Tivoli Netcool Support's Guide to Sub-Second Triggers in a multitier environment

by
Jim Hutchinson

Document release: 2.1

Table of Contents

1 Introduction

1.1 Overview

This document was revised to provide a working example of how to configure Netcool/OMNIbus to handle sub-second problem/resolution events in the Netcool/OMNIbus multitier environment.

The two solutions presented in the document are:

- ProbeSubSecondId solution : Events created by rules files that use LastOccurrence and ProbeSubSecondId
- Sub-second solution : Events created using SQL and the ProbeSubSecondId solution

1.2 Sub-second events

Sub-second problem/resolution events are problem/resolution events that occur in the same second.

Consider the following sequence of events;

Sequence	Timestamp	Type
1	12:01:01	Problem
2	12:01:02	Resolution
3	12:01:05	Problem
4	12:01:05	Resolution
5	12:01:05	Problem
6	12:01:05	Resolution

The default object servers triggers would show the example event as a **problem event** as they are not designed to manage problem/resolution events in the same second.

As event rates from devices increases, the probability of same second problem/resolution events being managed incorrectly has increased, resulting in issues with network management.

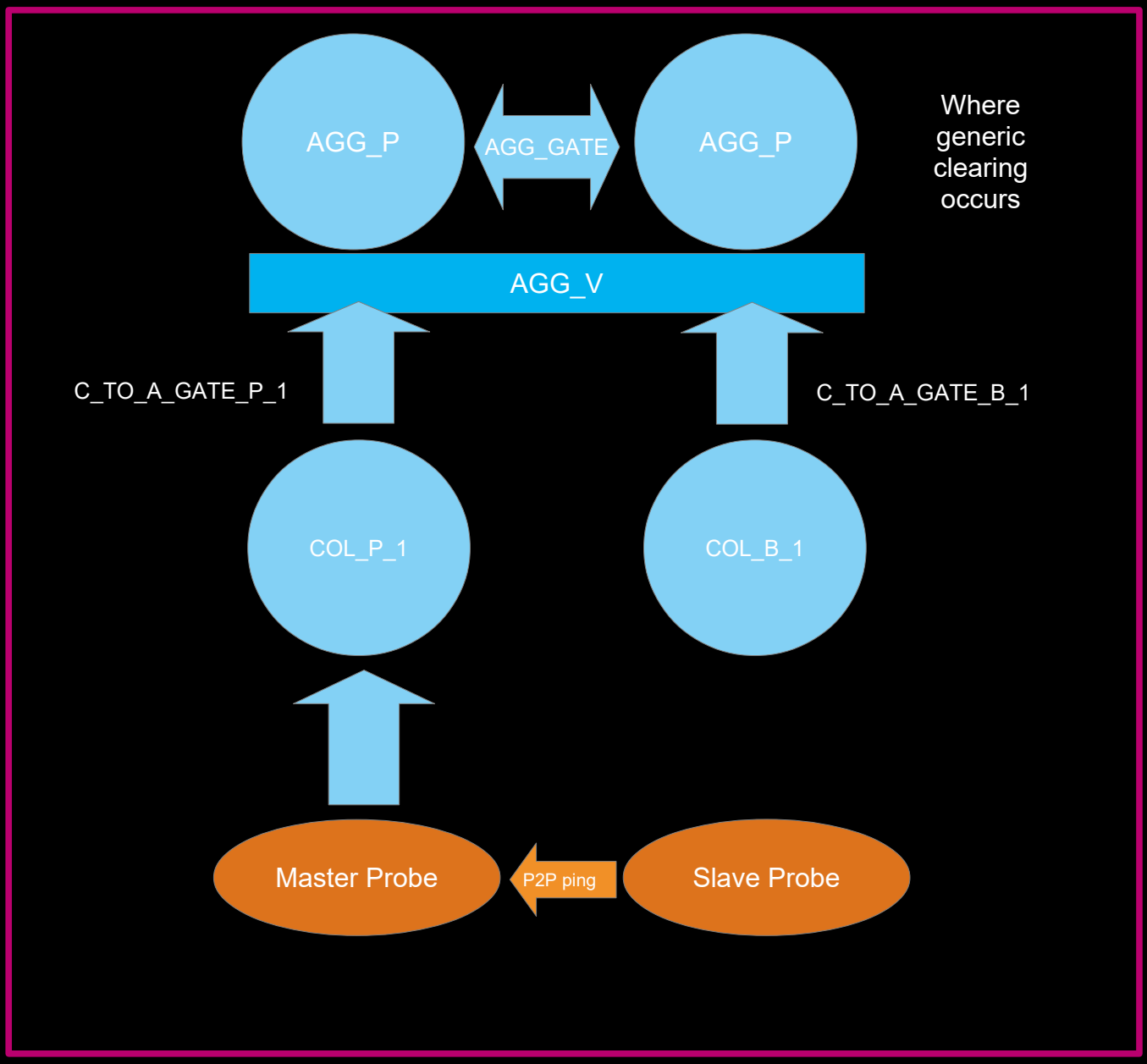
The solutions proposed in this document resolves the issue of same second problem/resolution events using counter triggers, as well as the probes rules file fields LastOccurence and ProbeSubSecondId.

