



HITACHI
Inspire the Next

Programming the ivy engine

February 12, 2020

Allart Ian Vogelesang
Stephen P. Morgan

ian.vogelesang@hitachivantara.com
stephen.morgan@hitachivantara.com

This is a "reference" that describes all features

- For beginners, start with
 - Introduction to ivy
 - Getting started with ivy
- Also, beginners should review the ivy demos.
- This presentation, "Programming the ivy engine" is kept updated by the developers in sync with features as they become available.

The ivyscript wrapper and the ivy engine

1. ivyscript programming language wrapper and library (separate presentation)
 - Automate workflow, embody expertise in code library
 - Do something, analyze what happened, decide what to do next
 - Similar to a subset of C/C++, with some minor differences.
 - Extensible - parser auto-generated from language grammar. (Flex+Bison)
 - Each ivyscript ivy engine control statement maps to an underlying ivy engine control API call.
2. ivyscript ivy engine control statements (this material)
 - Each ivyscript engine control statement maps to an underlying ivy engine control C++ API.
 - See "ivy_engine_API.txt" output file to see what calls to the ivy engine API your ivyscript program makes.
 - There is also an ivy REST API, with corresponding ivy REST API calls mapping one-to-one to ivy engine C++ API calls.
 - Thus it's possible to use Python instead of ivyscript to operate the ivy engine.

Invoking `ivy` on the Linux command line

- `ivy [options] ivyscript_filename`

- Ivyscript filenames must end in `.ivyscript`.

If you leave off the `.ivyscript` suffix, `ivy` will add it before looking for the file.

- Options: (case insensitive, ignores underscores, e.g. `-noLDEV` same as `-no_ldev`.)

`-log` – turns on logging for `ivy` developer problem diagnosis. (Or else `ivy_engine_set("log", "on")` in your `ivyscript` program.)

`-no_cmd` – stops `ivy` from automatically connecting to a command device.

`-one_thread_per_core`

Normally `ivydriver` on each test host starts an I/O driving subthread on all hyperthreads of every Linux CPU `core_id`, except the first two hyperthreads on core 0. The `-one_thread_per_core` option only starts an I/O driving subthread on the first hyperthread of every `core_id` except core 0. This option should only be used when measuring service times at very low I/O rates, along with the `-spinloop` option.

`-no_wrap`

In csv files, `ivy` wraps PG names e.g. `1-1` and LDEV names like `10:00` as character string formulas like `"1-1"` to stop Excel from interpreting them as dates and times. `-no_wrap` suppresses this.

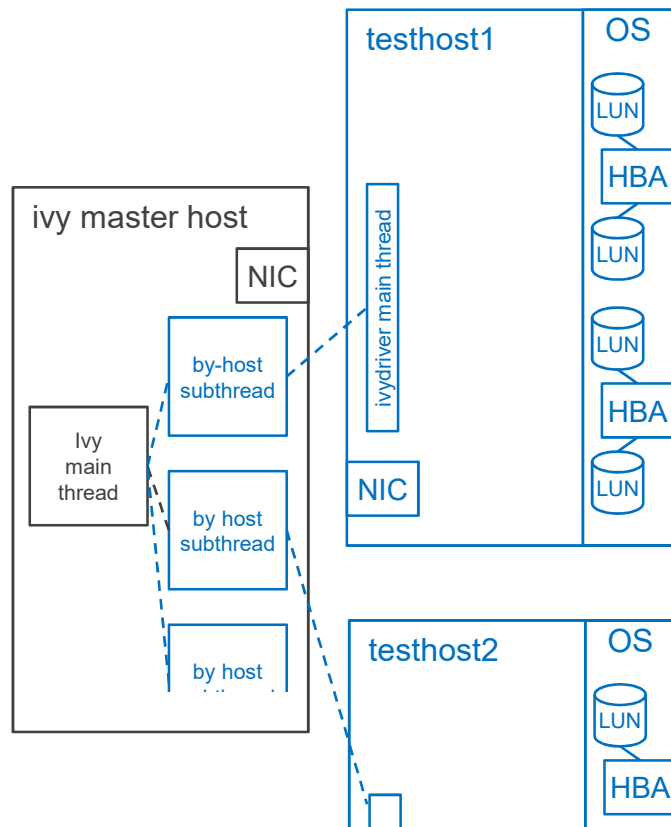
`-suppress_perf`, `-skip_LDEV`, & `-no_check_failed_component` – covered later in the [Go] parameter section.

ivyscript engine control statements

Each maps to its corresponding underlying ivy engine C++ API call.

- `[OutputFolderRoot] "/ivy_output";`
- `[Hosts] "sun159, cb[24-31]" [Select] << "serial number" : 123456, LDEV : 00:00-01:FF >>;`
- `[SetIosequencerTemplate] "random_steady" [parameters] "IOPS = 20, blocksize = 4KiB";`
- `[CreateWorkload] "cat" [iosequencer] "sequential" [parameters] "IOPS=max, maxTags=1";`
- `[DeleteWorkload] "cat" [select] "LDEV : 00:04";`
- `[CreateRollup] "host" [nocsv] [quantity] 8 [MaxDroop] 25%;`
- `[EditRollup] "serial_number+Port = { 410123+1A, 410123+2A }" [parameters] "maxTags=128";`
- `[DeleteRollup] "serial_number+Port";`
- `[Go] "stepname=whole_LUN_staggered_start, measure_seconds = 60";`

ivy engine startup – specify test host names, select LUNs

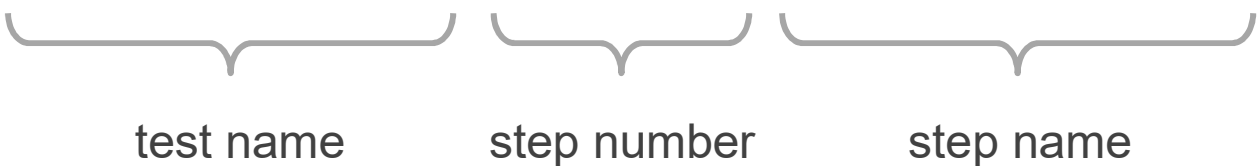


- `[Hosts] "testhost[1-8]"`
`[Select] << "serial number" : 123456,`
`LDEV : 00:00-01:FF >>;`
- Executing the ivyscript `[Hosts]` statement invokes the ivy engine API `startup()` method to use ssh to fire up the `ivydriver` executable remotely on the test hosts .
- Each test host runs a SCSI Inquiry tool to report back the decoded attributes of "all discovered LUNs".
 - The master "all discovered LUNs" LUN attribute list is written as a csv file with a header line for attribute names, and a detail line for each host LUN. Look here to see what you can select from.
- The `[select]` clause specifies a JSON-format* LUN attribute value filter to select "available test LUNs" from "all discovered LUNs".
 - Must specify at least "serial number" or "vendor" for safety reasons.
 - * ivy relaxes JSON – OK to omit surrounding braces {}, OK to omit double quotes around identifiers, LDEV ranges, parity group names, etc.

