[Go] "focus metric"

HITACHI
Inspire the Next

- If you want to run a fixed workload for a fixed number of subintervals, all you need is `warmup_seconds` and `measure_seconds`.

- Otherwise, we need to specify the "focus metric".

  1. The focus metric is what we are making a valid measurement of using "`measure`", the "seen enough and stop" feature.
     - Measure the focus metric to a required plus/minus accuracy with a specified % confidence level.
  2. When dynamically adjusting `total_IOPS` using the PID loop dynamic feedback controller (`dfc=pid`), the focus metric is the "feedback" in dynamic feedback control.

Granularity of the "focus metric"

HITACHI
Inspire the Next

- When `measure = on` or `dfc = pid` are used, measurement or PID loop DFC is performed at the granularity of each instance of the `focus_rollup`.
- For the default, `focus_rollup = all`, the measurement or DFC is at the overall level.
- When a `focus_rollup` is used that has multiple rollup instances,
  - With `measure=on`, a successful measurement identifies a subsequence of subintervals where for every rollup instance within the `focus_rollup`, the measurement is valid for that rollup instance.
  - When `dfc=pid` is used, dynamic feedback control is performed independently for each rollup instance in the `focus_rollup`.

`source` of the `focus_metric`

HITACHI
Inspire the Next

- `source = workload`
  - Specifies that we are selecting a focus metric from data collected by ivy workload threads on test hosts.
  - We always have rollup data from test host workload threads (*details in another presentation/video)*
- `source = RAID_subsystem`
  - Specifies that we are selecting the focus metric from real time performance data collected from a command device.
  - There's a small list of subsystem metrics specified in an ivy source code table that are filtered and rolled up from the raw bulk RMLIB data by rollup instance, and from which you select the focus metric. (*details in another presentation/video*

Selecting a "source=workload" metric

HITACHI
Inspire the Next

- category =
  - overall, read, write, random, sequential, random_read, random_write, sequential_read, sequential_write
- accumulator_type =
  - bytes_transferred, service_time, response_time
- accessor =
  - avg, count, min, max, sum, variance, standardDeviation
- It will be easier to explain first accessor then category, then accumulator_type

Accumulators and "accessor"

HITACHI
Inspire the Next

- An accumulator is an object that you push numbers into in order to be able to compute summary values.
- Every time that an I/O completes, ivy posts the service time into one accumulator, the bytes transferred into another accumulator, and if we are not running IOPS=max, it posts the response time into another accumulator.
- The selectable values for "`accessor`" are the names of the methods that you can use to retrieve something from an accumulator
  - `avg, count, min, max, sum, variance, standardDeviation`
  - `avg` gives you the average of the numbers that were pushed in the accumulator
  - `count` gives you how many numbers were pushed in.
  - Et cetera.

Attributes of individual I/Os:

HITACHI
Inspire the Next

- read vs. write
- blocksize
- LBA
  - Logical Block Address = sector number from 0 within LUN
- service_time (in seconds)
  - The duration from when ivy launched an I/O until ivy received the notification that the I/O was complete.
- response_time* (in seconds) (analogue to application-level response time)
  - The duration from the scheduled start time of an I/O until the time the I/O is complete.
  - An I/O may not be started at the scheduled time if there are no idle asynchronous I/O "slots" (~tags) available.
  - ***only I/Os with a non-zero scheduled start time will have a response_time attribute.**
  - When running iops=max, all I/Os have a scheduled start time of zero, meaning you don't get response_time

Ivy uses the Linux nanosecond resolution clock for all timing

How ivy posts results of each I/O

HITACHI
Inspire the Next

- Based on the attributes of each I/O, an accumulator category is selected.
  - Then the I/O is posted into the selected category "bucket" (into two or three accumulators in that bucket – more in a moment.)
- Currently, the breakdown for the array of categories for which there are accumulators are
  - read vs. write
  - random vs. sequential (The I/O sequencer tells you if it's a random or sequential sequencer.)
  - For each of those 4 there is a further breakdown as a histogram by service time and by response time
    - You see the histograms in the csv files.
    - Ivy doesn't currently expose the histogram in the PID loop, but if there is interest it can be added.

Other category breakdowns could be defined

HITACHI
Inspire the Next

- The rollup mechanism operates on a view of the categories as an array, and is blind to the significance of each position in the array.
  - It is easy to define a different mapping from the attributes of an individual I/O to the category bucket the I/O will be recorded in.