When using the sub-second solutions, ensure that the event loads expected in the production environment are tested, so that the impact of the new sub-second triggers is understood and catered for.

1.3 Example Object Server Configuration

The files provided within the sub-second solution are for the Netcool/OMNIbus multitier system;

- AGG_P and AGG_B joined by a bi-directional gateway, AGG_GATE
- COL_P_1 connected to AGG_V using a uni-directional gateway, C_TO_A_GATE_P_1
- COL_B_1 connected to AGG_V using a uni-directional gateway, C_TO_A_GATE_B_1
- MTTrapd probe configured for peer-to-peer (master/slave)
- JDBC gateway which is reporting to an Oracle database



2 The Sub-second Solution

The sub-second solution uses a custom table to store the sub-second counter, this counter is updated whenever the alerts.status table is updated. During event resolution the value of the counter can then be used to decide if the problem event is resolved or not. The solution includes a sub-second journals solution that allows the custom nc_jinserts procedure to be used where sub-second journals are most probable.

2.1 Key Objects

New alerts.status Columns:-

NcEventTimeStamp : Internal timestamp at the object server
NcEventSecondCounter : Integer counter to store the current seconds event count
NcReporting : Flag used to control historical event flow

New trigger groups:-

- nc_sub_second
- nc_primary_only
- nc_historical_reporting

New custom table:-

subsecond.counter

TargetTable	varchar(64) primary key,
Counter	int,
Timestamp	time

New triggers:-

- nc_status_reinserts
- nc_status_inserts
- nc_status_updates
- nc_journal_inserts
- nc_new_row
- nc_deduplication
- nc_generic_clear
- nc_agg_delete_clears

New procedure:-

- nc_jinsert

2.2 Installation

The installation of the triggers and procedure into the object servers can be performed using nco_sql or manually using the SQL workbench. There is a README.txt file in the main sub-second installation directory discusses the solution and installation as well.

2.2.1 Object Server Configuration

To install the new triggers and procedure in the multitier object servers;

```
cat subsecond.sql | nco_sql -server -user root -password '' -server AGG_P
cat subsecond.sql | nco_sql -server -user root -password '' -server AGG_B
```

and for the collection layer object servers:

```
cat subsecond.sql | nco_sql -server -user root -password '' -server COL_P_1
cat subsecond.sql | nco_sql -server -user root -password '' -server COL_B_1
```