ivy engine startup - the "test name"

- When ivy is invoked on the command line like
 - `ivy some/path/henri.ivyscript`
- The part of the ivyscript filename discarding the path and the .ivyscript suffix, is called the **"test name"**.
 - This must be composed entirely of letters a-zA-Z, Japanese hiragana / katakana / kanji, digits 0-9, hyphens -, and underscores _.
 - Note: test names (output filenames) using Japanese characters in Linux are encoded in UTF-8 which may not display properly on Windows systems.
- The test name is used as the subfolder name off of the `[OutputFolderRoot]` folder, and is used as part of the filenames of ivy output csv files.

"test name" – used in output filename prefixes

- The test name is also used as part of the prefix of ivy output filenames.
 - Fully qualified csv files names incorporate test name, and other fields so you can combine together in one folder any files from multiple ivy runs without name collisions as long as the test names are different.
 - `demo1_fixed_DF.step0003.blocksize_8_KiB.all=all.csv`


<u>demo1_fixed_DF</u>	<u>step0003</u>	<u>blocksize_8_KiB.all=all</u>
test name	step number	step name

"step name" and "step number"

- Later we'll also see things like "step name" and "step number".
 - Ivyscript has some handy builtin functions to retrieve things like this to build csv file names and then load a csv file into an object to retrieve rows, columns, and cells including identifying columns by column header title.
 - This is how you retrieve the result of a previous test step to see what happened and decide what to do next.
 - There's also a built-in function to let you write to ivy master log file.

Statements – [OutputFolderRoot]

- [OutputFolderRoot] <string literal>;
 - Specifies a root folder which must already exist.
 - The default is "." (the current folder).
 - Specifies the root folder in which ivy will make a subfolder to record the output from running an `.ivyscript` program.
- A string literal (string constant) is required, because the output root folder name is captured at compile time.
 - This way, the output folder structure and log files can be all in place before the `ivyscript` program starts running.
 - At most one [OutputFolderRoot] statement, anywhere in your program.

Specifying host names

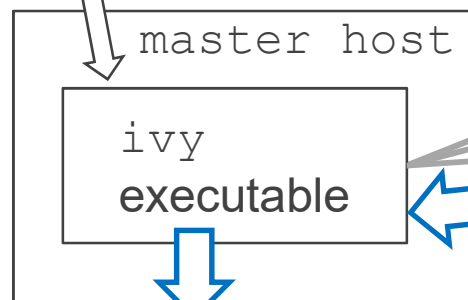
- `[Hosts] "sun159, horde[33-64], 192.168.0.0"`
`[Select] "xxx";`
- Host name forms
 - `sun159` single host
 - `Horde[33-64]` range of hosts with consecutive numeric suffixes
 - `192.168.0.0` IPV4 dotted quad

Vendor-independent LUN attribute discovery

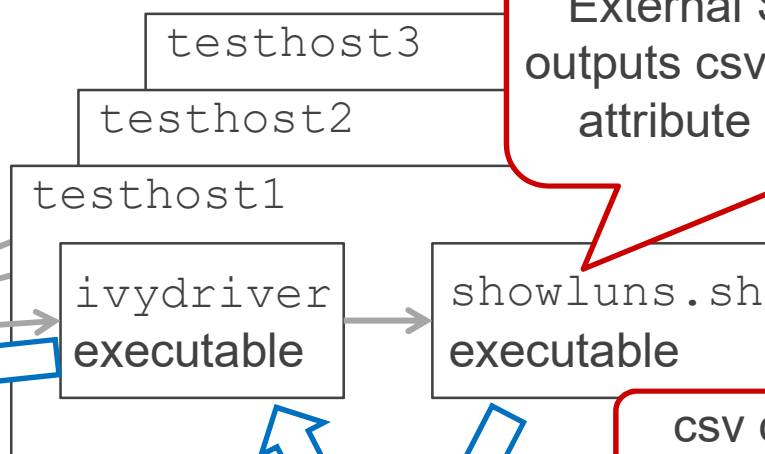
command line: `ivy test2`

`test2.ivyscript`

`[hosts] "testhost[1-3]"`



```
Host,LUN,HDS Product,LDEV,PG
testhost1,/dev/sdxy,VSP,00:00,1-1
testhost2,/dev/sdxy,VSP,00:01,1-2
testhost3,/dev/sdxy,VSP,00:02,1-3
```



External SCSI Inquiry tool
outputs csv file decoding LUN
attribute names & values

csv column headings
become selectable in ivy

```
Host,LUN,HDS Product,LDEV,PG
testhost1,/dev/sdxy,VSP,00:00,1-1
```

Sample attribute values from LUN lister tool

```
hostname = cb23
LUN Name = /dev/sdbu
Hitachi Product = HM700
Hitachi Vantara Product = "HUS VM"
Serial Number = 210030
Port = 1A
LDEV = 00:00
Nickname = James
LDEV type = Internal
RAID level = RAID-6(6+2)
PG Names = 1-1/1-2
Pool ID = 2
CLPR = CLPR0
Max LBA = 2097151
Size MB = 1073.741824

Size MiB = 1024.000000
Size GB = 1.073742
Size GiB = 1.000000
Size TB = 0.001074
Size TiB = 0.000977
Vendor = HITACHI
Product = OPEN-V
```

There are more attributes on newer subsystem types.

Get the latest version of the Hitachi LUN discovery tool,
open source software, found at
https://github.com/Hitachi-Data-Systems/LUN_discovery

LUN attribute matching

- The LUN discovery (SCSI Inquiry) tool output csv file header line defines the LUN attribute names:
 - e.g. "Hitachi Vantara Product, Serial Number, LDEV, ..."
- Each header line csv column title automatically becomes selectable as a LUN attribute in ivy.
 - This is what makes ivy vendor independent. To support another vendor's architecture and terminology in ivy, all you need is a SCSI Inquiry tool that makes a csv file with a header line defining the LUN attribute names, and a data line for each LUN showing the attribute values for that LUN.
- There are a handful of "custom" attribute value matchers matching specific token types for Hitachi subsystems, shown in the following charts.
 - Other vendors are encouraged to write their own.

