Cloudera Runtime 7.1.7

Troubleshooting Apache Kudu

Date published: 2020-02-28
Date modified: 2021-08-05

https://docs.cloudera.com/

Legal Notice

© Cloudera Inc. 2024. 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

Issues starting or restarting the master or the tablet server....4
Errors during hole punching test....4
Already present: FS layout already exists....5
Troubleshooting NTP stability problems....5

Disk space usage issue....5

Performance issues....6
Kudu tracing....6
Accessing the tracing web interface....6
RPC timeout traces....6
Kernel stack watchdog traces....7
Memory limits....8
Block cache size....8
Heap sampling....9
Slow name resolution and nscd....10

Usability issues....10
ClassNotFoundException: com.cloudera.kudu.hive.KuduStorageHandler....10
Runtime error: Could not create thread: Resource temporarily unavailable (error 11)....10
Tombstoned or STOPPED tablet replicas....10
Corruption: checksum error on CFile block....11

Symbolizing stack traces....11

Recover from a dead Kudu master in a multi-master deployment....12
Prepare for the recovery....13
Perform the recovery....13

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 l
ayout: 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*.

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 issue

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
```

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.Consens
usService.UpdateConsensus
from 192.168.1.102:43499 (request call id 3555909) took 1464ms (client tim
eout 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 que
ue
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 co
mmitted 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 rece
ived 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 repl
icates 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() fi
nished
0922 00:56:52.313788 (+    24us) inbound_call.cc:114] Queueing success re
sponse
```

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:
[<ffffffffa00b209d>] do_get_write_access+0x29d/0x520 [jbd2]
[<ffffffffa00b2471>] jbd2_journal_get_write_access+0x31/0x50 [jbd2]
[<ffffffffa00fe6d8>] __ext4_journal_get_write_access+0x38/0x80 [ext4]
[<ffffffffa00d9b23>] ext4_reserve_inode_write+0x73/0xa0 [ext4]
[<ffffffffa00d9b9c>] ext4_mark_inode_dirty+0x4c/0x1d0 [ext4]
[<ffffffffa00d9e90>] ext4_dirty_inode+0x40/0x60 [ext4]
[<ffffffff811ac48b>] __mark_inode_dirty+0x3b/0x160
[<ffffffff8119c742>] file_update_time+0xf2/0x170
[<ffffffff8111c1e0>] __generic_file_aio_write+0x230/0x490
[<ffffffff8111c4c8>] generic_file_aio_write+0x88/0x100
[<ffffffffa00d3fb1>] ext4_file_write+0x61/0x1e0 [ext4]
[<ffffffff81180f5b>] do_sync_readv_writev+0xfb/0x140
[<ffffffff81181ee6>] do_readv_writev+0xd6/0x1f0
[<ffffffff81182046>] vfs_writev+0x46/0x60
[<ffffffff81182102>] sys_pwritev+0xa2/0xc0
[<ffffffff8100b072>] 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_cach e_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% * --memory_pressure_percentage) * --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 theblock_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=<ta
bles>
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_repla ce_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.

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: 0000000002398852 0000000000ad8c69 0000000000aa62ba 0
00000000221aaa8 000000000221b1a8 00000000023a8f83 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/master/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, 00000000 00a2cdb0):

```
$ readelf -S usr/lib/debug/usr/lib/kudu/sbin-release/kudu-master.debug | gre
p .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.de
bug 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/master/master_service.cc"
   starts at address 0xaa62af <kudu::master::MasterServiceImpl::ConnectToMas
ter(kudu::master::ConnectToMasterRequestPB const*, kudu::master::ConnectToMa
sterResponsePB*, kudu::rpc::RpcContext*)+47>
   and ends at 0xaa62bb <kudu::master::MasterServiceImpl::ConnectToMaster(k
udu::master::ConnectToMasterRequestPB const*, kudu::master::ConnectToMasterR
esponsePB*, kudu::rpc::RpcContext*)+59>.
```

Recover from a dead Kudu master in a multi-master deployment

Kudu multi-master deployments function normally in the event of a master loss. However, it is important to replace the dead master. Otherwise a second failure may lead to a loss of availability, depending on the number of available masters. This workflow describes how to replace the dead master.

