of data. And it is impossible to switch between block managers without wiping and reinitializing the tablet servers.

Already present: FS layout already exists

When Kudu starts, it checks each configured data directory, expecting either for all to be initialized or for all to be empty. If a server fails to start with a log message such as the following, then this precondition check has failed.

```
Check failed: _s.ok() Bad status: Already present: Could not create new FS layout: FSManager root is not empty: /data0/kudu/data
```

This could be because Kudu was configured with non-empty data directories on first startup, or because a previously-running, healthy Kudu process was restarted and at least one data directory was deleted or is somehow corrupted, perhaps because of a disk error. If it is the latter, see [Changing directory configuration](#).

NTP clock synchronization

The local clock of the machine where Kudu master or tablet server is running must be synchronized using the Network Time Protocol (NTP) if you are using the system time source. The time source is controlled by the `--time_source` flag and is set to `system`, by default.

Kudu requires the maximum clock error (not to be mistaken with the estimated error) of the NTP-synchronized clock to be below a configurable threshold. The default threshold value is 10 seconds and it can be customized using the `--max_clock_sync_error_usec` flag.

When running Kudu with the system time source, Kudu will not start and will display the following message if the local clock is reported unsynchronized:

```
F0924 20:24:36.336809 14550 hybrid_clock.cc:191 Couldn't get the current time: Clock unsynchronized. Status: Service unavailable: Error reading clock. Clock considered unsynchronized.
```

If the machine's clock is synchronized, but the maximum clock error is too high, you will see a message such as:

```
Sep 17, 8:13:09.873 PM FATAL hybrid_clock.cc:196 Couldn't get the current time: Clock synchronized, but error: 11130000, is past the maximum allowable error: 10000000
```

or

```
Sep 17, 8:32:31.135 PM FATAL tablet_server_main.cc:38 Check failed: _s.ok() Bad status: Service unavailable: Cannot initialize clock: Cannot initialize HybridClock. Clock synchronized but error was too high (11711000 us).
```

Installing NTP-related packages

Kudu has been well-tested to work on machines whose clock is synchronized with `ntpd`, the NTP server from the ubiquitous NTP suite.

To install `ntpd` and other NTP-related utilities, use the command appropriate for your operating system:

OS	Command
Debian/Ubuntu	<code>sudo apt-get install ntp</code>
RHEL/CentOS	<code>sudo yum install ntp</code>

If `ntpd` is installed but not running, update it by running the `ntpddate` command and then restart it by using command appropriate for your operating system:

OS	Command
Debian/Ubuntu	<code>sudo service ntp restart</code>
RHEL/CentOS	<code>sudo service ntpd restart</code>

Make sure that the `ntpd` is in the list of services running when the machine starts. The `ntpd` command should be run first before starting `ntpd` to avoid long synchronization delay of the machine's local clock with the true time. The smaller the offset between local machine's clock and the true time, the faster the NTP server can synchronize the clock.

When talking about the 'synchronization' with true time using NTP, we are referring to:

- The synchronization status of the NTP server which drives the local clock of the machine
- The synchronization status of the local machine's clock itself as reported by the kernel's NTP discipline

The former can be retrieved using the `ntpstat`, `ntpq`, and `ntpd` utilities (they are included in the `ntp` package). The latter can be retrieved using the `ntpdate` utility (the `ntpdate` utility is also a part of the `ntp` package). For more information, see the manual pages of the mentioned utilities and the following section:

Sometime it takes too long to synchronize the machine's local clock with the true time even if the `ntpstat` utility reports that the NTP daemon is synchronized with one of the reference NTP servers. This manifests as the following: the utilities which report on the synchronization status of the NTP daemon claim that all is well, but `ntpdate` claims that the status of the local clock is unsynchronized and Kudu tablet servers and masters refuse to start, outputting an error like the one mentioned above. This situation often happens if the `ntpd` is run with the `-x` option. According to the manual page of `ntpd`, the `-x` flag configures the NTP server to only slew the clock. Without `-x`, the NTP server would do a step adjustment instead:

```
-x
Normally, the time is slewed if the offset is less than the step threshold,
which is 128 ms by default, and is stepped if the offset is above the threshold.
This option sets the threshold to 600 s, which is well within the accuracy
window to set the clock manually.
```



Note:

Since the slew rate of typical Unix kernels is limited to 0.5 ms/s, each second of adjustment requires an amortization interval of 2000 s. Thus, an adjustment as much as 600 s will take almost 14 days to complete.

In such cases, removing the `-x` flag will help synchronize the local clock faster.