LUN attribute [Select] uses JSON syntax

- JSON format is used to describe attribute names and selected values
 - { "LDEV type" : "DP-Vol", "port" : ["1A", "3A"] }
- In ivyscript, to make typing character strings containing double quote marks easy, use “raw strings” surrounded by << and >>, as in:

```
[select] << { "LDEV type" : "DP-Vol", "port" : [ "1A", "3A" ] } >>
```

Otherwise, you would need to type:

```
[select] "{ \"LDEV type\" : \"DP-Vol\", \"port\" : [ \"1A\", \"3A\" ] }"
```

ivy supports a “relaxed” JSON syntax

- Strict JSON format is welcome, but to spare some typing and make ivy a little friendlier, some types of things don't have to be in quotes. (But anything containing commas or spaces does need to be in quotes.)

- `robert`
- `00:00`
 - A single Hitachi LDEV (logical device)
- `00:00-01:FF`
 - A range of LDEVs
- `1-1`
 - A Hitachi parity group name
- `1A`
 - A Hitachi subsystem port name
- All integer/floating point values (not just JSON-specific numeric values)

[Select] LDEV

- There is a custom matcher for LDEV which understands
 - "00:1A-00:3F, 01:00, 05:00 - 05:FF"
 - Commas and / or spaces are used as delimiters between single LDEVs like 00:10, or consecutive LDEV subranges like 00:1A-00:3F
 - Spaces are optional around the hyphen in a consecutive range.
- These LDEV specifications don't need to be in quotes. Note they don't contain spaces or commas:
 - 01:FF
 - 00:1A-00:3F

[Select] PG

- Single PGs like 1-1 don't need to be in quotes.
 - [select] "PG : [1-1, 2-3] "
- There is a custom matcher for "PG" which understands
 - "1-*" matches PG names starting with 1-
 - "1-2:4" matches 1-2, 1-3, 1-4 (or 01-02, 01-03, 01-04)
 - "1-2:" matches 1-2, 1-3, ...
 - "1-:2" matches 1-1, 1-2
 - These special matching ranges **do** need to be in quotes, unlike a single PG constant like 1-1.

[hosts] – use of command devices is automatic

- After we have "available test LUNs", (which excludes command devices)
 - The [hosts] statement looks through the command devices that were part of "all discovered LUNs".
 - For each unique subsystem serial number represented in "available test LUNs",
 - For the first command device found that goes to that subsystem on a host where the Hitachi proprietary command device connector "ivy_cmddev" (not part of the ivy open source project) is available, and where the ivy_cmddev license key and RMLIB are installed, we fire the ivy command device connector ivy_cmddev up remotely on the test host that has the command device, and retrieve the RMLIB API data on the configuration of the subsystem. This is completely transparent to an ivyscript program.
 - **To disable the use of command devices, use the “-no_cmd” command line option when running ivy. To only collect configuration data from a command device, but not performance data while ivy is running, use the “-no_perf” option.**
- For each available test LUN, if we have subsystem configuration data for the LDEV behind that LUN, the subsystem LDEV configuration attribute value pairs are merged into the LUN's attributes – add attribute name suffix upon collision with different attribute value.
 - That means that if you have a command device, you can select on `drive_type` to create a workload.
 - "drive_type" even works for DP-Vols, as ivy follows the config info to find the pool vols and use their drive type.

Real time subsystem data filtered by rollup instance

- Later we will show you how to perform dynamic feedback control at the granularity of each instance of a rollup
 - DFC can be performed on real-time subsystem data at the granularity of the rollup instance using "subsystem component filters", which list the names of each port, each MP_core, each PG, etc. that are associated with the workloads and their underlying LUNs that comprise the rollup instance.
- This is done with a combination of having the configuration data and knowing the fixed relationships in all instances of a subsystem model.
 - The knowledge of which MP_cores comprise an MPU together with the LDEV MPU assignment allow us to filter MP_core data by rollup instance.

Statements - [SetIosequencerTemplate]

- [SetIosequencerTemplate] "random_steady"
[parameter]

The use of [SetIosequencerTemplate] is deprecated, meaning it's an old thing that still works, but users are requested to avoid using it.

fraction_read
fractionRead, etc.

- Sets the default
- If you are going to use [SetIosequencerTemplate] you create a workload.

Instead, please use [EditRollup] to set the same parameters across a selection of existing workloads once they have been created.

variations, you could
common, and then when
workload.

- Handy if you are starting at a different point in the LUN or have been reading the program, it's more clear what's going on if the [CreateWorkload] only sets what's different each time.

At some point in the future, [SetIosequencerTemplate] may be removed from ivy.

[iosequencer] **some common** [parameters]

- RangeStart **default 0.0 same as 0%** (0.0 means beginning of LUN, 1.0 means end of LUN)
RangeEnd **default 1.0 same as 100%**
 - Establishes the "coverage zone" within the LUN. You can layer different workloads in different parts of the same LUN.
 - RangeStart and RangeEnd can also be specified as a value in MB, MiB, GB, GiB, TB, or TiB relative to the beginning of the LUN. (One MB is 1000000, and one MiB is 1024 * 1024, etc.)
- blocksize **default "4 KiB" same as "4096"** – also supports "MiB" units.
- maxTags **default 1.**
 - The maximum number of I/Os that this workload on this LUN is allowed to **try** to issue at one time.
 - OS call to start I/Os may block if underlying HBA/device driver is out of tags. Workloads share LUNs and share the underlying HBA/device driver.
- IOPS **default 5**
 - IOPS = "max" - keep starting I/Os trying to keep queue depth at "maxTags".
- fractionRead **default 1.0 same as 100%.**

[iosequencer] random – two types

- `random_steady`
 - I/Os are issued to random locations on a steady drumbeat in time.
 - A "location" means an LBA (Logical Block Address, or sector number) that is aligned on a multiple of the blocksize being generated.
- `random_independent`
 - I/Os occur at random times as well as to random locations
 - Random independent distributions are easier to model mathematically.
 - The lower the IOPS rate or the shorter the observation period, the more erratic `random_independent` IOPS will appear.
 - In general, `random_independent` I/O patterns will have a slightly higher service time compared to `random_steady` workloads, because scheduled I/O start times are independent and in general can collide (bursty), whereas `random_steady` workloads space out I/O scheduled start times evenly.