Due to KUDU-1620, it is not possible to perform this workflow without also restarting the live masters. As such, the workflow requires a maintenance window, albeit a potentially brief one if the cluster was set up with DNS aliases.

Important:

- Replacing a master created without DNS aliases requires an unavailability window when tablet servers are restarted to pick up the replacement master at different hostname. See the multi-master migration workflow in *Migrating to multiple Kudu masters* for more details on deploying with DNS aliases.
- The workflow presupposes at least basic familiarity with Kudu configuration management. If using Cloudera Manager, the workflow also presupposes familiarity with it.
- All of the command line steps below should be executed as the Kudu UNIX user, typically kudu.

Prepare for the recovery

It is crucial to make sure that the master node is truly dead and does not accidentally restart while you are preparing for the recovery.

Procedure

1. If the cluster was configured without DNS aliases perform the following steps. Otherwise move on to step 2:
   a) Establish a maintenance window (one hour should be sufficient). During this time the Kudu cluster will be unavailable.
   b) Shut down all Kudu tablet server processes in the cluster.
2. Ensure that the dead master is well and truly dead. Take whatever steps needed to prevent it from accidentally restarting; this can be quite dangerous for the cluster post-recovery.
3. Choose an unused machine in the cluster where the new master will live. The master generates very little load so it can be co-located with other data services or load-generating processes, though not with another Kudu master from the same configuration. The rest of this workflow will refer to this master as the "replacement" master.
4. Perform the following preparatory steps for the replacement master:
   - If using the same dead master as the replacement master then delete the master's directories.
   - Ensure Kudu is installed on the machine, either via system packages (in which case the kudu and kudu-master packages should be installed), or via some other means.
   - Choose and record the directory where the master's data will live.

Perform the recovery

After you have identified a reference master, you need to copy the master data to the replacement master node. You need to bring the Kudu clusters down. Therefore, identify at least a one-hour maintenance window for this task.

Procedure

1. Remove the dead master from the Raft configuration of the master using the kudu master remove command.

   In the following example dead master master-2 is being recovered:

   ```
   $ sudo -u kudu kudu master remove master-1,master-2 master-2
   ```

2. On the replacement master host, add the replacement master to the cluster using the kudu master add command.

   Look for any success or error messages on the console or the replacement master log file. The command is designed to be idempotent so in case of an error after the issue mentioned in the error messages is fixed, run the same command again to make forward progress. After the completion of the procedure irrespective of whether the procedure is successful, the replacement master is shutdown. In the following example, replacement master mast er-2 is used. In case DNS alias is not being used, use the hostname of the replacement master.

   ```
   $ sudo -u kudu kudu master add master-1 master-2 --fs_wal_dir=/data/kudu/
   master/wal \
   --fs_data_dirs=/data/kudu/master/data
   ```

3. If the cluster was set up with DNS aliases, reconfigure the DNS alias for the dead master to point at the replacement master.
4. If the cluster was set up without DNS aliases, perform the following steps:
   a) Modify the value of the master_addresses configuration parameter for each live master removing the dead master and substituting it with the replacement master.

      The new value must be a comma-separated list of all of the masters. Each entry is a string of the form <hostname>:<port>:

      hostname

      The master's previously recorded hostname or alias

      port

      The master's previously recorded RPC port number
   b) Restart the remaining live masters.
5. Start the replacement master.
6. If the cluster was set up without DNS aliases, perform the following steps for tablet servers:
   a) Modify the value of the tserver_master_addr configuration parameter for each live master removing the dead master and substituting it with the replacement master.

      The new value must be a comma-separated list of all of the masters. Each entry is a string of the form <hostname>:<port>:

      hostname

      The master's previously recorded hostname or alias

      port

      The master's previously recorded RPC port number
   b) Restart all tablet servers.

What to do next

To verify that all masters are working properly, consider performing the following checks:

- Using a browser, visit each master's web UI and navigate to the /masters page. All the masters should now be listed there with one master in the LEADER role and the others in the FOLLOWER role. The contents of /masters on each master should be the same.
- Run a Kudu system check (ksck) on the cluster using the kudu command line tool.