For more information on best practices and examples of practical resolution of various NTP synchronization issues, see [clock-drift](#)

Monitoring NTP status

When the `ntp` package is installed, you can monitor the synchronization status of the machine's clock by running `ntpdate`.

For example, a system with a local clock that is synchronized may report:

```
ntp_gettime() returns code 0 (OK)
time de24c0cf.8d5da274 Tue, Feb 6 2018 16:03:27.552, (.552210980),
maximum error 224455 us, estimated error 383 us, TAI offset 0
ntp_adjtime() returns code 0 (OK)
modes 0x0 (),
offset 1279.543 us, frequency 2.500 ppm, interval 1 s,
maximum error 224455 us, estimated error 383 us,
status 0x2001 (PLL,NANO),
time constant 10, precision 0.001 us, tolerance 500 ppm,
```

Note the following most important parts of output:

- maximum error 224455 us: This value is well under the 10-second maximum error required by Kudu
- status 0x2001 (PLL,NANO): This indicates that the local clock is synchronized with the true time up to the maximum error above

In contrast, a system without NTP properly configured and running will output something like the following:

```
ntp_gettime() returns code 5 (ERROR)
```

```
time de24c240.0c006000 Tue, Feb 6 2018 16:09:36.046, (.046881),
maximum error 16000000 us, estimated error 16000000 us, TAI offset 0
ntp_adjtime() returns code 5 (ERROR)
modes 0x0 (),
offset 0.000 us, frequency 2.500 ppm, interval 1 s,
maximum error 16000000 us, estimated error 16000000 us,
status 0x40 (UNSYNC),
time constant 10, precision 1.000 us, tolerance 500 ppm,
```

The UNSYNC status means that the local clock is not synchronized with the true time. Because of that, the maximum reported error doesn't convey any meaningful estimation of the actual error.

The ntpstat utility reports a summary on the synchronization status of the NTP daemon itself. For example, a system which has ntpd running and synchronized with one of its reference servers may report:

```
$ ntpstat
synchronised to NTP server (172.18.7.3) at stratum 4
time correct to within 160 ms
polling server every 1024 s
```

Keep in mind that the synchronization status of the NTP daemon itself doesn't reflect the synchronization status of the local clock. The way NTP daemon drives the local clock is subject to many constraints, and it may take the NTP daemon some time to synchronize the local clock after it itself has latched to one of the reference servers.

If more detailed information is needed on the synchronization status of the NTP server (but not the synchronization status of the local clock), the ntpq or ntpdc tools can be used to get detailed information about what NTP server is currently acting as the source of the true time and which are considered as candidates (either viable or not):

```
$ ntpq -nc lpeers
      remote               refid           st t when poll reach  delay  offset  ji
-----
---
-108.59.2.24      130.133.1.10      2 u   13   64    1   71.743    0.373   0.0
16
+192.96.202.120  129.6.15.28       2 u   12   64    1   72.583   -0.426
0.028
-69.10.161.7      204.26.59.157     3 u   11   64    1   15.741    2.641   0
.021
-173.255.206.154 45.56.123.24      3 u   10   64    1   43.502    0.199   0.
029
-69.195.159.158  128.138.140.44    2 u    9   64    1   53.885   -0.016   0.0
13
*216.218.254.202 .CDMA.            1 u    6   64    1    1.475   -0.400
0.012
+129.250.35.250  249.224.99.213    2 u    7   64    1    1.342   -0.640   0
.018

$ ntpq -nc opeers
      remote               local           st t when poll reach  delay  offset  di
sp
-----
-----
-108.59.2.24      10.17.100.238     2 u   17   64    1   71.743    0.373 187
.573
+192.96.202.120  10.17.100.238     2 u   16   64    1   72.583   -0.426 187.
594
-69.10.161.7      10.17.100.238     3 u   15   64    1   15.741    2.641 187.5
69
-173.255.206.154 10.17.100.238     3 u   14   64    1   43.502    0.199 18
7.580
-69.195.159.158  10.17.100.238     2 u   13   64    1   53.885   -0.016 187
.561
```

```
*216.218.254.202 10.17.100.238 1 u 10 64 1 1.475 -0.400 187.543
+129.250.35.250 10.17.100.238 2 u 11 64 1 1.342 -0.640 187.588
```



Note: Both `lpeers` and `opeers` may be helpful as `lpeers` lists refid and jitter, while `opeers` lists clock dispersion.

Using chrony for time synchronization

Some operating systems offer `chronyd` as an alternative to `ntpd` for network time synchronization. The OS package is called `chrony` and contains both the NTP server `chronyd` and the `chronyc` utilities.