[iosequencer] "sequential"

- In ivy, a sequential workload must be all reads (`fractionRead=1.0` or `fractionRead=100%`) or all writes (`fractionRead=0%`).
- But, you can use a for loop to create a series of sequential threads starting at different points along the LUN, where each of the threads is either a read thread or a write thread
 - `SeqStartPoint = 0.23`
 - Range is from 0.0 to less than 1.0 - this is relative to the volume coverage zone defined from `RangeStart` to `RangeEnd`.
 - More commonly use the volume coverage parameters to have sequential threads wrap around in their own areas.

Workload placement in part of the LUN

- Use a loop to create 10 sequential threads where each of the 10 threads operates within its own 1/10th of the LUN – its own “zone”, so that when it gets to the end of its own zone, it should wrap around to the beginning of that zone. You can layer different workload types in different parts of a LUN.

```
[hosts] "sun159" [select] << { "serial_number" : 83011441 } >>;

int    zones = 10;
int    zone;
double start, end;

for (zone = 0; zone < zones; zone = zone + 1)
{
    start = double(zone)/double(zones);
    end = double(zone+1)/double(zones);

    [CreateWorkload] "zone" + string(zone)
        [iosequencer] "sequential"
        [parameters] "RangeStart=" + string(start)
                    + ",RangeEnd=" + string(end)
                    + ",IOPS=max, blocksize=64KiB, fractionRead=100%, maxTags=1";
}

[CreateRollup] "workload"; // this is to give us data by zone across all LUNs

[Go] "stepname=separate_zones, measure_seconds = 60";
```

Start a sequential thread at a point

SeqStartPoint

- Use a loop to create 10 sequential threads where each of the threads covers the entire LUN, wrapping around from the end of the entire LUN to the beginning of the LUN, but where each thread starts at a different equally spaced point.

```
[hosts] "sun159" [select] << { "serial_number" : 83011441 } >>;

int    zones = 10;
int    zone;
double start;

for (zone = 0; zone < zones; zone = zone + 1)
{
    start = double(zone)/double(zones);

    [CreateWorkload] "zone" + string(zone)
        [iosequencer] "sequential"
        [parameters] "SeqStartPoint=" + string(start)
                    + ",IOPS=max, blocksize=64KiB, fractionRead=100%, maxTags=1";
}

[CreateRollup] "workload"; // this is to give us data by zone across all LUNs

[go] "stepname=whole_LUN_staggered_start, measure_seconds = 60";
```

Sequential – mixing read threads & write threads

- Use a loop to create a group of sequential workload threads each operating within its own "zone", and where some threads do writes and some do reads.

```
- [hosts] "sun159" [select] << { "serial_number" : 83011441 } >>;

int zones = 12; int zone; double start, end; double seq_percent_read = 75%;

for (zone = 0; zone < zones; zone = zone + 1) {
    start = double(zone)/double(zones); end = double(zone+1)/double(zones);

    double rw; string p;

    if ( ( double(zone) / double(zones) ) < seq_percent_read ) { rw = 100%; p = "read_"; }
    else { rw = 0%; p = "write_"; }

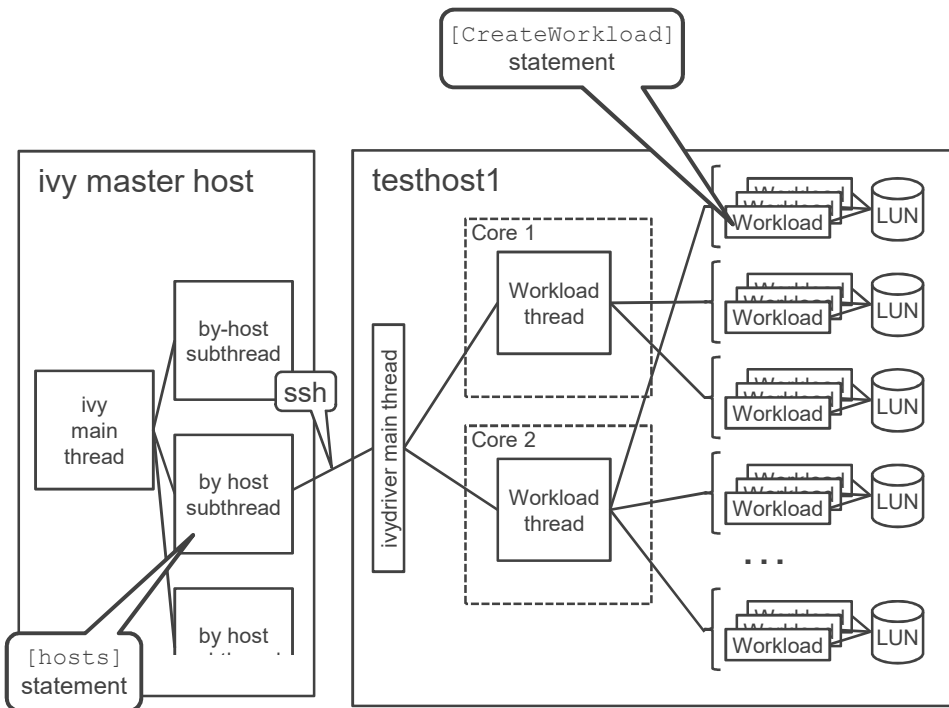
    [CreateWorkload] p + "zone" + string(zone)
        [iosequencer] "sequential"
        [parameters] "RangeStart=" + string(start)
                    + ",RangeEnd=" + string(end)
                    + ",IOPS=max, blocksize=64KiB, maxTags=1"
                    + ", fractionRead=" + string(rw);
}

[Go] "stepname=read_and_write_zones, measure_seconds = 60";
```

Sequential workloads and `maxTags`

- The default `maxTags` value is 1.
- ivy iosequencers generate a sequence of I/Os in scheduled start time order.
- For `IOPS=max`, the scheduled start time for each I/O is zero.
- For all iosequencers, if you specify, for example, `maxTags=4`, this means "keep issuing I/Os when it's their scheduled time, except wait to start the next I/O if there are already 4 I/Os running".
 - For a sequential workload, `IOPS=max`, `maxTags=4` means "issue I/Os for 4 consecutive blocks at once, and then when one of these completes, keep issuing more to try to keep 4 running at all times."
- Note: The ivy method was chosen so as not to suffer from the issue with vdbench when specifying "threads = n" for $n > 1$ with sequential workloads, where each vdbench thread reads block 0, then block 1, reading each block multiple times which is clearly not what customer workloads would be expected to do.

Statements – [CreateWorkload]



- `[CreateWorkload] "r_steady"`
`[select] << {"LDEV":"00:00-00:1F"} >>`
`[iosequencer] "random_steady"`
`[parameters] "fractionRead = 75%"`
- Apply a `[select]` filter matching against "available test LUN" attribute values.
- On each selected LUN, create an identical workload with the specified workload name.
 - Each type of `iosequencer` has its own set of valid parameter names.

