Collect Data when NSO Consumes High CPU

Contents

Introduction

This document describes the Network Services Orchestrator (NSO) data collection needed when CPU consumption increases to 100-150%.

Prerequisites

Requirements

There are no specific requirements for this document.

Components Used

This document is not restricted to specific software and hardware versions.

The information in this document was created from the devices in a specific lab environment. All of the devices used in this document started with a cleared (default) configuration. If your network is live, ensure that you understand the potential impact of any command.

Background Information

When multiple transactions are processed from NB, the NSO CPU consumption increases to approximately 100-150% of normal consumption. When this happens, you need to find the cause that downgrades CPU performance. And, at the same time, the NSO does not respond to RESTCONF (if used) queries correctly.

This article highlights all the important data that needs to be collected during the problem so that the issue can be properly troubleshooted and also suggest some remedy steps.

Data to be collected

From Linux perspective:

- lscpu
- top
- free -h
- vmstat
- cat /proc/meminfo
- pstree -c
- ps auxw | sort

Note: You can capture these details (except 'lscpu') at regular intervals in order to understand how the system behaves when the requests come from NB.

From NSO perspective:

- ncs --status | grep lock
- Enable the progress trace: admin@ncs(config)# commit dry-run cli { local-node { data progress { + trace all { + destination { + file progress-all.txt; + format log; + } + } } }}admin@ncs(config)# commit

- Capture the next information every 'n' seconds (it can be run as a script):
  seq=0
  while ncs --status >& /dev/null; do
  ncs --debug-dump ncs.dd.$((seq++));
  ncs --status > ncs.stat.$((seq++));
  sleep 30; #Configured according to user
  done

Next are some remedy steps that can also be performed to mitigate the issue:

1. Limit the number of sessions as follows (presently, you don't have this set):
   <session-limits>

   <session-limit>

   <context>rest</context>

   <max-sessions>100</max-sessions>

   </session-limit>

   </session-limits>

b. Enable audit rule to ascertain if NSO process was killed by something and if in case it was, record it in audit.log:

sudo auditctl -a exit,always -F arch=b64 -S kill -k audit_kill

To troubleshoot and analyze, you need the previous details along with the audit.log, devel.log (preferably set at level=trace), ncs-java-vm.log and NB logs.

Additional Information

Q. How does NSO actually handle RESTCONF requests from a NB application?

A. When a northbound application sends a RESTCONF request, it is treated as a unique transaction based on NSO. This means that NSO can lock the entire CDB, and not allow any other transactions until the current transaction is completed.If this is done, the transactional nature of NSO is preserved and it ensures that a rollback can be done in case of any issues.

The NSO commit-queue can process each subsequent transaction request as it completes, and you can track the transaction lock in the devel.log as they start/complete. In use cases where a large amount of queries are done, this introduces a large amount of overhead in NSO; and transactions are in the commit queue for longer than expected. In case the RESTCONF requests were grouped, the throughput would increase, as the transaction overhead would be lessened. Also, NSO would be able to do as much as it can at the same time, inside of a single transaction. For example, if a transaction contains 2 device configuration changes, NSO can lock the CDB, reach out to and edit both devices at the same time, then complete the transaction This is in contrast to 2 transactions that each contain 1 device and both are changed; as NSO can lock the CDB for the first transaction, edit the first device, complete the transaction, then do the same steps for the second device.

Related Information

- **Cisco Technical Support & Downloads**