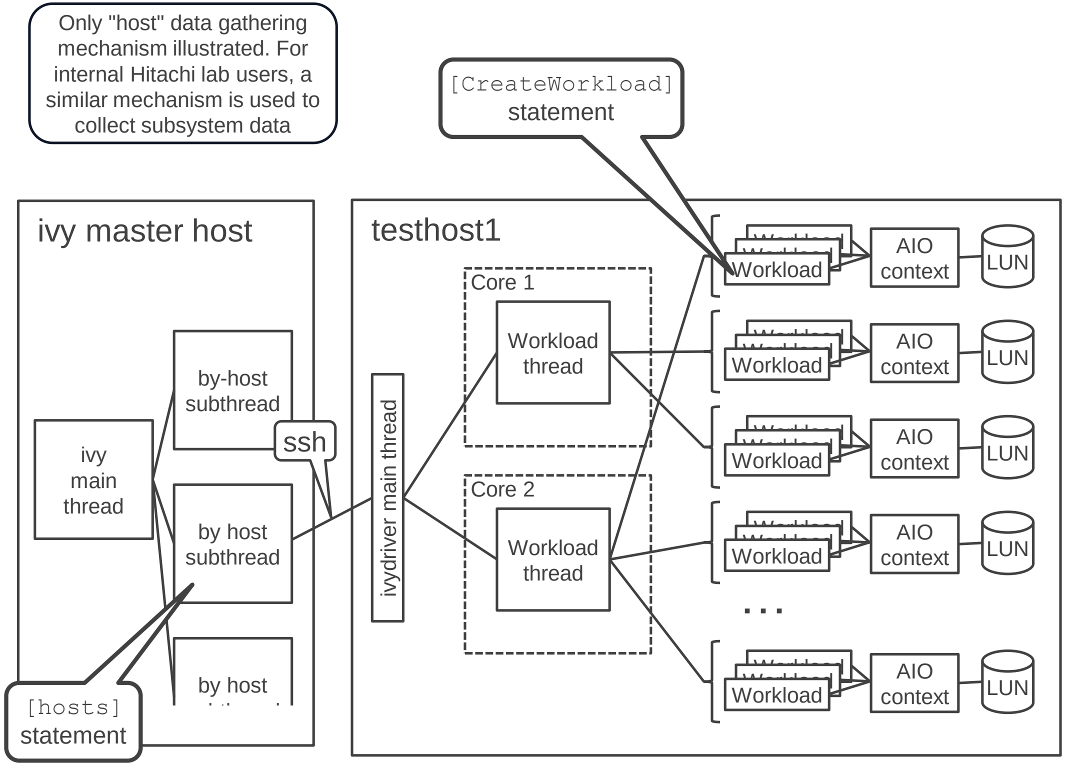
ivy Dynamic Feedback Control

Adaptive PID Loop

September 6, 2019

Allart Ian Vogelesang ian.vogelesang@hitachivantara.com

Ivy was designed for dynamic feedback control



- At the end of every (default 5-second) subinterval, data from each workload thread is rolled up centrally.
- A "rollup" is a grouping of all workloads into "rollup instances". Each workload appears in exactly one instance of every rollup.
  - There is always an "all" rollup with one instance "all=all".
- The data for each individual workload is posted into the owning instance in each rollup.
- If a Hitachi command device connector is active, subsystem performance data is also sent to the master host.
  - The detail by subsystem component is filtered by rollup instance, using the subsystem configuration data to match workloads to their underlying LUNs LDEVs, ports, PGs, MPUs, etc.
  - This by-rollup filtering of subsystem data enables DFC at the granularity of the rollup instance, even for subsystem data.

How Dynamic Feedback Control works

HITACHI
Inspire the Next

- At the end of a subinterval, the rolled up real time host & subsystem data is examined to decide what workload iosequencer parameters adjustments to make.

- The workload parameter updates are sent out using the `[EditRollup]` mechanism at the granularity of the DFC `focus_rollup` instance in real time to immediately affect running workloads.

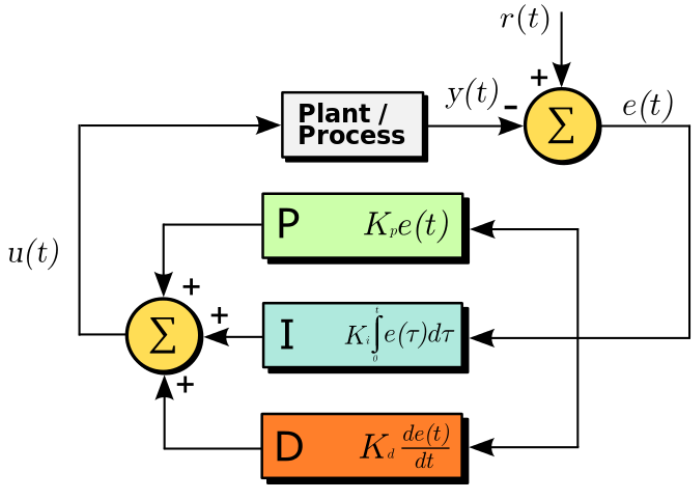
- Then the ivy engine decides if it should stop or continue at the end of the currently running subinterval.
  - The default is to run for `warmup_seconds` plus `measure_seconds`.
  - When the `measure` feature is used, as in `measure = service_time_seconds, accuracy = 1%, timeout= "30:00"`, then `warmup_seconds` and `measure_seconds` become minimums, to be extended as necessary up to `timeout_seconds` until a successful measurement has been made to the target accuracy.
  - The ivy engine sends "stop" or "continue" to `ivydriver` on each test host, which in turn propagates to each workload thread and the workloads it's operating. The "stop" or "continue" must arrive at each running workload thread before it gets to the end of the currently running subinterval.

- At present, only IOPS is adjusted by DFC. In principle, we could adjust nearly all workload parameters.