.ivyscript dedupe syntax

- ```
[CreateWorkload] "owl"
 [select] << { "port" : ["1A", "2A"] } >>
 [iosequencer] "random_steady"
 [parameters] << IOPS=max, fraction_read=50%,
 blocksize = "8 KiB"
 dedupe = 1.5 >>;
```
- The dedupe parameter (default dedupe = 1.0) controls the average number of copies of a generated pattern that are written.
  - dedupe must set to a value greater than or equal to 1.0
  - For dedupe\_method=serpentine it is an error if some workload threads are set to a different dedupe value than other workload threads with the same name.
  - The dedupe parameter is ignored for fraction\_read = 100%.

# dedupe\_method

- `constant_ratio`
  - For each block address within the LUN, flips back and forth between a set of fixed patterns.
  - When writing sequentially to fill and then writing randomly, the achieved dedupe ratio remains constant.
- `static (default)`
  - For each location in the LUN, always writes exactly the same pattern in that location.
  - The vast majority of blocks have a pattern that occurs only once in the LUN.
  - There is a small "duplicate set" of distinct original patterns from which duplicate copies are made.
  - The same number of duplicate copies of each original pattern in the "duplicate set" are made.
    - Or as close to the same number of duplicate copies as is possible to achieve the `dedupe_ratio`.
  - The default is `duplicate_set_size = 128`, but this can be set to any size.

## dedupe\_method

- round\_robin
  - Writes blocks from a “duplicate set” round-robin across all the LUNs in a workload.
  - When writing sequentially to fill and then writing randomly, the achieved dedupe ratio remains constant.
  - Duplicate blocks are always written statically, but not non-duplicates.
  - The duplicate set is automatically sized to be about 0.5% of the total blocks in all the LUNs in the workload.
  - This method provides the best performance with Hitachi Adaptive Data Reduction (ADR).
  - Achieves very close to the requested dedupe ratio (usually within 0.1%).



# dedupe\_method

- For `constant_ratio`, `static`, & `round_robin`
  - `blocksize` must be a multiple of `dedupe_unit_bytes`. (Default `dedupe_unit_bytes` = "8 KiB".)
  - Specify `fraction_zero_pattern` = 15% to have 15% of all 8 KiB (sub-)blocks to be written with an all zeros pattern.
  - The `dedupe` ratio applies to the sub-blocks that are not all zeros.
  - For example, with `fraction_zero_pattern` = 25% and `dedupe` = 1.5, 25% of all 8 KiB (sub-) blocks will be all zeros, of the remaining 75% of all (sub-)blocks, one third will be additional duplicate copies.
- For `constant_ratio` & `static`
  - The duplicate copies are written within a single workload instance, i.e. within the same LUN. In other words, the patterns written in one LUN don't appear in any other LUN.
- For `round_robin`
  - The duplicate copies are written across all the LUNs within the same workload.

## dedupe\_method

- `serpentine`
  - Generates a new unique pattern each time.
  - Writes the specified “dedupe” number of multiple copies at the same time, whether randomly or sequentially.
  - When writing sequentially to fill, achieved dedupe ratio is “dedupe”, then when subsequently writing randomly, achieved dedupe ratio decays.
  - The `serpentine` method’s “scope”, meaning the range over which the duplicate copies are written, covers all the workloads across all LUNs, all ports, all subsystems which share the same workload name.
- `target_spread`
  - Some patterns are unique, some repeat, targeting to achieve an average of “dedupe” copies.

## .ivyscript compressibility syntax

- ```
[CreateWorkload] "owl"  
  [select] << { "port" : [ "1A", "2A" ] } >>  
  [iosequencer] "random_steady"  
  [parameters]<< IOPS=max,fraction_read=50%,  
                 blocksize = "4 KiB"  
                 pattern = random >>;
```
- The `pattern` parameter selects a pattern generator to fill the contents of a block before it is written to the LUN.
- When `dedupe` is not set to the default `dedupe=1`, the default is `pattern = random`.
 - See next page.
- Don't use `pattern` values of `zeros`, `all_0xFF`, `all_0x0F`, or whatever with `dedupe` & compression.

.ivyscript pattern parameter

- `pattern = whatever`
 - Doesn't generate a data pattern for writes. Whatever just happens to be in the memory buffer is written. This is the default pattern when `dedupe=1`, which is the default value for `dedupe`.
- `pattern = random`
 - Random binary noise. Not compressible. This is the default pattern when `dedupe` is not set to `dedupe=1`.
- `pattern = trailing_blanks, compressibility = 50%`
 - Each 8 KiB part of block has an incompressible section and a section with repeated blanks.
 - `compressibility` specifies the % of the block that is repeating blanks.
- `pattern = ascii`
 - Random ascii characters. Fixed degree of compressibility
- `pattern = gobbledegook`
 - Pseudo-English text generated by randomly selecting words from a dictionary.
 - Fixed degree of compressibility.
- `pattern = zeros, pattern = all_0x0F, pattern = all_0xFF`
 - Self explanatory

Using the first 32 Ki Words appearing in the 1913 public domain edition of Webster's dictionary.