To configure and enable the triggers:

```
cat enable_subsecond_at_aggregation.sql | nco_sql -user root -password '' -server AGG_P
cat update_automation_procedures.sql | nco_sql -user root -password '' -server AGG_P
cat SQL/enable_nc_primay_only.sql | nco_sql -user root -password '' -server AGG_P

cat enable_subsecond_at_aggregation.sql | nco_sql -user root -password '' -server AGG_B
cat update_automation_procedures.sql | nco_sql -user root -password '' -server AGG_B

cat enable_subsecond_at_collection.sql | nco_sql -user root -password '' -server COL_P_1
cat enable_subsecond_at_collection.sql | nco_sql -user root -password '' -server COL_B_1
```

Once installed the nc_jinsert procedure must be integrated into all triggers and tools that perform journal entry where journals are at risk of being added in the same second. This is usually only for triggers rather than tools.

2.3 Object Server gateway configuration

The Collection and Aggregation layer gateways need to pass the new fields between each other to allow the Generic Clear at the Aggregation layer to work correctly.

In the example the following Object Server gateways need to be updated:

- AGG_GATE
- C_TO_A_GATE_P_1
- C_TO_A_GATE_B_1

with the new fields, as required:

```
'NcEventTimeStamp'  =   '@NcEventTimeStamp' ,
'NcEventSecondCounter'  = '@NcEventSecondCounter' ,

# Uncomment if you want to set the Historical reporting flag at the probes
# 'NcReporting'  = '@NcReporting' ,
```

The NcReporting flag is usually set in the probes rules file, but could equally be set using a custom trigger in the object servers, as required.

2.4 Testing

The sub-second installation directory includes the configuration and test scripts to check the configuration, and to allow further development, as required. Each sub-directory includes a README.txt describing how to configure and perform the individual tests. Once the dual-resilient object server and Oracle gateway are configured the system may be tested.

Included tests:-

2.4.1 Sub-second rules file problem/resolution

Use the rulesfile and raw capture file in rules sub-directory to test sub-second problem/resolution;

To create a same second problem event [the fix]:-

```
cat problem.cap | $OMNIHOME/probes/nco_p_stdin -rulesfile ./samesec.rules -server COL_P_1
```

Expected : PROBLEM event remains

To create a same second resolution event [to confirm normal operation]:-

```
ccat resolution.cap | $OMNIHOME/probes/nco_p_stdin -rulesfile ./samesec.rules -server COL_P_1
```

Expected : Event RESOLVED

Note: By default the NcReporting flag is not set, edit the samesec.rules to enable historical reporting of the test events

2.4.2 Sub-second SQL problem/resolution

Test the SQL insert problem and resolution using the given scripts:

```
./insert_resolution_problem.sh COL_P_1
```
Expected : PROBLEM event remains

```
./insert_problem_resolution.sh COL_P_1
```
Expected : Event RESOLVED

to test the historical reporting part of the integration use the scripts with _reporting:

```
./insert_resolution_problem_reporting.sh COL_P_1
```
Expected : PROBLEM event remains

```
./insert_problem_resolution_reporting.sh COL_P_1
```
Expected : Event RESOLVED

2.4.3 Sub-second and normal traps

Use the scripts provided to send sub-second and normal traps to the object server via the MTTrapd probe configured for peer-to-peer failover/failback. The scripts requires the net-snmp program snmptrap to be installed and in the test users path [see www.net-snmp.org].

sendsubsecondtrap2master : Sends the same ten TCP SNMPv1 traps to the localhost master mttrapd probes port [TCP:localhost:1621]

Expected : One event with tally/count ten

sendtrap2master : Send ten **UNIQUE** TCP SNMPv1 traps to the localhost master mttrapd probes port [TCP:localhost:1621]

Expected : Ten events with tally/count one

sendtrap2slave : Sends ten **UNIQUE** TCP SNMPv1 traps to the localhost slave mttrapd probes port [TCP:localhost:1622]

Expected : NO events seen in object server [events discarded while master probe is running]

2.5 Zip File Contents

The README.txt provides an overview of how to use the sub-second triggers.
The README.subsecond.txt describes the custom subsecond solution.

