

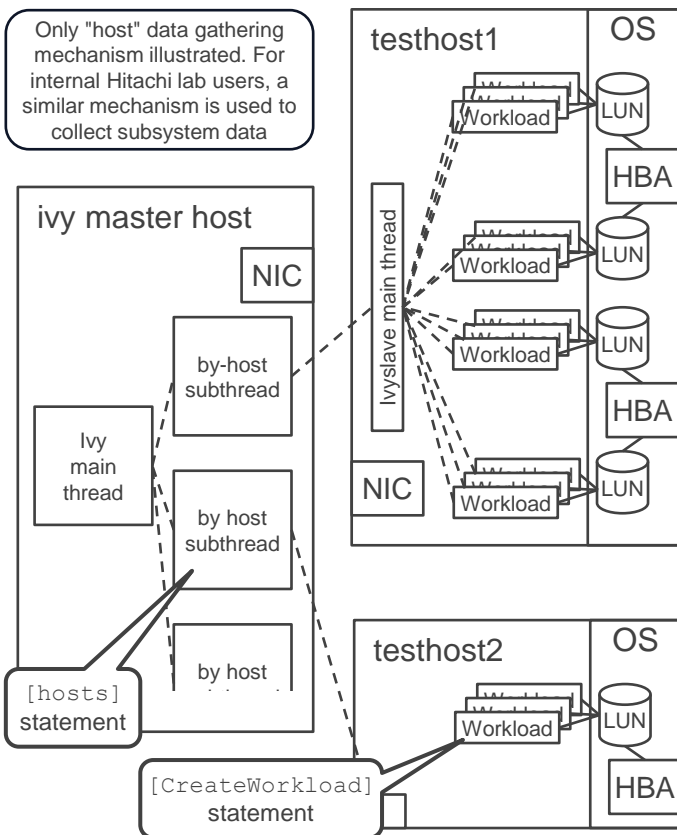
# ivy Dynamic Feedback Control

Adaptive PID Loop

**Allart Ian Vogelesang**

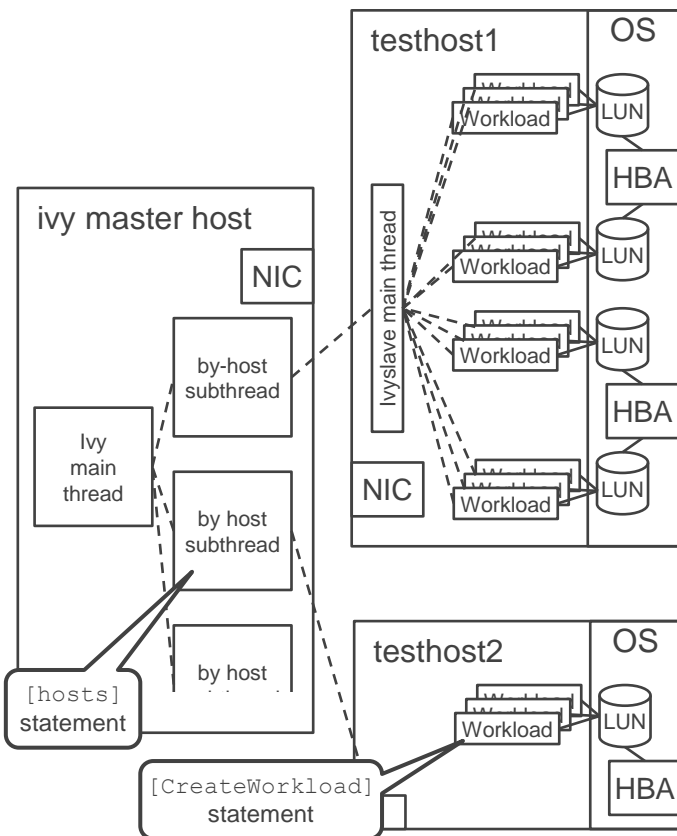
2017-04-13

# 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.
- The data for each individual workload is posted multiple times. It is posted exactly once into each "rollup".
  - A rollup instance is a set of workload threads.
  - Each workload thread appears in one instance of every rollup.
  - The "all" rollup has one instance "all=all" containing all workloads.
- If a Hitachi command device connector is active, subsystem performance data is 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.

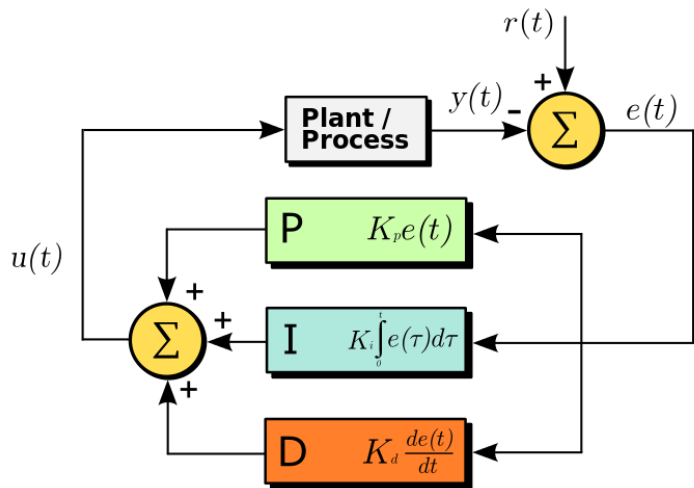
# Once host & subsystem data rollups complete



- Examine the rolled up real time host & subsystem data, and decide if any iosequencer parameters need to be adjusted.
- Using the [EditRollup] mechanism, optionally send out iosequencer parameter updates by rollup instance in real time to immediately affect running workload threads.
  - The `DFC=PID` plugin edits IOPS by "focus rollup" instance.
- Decide whether the ivy engine should stop or continue, at the end of the currently running subinterval.
  - `measure = service_time_seconds, accuracy = 1%`
- The ivy engine sends "stop" or "continue" to each ivyslave, which in turn propagates to each workload thread.
  - The "stop" or "continue" must arrive at each running workload thread before it gets to the end of the currently running subinterval.