pattern=random

```
offset 0x0000 0 ".G.n....:~wuH/...f.cM.....6." (ffd44709 6ecaf4c6 fb3a25be 7e777548 2fa3c79b 66a2634d 9b04101f 1dab36ef)
offset 0x0020 32 ".....N\QL..I.....?.....~.j.U4" (e816f69a 7f0a4e1b 5c514ce4 e549c1a4 90dd0cc5 3f13ba91 84857381 6c885534)
offset 0x0040 64 "~.p...!<.Z.=.:#;P...7...|>*.Q." (7e89701e bf84213c 1d5ae83d e184233b 50e6aeff 0f370bec 84857381 6c885534)
offset 0x0060 96 "...Z/.;}q.../.$.2...p.....JE." (b81d7d87 135a2fc7 3b7d71fc 10e62fce 24b60532 96d58370 84857381 6c885534)
offset 0x0080 128 ".4.I3.M....h].....I0...;..V$".l" (8434e949 33d64d02 dbed9d68 5df512ee 97a51949 3098881e 84857381 6c885534)
offset 0x00a0 160 ".....h.#l[...G.n....<....be...e" (d7e40db2 9b0fc868 1a236c5b e29147d5 6e9ba8c3 ef3c10aa 84857381 6c885534)
offset 0x00c0 192 ".../.....w.....y~W%;Ao..[x..Y.A" (d2c9a32f 17c099c2 b092779a e9c2f8af 797e5725 3b416f15 84857381 6c885534)
offset 0x00e0 224 "...?\\...{.H/mw.)}F....G.P.Y.v+\\. " (f4e1913f 5cd1147b c1482f6d 77b27d46 8d97a484 ba478d50 84857381 6c885534)
offset 0x0100 256 "...+M..Q.+..n~.t...v.....^=k.G" (92152b4d 95d251dc 2b10b76e 7ea87409 ede576ed 0869cfa9 84857381 6c885534)
offset 0x0120 288 "...b.8r`P{9....a.\\}@.*P%....$=" (94aff162 9a387260 507b390c ee80e961 1e5c5d40 408c2a50 84857381 6c885534)
offset 0x0140 320 "...||>.....MV.J.&..S..7..f.._" (ad1d7c7c 3eeb14ef f6aae09b 0f1c4d56 e84ae126 1ed65319 84857381 6c885534)
offset 0x0160 352 "...H..z.AB.....$......=CZ...|" (890948a3 d07aee41 42b91aed 2c9de494 2497ccc2 1da2e707 84857381 6c885534)
offset 0x0180 384 "...\\.....& ...0.&#.....s.?..0'" (125c925c e59fe31c f60fc926 20f0be09 4f932623 7d687d6 84857381 6c885534)
offset 0x01a0 416 "...^..R..|.....stq$....|....." (a9a6d3d3 5e8c527f 817cc7e8 d88c92be 73747124 c19ae1b3 7ce3d6fe f00d98a5)
offset 0x01c0 448 "...lt...I.A[S[.c.....\\.....l_x" (8a3174a6 0ddd49e6 415b535b 9b638f05 d1f08d5c 8d97a9db 85181698 6c5fae78)
offset 0x01e0 480 "...Y.P.<...w.hb>.n&..d.IE}e.w..g.b" (f0590f50 cc3ceffe bb77e368 623eb26e 2612f664 db49457d 65ae77a2 9467f462)
offset 0x0200 512 "...W..g...L&V#....+.q}...3....." (57e49d67 b18cd74c 26562304 8094ba2b 06717dea d72c33cc ddf5b4c1 098f0b1e)
offset 0x0220 544 "...[.m....dN..lw...5o....f..C...." (d0c85ba9 6de71307 9c644ead fe6c7704 c8ac356f ba12d7d3 66b6c543 84e717e7)
offset 0x0240 576 "...*u../4.O.q..4....w...x...m." (c1d71713 2a75c5b6 2f34fa4f 1271e980 34cf9b00 18d377c0 dc1a78be 1ba16dea)
offset 0x0260 608 "#..6.>...i.....P..l.giv...|.5.." (23bc1f36 ec3efe10 a169f489 ae0fd68c 50eaad6c b3676976 b310fd7c 8a35e1e3)
offset 0x0280 640 ".....N..Bs.....J^(<.k.cp..b..SU" (e6861bf1 a54ee8a1 4273c0c8 d3dffef7 4a5e7b3c fb6be163 70afb362 de955355)
offset 0x02a0 672 "...-B}.}..a<9.....k.....y.8.<.7.a" (a32d427d 907d0ca5 613c39f7 a98b90d3 6b17c2f6 19088c79 9e380a3c a237c861)
offset 0x02c0 704 ".....}.&);d... 'ki.....X...." (f20ed0c9 ee0dacc0 1c7d1a26 293b64da 8b7f276b 69f01aee d68cab58 9fade706)
offset 0x02e0 736 "...p.....u6...5}.U....~..._..b.." (d10770ca 0c86f111 b0fe7536 ca99bc35 7db9ee55 8fc7db8b 7ef8065f 8e62c6a5)
offset 0x0300 768 "...3.MsP.4N.u.Y...*.@...C#.....g." (0fed0433 d34d7350 bc344e0f 75be5918 e12afdd5 40dd99f8 4323a5fc a6086717)
offset 0x0320 800 "...e..jbd..E.....q...>K..%.9.. " (8f65f999 6a6264e7 e845f916 97dbceea 9ea58371 cdab3e4b e4e725da 390a60b9)
offset 0x0340 832 "...o..cM.....c.....s..5.E.ro..712." (6fb2ac63 4dc3b00c eb1713b5 e36397dd 0973cdfc 350645b6 726f0587 37315ac2)
offset 0x0360 864 "c... " ...B.r.x,.e.o..No.K...L..[" (63939ac6 22c5b295 42e672b9 782cb865 986feef2 4e6ff34b bde6e04c 11c45bd0)
offset 0x0380 896 "... (h..8\\Vw..\\Tc.?_'.Q....." (062868d4 fc385c19 5677e9c6 5ca75463 ab3f5f60 95517fe4 dd8ae4e5 f884d519)
offset 0x03a0 928 "...$......Q,-.....C..&....." (b1249cfa 88a1cbbf 07fc512c 2d93a7e1 09ebe643 2cdcb026 960bc381 c6d19b04)
offset 0x03c0 960 "...7hF.9.l.O8..... _'.`.....3j5." (ccdb8f07 3768461e 39ed6cf2 4f38e3ec f4e0209d a15fe560 a8fe1cdd 336a359c)
offset 0x03e0 992 "...6...P7.[.;.....).K.....oi..." (edd6367f e1e45037 ae5b923b d78ef3b2 e729b1fe 4b1fa79f 05e7f96f 69dcef9a)
```

Random binary data
(incompressible)

pattern=trailing_blanks,compressibility=33%,blocksize="1 KiB"