If using `chronyd` for time synchronization at Kudu nodes, the `rtcsync` option must be enabled in `chrony.conf`. Without `rtcsync`, the local machine's clock will always be reported as unsynchronized and Kudu masters and tablet servers will not be able to start. The following [code](#) explains the observed behavior of `chronyd` when setting the synchronization status of the clock on Linux.

NTP configuration best practices

In order to provide stable time synchronization with low maximum error, follow the NTP configuration best practices listed in this topic.

- Run `ntpd` before running NTP server. If the offset of the local clock is too far from the true time, it can take a long time before the NTP server synchronizes the local clock, even if it's allowed to perform step adjustments.
- In certain public cloud environments, use the highly-available NTP server accessible via link-local IP address or other dedicated NTP server provided as a service. If your cluster is running in a public cloud environment, consult the cloud provider's documentation for the recommended NTP setup. Both AWS and GCE clouds offer dedicated highly available NTP servers accessible from within a cloud instance via link-local IP address.
- Unless using highly-available NTP reference server accessible via link-local address, always configure at least four time sources for NTP server at the local machine. In addition to providing redundancy in case one of time sources becomes unavailable, this might make the configuration more robust since the NTP is designed to increase its accuracy with a diversity of sources in networks with higher round-trip times and jitter.
- Use the `iburst` option for faster synchronization at startup. The `iburst` option instructs `ntpd` to send an initial "burst" of time queries at startup. This results in a faster synchronization of the `ntpd` with its reference servers upon startup.
- If the maximum clock error goes beyond the default threshold set by Kudu (10 seconds), consider setting lower value for the `maxpoll` option for every NTP server in `ntp.conf`. For example, consider setting the `maxpoll` to 7 which will cause the NTP daemon to make requests to the corresponding NTP server at least every 128 seconds. The default maximum poll interval is 10 (1024 seconds).



Note: If using custom `maxpoll` interval, don't set `maxpoll` too low (e.g., lower than 6) to avoid flooding NTP servers, especially the public ones. Otherwise they may disallow the client (i.e. the `ntpd` daemon at your machine) and cease providing NTP service at all. If in doubt, consult the `ntp.conf` manual page.

Following are a few examples of the `ntpd` configuration files:

```
# Use my organization's internal NTP server (server in a local network).
server ntp1.myorg.internal iburst maxpoll 7
# Add servers from the NTP public pool for redundancy and robustness.
server 0.pool.ntp.org iburst maxpoll 8
server 1.pool.ntp.org iburst maxpoll 8
server 2.pool.ntp.org iburst maxpoll 8
server 3.pool.ntp.org iburst maxpoll 8
```

```
# AWS case: use dedicated NTP server available via link-local IP address.
server 169.254.169.123 iburst
```

```
# GCE case: use dedicated NTP server available from within cloud instance.
server metadata.google.internal iburst
```



Note: After configuring ntpd, first run the ntpdate tool with the same set of NTP servers (it's assumed that ntpd is not running when the ntpdate tool is run). Make sure the tool reports success: check its exit status and output. In case of issues connecting to the NTP servers, make sure NTP traffic is not being blocked by a firewall (NTP generates UDP traffic on port 123 by default) or other network connectivity issue. Then start the ntpd daemon and use the ntpq tool described above to verify that the NTP daemon is able to connect to its reference servers.

Troubleshooting NTP stability problems

As of Kudu 1.6.0, Kudu daemons can continue to operate during a brief loss of clock synchronization. If clock synchronization is lost for several hours, the Kudu daemons may crash. If a daemon crashes due to clock synchronization issues, consult the ERROR log for a dump of related information which may help to diagnose the issue.



Note: If using other than link-local NTP server, it may take some time for ntpd to synchronize with one of its reference servers in case of network connectivity issues. In case of a spotty network between the machine and the reference NTP servers, ntpd may become unsynchronized with its reference NTP servers. If that happens, consider finding other set of reference NTP servers: the best bet is to use NTP servers in the local network or *.pool.ntp.org servers.

