Fast failover after cluster node lock-up and protection against client hangs

Executive summary

Oracle client sessions can in certain conditions (such as a locked up, but still partially active kernel) indefinitely wait for a server response rather than failing over to another active cluster node. This page documents the analysis of the issue and discusses two different approaches to insure fast failover for applications which need to insure quick failover.

- The preferred solution for highly available applications is to implement time-outs on database access in the application.
- An additional solution, which can be implemented without client application changes, is based on configuring Oracle*net to make use of TCP 'keepalive' functionality.

Failover issues

Oracle clients issuing a call against a Oracle DB server (RAC node) may wait indefinitely if the kernel is still responding to TCP packets. This is a very specific fault condition which has recently been observed more frequently. The likely cause for this conditions are bugs in the kernel or kernel modules or by faulty hardware, which are currently under investigation by RedHAT/Oracle. The symptoms are that the kernel stays up, but user processes are not scheduled on CPU any more. Low level kernel operations such as IP and TCP communications are still available (the machine will respond to ping for example) but ssh or Oracle operations are 'frozen'. Oracle clients connected to the faulty node will hang* following the cluster node freeze, because their Oracle*net requests can go through to the server without errors (TCP packets will be acknowledged) but the answer from Oracle processes will never come back (as the process is not scheduled on CPU any more). When the faulty node reboots oracle clients are not notified until they make another request. Depending on their status at the time of the node fault they may continue to wait for Oracle*net messages. One solution to this is that the client code detects exceptionally long requests eg due to network or server failure and time-out and reconnect accordingly. Oracle clients connected to **other nodes are unaffected**. Also any new Oracle*net connections to services that have failed will work fine (i.e. fail over to a healthy node immediately).

Solution 1 - Application layer

Code the client to wait only a set period of time. This is the architecturally strongest solution because it will handle all hang situations not only the particular case of locked-up kernels. There are 2 main ways that client code can be written to wait only a set period of time before taking alternative action:

- Set a timeout before any call to the database and generate a break / interrupt signal if this timeout expires. Eg: Use an Oracle8 OCIBreak()
- Use non-blocking OCI and monitor a timer whilst polling for a response. If the timer expires issue an OCIBreak()
- See also OCI_code_with_timeouts.txt

*Note: A custom **client 'watchdog'** that detects hanging client connection and restarts them automatically is also a viable alternative to consider

Solution 2 - TCP keep alive and Oracle*net client parameters

Oracle*net can be configured to use TCP keep alive functionality: Oracle clients will periodically send 'keep alive packets' on idle connection with the db server (node). When the receiving end of the DB connections

fails, keepalive will generate an error and therefore properly disconnect the client. If Oracle TAF is configured, the Oracle client will transparently reconnect to another database node. The disconnect and failover of the client happens only after the faulty node has been switched off or rebooted.

Oracle*net by default does not use keepalive when connecting to the DB. The configuration is a **client side** configuration and is done in two parts:

- add the parameter (ENABLE=BROKEN) to the db alias
  - this parameter has already been added to the 'central' cern tnsnames.ora file (i.e.no further actions required for users of the central tnsnnames.ora).
  - Clients using private copies or private db aliases (JDBC thin for example) instead will need to refresh them accordingly (see also sample_tnsnames_ora_keepalive.txt)
- tune TCP parameters for keepalive (the default is 2 hours, see below).
  - the following parameter have been tested for Linux.

```
echo 500 > /proc/sys/net/ipv4/tcp_keepalive_time
echo 10 > /proc/sys/net/ipv4/tcp_keepalive_intvl
echo 9 > /proc/sys/net/ipv4/tcp_keepalive_probes
```

Notes:

- Please note that TCP settings are system-wide, should be tested against the other applications on the client and should not be set too low with the risk of flooding the server(s) with keepalive packages.
- Kernel parameters should be set in agreement with the system administrator of the server.
- Windows clients will have similar settings in the registry.
- En example of quattor configuration (where relevant) can be found here below:

```
"/software/components/sysctl/active" = true;
"/software/components/sysctl/netDOTipv4DOTtcp_keepalive_time"   = "500";
"/software/components/sysctl/netDOTipv4DOTtcp_keepalive_intvl"  = "10";
"/software/components/sysctl/netDOTipv4DOTtcp_keepalive_probes" = "9";
```

Other network tuning parameters for TAF

- the following tcp parameters can be tuned to activate faster detection of network errors and therefore swifter TAF failover (quattor users will need to update cdb with a syntax similar to above)
- The tcp_syn_retries parameter specifies the maximum number of times that SYN packets will be sent for an active TCP connection. Tuning it reduces the delay when the target server is not available (default in Linux=5, i.e. 3 minutes)
- tcp_retries2 parameter specifies the maximum number of times a TCP packet is retransmitted before giving up on the connection (default in Linux = 15)

```
echo 2 > /proc/sys/net/ipv4/tcp_syn_retries
echo 5 > /proc/sys/net/ipv4/tcp_retries2
```

Summary

Two different solutions are proposed to insure a quick failover of Oracle clients eg under kernel lock-up conditions. Application owners should choose which solution to implement based on the specific availability and failover requirements of their application. Configuration **changes of client code and/or client machine tcp setting may be required** to implement short failover times under all conditions. We will be happy to consult application owners on the detailed implementation of the proposed solutions.

Examples and additional notes:

What can cause a server hang

- ♦ a kernel bug or a kernel driver bug can cause a system hang (see kernelhang.zip)
- ♦ Oracle 10g R2 RAC (up to 10.2.0.3 included) on dual CPU boxes can hang the server because of LMS processes spinning on CPU. This is because by default Oracle activates 2 LMS processes (GCS/cache fusion workers) in real time (RR scheduling) and apparently can they can enter a race condition/loop on CPU. The workaround is to change LMS processes to TS scheduling. On 10.2.0.3: `alter system set "_high_priority_processes"='' scope=spfile sid='*'`, then bounce the instances.

- OCI_code_with_timeouts.txt: example on how to code OCI clients with timeouts

- sample_tnsnames_ora_keepalive.txt: sample dbalias with TAF + client load balancing + ENABLE=BROKEN activated

- kernelhang.zip: This Linux kernel module generates a kernel hang - for testing purposes only

- a programmable kernel delay can be simulated with: **modprobe hangcheck-delay hangcheck_delay=** , but this causes a different hang than the kernelhang module above, which instead generates a more realistic kernel hang: for example the tcp stack is not available when hanging with hangcheck-delay, while it is available with the hang generated by kernelhang module.

This topic: PSSGroup > OCIClientHangProtection
Topic revision: r11 - 2007-02-28 - LucaCanali