buffer contents:

```
offset 0x0 (0) "[.....7hH1L.U=.....~QPC...W.H.$." (5b81c8e7 f3193768 486c4cc4 553df104 93e1937e 5150431e 05f457ce 48b024d0)
offset 0x20 (32) ".4.....<.3.../.....#...." (89348d95 018b9d06 c8cff3ba 88a5113c e133bb01 fd2fbfb7 f5001d23 1a089c11)
offset 0x40 (64) "X..W.&.K.?.v.....g.z..d.....]qe" (58ccfe57 b926a94b 063fa276 d719abb0 0d1d67e8 7ab5b064 c49bd00e c05d7165)
offset 0x60 (96) "4....+.M.!B35.I..j...l.a..K..." (34d7bc0f db2bd11b 4deea721 42333596 49cbd96a 98f4c331 f461b512 4bd4f501)
offset 0x80 (128) ".g..vv...Q...|.....6..f..o..b..{" (c0678a03 7676d299 a75108ce 057cea9a dfe81336 cba3660f d86fff62 d7b97b11)
offset 0xA0 (160) ".....fQ.;;k..Z.3...fA".....X..." (12b00084 de6651cc 3a3b7d6b f8a85abe 33d4b2cb 6641220f baf4e815 5809c61e)
offset 0xC0 (192) "...&E=;q.w.-~.^q...8...jJ.{...yD..." (08c62645 3d3b719f 772da47e 0b5e71c0 df843884 19866a4a 867ba1f4 b578...)
offset 0xE0 (224) ".C$......+&...~]a..."2.Z..6..." (d84324f2 b0f517c1 96b7e0f7 012b2601 8ab57e5d 61ecd68a 2232d65a 6b9036a0)
offset 0x100 (256) ".a...[Y..pl.....%IN.Z{...3..." (cf610004 cf5b5913 bc706c19 0d92ebcb 1aaaad25 494e125a 28adedd4 a233efe3)
offset 0x120 (288) "...vO.....Y7....|...}... 5.\.(+i" (c7d9764f 94d9ceb1 923afbe0 5937b2b0 aff07c1a 8f7d160d 2035ce5c cc282b69)
offset 0x140 (320) "...\\.....M...{y/%F...L..=.m..." (e5d85cd8 a0ae13e2 144d84d8 027b792f 25461ac6 1a4caefb 3ddc816d bdabbb85)
offset 0x160 (352) "...o.....9.....*.j...s...%.....A..." (cab2c06f a3e8d79a 399edda9 ccbf2abc 6a98f173 b5bd25a2 d2e6cf86 415ce5d3)
offset 0x180 (384) "SZ...x.S/+..b,<..A6....,....F...|p^5" (535ab9ee c778eb53 2f2b0362 2c3c1d41 3699b693 0b2c92ea 9a46f5b8 7c705e35)
offset 0x1A0 (416) ". F.....O&.....gVP.J...e'...n..." (e22046ce b98dd817 ba4f26b6 de7fb8fa 80675650 aa4ab6f4 806527ce f49c6eb4)
offset 0x1C0 (448) "...N..G...3 ...L...*.t.db.#...!..." (5fe34eb8 1147d115 d4332020 10ea0f4c ead6a62a b074a664 62df2311 c32100cf)
offset 0x1E0 (480) ".W....A..(DA....axY]....h9'....)" (a557019d b6e34190 28974441 1fa2bca0 6178595d 0ff416af 683927c2 acc9f229)
offset 0x200 (512) ">O..._.j...9^....K;+...@^....K<..." (3e4f0a02 5fa4f66a 04a6d239 5eafdda7 0c4b3b2b 8782ff40 5e9019a4 194b3cd6)
offset 0x220 (544) "DL|N....._uV(.Og.... q.%...v..." (444c7c4e 8b96f9fd 148f5f04 7556288e 4f67c5c8 9acb2071 ed25acc4 8f769cc4)
offset 0x240 (576) "M...r.x..F.v ...4_.P....0.z..K2" (4dflade6 72c978f5 d846fb76 20940cc6 ed345ff1 500da10a 8930ef7a 0bb14b32)
offset 0x260 (608) "e.....[:y.P..H.^*.dX.....M..Ye" (65f2f5e8 b45b3a79 a650e78e 488f5e2a c8645883 87ebef0e 929aae4d 0aef5965)
offset 0x280 (640) "j.....H)P8....e.....9N..." (6abfd4c8 fe06c648 295038e4 fae3b165 ecc390b7 f7e799a3 fd1ca8e5 c4394eb5)
offset 0x2A0 (672) "\\d...{..`..y" (5c6418cb 7bfff601 df792020 20202020 20202020 20202020 20202020 20202020)
offset 0x2C0 (704) " (20202020 20202020 20202020 20202020 20202020 20202020 20202020 20202020)
offset 0x2E0 (736) " (20202020 20202020 20202020 20202020 20202020 20202020 20202020 20202020)
offset 0x300 (768) " (20202020 20202020 20202020 20202020 20202020 20202020 20202020 20202020)
offset 0x320 (800) " (20202020 20202020 20202020 20202020 20202020 20202020 20202020 20202020)
offset 0x340 (832) " (20202020 20202020 20202020 20202020 20202020 20202020 20202020 20202020)
offset 0x360 (864) " (20202020 20202020 20202020 20202020 20202020 20202020 20202020 20202020)
offset 0x380 (896) " (20202020 20202020 20202020 20202020 20202020 20202020 20202020 20202020)
offset 0x3A0 (928) " (20202020 20202020 20202020 20202020 20202020 20202020 20202020 20202020)
offset 0x3C0 (960) " (20202020 20202020 20202020 20202020 20202020 20202020 20202020 20202020)
offset 0x3E0 (992) " (20202020 20202020 20202020 20202020 20202020 20202020 20202020 20202020)
```

Leading part of block is
random binary data
(incompressible)

