A Rule For Zabbix Trigger Dependencies

James Pearce, BEng. BBA.
Sir Technology, Perth, Western Australia
james.pearce@sirtech.com.au

Abstract

If the timing for dependent triggers are not setup correctly in Zabbix, the triggers may not fire in the expected order when an upstream event occurs. A rule is needed to guarantee trigger order. Through experimental analysis, such a rule has been developed and tested in the 'worst case' scenario for a trigger dependency configuration in Zabbix.

The Issue

- Zabbix triggers in version 2.0.1 are based on currently stored values of items
- The expectation Sir Technology had during trials is that Zabbix will 'know' about trigger dependencies and will handle the item polling accordingly i.e. possibly checking slave dependent items immediately to obtain a currently result when a master trigger is given a reason to trigger.
- There is no automatic handling of trigger dependencies that will ensure that dependent values are actually current.

Objective

The desirable behavior we are trying to achieve is such that, even in a worst-case scenario, trigger A will fire first. This requires that the trigger for A be allowed to 'see' enough failures on the item to fire before the trigger for B does.

Network Environment Setup

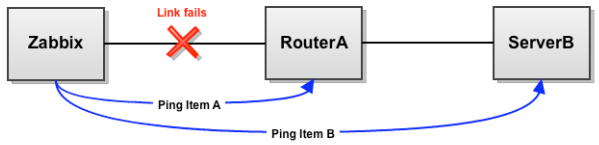

Figure 1: The network environment setup for the tests

Scenario Configuration and Results Overview

#	P(A)	P(B)	T(A)	T(B)	Result	Notes	Fig.
1	30	60	60	90	OK		N/A
2	30	30	60	90	OK		5
3	30	30	61	90	MAYBE	Creates a race condition	6
4	30	60	60	60	FAIL		N/A
5	20	60	60	60	FAIL		4
6	30	30	90	90	OK		N/A
7	60	60	0	0	FAIL	Immediate trigger	7
8	30	30	*	*	FAIL	*Trigger after 2 failed events	N/A
9	30	60	*	*	OK	*Trigger after 2 failed events	N/A

Table 1: Scenario Configuration and Results

Zabbix Notes

The way Zabbix actually handles time calculation in Zabbix has not been represented in the test setup. Zabbix only checks trigger values every 30 seconds for time based values. Zabbix also calculates time elapsed based on the number of event polls that have passed e.g. 4 events polled at 30 second polling intervals is seen to be 120 seconds of elapsed time in the eyes of Zabbix. In reality this is 90 seconds until just before the 5th value is polled. Because these rules do not change the relationship between the triggers, the rules in this paper still function correctly.

Diagramming Standard

By depicting each scenario in a fixed scale timeline, it is easier to visualize the relationship between the polling time, the trigger, and how Zabbix is 'seeing' these variables amongst multiple configuration items.

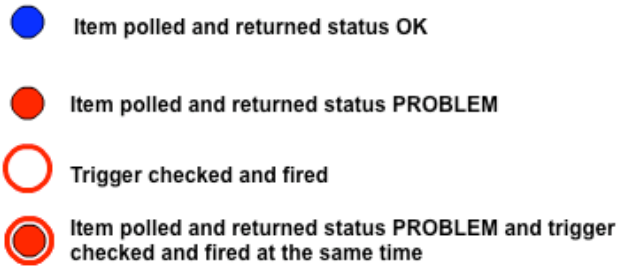

Figure 2: Diagram key

In each diagram, the first polling event that occurs for Item A is at 9 seconds and the first polling event for Item B occurs at 10 seconds. The link failure depicted in Figure 1 occurs at 10 seconds also. To be consistent with a worst-case scenario, the initial poll for Item B in each scenario run will always occur just after the link failure, and so return a failure.