Ivy engine characteristics

- Workload threads operate driving and harvesting I/Os without ever waiting for their parent `ivydriver` thread.
  - The ivy engine must have delivered "continue" or "stop" before the end of the subinterval is reached.
  - Workload threads don't communicate with each other.
- At the end of a subinterval, the `ivydriver` main thread samples CPU % busy and CPU temperature for each core hyperthread, and sends it up the master host. Then it sends up the results for the just-completed subinterval for each workload on that test host.
- If a command device connector is being used, based on historical latencies, the ivy master host schedules the start of a subsystem performance data gather for "just in time" availability at the end of the subinterval together with the test host data from `ivydriver` instances.
- DFC parameter updates take effect immediately when sent out. Propagation is in milliseconds.

Dynamic Feedback Control using a PID Loop



P is for Proportional
I is for Integral
D is for Differential

- Once all the measurement data have been received at the ivy master host at the end of each subinterval, a new IOPS value is calculated and then immediately sent out using the "edit rollup" mechanism to running workload threads.
- The new IOPS is the sum of three things:
  1. "P" times the error signal.
  2. "I" times the cumulative error since the test began
  3. "D" times the rate of change of the error signal
- See http://en.wikipedia.org/wiki/PID_controller

Sample ivyscript program to use DFC on PG busy

HITACHI
Inspire the Next

```
[hosts] "sun159" [select] "serial_number = 410116";

[CreateWorkload] "steady" [select] "port : 1A" [parameters] "fraction_read=0%, blocksize = 8 KiB, maxTags=4";

[Go!] "IOPS_curve=(2%,98%),measure=PG_busy_percent, accuracy_plus_minus = 1%, warmup_seconds = 30,
measure_seconds = 120, timeout_seconds = 7200";

string summary_filename = ivy_engine_get("summary_csv");

double low_IOPS    = double(csv_cell_value(summary_filename,1,"Overall IOPS"));
double low_target  = double(csv_cell_value(summary_filename,1,"subsystem avg PG busy %"));
double high_IOPS   = double(csv_cell_value(summary_filename,2,"Overall IOPS"));
double high_target = double(csv_cell_value(summary_filename,2,"subsystem avg PG busy %"));

double target_value;
for target_value = { 10%, 20%, 30%, 40% }
{
   [Go]  "stepname = \"IOPS at " + string(100*target_value) + "% PG busy\""
      + ", measure = PG_busy_percent, warmup_seconds = 30, measure_seconds=120, timeout_seconds = \"30:00\""
      + ", accuracy_plus_minus = 1%, dfc = pid, target_value = \"" + string(target_value) + "\""
      + ", low_target   = \"" + string(low_target) + "\", high_target  = \"" + string(high_target)  + "\""
      + ", low_IOPS     = \"" + string(low_IOPS) + "\", high_IOPS = \"" + string(high_IOPS) + "\"";
}
```

Inputs to DFC

HITACHI
Inspire the Next

- `low_IOPS, low_target, high_IOPS, high_target`
  - These are the endpoints of an IOPS – target curve, where "target" is the metric that you're performing DFC on, such as `service_time_seconds` (an ivy test host metric), or or `PG_busy_percent` (a command device connector metric available to authorized Hitachi internal lab users.)

```
[Go!] "IOPS_curve=(2%,98%),measure=PG_busy_percent, accuracy_plus_minus = 1%, warmup_seconds = 30,
measure_seconds = 120, timeout_seconds = 7200";

string summary_filename = ivy_engine_get("summary_csv");

double low_IOPS    = double(csv_cell_value(summary_filename,1,"Overall IOPS"));
double low_target  = double(csv_cell_value(summary_filename,1,"subsystem avg PG busy %"));
double high_IOPS   = double(csv_cell_value(summary_filename,2,"Overall IOPS"));
double high_target = double(csv_cell_value(summary_filename,2,"subsystem avg PG busy %"));
```

- Here we use `IOPS_curve = (2%,98%)` to run 3 steps. One at `IOPS=max`, one at 2% of max IOPS, and one at 98% of max IOPS.

- We then retrieve the low/high values from the `all=all` summary csv file.

The "PID control metric"

- Both DFC and the "measure" feature use the same "focus metric"

- `dfc = pid, target_value = 0.001`
  `dfc = pid, target_value = 50%`

- Often the focus metric is set using one of the "shorthand" settings.

```
measure = service_time_seconds
measure = response_time_seconds
measure = MP_core_busy_percent
measure = PG_busy_percent
measure = CLPR_WP_percent*
```

* For `CLPR_WP_percent` the current PID loop should work, but it may take a long time before the IOPS settles down.

- The shorthand `measure = IOPS` and `measure = MB_per_second` can't be used as a PID control metric because, obviously, it's the IOPS that is the "input".

Adaptive PID function

- The calibration values `low_IOPS`, `low_target`, `high_IOPS`, and `high_target` are used together with `target_value` to establish starting "ballpark" rough estimated parameters for the PID loop.
  - Starting IOPS.
  - Starting "I" parameter, the cumulative error gain.
  - A starting value for the PID loop cumulative error that will yield the desired starting IOPS at the starting gain.

- When the PID loop starts running, an adaptive method is used to adjust the gain to rapidly approach the target PID control metric value, and then settle in and lock on stably.

- Measurement only can start after the last adaptive gain adjustment.