- Workload threads operate driving and harvesting I/Os without ever waiting for their parent ivyslave 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.
    - Dedupe uses a statistical method to ensure the right average number of copies are written.
- In order to avoid disturbing any workload thread, ivyslave waits for a short "catnap" after the end of the subinterval to make sure the independently dispatched workload threads have finished posting before ivyslave harvests and sends the posted data to ivymaster.
  - After the catnap is complete, it only takes milliseconds for all the data to roll up centrally.
- Command device collection is timed based on historical latencies for central rolled up "just in time" availability at the end of the subinterval.
- 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 rollout" mechanism to running workload threads.
- The new IOPS is the sum of three things:
  - "P" times the error signal.
  - "I" times the cumulative error since the test began
  - "D" times the rate of change of the error signal
- See [http://en.wikipedia.org/wiki/PID\\_controller](http://en.wikipedia.org/wiki/PID_controller)

# First perform 3 calibration measurements

1. Measure max IOPS     `measure = IOPS, IOPS = max`
    - This measured max IOPS value is only used to perform steps 2) and 3)
  2. Measure the target PID control metric at the lower IOPS limit of the operating range (e.g. measure at 1% of max IOPS).
    - Later you will specify `low_IOPS = xxxx, low_target = yyyy`
  3. Measure the PID control metric at upper IOPS limit of the operating range (e.g. measure at 90% of max IOPS).
    - Later you will specify `high_IOPS = aaaa, high_target = bbbb`
- The `target_value` you specify for the PID loop must be in the range from `low_target` to `high_target`.
  - These measurements are for the total IOPS (`all=all` instance).

# Select 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".

- 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.

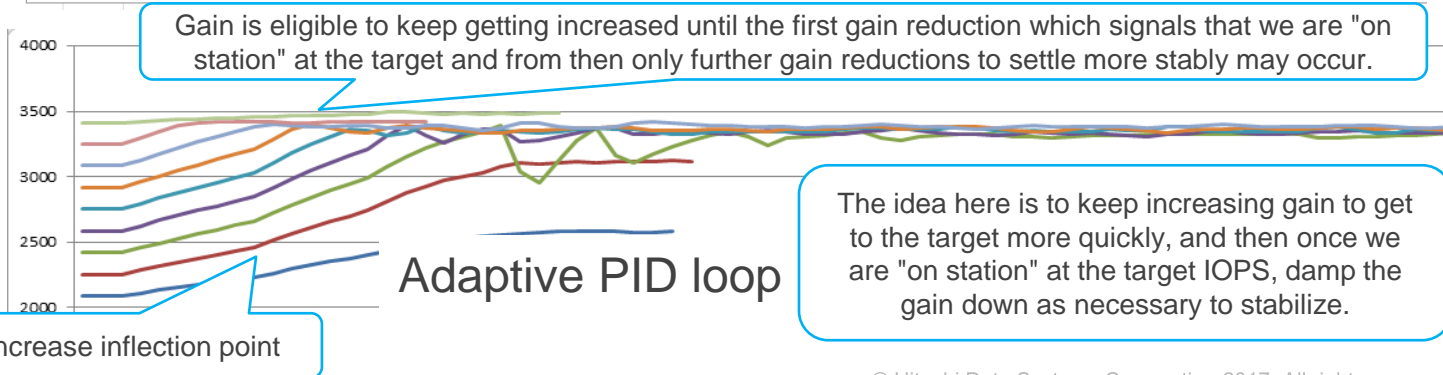
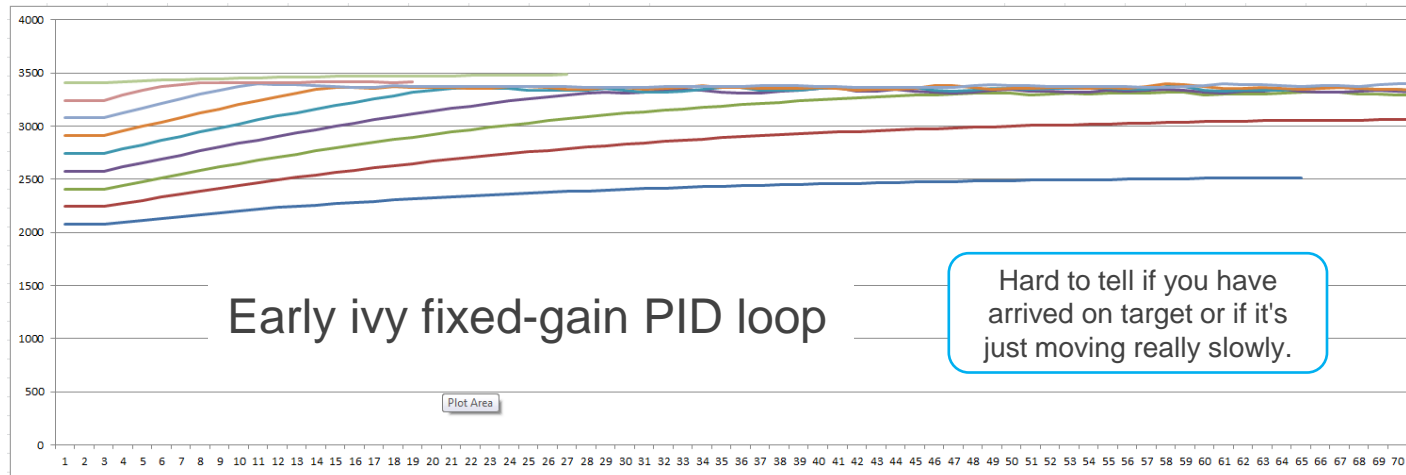


