experiment

May 10, 2023

1 Test Inspektor Gadget scaling with `exec gadgettracermanager`

The goal of this experiment is to test until which number of nodes Inspektor Gadget scales when using `exec gadgetttracermanager` to get the events stream.

1.1 Experimental environment

1.1.1 Hardware environment

Inspektor Gadget will be tested on an AKS cluster using `Standard_DS2_v2` as node Azure image (*i.e* 2 CPU cores and 7 GB of DRAM). The number of nodes will vary over the experience with the following values: 2, 5, 11, 17, 23, 31 and 39. The number of nodes were chosen to be one over two prime number until 39 (which is non prime though).

1.1.2 Software environment

To test scaling, the idea is to deploy a pod on each nodes which will generate a lot of `exec()` and to count the number of `exec()` events reported by Inspektor Gadget. So, on each node, a pod running `stress-ng` is deployed to generate as many `exec()` it can during 1 second on all the node CPU (in our case 2). Before that, Inspektor gadget is deployed on the cluster to monitor all these pods (which belong to the same namespace). The experiment is described in `script.sh`.

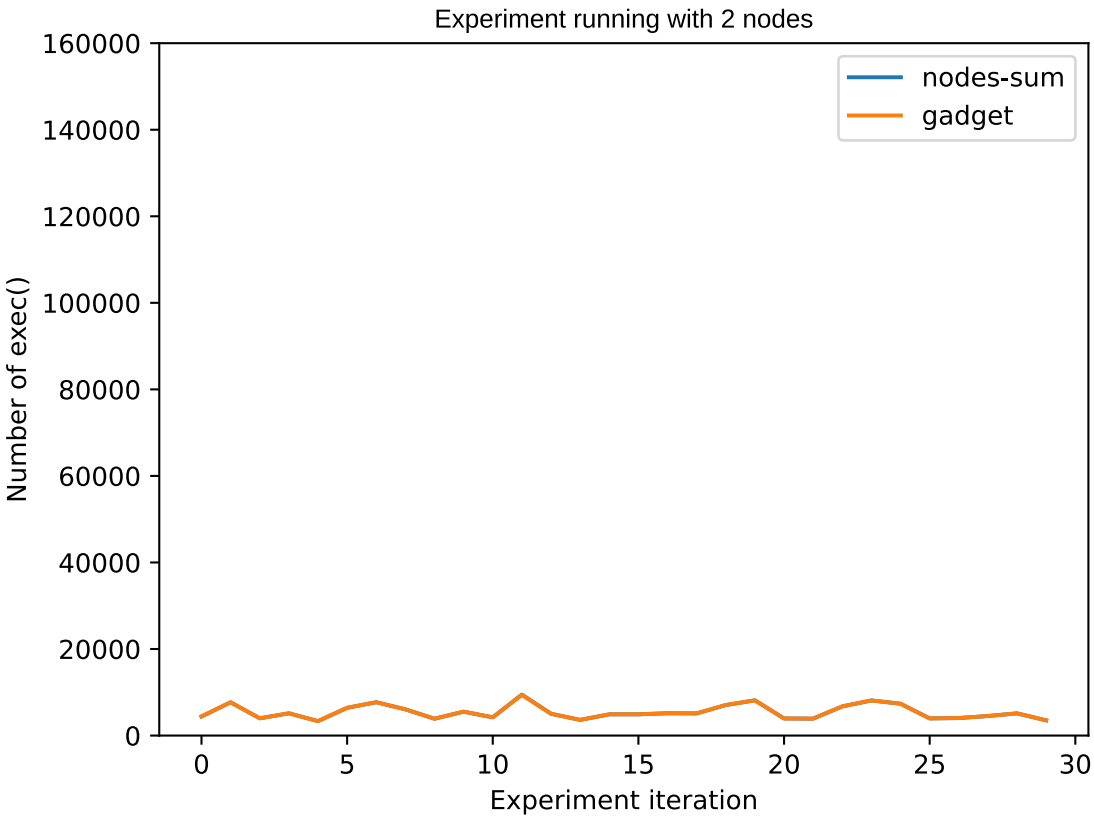
1.1.3 Statistics

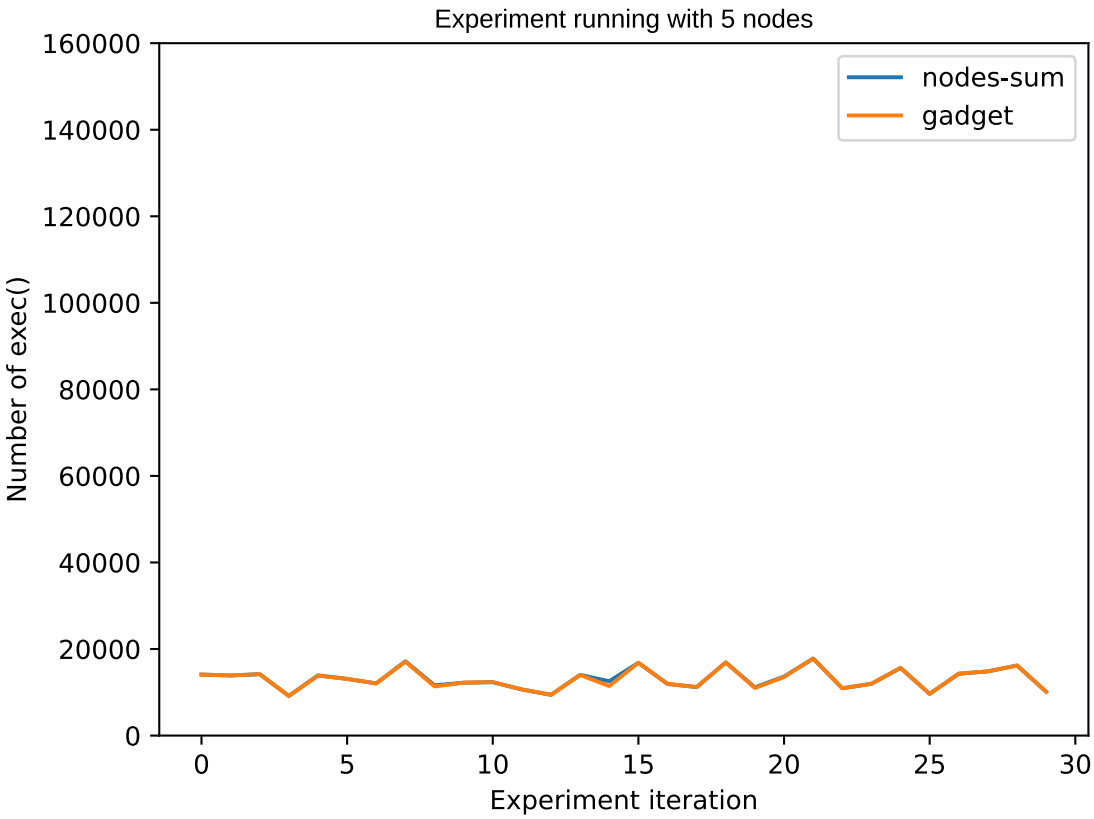
For each number of node, the above experiment is run 30 times to compute some statistics.