Disk space usage

When using the log block manager (the default on Linux), Kudu uses sparse files to store data. A sparse file has a different apparent size than the actual amount of disk space it uses. This means that some tools may inaccurately report the disk space used by Kudu. For example, the size listed by `ls -l` does not accurately reflect the disk space used by Kudu data files:

```
$ ls -lh /data/kudu/tserver/data
total 117M
-rw----- 1 kudu kudu 160M Mar 26 19:37 0b9807b8b17d48a6a7d5b16bf4ac4e6d.da
ta
-rw----- 1 kudu kudu 4.4K Mar 26 19:37 0b9807b8b17d48a6a7d5b16bf4ac4e6d
.metadata
-rw----- 1 kudu kudu 32M Mar 26 19:37 2f26eeacc7e04b65a009e2c9a2a8bd20.
data
-rw----- 1 kudu kudu 4.3K Mar 26 19:37 2f26eeacc7e04b65a009e2c9a2a8bd20.m
etadata
-rw----- 1 kudu kudu 672M Mar 26 19:37 30a2dd2cd3554d8a9613f588a8d136ff.da
ta
-rw----- 1 kudu kudu 4.4K Mar 26 19:37 30a2dd2cd3554d8a9613f588a8d136ff
.metadata
-rw----- 1 kudu kudu 32M Mar 26 19:37 7434c83c5ec74ae6af5974e4909cbf82.
data
-rw----- 1 kudu kudu 4.3K Mar 26 19:37 7434c83c5ec74ae6af5974e4909cbf82.m
etadata
-rw----- 1 kudu kudu 672M Mar 26 19:37 772d070347a04f9f8ad2ad3241440090.da
ta
-rw----- 1 kudu kudu 4.4K Mar 26 19:37 772d070347a04f9f8ad2ad3241440090
.metadata
-rw----- 1 kudu kudu 160M Mar 26 19:37 86e50a95531f46b6a79e671e6f5f4151.
data
-rw----- 1 kudu kudu 4.4K Mar 26 19:37 86e50a95531f46b6a79e671e6f5f4151.m
etadata
-rw----- 1 kudu kudu 687 Mar 26 19:26 block_manager_instance
```

Notice that the total size reported is 117MiB, while the first file's size is listed as 160MiB. Adding the `-s` option to `ls` will cause `ls` to output the file's disk space usage.

The `du` and `df` utilities report the actual disk space usage by default.

```
$ du -h /data/kudu/tserver/data118M /data/kudu/tserver/data
```

The apparent size can be shown with the `--apparent-size` flag to `du`.

```
$ du -h --apparent-size /data/kudu/tserver/data1.7G /data/kudu/tserver/data
```

Reporting Kudu crashes using breakpad

Kudu uses the Google breakpad library to generate a minidump whenever Kudu experiences a crash. A minidump file contains important debugging information about the process that crashed, including shared libraries loaded and their versions, a list of threads running at the time of the crash, the state of the processor registers and a copy of the stack memory for each thread, and CPU and operating system version information. These minidumps are typically only a few MB in size and are generated even if core dump generation is disabled. Currently, generating minidumps is only possible on Linux deployments.

By default, Kudu stores its minidumps in a subdirectory of the configured `glog` directory called `minidumps`. This location can be customized by setting the `--minidump_path` flag. Kudu will retain only a certain number of minidumps before deleting the older ones, in an effort to avoid filling up the disk with minidump files. The maximum number of minidumps that will be retained can be controlled by setting the `--max_minidumps` gflag.

Minidumps contain information specific to the binary that created them and are therefore not useful without access to the exact binary that crashed, or a very similar binary.

Kudu developers can access the minidump tools in their development environment because they are installed as part of the Kudu thirdparty build. They can be found in the Kudu development environment under `uninstrumented/bin`. For example, `thirdparty/installed/uninstrumented/bin/minidump-2-core`.

If minidumps are enabled, it is possible to force Kudu to create a minidump without killing the process. To do that, send a `USR1` signal to the `kudu-tserver` or `kudu-master` process. For example:

```
sudo pkill -USR1 kudu-tserver
```

Viewing the minidump stack trace with the GNU debugger

Although a minidump contains no heap information, it does contain thread and stack information. You can convert a minidump to a core file to view it with GDB.

To convert the minidump (`.dmp` file) to a core file:

```
minidump-2-core -o 02cb4a97-ee37-6454-73a9d9cb-590c7dde.core \  
02cb4a97-ee37-6454-73a9d9cb-590c7dde.dmp
```

To view the core file with GDB (on a parcel deployment):

```
gdb /opt/cloudera/parcels/KUDU/lib/kudu/sbin-release/kudu-master \  
-s /opt/cloudera/parcels/KUDU/lib/debug/usr/lib/kudu/sbin-release/kudu-ma-  
ster.debug \  
02cb4a97-ee37-6454-73a9d9cb-590c7dde.core
```

Troubleshooting performance issues

This topic helps you to troubleshoot issues and improve performance using Kudu tracing, memory limits, block size cache, heap sampling, and name service cache daemon (`nscd`).

Kudu tracing

The Kudu master and tablet server daemons include built-in support for tracing based on the open source Chromium Tracing framework. You can use tracing to diagnose latency issues or other problems on Kudu servers.

Accessing the tracing web interface

The tracing interface is part of the embedded web server in each of the Kudu daemons, and can be accessed using a web browser. Note that while the interface has been known to work in recent versions of Google Chrome, other browsers may not work as expected.