# Adaptive behaviour – gain too low

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

## Ease of use:

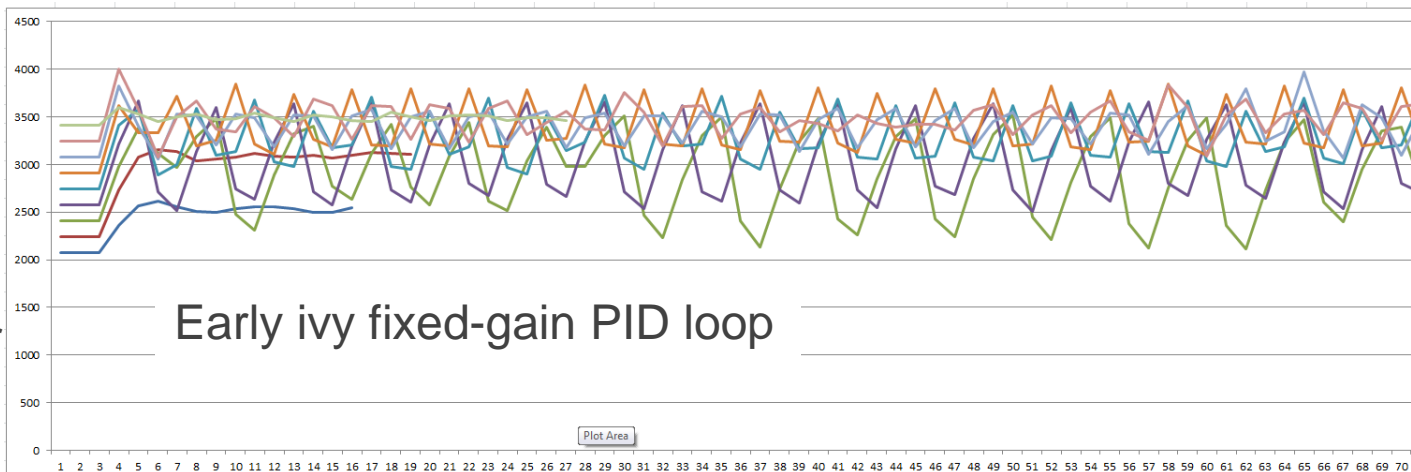
- `gain_step = 2` works exactly the same as `gain_step = 0.5` or `gain_step = 50%`



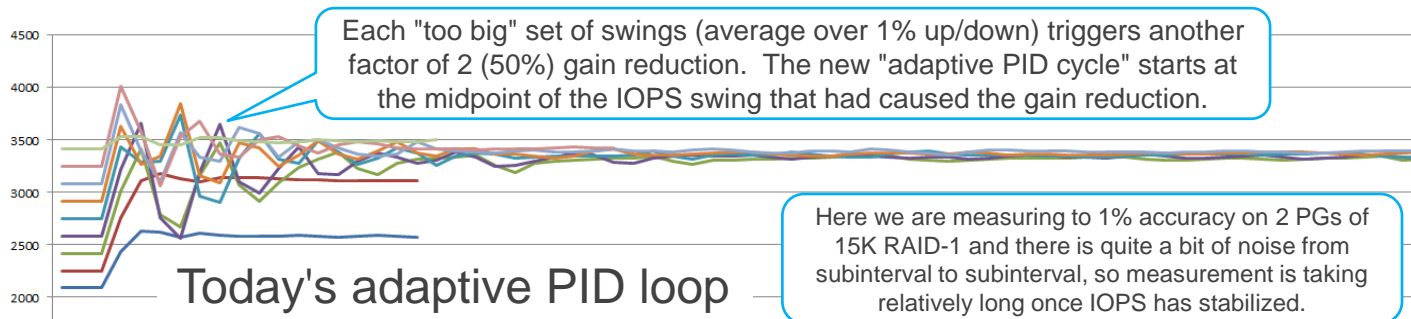
- In some cases if there is noise from subinterval to subinterval in the measurement value even at a fixed IOPS setting, the "max\_monotone" gain increase mechanism may be slow to trigger due to interfering noise-induced IOPS fluctuations.
- There is a secondary mechanism that operates on the principle 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

- 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 PID control metric at 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`

- Start up ivy and create workloads:

```
[outputFolderRoot] "/some/where/ivyoutput";

double accuracy = 1%;

[hosts] "sun159"
      [select] "serial_number : 83011441";

[CreateWorkload] "steady"
      [select] "LDEV : [ 0008, 0009 ]"
      [iogenerator] "random_steady"
      [parameters] "fractionRead = 100%, blocksize = 4KiB, IOPS = max, maxtags = 128";

[CreateRollup] "LDEV";
```

## ■ Measure max IOPS:

```
[Go!] "stepname = \"IOPS = max\", measure = IOPS, accuracy_plus_minus = 1%";  
  
// Retrieve the "Overall IOPS" value from the step 0 row of the summary.csv file for the all=all rollup.  
  
string summary_filename = testFolder() + "/all/" + testName() + ".all=all.summary.csv";  
  
double max_IOPS = double(csv_cell_value(summary_filename, 0, "Overall IOPS"));
```