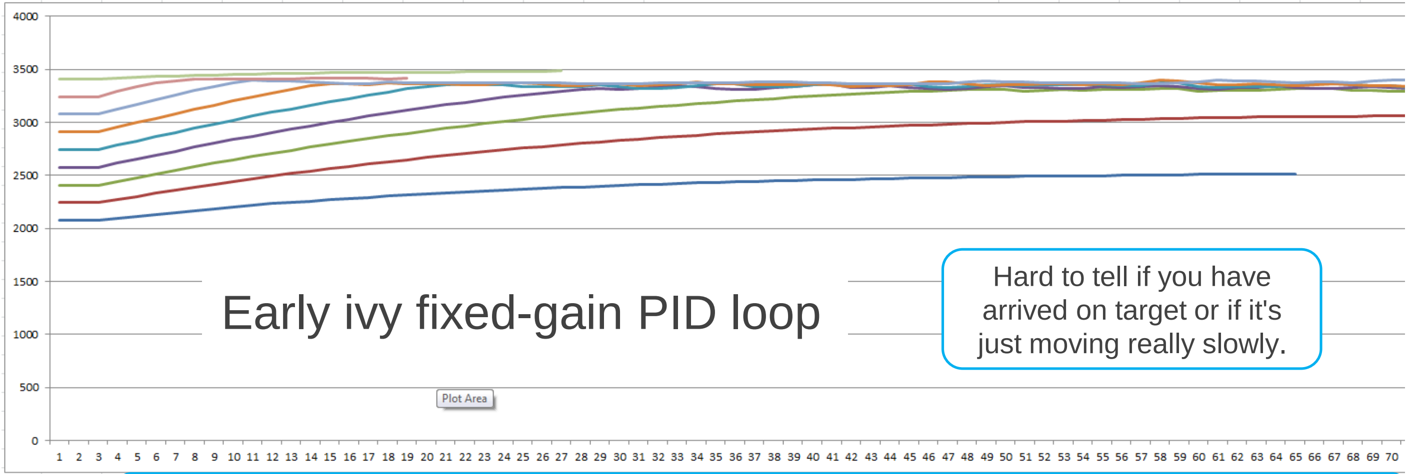
Adaptive behaviour – gain too low

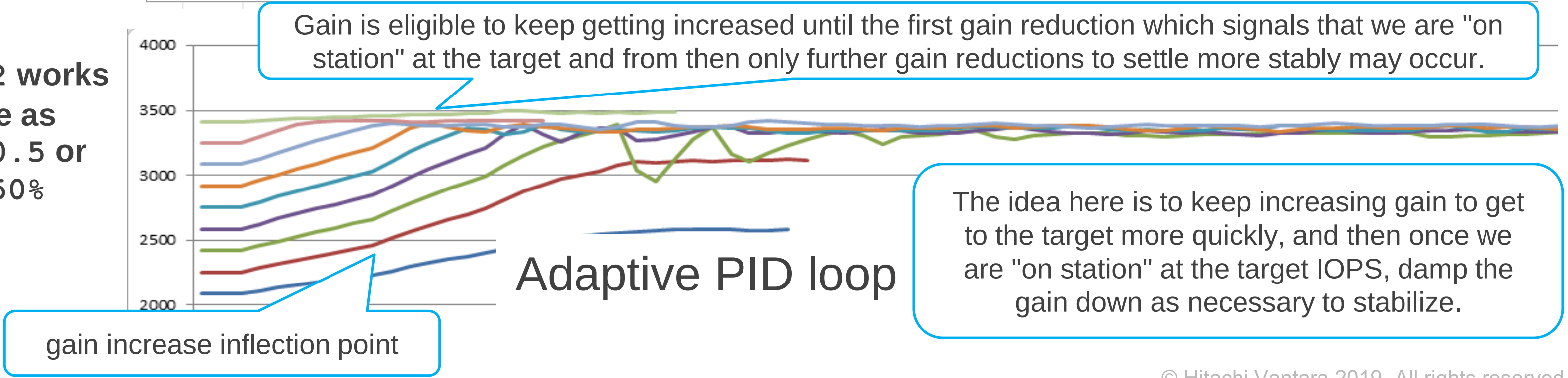
HITACHI
Inspire the Next

- If IOPS initially goes continuously in the same direction for more than `max_monotone` subintervals (default 5), gain is increased by the the `gain_step` factor and a new "adaptive PID subinterval cycle" is started.

Ease of use:

- `gain_step = 2` works exactly the same as `gain_step = 0.5` or `gain_step = 50%`





balanced_step_direction_by

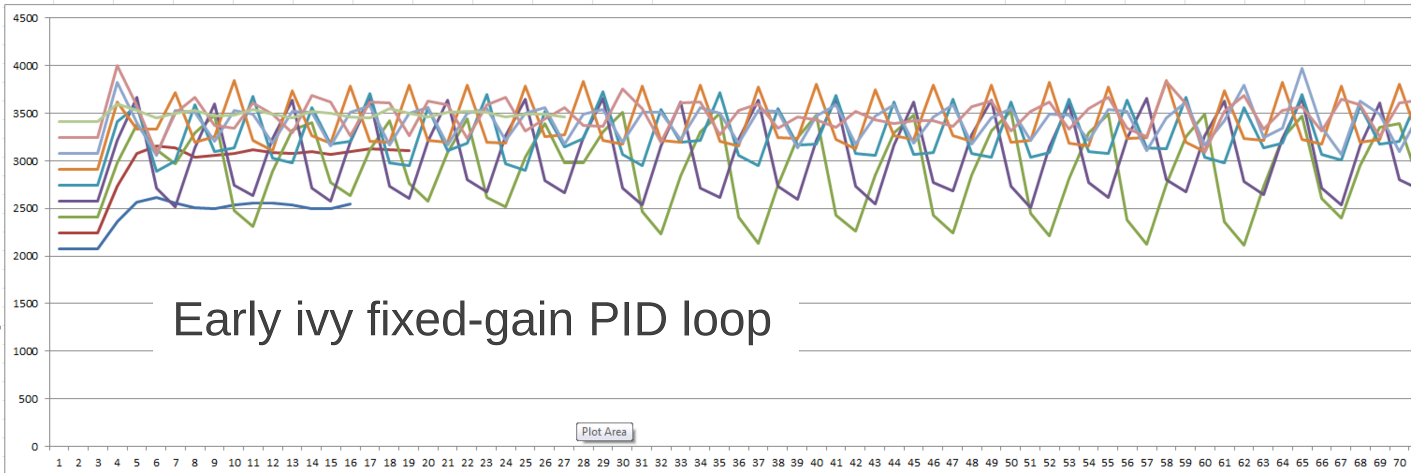
- In some cases if there is noise from subinterval to subinterval in the measurement value even at a fixed IOPS setting, the "`max_monotone`" gain increase mechanism may be slow to trigger due to interfering noise-induced IOPS fluctuations.
- There is a secondary mechanism that operates on the principle that we know we need to increase the gain if on average, having run at least `balanced_step_direction_by` subintervals in the current adaptive PID gain adjustment cycle, if over 2/3 of the IOPS adjustments up or down from subinterval to subinterval are in the same direction, this indicates that we are still steering towards the target and to get there faster we need to increase the gain.
- The default `balanced_step_direction_by` value of 12 is bigger than the default `max_monotone` value of 5 to accommodate noise.