Daemon	URL
Tablet Server	<TABLET-SERVER-1.EXAMPLE.COM>:8050/tracing.html
Master	<MASTER-1.EXAMPLE.COM>:8051/tracing.html

RPC timeout traces

If client applications are experiencing RPC timeouts, the Kudu tablet server WARNING level logs should contain a log entry which includes an RPC-level trace.

For example:

```
W0922 00:56:52.313848 10858 inbound_call.cc:193] Call kudu.consensus.ConsensusService.UpdateConsensus
from 192.168.1.102:43499 (request call id 3555909) took 1464ms (client timeout 1000).
W0922 00:56:52.314888 10858 inbound_call.cc:197] Trace:
0922 00:56:50.849505 (+ 0us) service_pool.cc:97] Inserting onto call queue
0922 00:56:50.849527 (+ 22us) service_pool.cc:158] Handling call
0922 00:56:50.849574 (+ 47us) raft_consensus.cc:1008] Updating replica for 2 ops
0922 00:56:50.849628 (+ 54us) raft_consensus.cc:1050] Early marking committed up to term: 8 index: 880241
0922 00:56:50.849968 (+ 340us) raft_consensus.cc:1056] Triggering prepare for 2 ops
0922 00:56:50.850119 (+ 151us) log.cc:420] Serialized 1555 byte log entry
0922 00:56:50.850213 (+ 94us) raft_consensus.cc:1131] Marking committed up to term: 8 index: 880241
0922 00:56:50.850218 (+ 5us) raft_consensus.cc:1148] Updating last received op as term: 8 index: 880243
0922 00:56:50.850219 (+ 1us) raft_consensus.cc:1195] Filling consensus response to leader.
0922 00:56:50.850221 (+ 2us) raft_consensus.cc:1169] Waiting on the replicates to finish logging
0922 00:56:52.313763 (+1463542us) raft_consensus.cc:1182] finished
0922 00:56:52.313764 (+ 1us) raft_consensus.cc:1190] UpdateReplicas() finished
0922 00:56:52.313788 (+ 24us) inbound_call.cc:114] Queueing success response
```

These traces can indicate which part of the request was slow. Make sure you include them when filing bug reports related to RPC latency outliers.

Kernel stack watchdog traces

Each Kudu server process has a background thread called the Stack Watchdog, which monitors other threads in the server in case they are blocked for longer-than-expected periods of time. These traces can indicate operating system issues or bottle-necked storage.

When the watchdog thread identifies a case of thread blockage, it logs an entry in the WARNING log as follows:

```
W0921 23:51:54.306350 10912 kernel_stack_watchdog.cc:111] Thread 10937 stuck at /data/kudu/consensus/log.cc:505 for 537ms:
```

```

Kernel stack:
[<fffffffa00b209d>] do_get_write_access+0x29d/0x520 [jbd2]
[<fffffffa00b2471>] jbd2_journal_get_write_access+0x31/0x50 [jbd2]
[<fffffffa00fe6d8>] __ext4_journal_get_write_access+0x38/0x80 [ext4]
[<fffffffa00d9b23>] ext4_reserve_inode_write+0x73/0xa0 [ext4]
[<fffffffa00d9b9c>] ext4_mark_inode_dirty+0x4c/0x1d0 [ext4]
[<fffffffa00d9e90>] ext4_dirty_inode+0x40/0x60 [ext4]
[<ffffff811ac48b>] __mark_inode_dirty+0x3b/0x160
[<ffffff8119c742>] file_update_time+0xf2/0x170
[<ffffff8111c1e0>] __generic_file_aio_write+0x230/0x490
[<ffffff8111c4c8>] generic_file_aio_write+0x88/0x100
[<ffffff8110d3fb1>] ext4_file_write+0x61/0x1e0 [ext4]
[<ffffff81180f5b>] do_sync_readv_writev+0xfb/0x140
[<ffffff81181ee6>] do_readv_writev+0xd6/0x1f0
[<ffffff81182046>] vfs_writev+0x46/0x60
[<ffffff81182102>] sys_pwritev+0xa2/0xc0
[<ffffff8100b072>] system_call_fastpath+0x16/0x1b
[<ffffffffffffffff>] 0xffffffffffffffff
User stack:
@      0x3a1ace10c4 (unknown)
@      0x1262103 (unknown)
@      0x12622d4 (unknown)
@      0x12603df (unknown)
@      0x8e7bfb (unknown)
@      0x8f478b (unknown)
@      0x8f55db (unknown)
@      0x12a7b6f (unknown)
@      0x3a1b007851 (unknown)
@      0x3a1ace894d (unknown)
@      (nil) (unknown)

```

These traces can be useful for diagnosing root-cause latency issues in Kudu especially when they are caused by underlying systems such as disk controllers or filesystems.