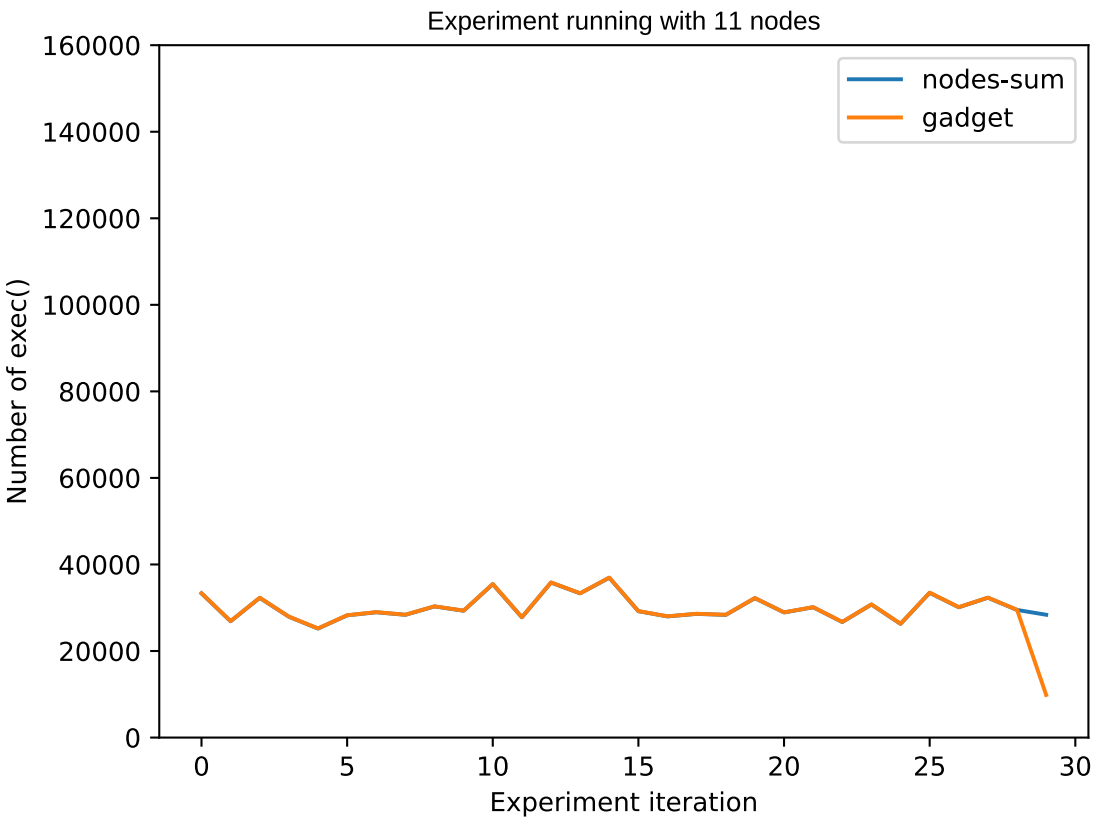
1.2 Results

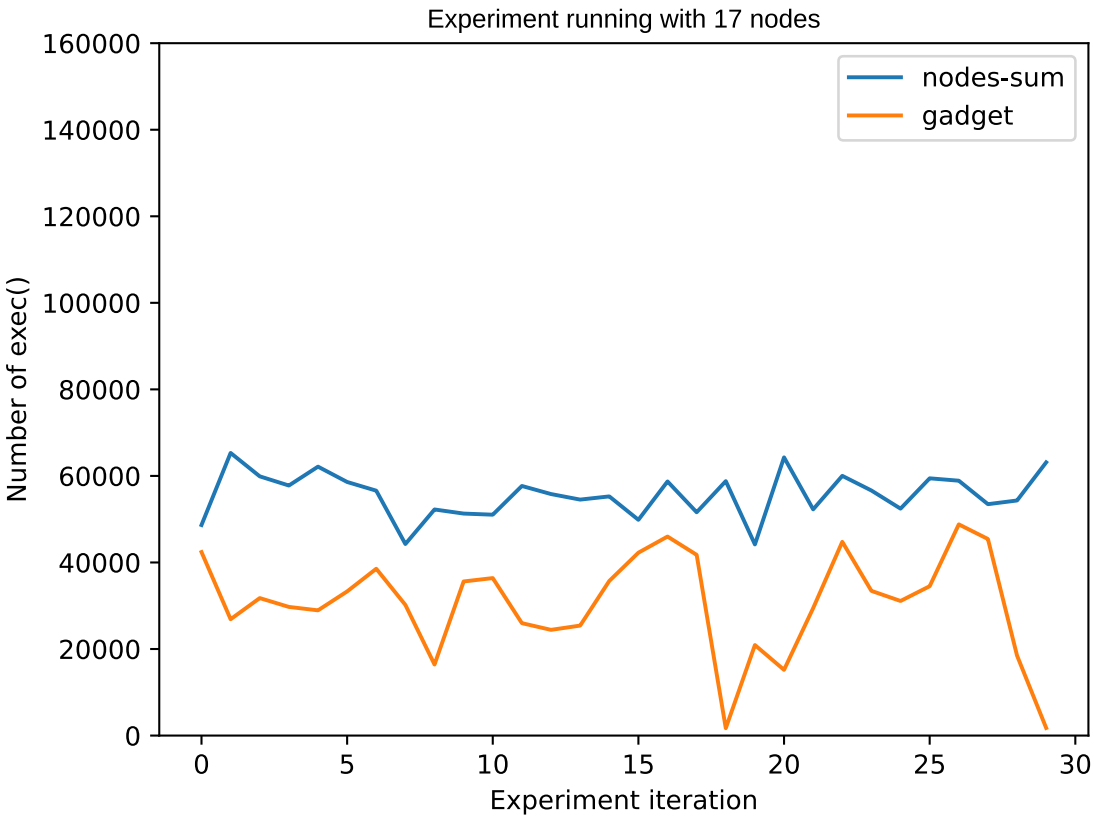
The results are depicted in the following graphs. The `x` axis indicates the experiment number from 1 to 30 while the `y` axis indicates the number of `exec()`. Each graph presents two curves:

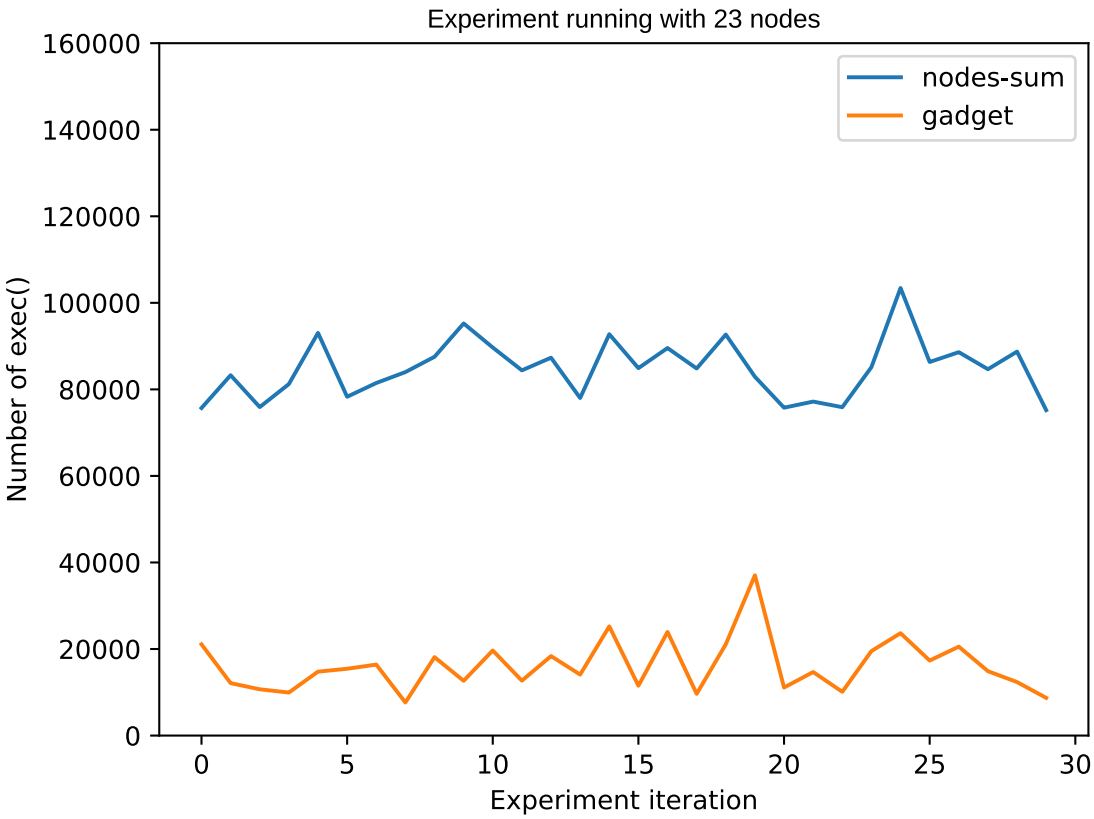
1. One which is the sum of all `exec()` generated for all nodes.
2. The other which is the number of `exec()` reported by Inspektor Gadget.