The other files are.

```
enable_subsecond_at_aggregation.sql
enable_subsecond_at_collection.sql
subsecond.sql
update_automation_procedures.sql

./GATES:
example.map

./G_JDBC:
G_JDBC.props
default
jdbc.map
jdbc.rdrwtr.tblrep.def
jdbc.startup.cmd

./G_JDBC/default:
G_JDBC.props
audit.G_JDBC.props
audit.jdbc.map
jdbc.map
jdbc.rdrwtr.tblrep.def
jdbc.startup.cmd
reporting.G_JDBC.props
reporting.jdbc.map

./G_ORAC:
CSHRC-10g
README.txt
nco_g_icmd.props
nco_g_oracle.map
nco_g_oracle.props
nco_g_oracle.startup.cmd
nco_g_oracle.thosts

./example-P2P:
README.txt
mttrapd_master.props
mttrapd_p2p.rules
mttrapd_slave.props
utils

./example-P2P/utils:
sendproblemtraps
sendresolutiontraps
sendsubsecondtrap2master
sendtrap2master
sendtrap2slave
```

```
./example-ProbeSubSecondId:
README.txt
mttrapd.rules

./test-rules:
README.txt
problem.cap
resolution.cap
samesec.rules

./test-sql:
README.txt
insert_problem.sh
insert_problem_resolution.sh
insert_problem_resolution_reporting.sh
insert_resolution.sh
insert_resolution_problem.sh
insert_resolution_problem_reporting.sh
```

The default ProbeSubSecondId solution example:

```
./PSSI-SQL:
README.pssi.txt
pssi_generic_clear.sql
disable_pssi_generic_clear.sql
enable_pssi_generic_clear.sql
generic_clear.sql
```

3 The default ProbeSubSecondId Solution

The ProbeSubSecondId field is the default field used to hold the sub-second counter. By default LastOccurrence is used, as LastOccurrence and ProbeSubSecondId are set by probes automatically for all supported versions of Netcool/OMNIbus. Therefore, event times from the element managers and event sources should be stored in custom fields, and custom event resolution used, if required.

The ProbeSubSecondId solution requires;

- Events to arrive at the probe in sequence
- LastOccurrence to be set by the probe, use getdate for pre-v7.3.1
- ProbeSubSecondId to be set to a unique sequential number within each second

In this way the sequence of problem/resolution events can be managed.

3.1 Considerations

3.1.1 Default Behaviour

In Netcool/OMNIbus v7.3.0, the default behaviour is to have the fields updated at the probe. There are no rulesfile code required as the probe populates the values automatically;

@LastOccurrence is set to an internal timestamp

@ProbeSubSecondId is set to a sequential integer for each second

The deduplication trigger then uses the ProbeSubSecondId number to determine if an event is new or old, and discards it if it is older than the one stored already in the object server in the same second.

It is important to ensure that either the default behaviour, or the custom solutions are used. Using BOTH methods will result in event loss.

3.1.2 Normal Behaviour

Nowadays events use the probes timestamp to determine event sequencing, which is stored in the LastOccurrence field. Check the probes rules file to see if a probe $TOKEN is used to set LastOccurrence. If this is the case, then create a custom field to hold the Events Time , e.g. **@NcEventTime**.

If LastOccurrence is unset within the probes rules file, it will get set to the ‘getdate’ value automatically in Netcool/OMNIbus v7.3.x or above, and to zero in earlier versions.

Within the Generic Clear trigger the event source [**@Node**] is used to differentiate problem/resolutions. Therefore in the traditional case, same second events at the event source are less probable.

It is therefore the decision of the network administrator to determine the likelihood of same second events and the impact on the ability of the object server to monitor the system accurately.