## ■ Measure target metric at 1% of max IOPS:

```
double low_IOPS = 1% * max_IOPS;

[EditRollup] "all=all" [parameters] "total_IOPS=" + string(low_IOPS);

[Go] "stepname=\"low_target at 1% of max\", measure = service_time_seconds, "
    + " timeout_seconds = \"10:00\", accuracy_plus_minus = 1%";

// Retrieve "low_target":

string result = last_result(); if ("success" != result)
    { s = "Failed to obtain measurement running at 1% of max IOPS.\n";
      print(s); log(masterlogfile(),s); exit(); }

double low_target_ms = double(csv_cell_value(summary_filename, 1,"Overall Average Service Time (ms)"));

double low_target = low_target_ms / 1000.0;
```

## ■ Measure target metric at 95% of max IOPS:

```
double high_IOPS = 95% * max_IOPS;

[EditRollup] "all=all" [parameters] "total_IOPS=" + string(high_IOPS);

[Go] "stepname=\"high_target at 95% of max\", measure = service_time_seconds, "
    ", timeout_seconds = \"10:00\", accuracy_plus_minus = 1%";

// Retrieve "high_target":

string result = last_result(); if ("success" != result)
    { s = "Failed to obtain measurement running at 95% of max IOPS.\n";
      print(s); log(masterlogfile(),s); exit(); }

double high_target_ms = double(csv_cell_value(summary_filename,2,"Overall Average Service Time (ms)"));

double high_target = high_target_ms / 1000.0;
```



- Run PID loop for a range service times across the operating range:

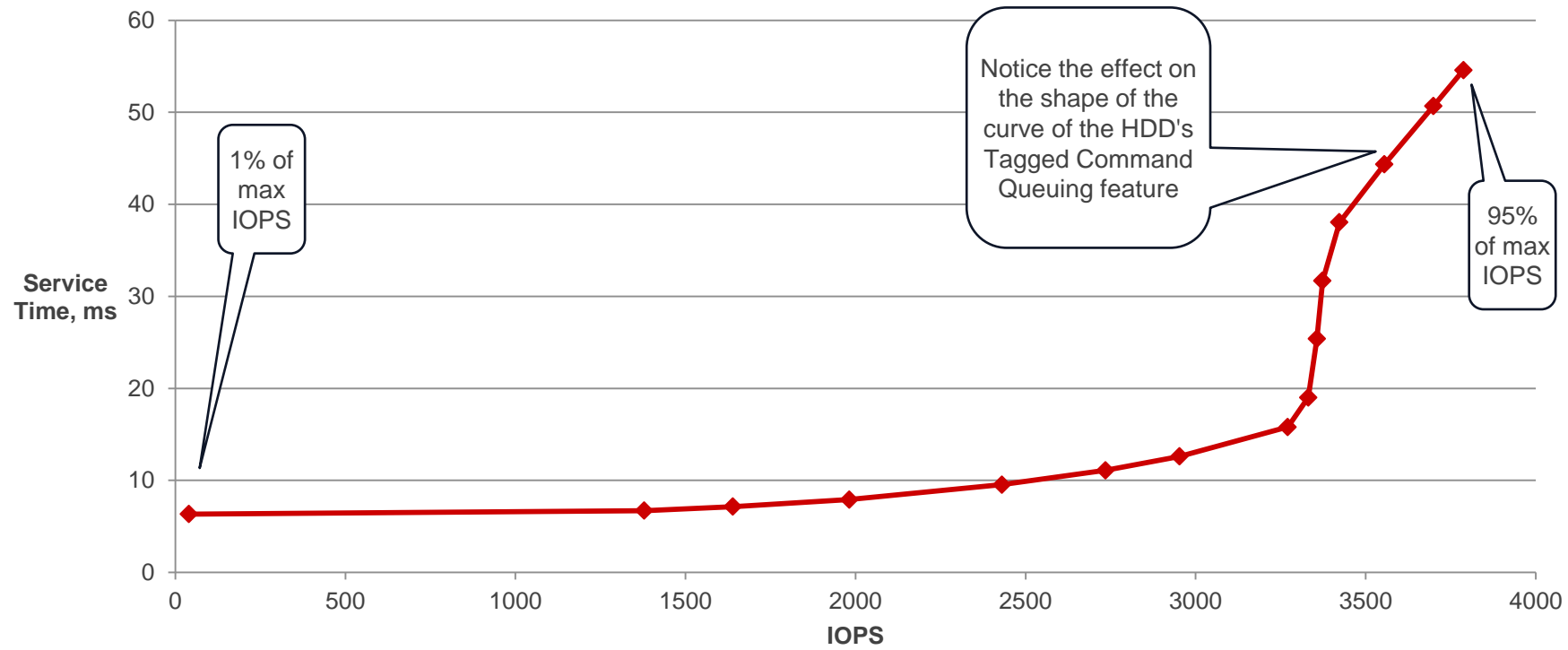
```
double multiplier, target_value;
string stepname;

for multiplier = { 1.0625, 1.125, 1.25, 1.5, 1.75, 2, 2.5, 3, 4, 5, 6, 7, 8 }
{
    target_value = low_target * multiplier;

    stepname = to_string_with_decimal_places(100.0*multiplier,3) + " x baseline service time";

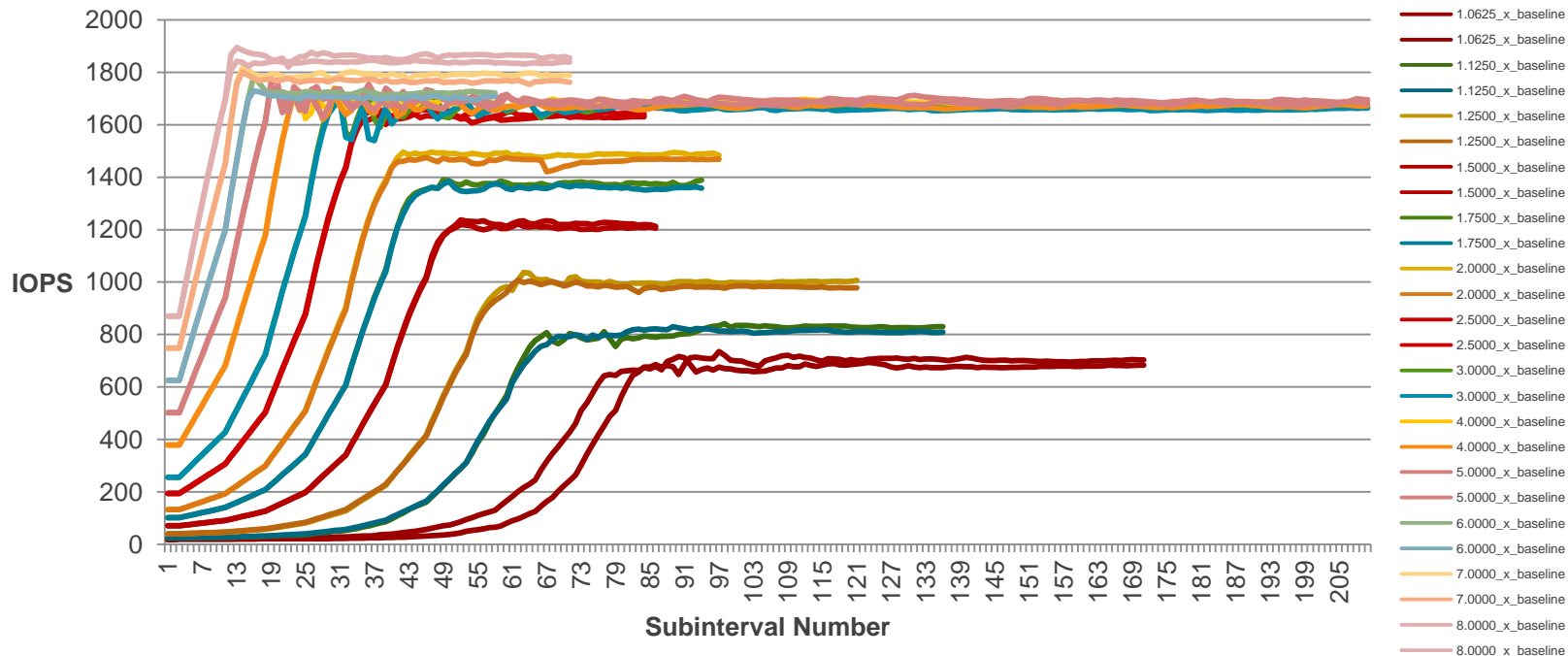
    [go] "stepname = \"" + stepname + "\""
        + ", focus_rollup = LDEV"
        + ", measure = service_time_seconds"
        + ", accuracy_plus_minus = 1%"
        + ", measure_seconds = 120"
        + ", timeout_seconds = \"1:00:00\""
        + ", dfc = PID"
        + ", max_ripple = 0.5%"
        + ", target_value = " + string(target_value)
        + ", low_IOPS = "      + string(low_IOPS)
        + ", low_target = "    + string(low_target)
        + ", high_IOPS = "     + string(high_IOPS)
        + ", high_target = "  + string(high_target);
}
```