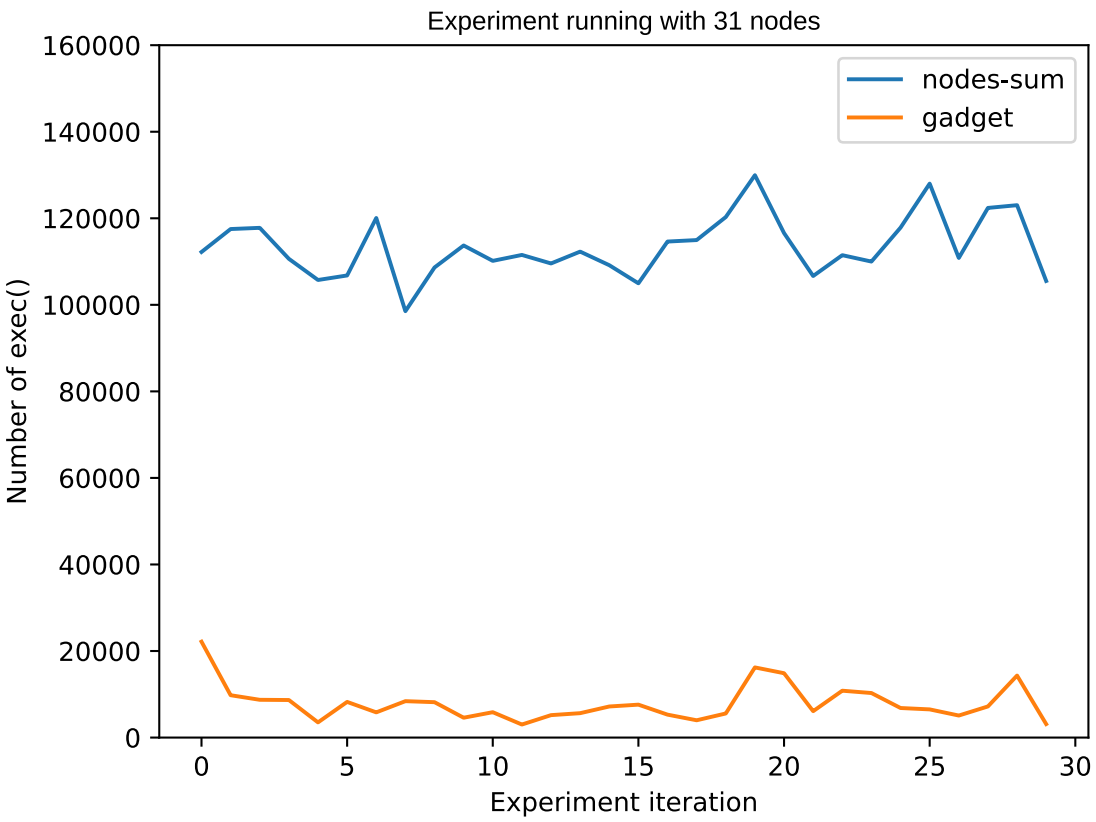

Experiment running with 2 nodes
nodes-sum
gadget
Number of exec()
160000
140000
120000
100000
80000
60000
40000
20000
0
0
5
10
15
20
25
30
Experiment iteration