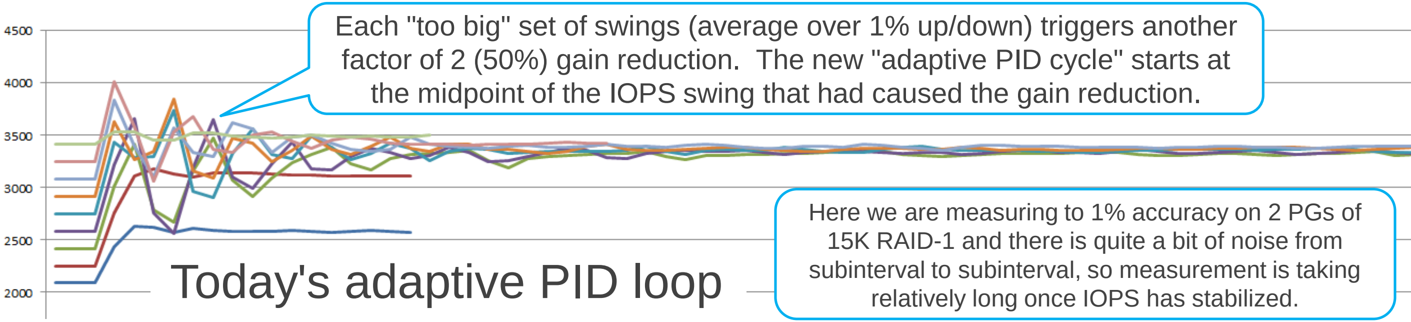
Adaptive behaviour – gain too high

HITACHI
Inspire the Next

- If both the average IOPS "up swing" and the average "down swing" over multiple swings in both directions is bigger than "`max_ripple`" (default 1%), then the gain is reduced by a factor of "`gain_step`" (default factor of 2) and a new "adaptive PID subinterval cycle" is started.

- Measurement can only begin after all gain adjustments are complete (and average IOPS swings up and down are smaller than "`max_ripple`")



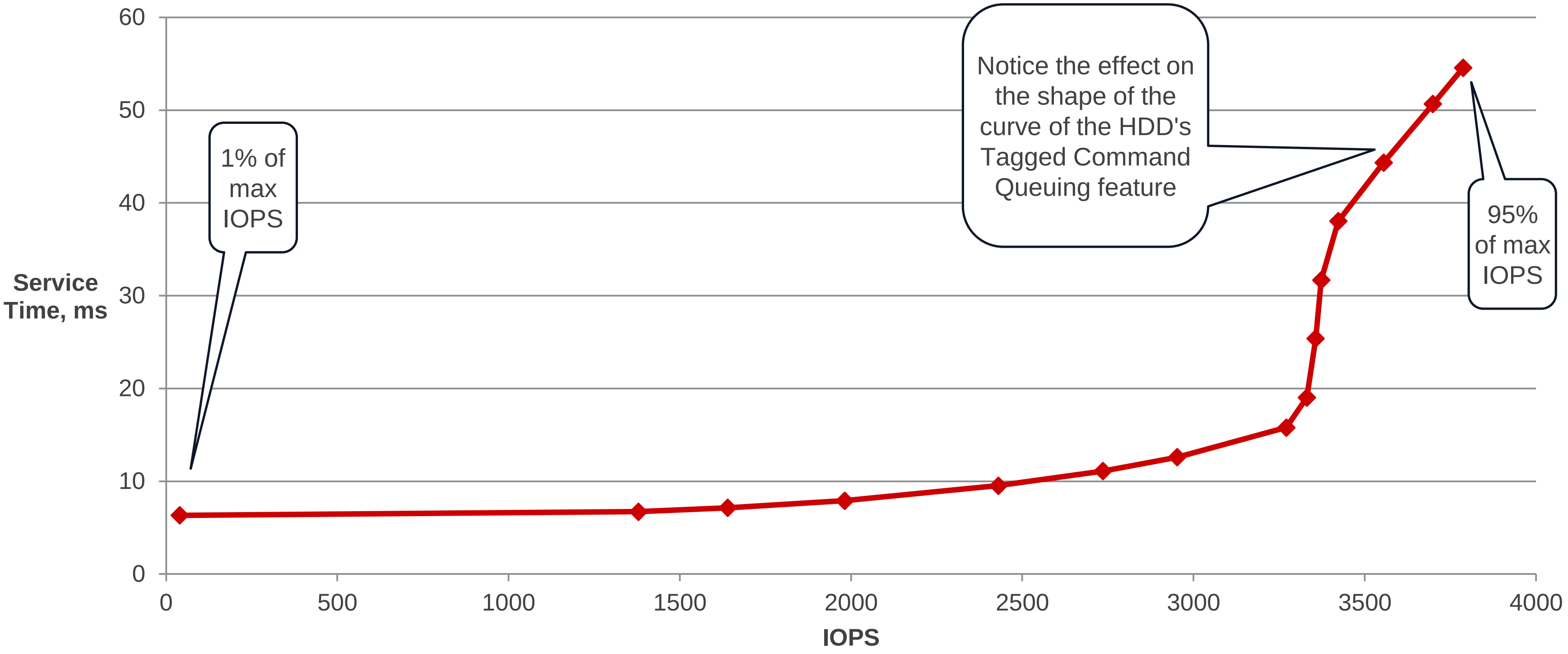


Summary of operating `DFC = PID`

- Measure max IOPS
- Measure low &high PID control metric at, for example, 1% and at 90% of max IOPS
- `[Go] "measure = service_time_seconds, accuracy_plus_minus = 1%, dfc = pid, target_value = `tt`, low_IOPS = `xx`, low_target = `yy`, high_IOPS = `aa`, high_target = `bb`"`
- Advanced user options to control adaptive PID (defaults shown)
  - `gain_step = 2, max_ripple = 1%, max_monotone = 5, balanced_step_direction_by = 12, ballpark_seconds = 60`

