Version 02.02.00 – skew – interlock protocol change

The “skew” or “skew\_weight” workload parameter is newly introduced in ivy version 02.02.00.

The default skew value is 1.0.

Skew values must be positive numbers, that is greater than zero.

In this version of ivy, support for skew\_weight has been added to the “edit rollup” function.

Edit rollup has two inputs, the text of the parameter settings to be sent out, e.g.“blocksize = 4096”, and the selection of workload instances that you want them sent to.

The default selection is to send to all workloads. This has the same effect selecting “all=all”.

There is one special parameter, “total\_IOPS” that “edit rollup” looks out for in the “parameter=value” text provided by the user.

Within the provided parameter text, where it says “total\_IOPS = xxx.xx”, that bit would be rewritten as “IOPS = yyy.yy’ in order to be sent out to individual remote test host workloads.

In earlier versions of ivy, the total\_IOPS value would be evenly divided by the number of workloads to arrive the IOPS value sent out to all the workload threads.

* Old: total\_IOPS spread evenly over workloads without reference to LUNs or test hosts.

What’s new in version 02.02.00 of ivy is the use of the “skew” or “skew\_weight” in distributing total\_IOPS values across workloads.

Now when the user specifies “total\_IOPS = 123456.78”, ivy first iterates over all selected workloads, using the workload ID like sun159+/dev/sdd+barney to populate a data structure, keeping track of what workload names like “barney” there are for each LUN, as identified by the sun159+/dev/sdd part of the workload ID, and recording the “skew\_weight” parameter value for each workload on the LUN.

First, the total\_IOPS value is divided up evenly over all host LUNs – the total IOPS summed over the workloads on every host LUN will be the same.

* New: total\_IOPS spread evenly over all selected host LUNs, and then within any particular LUN according to the skew\_weight values for each workload on that LUN.

There’s a “skew\_weight sum over workloads” total that’s available for each LUN.

Then the portion of the LUN IOPS going to each workload is that workload’s skew\_weight value divided by the sum of skew\_weights over that LUN.

This makes it easy to give, say, a random workload a skew of 80%, and a sequential workload a skew of 20%, and then saying edit\_rollup("all=all", "total\_IOPS = 1000000.") resulting of a total of 800000 random workload IOPS and total 200000 sequential workload IOPS.

A change was needed in the interlock protocol to implement this.

Previously, the exact same edited parameter string would go out to all selected workload IDs.

The select expression generates a distribution list of workload IDs, that’s apportioned out to each test host.

The interlock was for ivy main thread to instruct all the test host threads to send out the edited parameters to the set of selected workloads on that test host.

Thus all test hosts would receive their respective “edit workload” command at the same time. There was only one version of the edited parameter text that was sent to all selected workloads.

Then the main thread would wait for the “edit workload” to finish on all tests hosts.

With the “skew\_weight” total\_IOPS distribution, some workloads have their IOPS setting to a different values than other workloads. So now ivy iterates over all the target workloads, computes the edited parameter text with the total\_IOPS = xxx rewritten to IOPS = yyy, and it accumulates for each distinct value for the newly edited text, which workload IDs on that host get that version of the text.

This means interlocking with each remote ivyslave host multiple times, with no fixed number of distinct IOPS expected. The interlock is for the minimum number of times, once for each distinct edited parameter text.

So now the ivy main thread iterates serially over test hosts, and then for each distinct edited parameter IOPS value, it sends an “edit workload” command to ivyslave for the set of workload IDs being set to that value.

If in the protocol time numbers, if this becomes an issue, it would be a “simple matter of programming” to endeavour to perform this in parallel.