# Test result



# Monitoring PID loop behaviour

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



- We may decide to write a "library" function that does the four steps
  1. `measure = IOPS, IOPS = max`
  2. Measure low IOPS / low target
  3. Measure high IOPS / high target
  4. `dfc = PID, target_value = t, low_IOPS = x,`  
`low_target = y, high_IOPS = a, high_target = b`

# How it works

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

- 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.

- 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.

- 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.



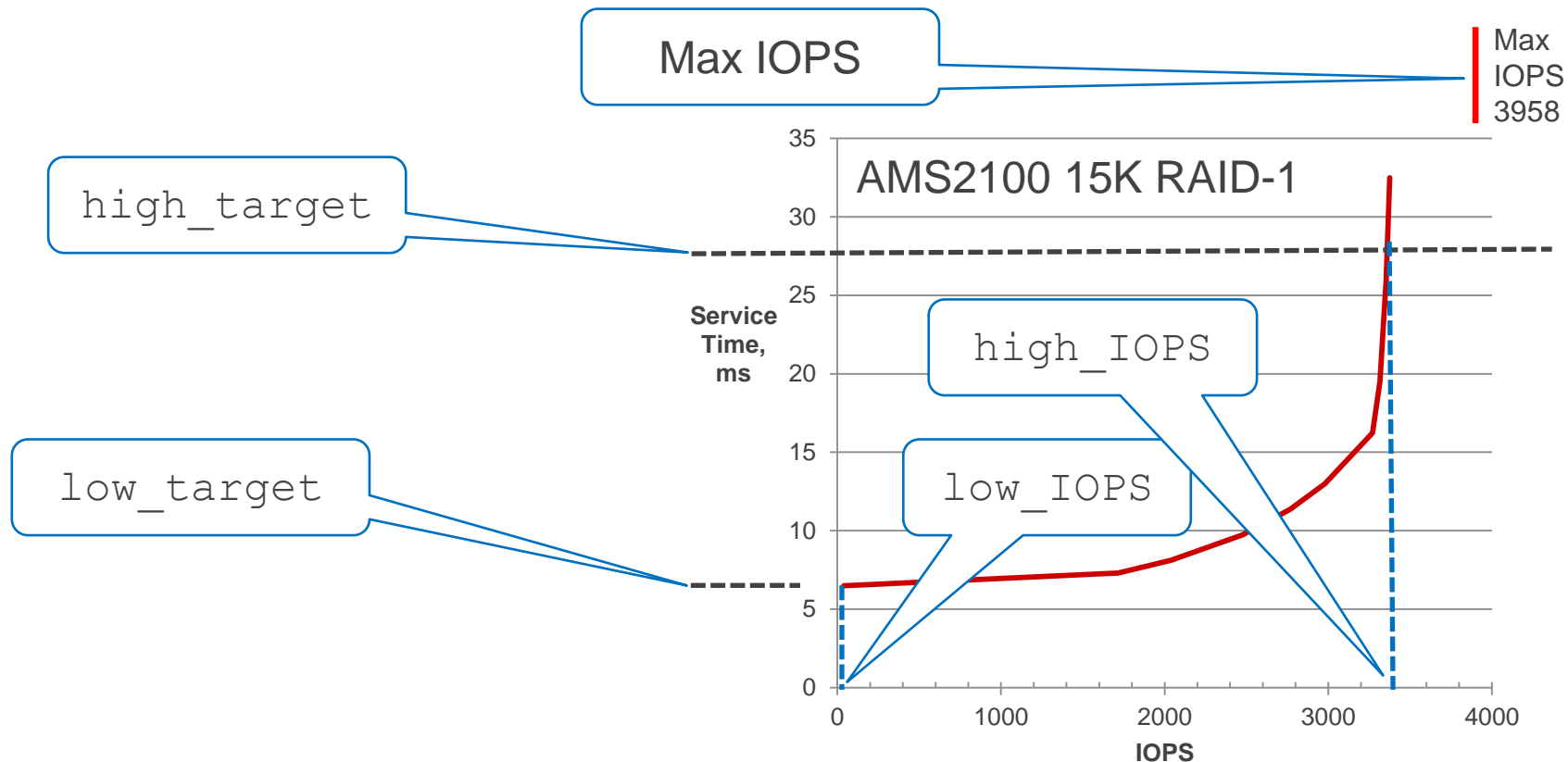
- We define the "operating range" for IOPS to be from 1% to 90% (or so) of max IOPS.
- We measure the PID control metric at 1% of max IOPS and at 90% of max IOPS.
- We define the "operating range" for the PID control metric to be from the measurement value at 1% of max IOPS to the measurement at 90%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