3.2 Rulesfile code [pre-Netcool/OMNIbus v7.3.x]

The following rules file code can be used in probe rules files where LastOccurrence and ProbeSubSecondId are not set by Netcool/OMNIbus.

```
# Set the time field NcEventTimeStamp to be used in the Generic Clear trigger to getdate
@LastOccurrence = getdate

# For pre-Netcool/OMNIbus v7.3.x need to set ProbeSubSecondId
if (match(%LastOccurrence,@LastOccurrence))
{
    %SubSecondCounter = int(%SubSecondCounter) + 1
}
else
{
    %LastOccurrence = @LastOccurrence
    %SubSecondCounter = 1
}

@ProbeSubSecondId = %SubSecondCounter
```

3.3 System updates

The file pssi_generic_clear.sql, is provided to create a Generic Clear trigger that takes account of the ProbeSubSecondId counter. The new trigger is called pssi_generic_clear and is inserted into the primary_only trigger group, disabled. To use you must disable the default generic_clear trigger and enable the new pssi_generic_clear in all object servers where problem/resolution needs to be active [by default AGG_P/AGG_B].

The pssi_generic_clear trigger needs to be used in all of the object server's where Generic Clear is to occur.

The SQL file also adds a new column ProbeSubSecondId in the alerts.problem_events table for use by the pssi_generic_clear trigger:

```
alter table alerts.problem_events add ProbeSubSecondId int;
go
```

The new Generic Clear trigger takes account of the ProbeSubSecondId as well as the LastOccurrence using additional logic, other than this it behaves the same as the default Generic Clear using a temporary table called alerts.problem_events which is used to hold all of the problem events [@Type=1]. It is important to be aware that the behaviour of the Generic Clear trigger is to process problem events as quickly as possible, which means that events designated as problem events [@Type=1] must have a corresponding resolution event [@Type=2]. With the Network Operations team actively seeking, and closing problem events.

If there are large volumes of standing problem events [@Type=1] the Generic Clear trigger becomes ineffective and the alerts.problem_events becomes unnecessarily large, which will cause issues with memory and other temporal triggers.

3.3.1 pssi_generic_clear SQL

The following SQL is provided to allow the default generic_clear trigger to be replaced by the pssi_generic_clear trigger that takes account of the ProbeSubSecondId value:

```
alter table alerts.problem_events add ProbeSubSecondId int;
go

create or replace trigger pssi_generic_clear
group primary_only
priority 1
comment 'Generic Problem/Resolution using LastOccurrence and ProbeSubSecondId'
every 5 seconds
begin
        -- Populate a table with Type 1 events corresponding to any uncleared Type 2 events
        for each row problem in alerts.status where
            problem.Type = 1 and problem.Severity > 0 and
            (problem.Node + problem.AlertKey + problem.AlertGroup + problem.Manager) in
            ( select Node + AlertKey + AlertGroup + Manager
               from alerts.status where Severity > 0 and Type = 2 )
        begin
                insert into alerts.problem_events
                (Identifier,LastOccurrence,AlertKey,AlertGroup,Node,Manager,Resolved,ProbeSubSecondId)
                values ( problem.Identifier, problem.LastOccurrence, problem.AlertKey, problem.AlertGroup,
                problem.Node, problem.Manager, false, problem.ProbeSubSecondId );

        end;

        -- For each resolution event, mark the corresponding problem_events entry as resolved
        -- and clear the resolution
        for each row resolution in alerts.status where resolution.Type = 2 and resolution.Severity > 0
        begin

-- Update problem event to Resolved
                update alerts.problem_events set Resolved = true where
                (LastOccurrence < resolution.LastOccurrence and
                Manager = resolution.Manager and Node = resolution.Node and
                AlertKey = resolution.AlertKey and AlertGroup = resolution.AlertGroup) or
                (LastOccurrence = resolution.LastOccurrence and
                ProbeSubSecondId < resolution.ProbeSubSecondId and
                Manager = resolution.Manager and Node = resolution.Node and
                AlertKey = resolution.AlertKey and AlertGroup = resolution.AlertGroup);
-- Set resolution Severity to cleared
                set resolution.Severity = 0;

        end;

        -- Clear the resolved events
        for each row problem in alerts.problem_events where problem.Resolved = true
        begin
                update alerts.status via problem.Identifier set Severity = 0;
        end;

        -- Remove all entries from the problems table
        delete from alerts.problem_events;
end;
go

alter trigger pssi_generic_clear set enabled false;
go
-- End of file
```