Memory limits

Kudu has a hard and soft memory limit. The hard memory limit is the maximum amount a Kudu process is allowed to use, and is controlled by the `--memory_limit_hard_bytes` flag. The soft memory limit is a percentage of the hard memory limit, controlled by the flag `memory_limit_soft_percentage` and with a default value of 80%, that determines the amount of memory a process may use before it will start rejecting some write operations.

If the logs or RPC traces contain messages such as the following example, then Kudu is rejecting writes due to memory back pressure. This may result in write timeouts.

```
Service unavailable: Soft memory limit exceeded (at 96.35% of capacity)
```

There are several ways to relieve the memory pressure on Kudu:

- If the host has more memory available for Kudu, increase `--memory_limit_hard_bytes`.
- Increase the rate at which Kudu can flush writes from memory to disk by increasing the number of disks or increasing the number of maintenance manager threads `--maintenance_manager_num_threads`. Generally, the recommended ratio of maintenance manager threads to data directories is 1:3.
- Reduce the volume of writes flowing to Kudu on the application side.

Finally, in Kudu versions 1.7 and lower, check the value of the `--block_cache_capacity_mb` setting. This setting determines the maximum size of Kudu's block cache. While a higher value can help with read and write performance, setting it too high as a percentage of the `--memory_limit_hard_bytes` setting is harmful. Do not raise `--block_cache_capacity_mb` above `--memory_pressure_percentage` (default 60%) of `--memory_limit_hard_bytes`, as this will cause Kudu to flush aggressively even if write throughput is low. The recommended value for `--block_cache_capacity_mb` is below the following:

$(50\% * \text{--memory_pressure_percentage}) * \text{--memory_limit_hard_bytes}$

With the defaults, this means the `--block_cache_capacity_mb` should not exceed 30% of `--memory_limit_hard_bytes`.

In Kudu 1.8 and higher, servers will refuse to start if the block cache capacity exceeds the memory pressure threshold.

Block cache size

Kudu uses an LRU cache for recently read data. On workloads that scan a subset of the data repeatedly, raising the size of this cache can offer significant performance benefits. To increase the amount of memory dedicated to the block cache, increase the value of the `--block_cache_capacity_mb` flag. The default is 512 MiB.

Kudu provides a set of useful metrics for evaluating the performance of the block cache, which can be found on the /metrics endpoint of the Web UI. The following is an example set:

```
{
  "name": "block_cache_inserts",
  "value": 64
},
{
  "name": "block_cache_lookups",
  "value": 512
},
{
  "name": "block_cache_evictions",
  "value": 0
},
{
  "name": "block_cache_misses",
  "value": 96
},
{
  "name": "block_cache_misses_caching",
  "value": 64
},
{
  "name": "block_cache_hits",
  "value": 0
},
{
  "name": "block_cache_hits_caching",
  "value": 352
},
{
  "name": "block_cache_usage",
  "value": 6976
}
```

To judge the efficiency of the block cache on a tablet server, first wait until the server has been running and serving normal requests for some time, so the cache is not cold. Unless the server stores very little data or is idle, `block_cache_usage` should be equal or nearly equal to `block_cache_capacity_mb`. Once the cache has reached steady state, compare `block_cache_lookups` to `block_cache_misses_caching`. The latter metric counts the number of blocks that Kudu expected to read from cache but which weren't found in the cache. If a significant amount of lookups result in misses on expected cache hits, and the `block_cache_evictions` metric is significant compared to `block_cache_inserts`, then raising the size of the block cache may provide a performance boost. However, the utility of the block cache is highly dependent on workload, so it's necessary to test the benefits of a larger block cache.



Attention: Do not raise the block cache size `--block_cache_capacity_mb` higher than the memory pressure threshold (defaults to 60% of `--memory_limit_hard_bytes`). As this would cause poor flushing behavior, Kudu servers version 1.8 and higher will refuse to start when misconfigured in this way.

Heap sampling

For advanced debugging of memory usage, administrators may enable heap sampling on Kudu daemons. This allows Kudu developers to associate memory usage with the specific lines of code and data structures responsible. When

reporting a bug related to memory usage or an apparent memory leak, heap profiling can give quantitative data to pinpoint the issue.



Caution: Heap sampling is an advanced troubleshooting technique and may cause performance degradation or instability of the Kudu service. Currently it is not recommended to enable this in a production environment unless specifically requested by the Kudu development team.