- 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

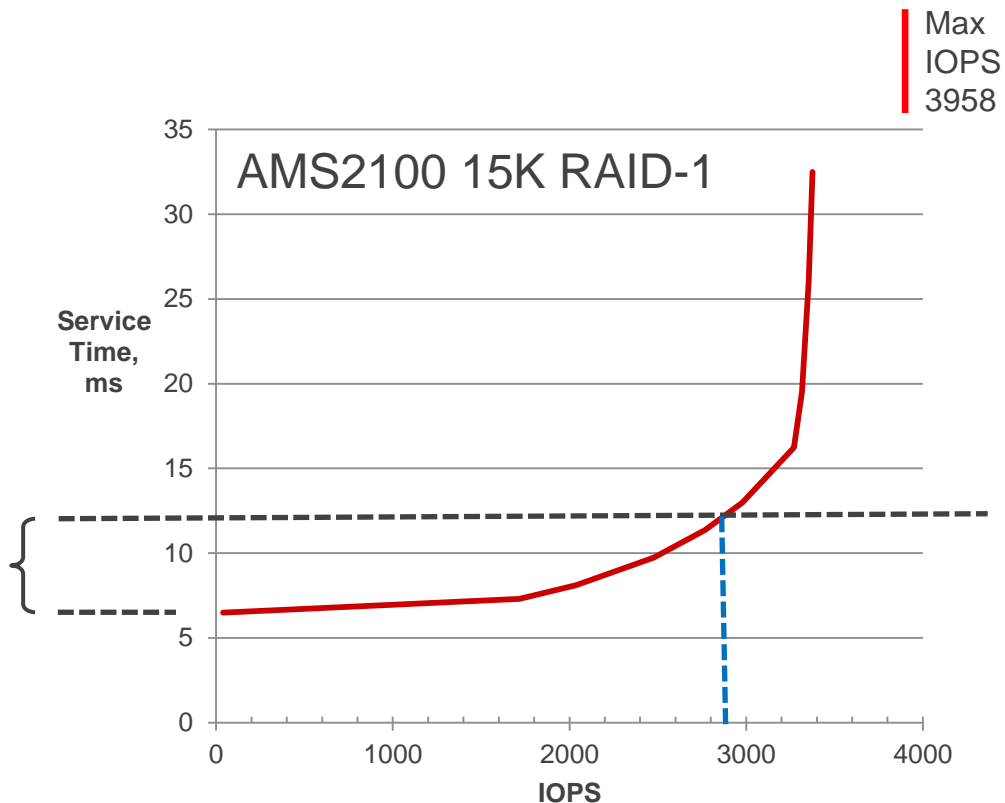


# Key concept is "initial error"

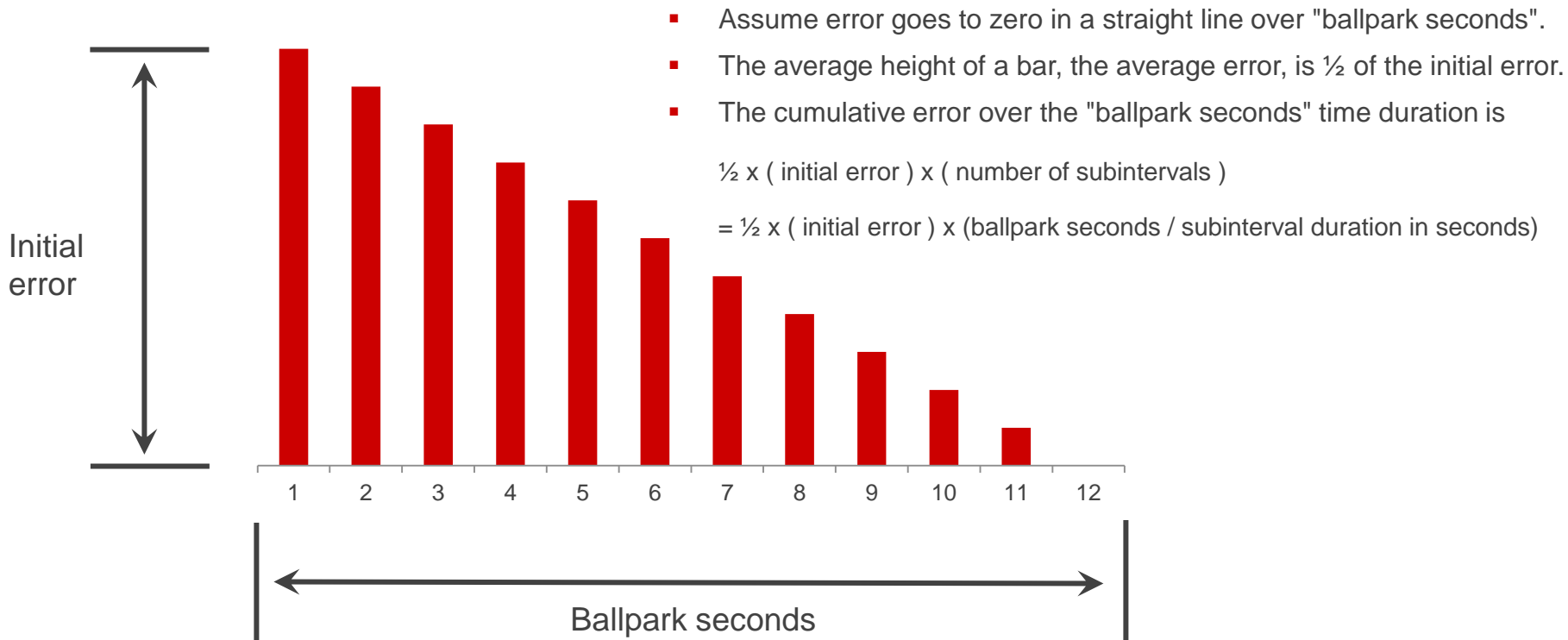
The "initial error" is the difference between