Example from `measure=service_time_seconds` with HDDs

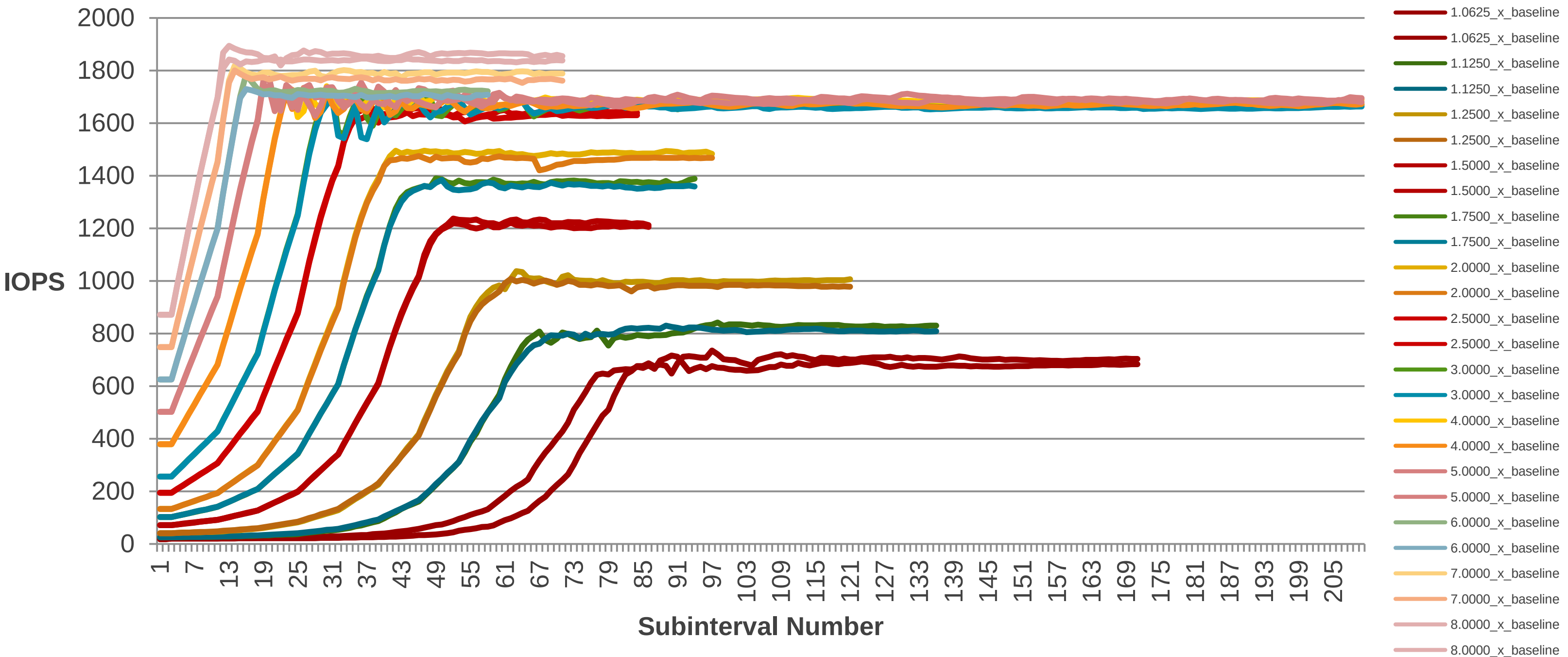
HITACHI
Inspire the Next



Monitoring PID loop behaviour

HITACHI
Inspire the Next

- Look in the (main) test output folder for <test name>`.PID.csv`.



How it works

What are P, I, and D used for in a PID loop?

HITACHI
Inspire the Next

- P
  - Used to respond to a perturbation or to follow a moving target.
    - Turn steering wheel a bit right for now to drift back to center of lane.
- I
  - Used to make the long term average measurement reach a stable target.
- D
  - Used to damp instability by limiting the "slew rate" or rate at which we allow the measured value to change towards the target value.

Use of P, I, and D in ivy

- P
  - We expect "noise" in the measurement at a fixed IOPS value, we don't have a moving target, and past history should not affect future measurements.
  - P is set to zero.
- I
  - Our focus in ivy is on setting "I" to make the average measurement value lock onto the target value promptly, but stably.
- D
  - Write Pending can have a significant time lag, so we should classify as "advanced" the topic of using WP as the PID control metric because we'll probably need to use D.

Cumulative error gain

- You want the cumulative error over "sufficiently many" subintervals to drive IOPS.
- If you make the gain too low, the system will be too sluggish to respond.
- If you make the gain too high, IOPS will chase "noise" in individual results from subinterval to subinterval.

The ballpark method

HITACHI
Inspire the Next

- We define the "operating range" for IOPS to be from 1% to 90% (or so) of max IOPS.

- We measure the PID control metric at 1% of max IOPS and at 90% of max IOPS.

- We define the "operating range" for the PID control metric to be from the measurement value at 1% of max IOPS to the measurement at 90% of max.

- We use a straight line between the "low" and "high" measurement points as a very rough estimate to set our initial gain & initial IOPS.

Auto-ranging concept

- Depending on which PID control metric is selected, the numeric range of the target value may vary.
  - For `PG_busy_percent`, the `target_value` used might be 0.8 (80%).
  - For `service_time_seconds` on FMD / SSD, the `target_value` might be be about a thousand times smaller at 0.001 (1 ms).

- Depending on the IOPS scalability of the platform being tested, the IOPS numeric range may vary.
  - A single small 7200 RPM HDD Parity Group might have max IOPS = 500.
  - A large subsystem with FMD / SSD might have a max IOPS in the millions.

