7 Predictive trigger functions

Overview

Sometimes there are signs of the upcoming problem. These signs can be spotted so that actions may be taken in advance to prevent or at least minimize the impact of the problem.

Zabbix has tools to predict the future behaviour of the monitored system based on historic data. These tools are realized through predictive trigger functions.

1 Functions

Two things one needs to know is how to define a problem state and how much time is needed to take action. Then there are two ways to set up a trigger signalling about a potential unwanted situation. First: trigger must fire when the system after “time to act” is expected to be in problem state. Second: trigger must fire when the system is going to reach the problem state in less than “time to act”. Corresponding trigger functions to use are **forecast** and **timeleft**. Note that underlying statistical analysis is basically identical for both functions. You may set up a trigger whichever way you prefer with similar results.

2 Parameters

Both functions use almost the same set of parameters. Use the list of supported functions for reference.

2.1 Time interval

First of all you should specify the historic period Zabbix should analyse to come up with prediction. You do it in a familiar way by means of `sec` or `#num` parameter and optional `time_shift` like you do it with **avg**, **count**, **delta**, **max**, **min** and **sum** functions.

2.2 Forecasting horizon

(**forecast** only)
Parameter `time` specifies how far in the future Zabbix should extrapolate dependencies it finds in historic data. No matter if you use `time_shift` or not, `time` is always counted starting from the current moment.

2.3 Threshold to reach

(**timeleft** only)
Parameter `threshold` specifies a value the analysed item has to reach, no difference if from above or from below. Once we have determined f(t) (see below) we should solve equation f(t) = `threshold`

and return the root which is closer to now and to the right from now or 999999999999.9999 if there is no such root.

When item values approach the threshold and then cross it, **timeleft** assumes that intersection is already in the past and therefore switches to the next intersection with `threshold` level, if any. Best practice should be to use predictions as a complement to ordinary problem diagnostics, not as a substitution.[1)]

2.4 Fit functions

Default `fit` is the *linear* function. But if your monitored system is more complicated you have more options to choose from.

fit	**x = f(t)**
linear	x = a + b*t
polynomialN[2)]	$x = a_0 + a_1*t + a_2*t^2 + ... + a_n*t^n$
exponential	x = a*exp(b*t)
logarithmic	x = a + b*log(t)
power	$x = a*t^b$

2.5 Modes

(**forecast** only)
Every time a trigger function is evaluated it gets data from the specified history period and fits a specified function to the data. So, if the data is slightly different the fitted function will be slightly different. If we simply calculate the value of the fitted function at a specified time in the future you will know nothing about how the analysed item is expected to behave between now and that moment in the future. For some `fit` options (like *polynomial*) a simple value from the future may be misleading.

mode	**forecast result**
value	f(now + `time`)
max	$max_{now <= t <= now + time}$ f(t)
min	$min_{now <= t <= now + time}$ f(t)
delta	*max - min*
avg	average of f(t) (now <= t <= now + `time`) according to definition

3 Details

To avoid calculations with huge numbers we consider the timestamp of the first value in specified period plus 1 ns as a new zero-time (current epoch time is of order 10^9, epoch squared is 10^{18}, double precision is about 10^{-16}). 1 ns is added to provide all positive time values for *logarithmic* and *power* fits which involve calculating log(t). Time shift does not affect *linear*, *polynomial*, *exponential* (apart from easier and more precise calculations) but changes the shape of *logarithmic* and *power* functions.

4 Potential errors

Functions return -1 in such situations:

- specified evaluation period contains no data;
- result of mathematical operation is not defined[3)];
- numerical complications (unfortunately, for some sets of input data range and precision of double-precision floating-point format become insufficient)[4)].

No warnings or errors are flagged if chosen fit poorly describes provided data or there is just too few data for accurate prediction.

5 Examples and dealing with errors

To get a warning when you are about to run out of free disk space on your host you may use a trigger expression like this:

```
{host:vfs.fs.size[/,free].timeleft(1h,,0)}<1h
```

However, error code -1 may come into play and put your trigger in a problem state. Generally it's good because you get a warning that your predictions don't work correctly and you should look at them more thoroughly to find out why. But sometimes it's bad because -1 can simply mean that there was no data about the host free disk space obtained in the last hour. If you are getting too many false positive alerts consider using more complicated trigger expression[5)]:

```
{host:vfs.fs.size[/,free].timeleft(1h,,0)}<1h and
{host:vfs.fs.size[/,free].timeleft(1h,,0)}<>-1
```

Situation is a bit more difficult with **forecast**. First of all, -1 may or may not put the trigger in a problem state depending on whether you have expression like

```
{host:item.forecast(...)}<...
```

or like

```
{host:item.forecast(...)}>...
```

Furthermore, -1 may be a valid forecast if it's normal for the item value to be negative. But probability of this situation in the real world situation is negligible (see how operator **=** works). So add

```
... or {host:item.forecast(...)}=-1
```

or

```
... and {host:item.forecast(...)}<>-1
```

if you want or don't want to treat -1 as a problem respectively.

See also

1. Predictive trigger functions (pdf) on zabbix.org

1)

For example, a simple trigger like

```
{host:item.timeleft(1h,,X)} < 1h
```

may go into problem state when the item value approaches X and then suddenly recover once value X is reached. If the problem is item value being below X use:

```
{host:item.last()} < X or {host:item.timeleft(1h,,X)} < 1h
```

If the problem is item value being above X use:

```
{host:item.last()} > X or {host:item.timeleft(1h,,X)} < 1h
```

2)

Polynomial degree can be from 1 to 6, *polynomial1* is equivalent to *linear*. However, use higher degree polynomials with caution. If the evaluation period contains less points than needed to determine polynomial coefficients, polynomial degree will be lowered (e.g *polynomial5* is requested, but there are only 4 points, therefore *polynomial3* will be fitted).

3)

For example fitting *exponential* or *power* functions involves calculating log() of item values. If data contains zeros or negative numbers you will get an error since log() is defined for positive values only.

4)

For *linear*, *exponential*, *logarithmic* and *power* fits all necessary calculations can be written explicitly. For *polynomial* only *value* can be calculated without any additional steps. Calculating *avg* involves computing polynomial antiderivative (analytically). Computing *max*, *min* and *delta* involves computing polynomial derivative (analytically) and finding its roots (numerically). Solving f(t) = 0 involves finding polynomial roots (numerically).

5)

But in this case -1 can cause your trigger to recover from the problem state. To be fully protected use:

```
{host:vfs.fs.size[/,free].timeleft(1h,,0)}<1h and ({TRIGGER.VALUE}=0 and
{host:vfs.fs.size[/,free].timeleft(1h,,0)}<>-1 or {TRIGGER.VALUE}=1)
```