the target value of the PID control metric, and the baseline value.

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



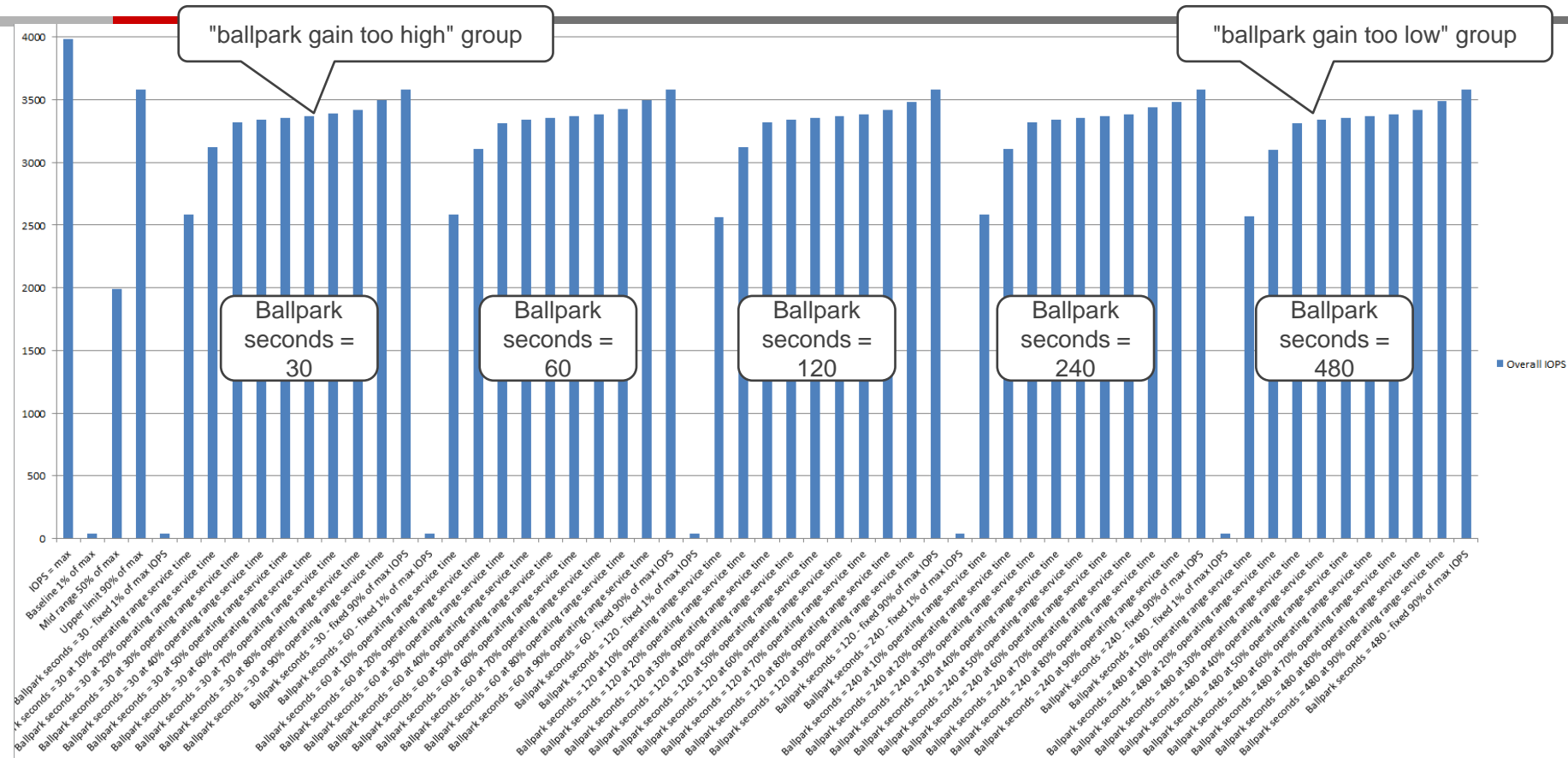
# "ballpark seconds" used to set gain sensitivity