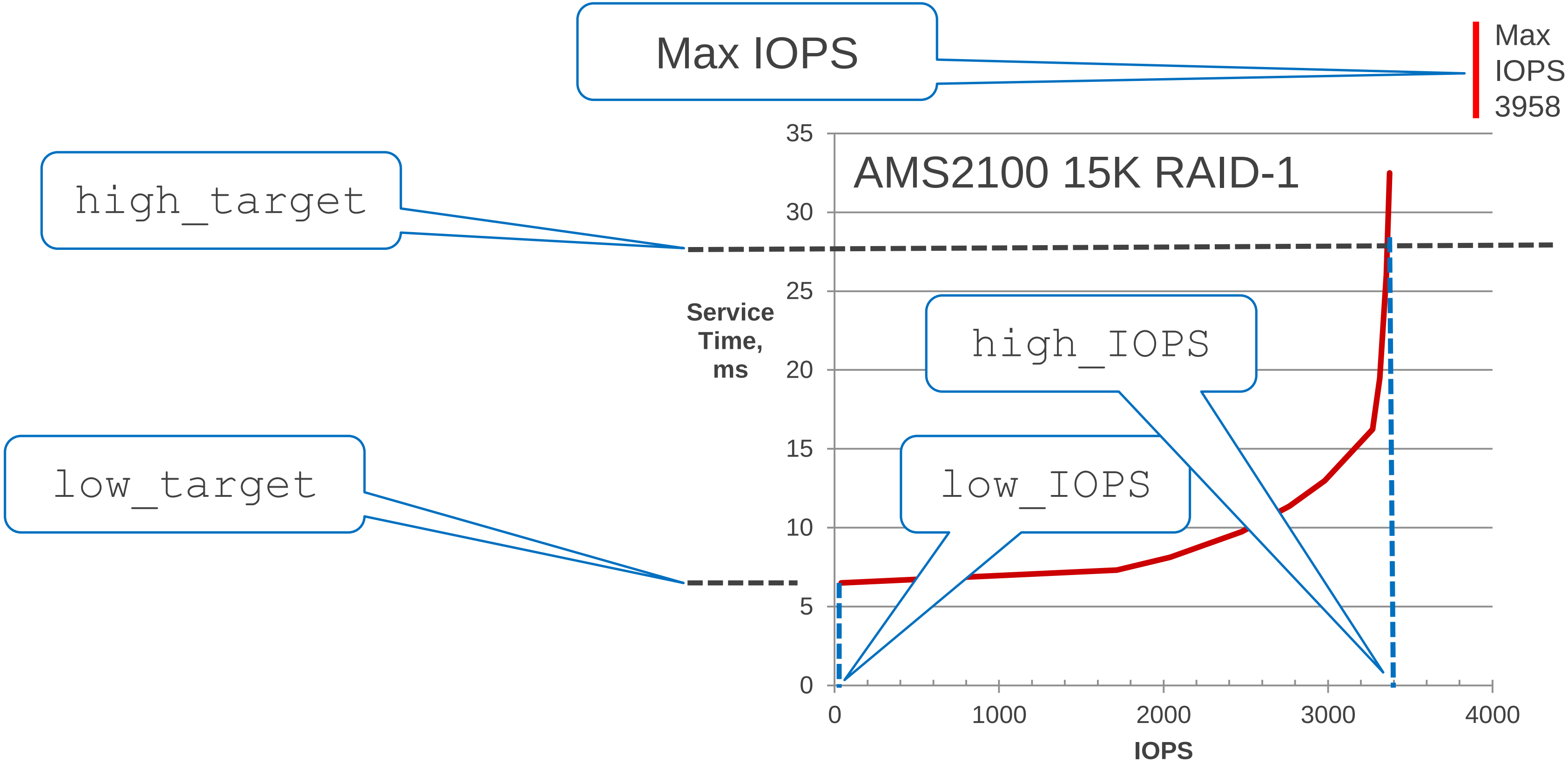
`dfc = pid` uses the "cumulative error" gain

- The ivy PID loop formula is **IOPS = gain x cumulative error**.

- Thus the gain needs to be appropriate for both the numeric size of the possible error signal, as well as the IOPS scalability of the platform under test.

- ivy uses an approximation method to pick a rough value for the gain, which will then automatically be adjusted as the PID loop runs.

Example with `measure = service_time_seconds`



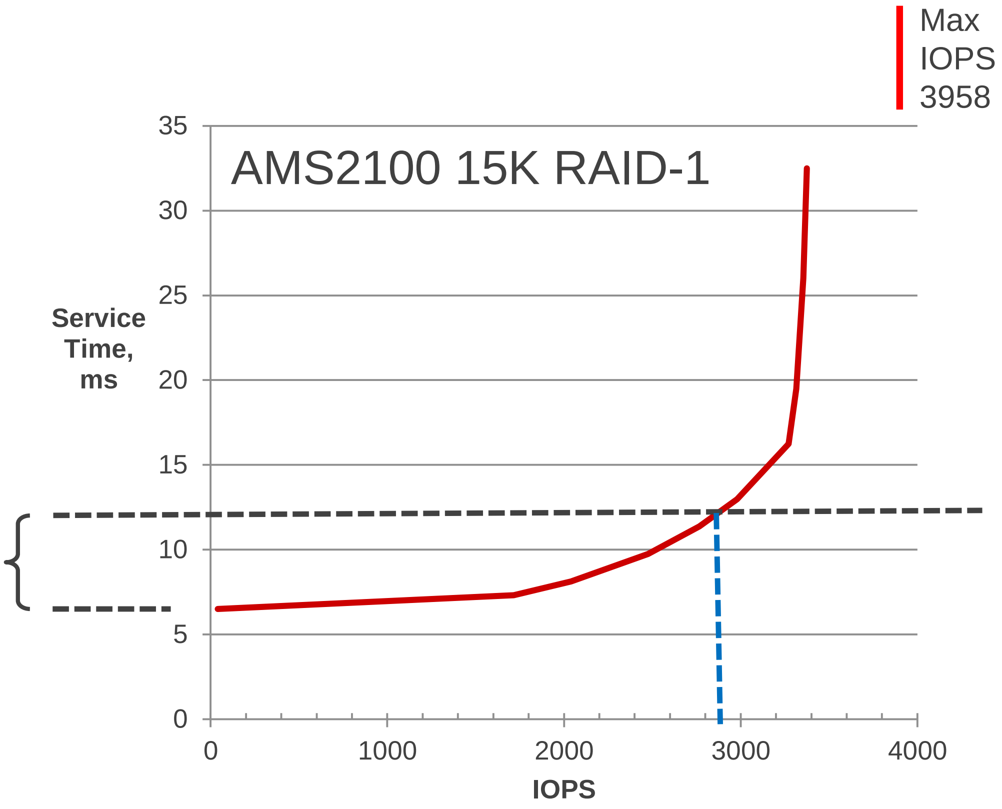
CONFIDENTIAL – For use by Hitachi Data Systems employees and other audiences under NDA only.