To enable heap sampling on a Kudu daemon, pass the flag `--heap-sample-every-n-bytes=524588`. If heap sampling is enabled, the current sampled heap occupancy can be retrieved over HTTP by visiting `http://tablet-server.example.com:8050/pprof/heap` or `http://master.example.com:8051/pprof/heap`. The output is a machine-readable dump of the stack traces with their associated heap usage.

Rather than visiting the heap profile page directly in a web browser, it is typically more useful to use the `pprof` tool that is distributed as part of the `gperftools` open source project. For example, a developer with a local build tree can use the following command to collect the sampled heap usage and output an SVG diagram:

```
thirdparty/installed/uninstrumented/bin/pprof -svg 'http://localhost:8051/pprof/heap' > /tmp/heap.svg
```

The resulting SVG may be visualized in a web browser or sent to the Kudu community to help troubleshoot memory occupancy issues.



Tip: Heap samples contain only summary information about allocations and do not contain any data from the heap. It is safe to share heap samples in public without fear of exposing confidential or sensitive data.

Slow name resolution and nscd

For better scalability on nodes hosting many replicas, we recommend that you use `nscd` (name service cache daemon) to cache both DNS name resolution and static name resolution (via `/etc/hosts`).

When DNS lookups are slow, you will see a log message similar to the following:

```
W0926 11:19:01.339553 27231 net_util.cc:193] Time spent resolve address for kudu-tserver.example.com: real 4.647s user 0.000s sys 0.000s
```

`nscd` can alleviate slow name resolution by providing a cache for the most common name service requests, such as for passwords, groups, and hosts.

Refer to your operating system documentation for how to install and enable `nscd`.

Usability issues

This topic lists some common exceptions and errors that you may encounter while using Kudu and helps you to resolve issues related to usability.

ClassNotFoundException: com.cloudera.kudu.hive.KuduStorageHandler

You will encounter this exception when you try to access a Kudu table using Hive. This is not a case of a missing jar, but simply that Impala stores Kudu metadata in Hive in a format that is unreadable to other tools, including Hive itself, and Spark. Currently, there is no workaround for Hive users. Spark users can work around this by creating temporary tables.

Runtime error: Could not create thread: Resource temporarily unavailable (error 11)

You may encounter this error when Kudu is unable to create more threads, usually on versions older than Kudu 1.7. It happens on tablet servers, and is a sign that the tablet server hosts too many tablet replicas.

To fix the issue, you can raise the `nproc` `ulimit` as detailed in the documentation for your operating system or distribution.

However, the better solution is to reduce the number of replicas on the tablet server. This may involve rethinking the table's partitioning schema. For the recommended limits on number of replicas per tablet server, see the [Scaling recommendations and limitations](#) topic.

Tombstoned or STOPPED tablet replicas

You may notice some replicas on a tablet server are in a STOPPED state and remain on the server indefinitely. These replicas are tombstones. A tombstone indicates that the tablet server once held a bona fide replica of its tablet.

For example, in case a tablet server goes down and its replicas are re-replicated elsewhere, if the tablet server rejoins the cluster, its replicas will become tombstones. A tombstone will remain until the table it belongs to is deleted, or a new replica of the same tablet is placed on the tablet server. A count of tombstoned replicas and details of each one are available on the /tablets page of the tablet server web UI. The Raft consensus algorithm that Kudu uses for replication requires tombstones for correctness in certain rare situations. They consume minimal resources and hold no data. They must not be deleted.

Corruption: checksum error on CFile block

In versions prior to Kudu 1.8.0, if the data on disk becomes corrupt, you will encounter warnings containing "Corruption: checksum error on CFile block" in the tablet server logs and client side errors when trying to scan tablets with corrupt CFile blocks. Fixing this corruption is a manual process.

To fix the issue, first identify all the affected tablets by running a checksum scan on the affected tables or tablets using the [ksck](#) tool.

```
sudo -u kudu kudu cluster ksck <master_addresses> -checksum_scan -tables=<tables>
sudo -u kudu kudu cluster ksck <master_addresses> -checksum_scan -tablets=<tablets>
```

If there is at least one replica for each tablet that does not return a corruption error, you can repair the bad copies by deleting them and forcing them to be re-replicated from the leader using the `remote_replica delete` tool.

```
sudo -u kudu kudu remote_replica delete <tserver_address> <tablet_id> "Cfile Corruption"
```

If all of the replica are corrupt, then some data loss has occurred. Until [KUDU-2526](#) is completed, this can happen if the corrupt replica became the leader and the existing follower replicas are replaced.