- P(X) represents the polling time in seconds of item X
- T(X) represents the amount of time event X must be past a threshold before the trigger fires (referred to as the "trigger time"). In Zabbix this is implemented as:
  - `Event.min(30) > 2` for a trigger to wait 30 seconds before firing when a value has been too high for too long
  - `Event.max(30) < 2` for a trigger to wait 30 seconds before firing when a value has been too low for too long

"Default" Zabbix Configuration

By default, when creating a trigger in Zabbix, it will be set to fire as soon as an event that breaches a threshold value is reached. This is represented in Zabbix as:

```
Event.last(0).value > 2
```

When using an immediate trigger, there is no adjustment on the polling frequency that can compensate for the worst-case scenario: The polling times are out of sync and Ping Item B is the first to see a failure. This is scenario #7. A variation of this is when the polling occurs at the same time. Which trigger will fire in this case? This is an undefined state that would be determined at runtime and cannot be planned for (i.e. a race condition). Scenario #3 has this problem also.

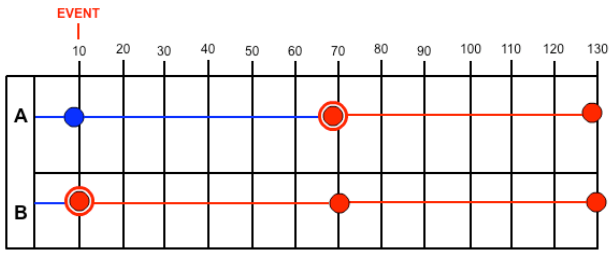

Figure 3: Scenario #7, Immediate Triggers

Increasing The Polling Frequency

The two variables available to us for adjustment in Zabbix are the polling frequency, and the amount of time an item must be in the "PROBLEM" state before a trigger fires.

An initial reaction to how to attain the objective may be: "If we make the polling frequency for Item A sufficiently fast relative to Item B, it will have more time to trigger than Item B, and so therefore solve the problem". This does not work. The problem with this is that there is no way of guaranteeing the sequence of polling between A and B when an event occurs. Making Item A poll every 20s and Item B poll every 60s does not guarantee that Item A will be polled first when an event occurs. As long as the trigger time for both items is the same, if Item B polls before A, it will simply mean a delay before Item B's trigger fires first (scenario #5).

This gives us a principle to work with: **When both trigger times are the same, it is not possible to force a trigger to fire first simply by making it poll faster.**

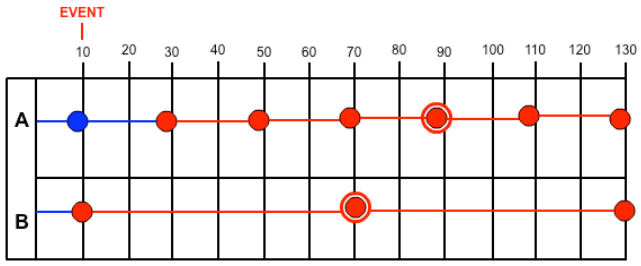

Figure 4: Scenario #5,

Decreasing The Trigger Time

Adjusting the second variable in isolation will work depending on the values chosen. In our working scenario #2, we see that even with an equal polling frequency for both A and B, it is possible to guarantee that the trigger for A will fire first. This gives us a second principle to work with:

By making the trigger time shorter when the polling frequencies are equal, it is possible to force which trigger fires first.

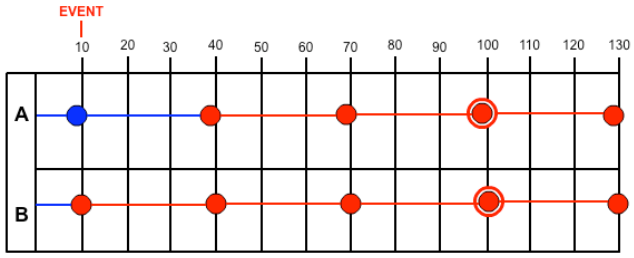

Figure 5: Scenario #2

Breaking The Working Model

By adjusting the time for trigger A one second at a time, it is possible to see what kind of margins exist in scenario #4. When the trigger time for Item A is set to 61 seconds, the model breaks. This is because there is only a 1 second gap between when the 2 triggers fire in scenario #1. The race condition that is created prevents us from being sure of which trigger will fire first. The broken model is represented as scenario 3. Adjusting the trigger for Item A to 62 seconds would definitely result in the trigger for Item B firing first.