- 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  **$\text{IOPS} = I \times \text{cumulative error}$** , we calculate starting gain  **$I = \text{estimated IOPS} / \text{estimated cumulative error}$** .

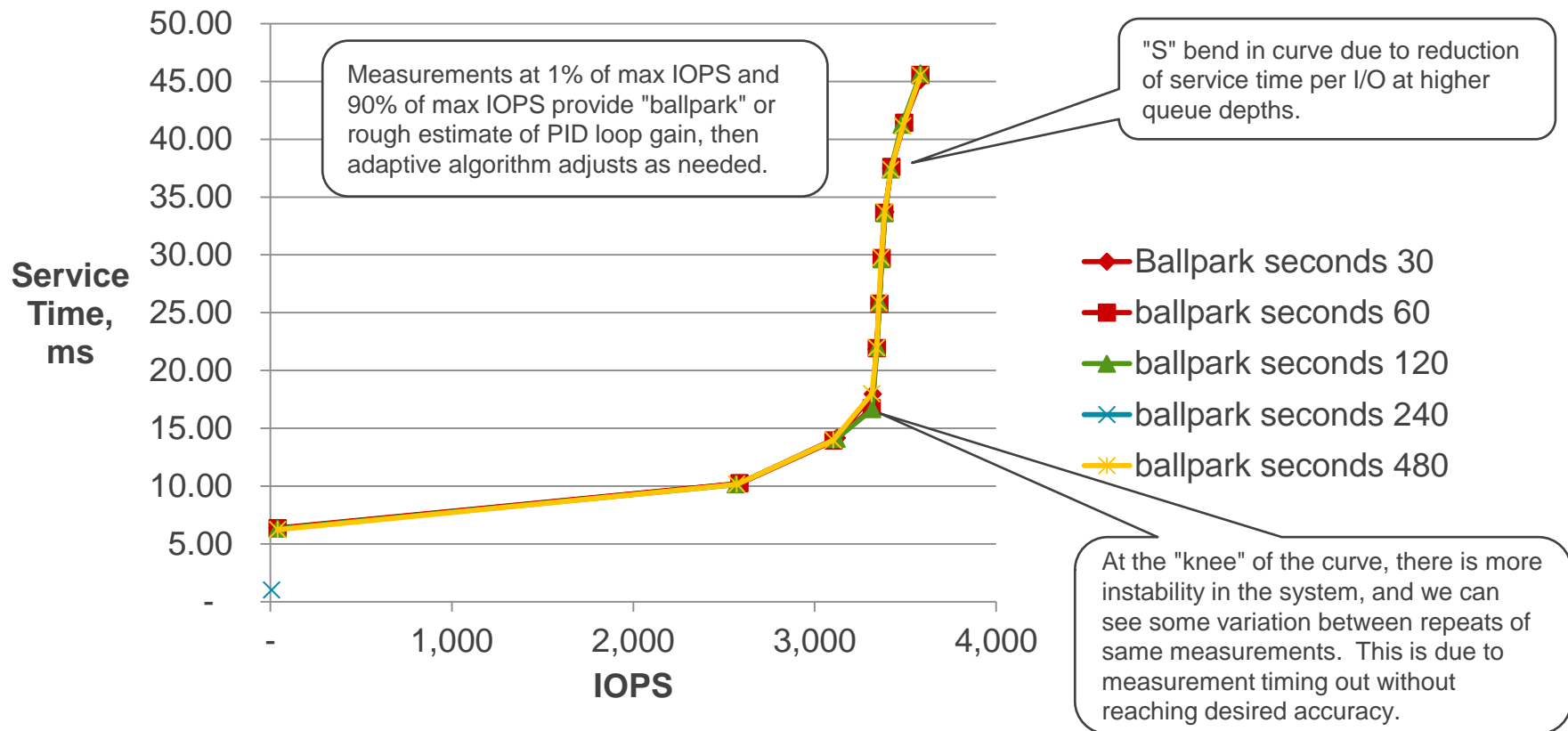
- The rough initial estimate followed by the use of the adaptive gain adjustment ensures a rapid approach to the target, followed fine-tuning to stably lock in on the target.
- It doesn't appear to be worth the time to make more than the two calibration measurements.



# Solid measurements for range of sensitivities



# Same data as previous chart – repeatability



**HITACHI**  
Inspire the Next 