If data has been lost, you can repair the table by replacing the corrupt tablet with an empty one using the `unsafe_replace_tablet` tool.

```
sudo -u kudu kudu tablet unsafe_replace_tablet <master_addresses> <table t_id>
```

From versions 1.8.0 onwards, Kudu will mark the affected replicas as failed, leading to their automatic re-replication elsewhere.

Generating a table list

To generate a list of tables to backup using the `kudu table list` tool along with `grep` can be useful.

Following is an example that generates a list of all tables that start with `my_db`:

```
kudu table list <master_addresses> | grep "^my_db\.*" | tr '\n' ' '
```



Note: This list could be saved as a part of your backup process so that it can be used while restoring.

Spark tuning

In general the Spark jobs were designed to run with minimal tuning and configuration. You can adjust the number of executors and resources to increase parallelism and performance using Spark's configuration options.

If your tables are super wide and your default memory allocation is fairly low, you may see jobs fail. To resolve this, increase the Spark executor memory. A conservative rule of thumb is 1 GiB per 50 columns.

If your Spark resources drastically outscale the Kudu cluster, then you may want to limit the number of concurrent tasks allowed to run on restore.

Related Information

[Spark's configuration options](#)

Symbolizing stack traces

This topic helps you to identify whether there is a high contention among the threads to acquire a lock and a way to symbolize stack addresses.

Sometimes you might see the following in the logs:

```
0323 03:59:31.091198 (+607857us) spinlock_profiling.cc:243] Waited 492 ms on
lock 0x4cb0960. stack: 00000000002398852 0000000000ad8c69 0000000000aa62ba 0
000000000221aaa8 0000000000221b1a8 000000000023a8f83 00007fa8b818be24 00007fa8b
646a34c
```

That is usually a sign of high contention among threads to acquire a lock, and in this case the reported time shows how long a thread spent on a CPU before acquiring the lock. The call stack addresses that are listed help to restore the stack trace of the waiting thread and locate the problem in the code.

It is possible to translate the addresses into the name of functions and lines in the code having the binary that produced the output (in this example, it is kudu-master). If the binary is stripped of symbols and debug information, it is possible to do so if the debug information for the binary is available separately.

Assuming both the stripped release binary and the debug information are available as RPMs, unpack them into a directory; for example, sysroot:

```
$ mkdir sysroot && cd sysroot
$ rpm2cpio ../kudu-1.10.0.el7.x86_64.rpm | cpio -idmv
$ rpm2cpio ../kudu-debuginfo-1.10.0.el7.x86_64.rpm | cpio -idmv
```

Use `addr2line` to find the line in the code for the stack address. In case if the binary is not stripped of debug information, supply the actual binary with an `-e` option instead of the debug info file as follows:

```
addr2line -C -f -e usr/lib/debug/usr/lib/kudu/sbin-release/kudu-master.debug
0x0000000000aa62ba
kudu::master::MasterServiceImpl::ConnectToMaster(kudu::master::ConnectToMast
erRequestPB const*, kudu::master::ConnectToMasterResponsePB*, kudu::rpc::Rpc
Context*)
/usr/src/debug/kudu-1.10.0/src/kudu/test/master_service.cc:504
```

To achieve the same with gdb, first find the address of the .text section in the symbol file (in the example, 0000000000a2cdb0):

```
$ readelf -S usr/lib/debug/usr/lib/kudu/sbin-release/kudu-master.debug | grep .text
[13] .text          NOBITS          0000000000a2cdb0  000002c0
```

Then start up gdb, pointing it to the kudu-master executable (that's the executable that produced the output in the log file):

```
gdb usr/lib/kudu/sbin-release/kudu-master
```

Now load the .debug symbols into gdb using the address found above. Tell gdb where to find source files and set the sysroot:

```
(gdb) add-symbol-file usr/lib/debug/usr/lib/kudu/sbin-release/kudu-master.debug 0x0000000000a2cdb0
(gdb) set substitute-path /usr/src/debug/kudu-1.10.0 usr/src/debug/kudu-1.10.0
(gdb) set sysroot .
```

To translate the address into line number and function information, use `info line * <address>`:

```
(gdb) info line * 0x0000000000aa62ba
Line 504 of "/usr/src/debug/kudu-1.10.0/src/kudu/test/master_service.cc"
    starts at address 0xaa62af <kudu::master::MasterServiceImpl::ConnectToMaster(kudu::master::ConnectToMasterRequestPB const*, kudu::master::ConnectToMasterResponsePB*, kudu::rpc::RpcContext*)+47>
    and ends at 0xaa62bb <kudu::master::MasterServiceImpl::ConnectToMaster(kudu::master::ConnectToMasterRequestPB const*, kudu::master::ConnectToMasterResponsePB*, kudu::rpc::RpcContext*)+59>.
```