3.3.2 Multitier considerations

Check that LastOccurrence and ProbeSubSecondId are passed by the object server gateways in the Collection and Aggregation layers.

i.e.

```
        'LastOccurrence'    =    '@LastOccurrence',
        'ProbeSubSecondId' =    '@ProbeSubSecondId',
```

Ensure that the ServerName and ServerSerial are preserved between layers and object servers.

4 Example nco_pa.conf

The following nco_pa.conf file is for an aggregation layer test system.

```
nco_process 'AGG_P'
{
        Command '$OMNIHOME/bin/nco_objserv -name AGG_P -pa NCO_PA' run as 'nrv731'
        Host            =       'localhost'
        Managed         =       True
        RestartMsg      =       '${NAME} running as ${EUID} has been restored on ${HOST}.'
        AlertMsg        =       '${NAME} running as ${EUID} has died on ${HOST}.'
        RetryCount      =       0
        ProcessType     =       PaPA_AWARE
}

nco_process 'AGG_B'
{
        Command '$OMNIHOME/bin/nco_objserv -name AGG_B -pa NCO_PA' run as 'nrv731'
        Host            =       'localhost'
        Managed         =       True
        RestartMsg      =       '${NAME} running as ${EUID} has been restored on ${HOST}.'
        AlertMsg        =       '${NAME} running as ${EUID} has died on ${HOST}.'
        RetryCount      =       0
        ProcessType     =       PaPA_AWARE
}
nco_process 'AGG_GATE'
{
        Command '$OMNIHOME/bin/nco_g_objserv_bi -propsfile $OMNIHOME/gates/AGG_GATE/AGG_GATE.props' run as
'nrv731'
        Host            =       'localhost'
        Managed         =       True
        RestartMsg      =       '${NAME} running as ${EUID} has been restored on ${HOST}.'
        AlertMsg        =       '${NAME} running as ${EUID} has died on ${HOST}.'
        RetryCount      =       0
        ProcessType     =       PaPA_AWARE
}

nco_process 'MTTRAPD_MASTER'
{
        Command '$OMNIHOME/probes/nco_p_mttrapd -propsfile $OMNIHOME/probes/solaris2/mttrapd_master.props' run
as 'nrv731'
        Host            =       'localhost'
        Managed         =       True
        RestartMsg      =       '${NAME} running as ${EUID} has been restored on ${HOST}.'
        AlertMsg        =       '${NAME} running as ${EUID} has died on ${HOST}.'
        RetryCount      =       0
        ProcessType     =       PaPA_AWARE
}
nco_process 'MTTRAPD_SLAVE'
{
        Command '$OMNIHOME/probes/nco_p_mttrapd -propsfile $OMNIHOME/probes/solaris2/mttrapd_slave.props' run as
'nrv731'
        Host            =       'localhost'
        Managed         =       True
        RestartMsg      =       '${NAME} running as ${EUID} has been restored on ${HOST}.'
        AlertMsg        =       '${NAME} running as ${EUID} has died on ${HOST}.'
        RetryCount      =       0
        ProcessType     =       PaPA_AWARE
}

nco_service 'Core'
{
        ServiceType     =       Master
        ServiceStart    =       Auto
        process 'AGG_P' NONE
        process 'AGG_B' NONE
        process 'AGG_GATE' NONE
        process 'MTTRAPD_MASTER' NONE
        process 'MTTRAPD_SLAVE' NONE
}
```

```
nco_service 'InactiveProcesses'
{
        ServiceType     =       Non-Master
        ServiceStart    =       Non-Auto
}

nco_routing
{
        host 'localhost' 'NCO_PA'
}
```