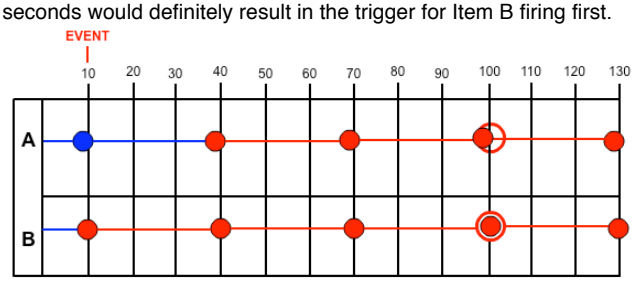

Figure 6: Scenario #3,

Trigger / Polling Time Relationship

In a real world deployment, the polling times of item A and B will not always be equal. It is necessary to know if the principle of adjusting the trigger time can still be used in this environment. Specifically: given a particular set of polling frequencies for item A and item B, how can we know what to set the trigger time values to?

When the trigger time for both items and the polling time for both items are the same, the problem is the delay until item A is polled again. This "lag time" is depicted by the blue line in scenario #7. When the trigger time is adjusted in scenario #3, the adjustment effectively compensates for the lag at the beginning. There is a 30 second lag at the beginning for item A where Zabbix see's the item as "OK", when it is really not, but the trigger time is 30s shorter.

Algorithmically we can see that it is possible to bring the time that trigger A fires to the same time as trigger B with the following formula:

T(A) = T(B) – P(A)

I.e. we make trigger A the same as trigger B, then compensate for the delay that A incurs because of it's polling frequency by making it shorter by that amount.

This formula simply gives us values for the trigger times to ensure that trigger A will fire at the same time as trigger B. To obtain the objective of ensuring trigger A fires first, we must alter the equation in the formula. A re-arrangement to calculate T(B) is also possible.

T(A) < T(B) – P(A)
T(B) > T(A) + P(A)

As the minimum step size in Zabbix for trigger times is 1 second, this can be achieved by simply solving one of the above as an equality and subtracting or adding an integer value C to the result. With a value for C of 1, trigger A will always fire at least 1 second before trigger B. This value C creates the "margin" between when the two triggers fire as evidenced in scenario #2. With a 1 second margin, a longer-than-usual poll or too much system load on the Zabbix server could potentially cause a delay between the actual poll and the item update in Zabbix resulting in scenario #2 turning into a race conditions in scenario #3, or even a complete failure.

Using Discrete Events as Triggers

When using only the discrete polling events, trigger time is measure in increments of the polling time, irrespective of what the polling times actually are. Previous experiment tests are based on a trigger which uses the currently stored value of an item, even between polling events. That is, we only test the trigger when an item is polled. In this case, we might say the trigger is "after 2 events" or, for an immediate trigger, "after 1 event". Scenario #7 where the trigger time is 0 seconds represents the latter. The implementation in Zabbix for this kind of trigger is:

```
Event.last(0).value > 2 || event.last(1).value > 2
```

From the previous rule that we know that the trigger time for B must be long enough for the polling of A to 'make up' the time lag, In other words, we must find the rule that causes trigger A to always fire at the same time as trigger B, and then make it less than this value. With discrete events only, this is calculable event.

Trigger A fires after: T(A) x P(A)
Trigger B fires after: T(B) x P(B)
Equalizing trigger fire point: T(A) x P(A) = : T(B) x P(B)
Theoretical rule: P(A) < P(B) x T(B) / T(A)

Testing of this theory yielded a flaw because the discrete events are measured in whole polling cycles, but the calculation for the polling time in the equation yields a time value in seconds. A simple solution to compensate for this is to adjust the rule to the following:

The calculated value of P(A) must be lowered to the nearest multiple of P(B) **Or,**

For a calculation of P(B), the value of P(B) must be raised to the nearest multiple of P(A).

Conclusions

The rule for ensuring trigger A fires before trigger B are as follows:

- Based on timing of triggers (seconds): T(A) < T(B) - P(A)
- Based on sequencing of triggers (events): P(A) < P(B) x T(B) / T(A)

It is expected that these rules should successfully scale past a single dependency if they are adapted appropriately, but this has not been tested.

It is also expected that these rules work for the 1.8.x branch of Zabbix as the architecture seems to be the same, but this also has not been tested.