- Future:
  - We could just as easily define a histogram of a 100 buckets by LBA range - we could break out the data by each 1% of the LBA range across the volume.
    - If we had an I/O sequencer that was playing back a customer I/O trace, we could show if workload characteristics were different in different areas of the LUN.
    - If we simply run sequential transfers across the LUN, we could see the sustained data rate "staircase" showing the zones in underlying HDDs.

During rollups, the categories are preserved

HITACHI
Inspire the Next

- For the `all=all` instance, you still have all the category breakdowns.
- Then in addition to the category bucket array, there are virtual categories, implemented as functions, which rollup underlying category buckets.
  - `overall` – sum over all categories in the bucket array
  - `read, write`
  - `random, sequential`
  - `random_read, random_write, sequential_read, sequential_write`
- You can see these virtual category rollups in column groups in ivy csv files.

`source=workload` available `category` values

HITACHI
Inspire the Next

- `overall`
  `read, write`
  `random, sequential`
  `random_read, random_write, sequential_read, sequential_write`

- These are actually the virtual categories.
  - If there is a need, we could provide access to the more fine-grained underlying category bucket array, or we could define other virtual categories as aggregations of the buckets.

source=workload - selecting accumulator

HITACHI
Inspire the Next

- Category buckets have 3 accumulators
- accumulator = bytes_transferred
  - For every I/O, the blocksize is posted to bytes_transferred.
  - Use sum attribute and divide by elapsed seconds to get bytes per second. Use count instead and get IOPS.
- accumulator = service_time
  - For every I/O the duration from when ivy started it to when it completed.
  - service_time and response_time values for I/Os are posted in units of seconds, with nanosecond resolution.
  - Use "avg" and multiply by 1000 to get average service time in ms.
- accumulator = response_time (~ application response time)
  - Only posted for those I/Os that have a non-zero "scheduled time".
  - Duration from scheduled time to I/O completion time.
  - The I/O sequencer computes the scheduled time, and when that time is reached, the I/O is started if there is an idle Asynchronous I/O "slot" (~tag) available. If not, it waits.
  - For IOPS=max, I/Os have a scheduled time of 0 (zero), so then you don't get any response_time events.

Summary: source=workload

HITACHI
Inspire the Next

- category =
  - overall, read, write, random, sequential, random_read, random_write, sequential_read, sequential_write
- accumulator_type =
  - bytes_transferred, service_time, response_time
- accessor =
  - avg, count, min, max, sum, variance, standardDeviation

source = RAID_subsystem

HITACHI
Inspire the Next

- Subsystem performance data is collected from a command device, and for each subsystem with a command device, there is a subfolder within the test step folder, where each csv file has one line per subinterval within that test step.
  - You cannot select the focus metric from this raw, bulk subsystem performance data.
- A small subset of metrics are extracted from the bulk subsystem data, and filtered and summarized by rollup instance
  1. To serve as candidates for selection as the focus metric
  2. To be printed as columns in rollup instance csv files side-by-side with the columns of host-workload data.
- This is controlled by a table in ivy source code, which has two levels that you pick from
  - `subsystem_element`, and within that, `element_metric`.
- For each metric in the table, you can optionally set a flag to have the value inserted a column side by side with the normal workload data for each rollup instance.

Subsystem metrics by rollup instance 2015-11-19

HITACHI
Inspire the Next

- MP_core
  - busy_percent, io_buffers
- CLPR
  - WP_percent
- PG
  - busy_percent,
    random_read_busy_percent, random_write_busy_percent, seq_read_busy_percent,
    seq_write_busy_percent
- LDEV
  - read_service_time_ms, write_service_time_ms,
    random_blocksize_KiB, sequential_blocksize_KiB,
    random_read_IOPS, random_read_decimal_MB_per_second , random_read_blocksize_KiB,
    random_read_hit_percent,
    random_write_IOPS, random_write_decimal_MB_per_second, random_write_blocksize_KiB,
    sequential_read_IOPS, sequential_read_decimal_MB_per_second, sequential_read_blocksize_KiB,
    sequential_write_IOPS, sequential_write_decimal_MB_per_second,
    sequential_write_blocksize_KiB,

Subsystem data filtered by rollup instance

HITACHI
Inspire the Next

- The way this works is via a "config filter" that is prepared in advance before a subinterval sequence starts.
- For each thing you get data for, such as PG, or LDEV, or MPU, etc., the config filter has the set of instances of PG or LDEV or MPU names that were either
  - directly observed as a SCSI Inquiry attribute of the LUNs underlying the workloads in the rollup instance, or
  - observed as an attribute of an underlying LDEV obtained via the RMLIB API, or
  - which were inferred from static tables of relationships for the particular subsystem model.