Key concept is "initial error"

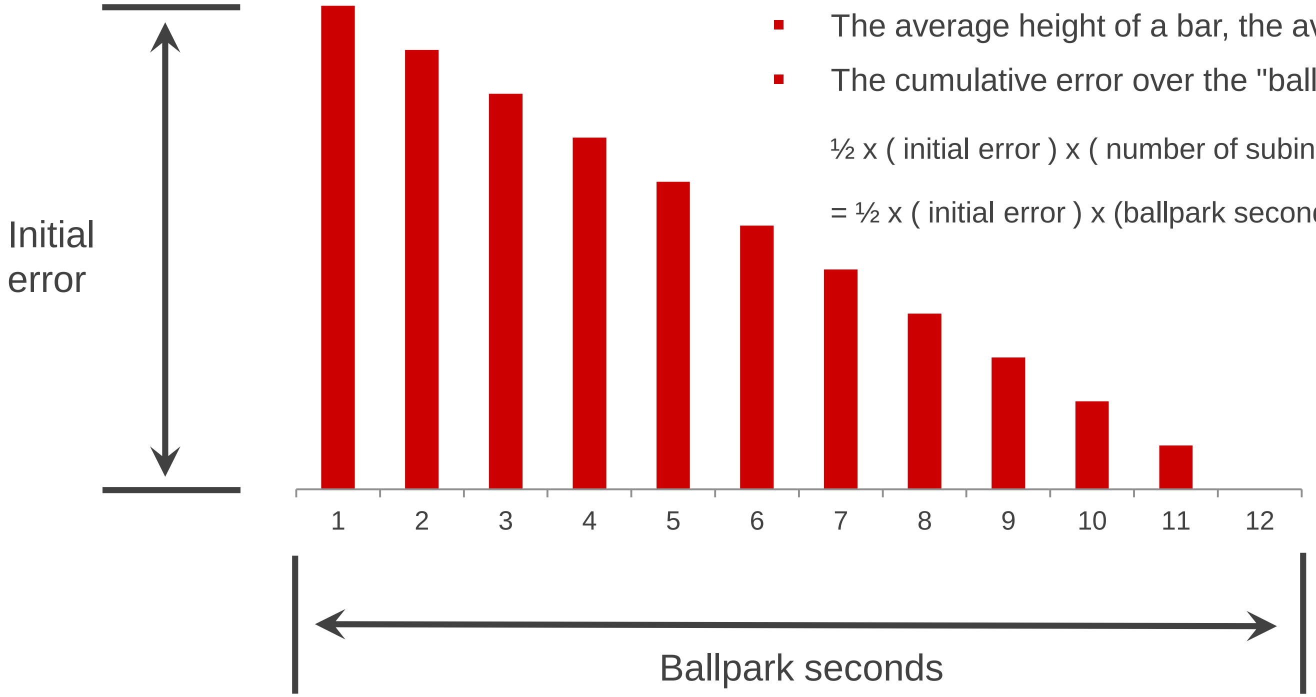
The "initial error" is the difference between the target value of the PID control metric, and the baseline value.

The initial error sign is negative.

In this example the initial error of service_time_seconds is -0.006.



Assume straight line initial error to zero



- Assume error goes to zero in a straight line over "ballpark seconds".
- The average height of a bar, the average error, is ½ of the initial error.
- The cumulative error over the "ballpark seconds" time duration is

½ x (initial error) x (number of subintervals)

= ½ x (initial error) x (ballpark seconds / subinterval duration in seconds)

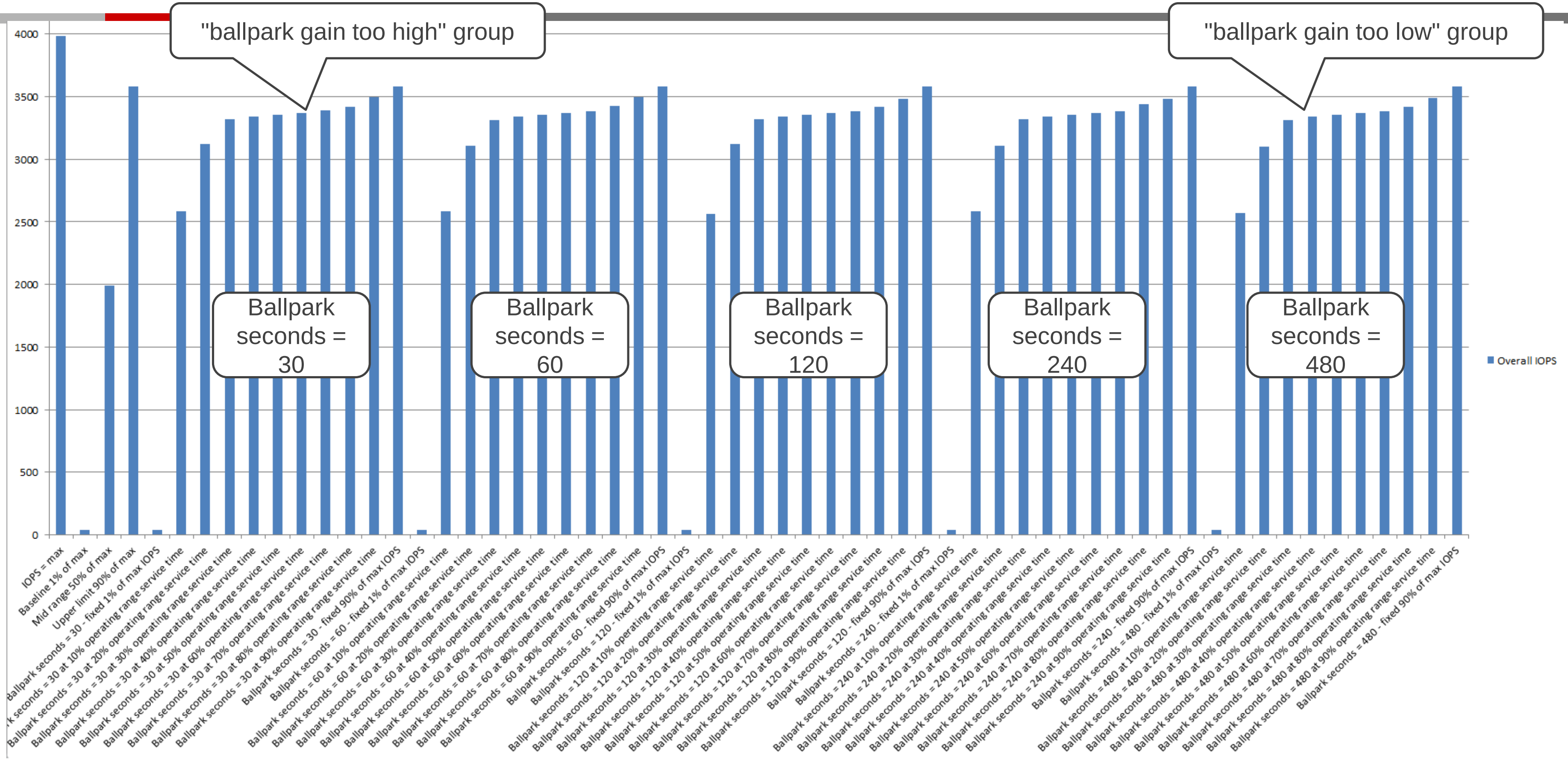
"ballpark seconds" used to set gain sensitivity