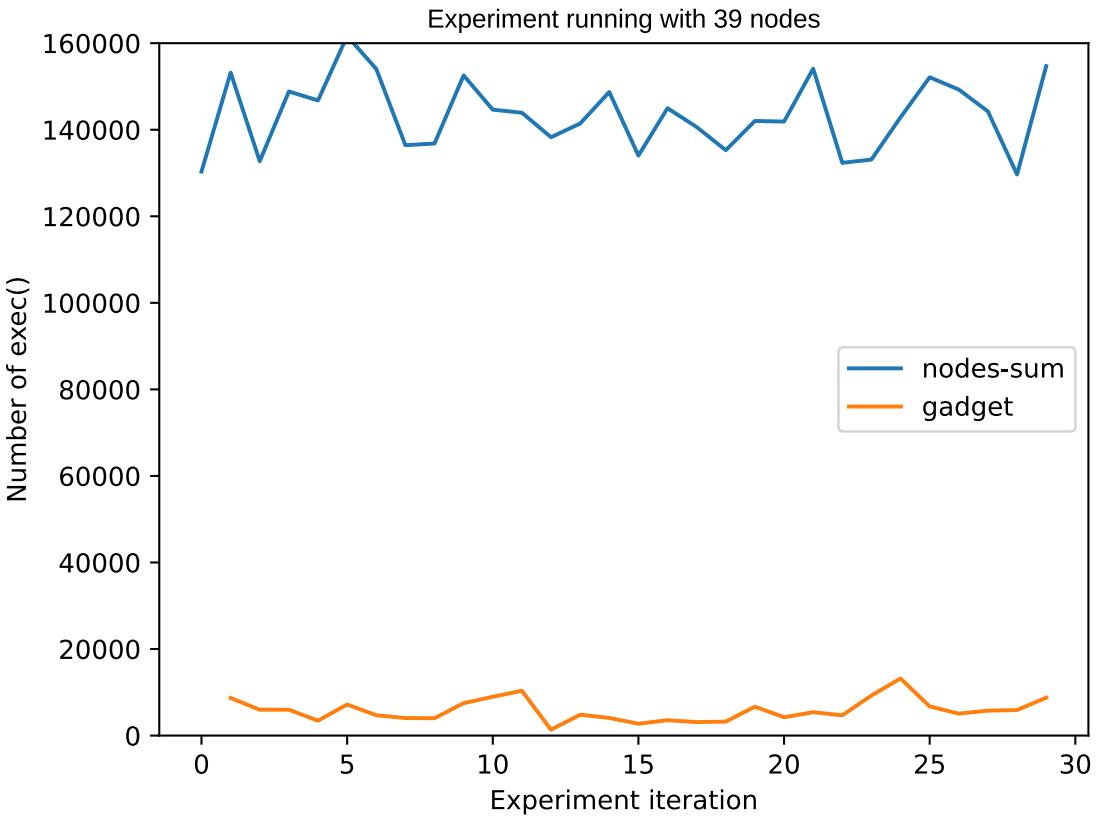

Experiment running with 5 nodes
nodes-sum
gadget
Number of exec()
160000
140000
120000
100000
80000
60000
40000
20000
0
0
5
10
15
20
25
30
Experiment iteration


Experiment running with 11 nodes
nodes-sum
gadget
Number of exec()
Experiment iteration
160000
140000
120000
100000
80000
60000
40000
20000
0
0
5
10
15
20
25
30


Experiment running with 17 nodes
nodes-sum
gadget
Number of exec()
Experiment iteration
160000
140000
120000
100000
80000
60000
40000
20000
0
0
5
10
15
20
25
30


Experiment running with 23 nodes
nodes-sum
gadget
Number of exec()
Experiment iteration
160000
140000
120000
100000
80000
60000
40000
20000
0
0
5
10
15
20
25
30


Experiment running with 31 nodes
nodes-sum
gadget
Number of exec()
Experiment iteration
160000
140000
120000
100000
80000
60000
40000
20000
0
0
5
10
15
20
25
30