- Early experiments with ivy showed that a good tradeoff between responsiveness and stability was to use "`ballpark_seconds`" = 60.
  - Lower `ballpark_seconds`, faster initial response / higher gain.
  - Higher `ballpark_seconds`, slower initial response, lower gain.
- On the previous chart we calculated the estimated cumulative error over the first "ballpark seconds".
- Next, we calculate a rough estimated IOPS drawing a straight line between the "low" and "high" calibration points.
- Then since **IOPS = I x cumulative error**, we calculate starting gain **I = estimated IOPS / estimated cumulative error**.

Effectiveness

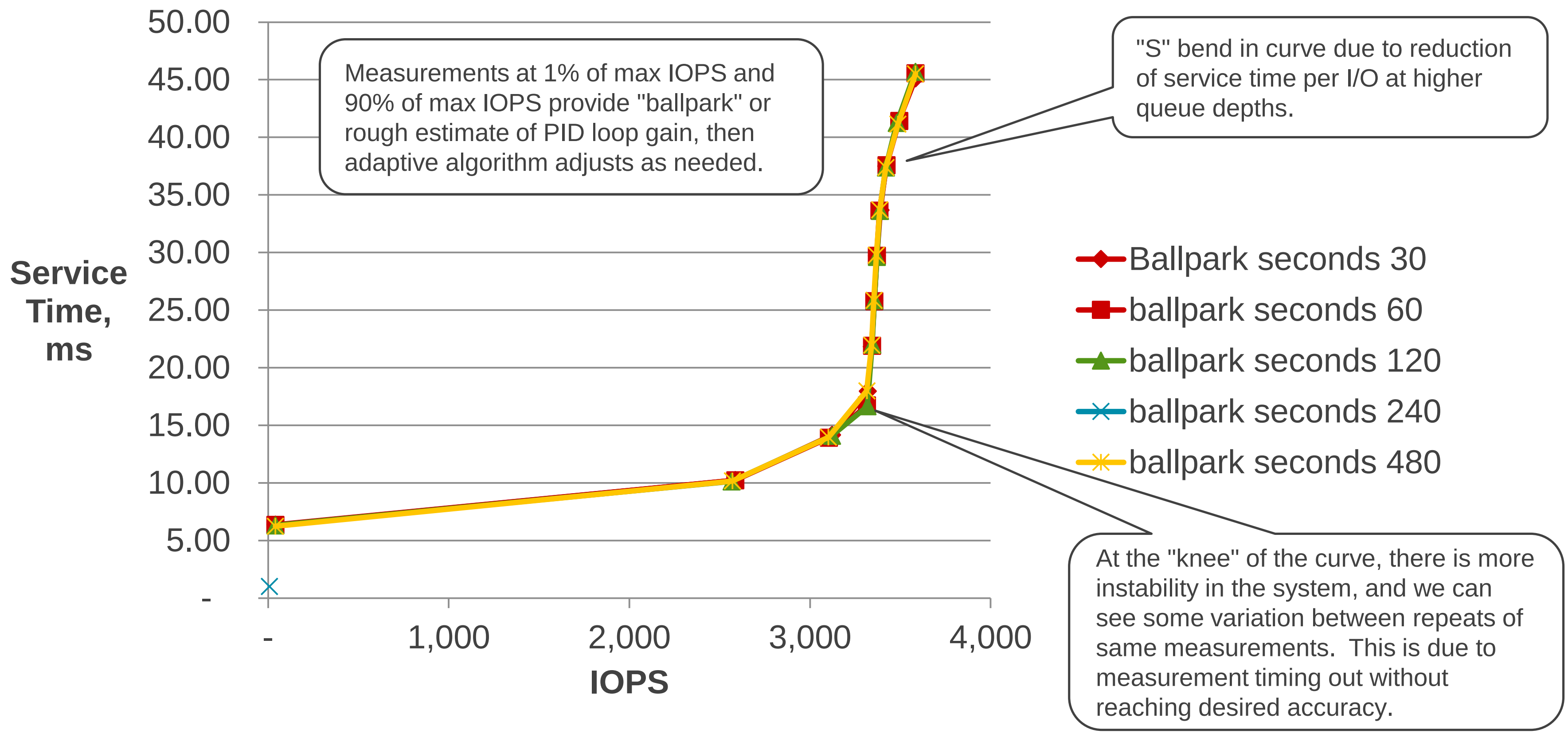
- The rough initial estimate followed by the use of the adaptive gain adjustment ensures a rapid approach to the target, followed fine-tuning to stably lock in on the target.
- It doesn't appear to be worth the time to make more than the two calibration measurements.

Solid measurements for range of sensitivities



Same data as previous chart – repeatability

HITACHI
Inspire the Next



HITACHI

Inspire the Next