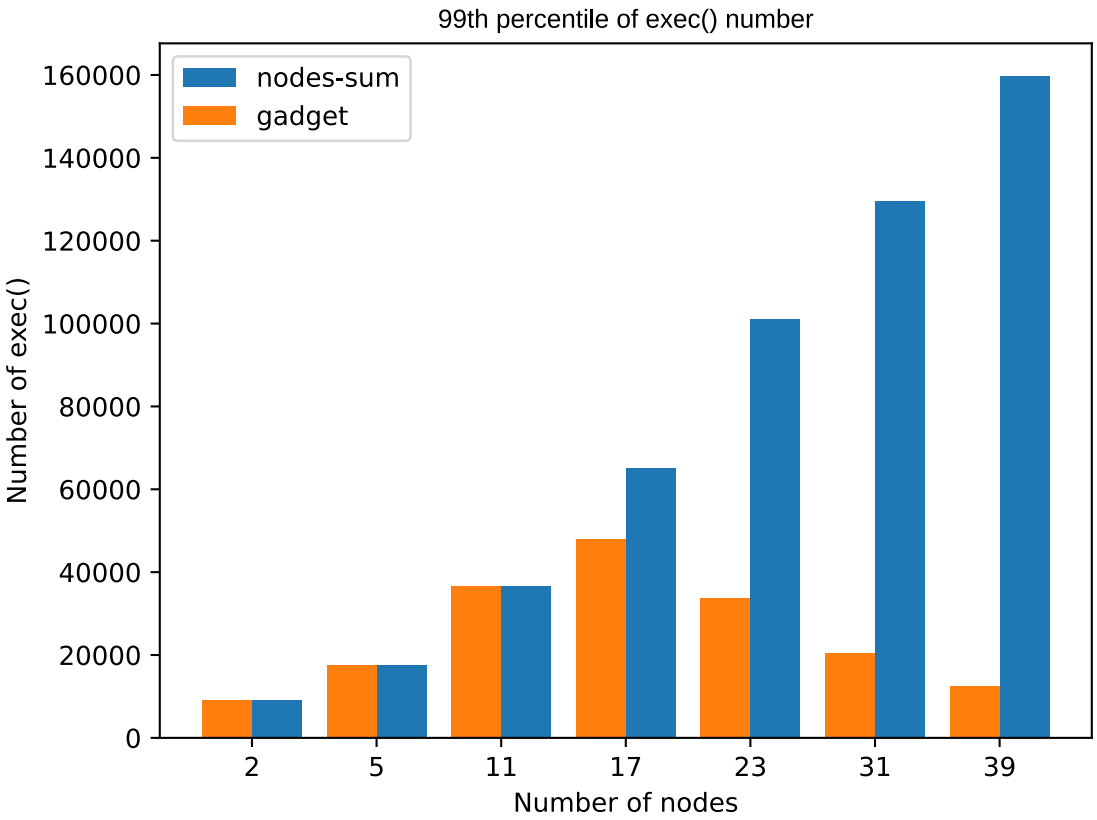


1.3 Interpretation

First of all, the curves are quite flat, so we can conclude there are not so much variation across the iterations.

The number of events reported by Inspektor Gadget follows the number of `exec()` generated until the experiment involving 17 nodes. For experiments with less than 17 nodes, Inspektor Gadget reports less events than actual number of generated `exec()`.

Let's compute the 99th percentile for all number of nodes:



With the 99th percentile, we can clearly see Inspektor Gadget does not scale passed 11 nodes. For 17 nodes, the 99th percentile of generated `exec()` is 65014.75 while Inspektor Gadget reports 47986.23 events. This represents a difference of 17028.52 events or 35.5%. To compare, for 2 nodes, the 99th percentile of generated events is 9082.91 and Inspektor Gadget reports 9036.37, hence a difference of 46.54 events or 0.5%.

1.4 Conclusion

Inspektor Gadget reports less events than actual number of generated `exec()`, this can be caused by difference of counting between `stress-ng` and Inspektor Gadget. Inspektor Gadget scaling is also quite limited, as it reports far less events than actually generated when the cluster has 17 nodes. So, using `exec gadgettracermanager` to get the events stream is clearly a bottleneck.