compressibility = 33%
means 33% trailing
blanks

pattern=ascii

```
offset 0x0000 0 "aPJYuQcQ4yaw>0a}Lgc7;z[[9PF0j{\4" (61504a59 75516351 34796157 3e4f617d 4c476337 3b7a5b5b 3950464f 6a7b5c34)
offset 0x0020 32 "b|n=Vp%]iWLB^JIYRkXiH-$b%Qbjz(b!" (627c6e3d 5670255d 69574c42 5e4a4959 526b5869 482d2462 2571426a 7a286221)
offset 0x0040 64 "A+>/bf.%nu^GV(t_ZFT7 ALZ76Y)Ran " (412b3e2f 62662e25 6e755e47 5628745f 5a465437 20416c5a 37365929 52616e20)
offset 0x0060 96 "ngxq??<Clv?'?M&w !po=ouj3fI/cnR=" (6e677871 3f3f3c43 6c763f27 3f4d2677 2021706f 3d6f756a 3366402f 636a523d)
offset 0x0080 128 "DI<:l?*([:Znm %5i*oLN<'%!4"EQjVt" (44493c3a 6c3f2a28 5b3a5a6e 6d202535 6921706f 3d6f756a 3366402f 636a523d)
offset 0x00a0 160 "UE]pNzawt<![XfUaB$';~yQ/t8,0)*HE" (55455d70 4e7a6177 743c215b 58665561 4221706f 3d6f756a 3366402f 636a523d)
offset 0x00c0 192 ",/~IB|V02w ym`XZ`xRIn{S#mpe_RQ_) (2c2f7e49 427c564f 32772079 6d60585a 6071706f 3d6f756a 3366402f 636a523d)
offset 0x00e0 224 "U/>LPMdpI>FzJZY(3K=FY0}DD+eM:wU/" (552f3e4c 504d6470 6c3e467a 4a5a5928 3341706f 3d6f756a 3366402f 636a523d)
offset 0x0100 256 "NQjVxX{YQ50}NMQ`SYG7t|3k<u@H1hsR" (4e516a56 78587b59 5135305d 4e4d5160 5351706f 3d6f756a 3366402f 636a523d)
offset 0x0120 288 "AKphT_Y?pq+_xs#x08)-ExmlwSrw:%}H" (414b7068 545f593f 70712b5f 78732378 66f1706f 3d6f756a 3366402f 636a523d)
offset 0x0140 320 "8D,f',9/FE%HV7#$. `1^MB[7_m!nZ{" (38442c66 602c392f 46452548 5621706f 3d6f756a 3366402f 636a523d)
offset 0x0160 352 "0\7afoGRP/iX_;8?<{)E a:[L|?4_K{" (4f5c3761 666f4752 503f1706f 3d6f756a 3366402f 636a523d)
offset 0x0180 384 "zKqHz8%+W/oca,%J*(C%`Z+|o9Is7q0S" (7a4b7148 7a380148 012c254a 2a21706f 3d6f756a 3366402f 636a523d)
offset 0x01a0 416 "CVz/Z%cqy3|@!R.<Tsc7=/!SEyD:(2lq" (4356f1706f 3d6f756a 3366402f 636a523d)
offset 0x01c0 448 " OevIIX31`17wP)n]WOK[ 'T}YbU@S8H" (204f6576 49495833 3160313f 77507d6e 5d574f4b 5b27547d 59625540 5338683f)
offset 0x01e0 480 "7XVR9[sd"k^Xt%u/B>(wAwZv91[Qsk~S" (37585652 395b7344 226b5e58 7425752f 423e2877 41775a76 39315b51 736b7e53)
offset 0x0200 512 "q{m*=uU8WD6xXHvBz09hKgBHTCib+}Sf" (717b6d2a 3d755538 57443678 58487642 7a4f3968 4b674248 54436962 2b5d5366)
offset 0x0220 544 "E`.z}^!6I-LX)X9kDW}%VT(lxGC6`K" (45602e7a 7d5e2136 492d4c58 2958396b 44577d25 56542831 69784743 3660294b)
offset 0x0240 576 "JkN^pNd@/}e+,3,HNA9W*-k\mzD~4i*V" (4a6b4e5e 704e6440 2f7d652b 2c332c48 4e413957 2a2d6b5c 6d7a447e 34692a56)
offset 0x0260 608 "FFb+sEQW&JE`kLL))9S1JY^~fv3":))" (4646622b 73455157 264a4560 6b4c4c29 29395331 4a5c595e 2d665633 223a297d)
offset 0x0280 640 "{uIM9 >\9XUHp&Uo0JH~.iT9H){f'<io" (7b75494d 39203e5c 39585548 7026556f 4f4a487e 2e695439 487d7b66 273c696f)
offset 0x02a0 672 "lUbl)&\RcmQ~,%2"XB!t0ds#+=s%VGlM" (3155626c 29265c52 636d517e 2c253222 58422174 4f647323 3d2b7325 56476c6d)
offset 0x02c0 704 "v{=S\hEJ7}&'qJqK{1}>BF,Hw`/C{?P" (767b3d53 5c5c6845 4a377d26 27714a71 4b7b3129 3e42462c 4857602f 43283f50)
offset 0x02e0 736 "*(Bp.p">t=5XQBdu48@'\+. Z8$!lcv?" (2a284270 2e70223e 743d3558 51426455 34384027 5c2b2e5f 5a382421 3163763f)
offset 0x0300 768 "|n}SR;^~>+n5f}taf5U=$f=po|FBG/L%" (7c6e7d53 523b5e2d 3e2b6e35 667d7461 6635553d 24663d70 6f7c4642 472f4c25)
offset 0x0320 800 ""<8*"7/yNR |DW|Sgamv\k|3ilbV0-ZL" (223c382a 22372f79 4e52207c 44577c53 67616d76 5c6b5d33 69316256 4f2d5a6c)
offset 0x0340 832 "dBogZ&l 7L6f6s.F"VNC08..md1*740V" (64426f67 5a263120 374c3666 36732e46 22564e43 4f382e2e 6d44312a 3f344f56)
offset 0x0360 864 "w%?ODQ&imX}W:e}{c(9\}tN+`p+66Wi!" (77253f4f 44512669 6d587d57 3a657d63 28395c5d 744e2b60 702b3636 5769216c)
offset 0x0380 896 "Su, cHdCRXx>.0/QQK$mFgy6~b#..?B2" (53752c20 63486443 5258783e 2e4f2f51 514b246d 46677936 7e62232e 7d3f4232)
offset 0x03a0 928 "QnYH&:.L&xvoho58(I`h*U~;|rt`+W)" (516e5948 263a2e6c 2678766f 686f3538 28496068 2a557e60 3b7c7274 602b577d)
offset 0x03c0 960 "6J**<hVYXQ0{]wa<bV7JwP[-U{yd F<A" (364a2a2a 3c685659 58513028 5d77613c 6256374a 77505b2d 557b7964 20463c41)
offset 0x03e0 992 "I=FaNDTK7FP-P{qJMjYfkbnnVH-_J3qn" (493d4661 4e44544b 3746502d 507b714a 4d6a5966 6b626e6e 56482d5f 4a33716e)
```

Randomly selected
printable ASCII characters
(fixed degree of
compressibility)

pattern=gobbledegook

```
offset 0x0000 0 "agens fever APPLICATORY indignan" (6167656e 73206665 76657220 4150504c 49434154 4f525920 696e6469 676e616e)
offset 0x0020 32 "t bacoro anamnestic ADVERTISEMENT W" (74206261 636f726f 20616e61 6d6e6573 74696320 41445645 5254454e 43452057)
offset 0x0040 64 "EBSTER weighed DELEGATE ail radi" (45425354 45522077 65696768 65642044 454c4547 41544520 61696c20 72616469)
offset 0x0060 96 "ant ANGLOMANIAC publicity fixus " (616e7420 414e474c 4f4d414e 49414320 7075636e 60636074 70206660 70757320)
offset 0x0080 128 "alouer reality arma satiety aumo" (616c6f75 65722072 65616c69 747920 60636074 70206660 70757320 70757320)
offset 0x00a0 160 "snier isotropic Seeley actinism " (736e6965 72206973 6f74726f 706960 60636074 70206660 70757320 70757320)
offset 0x00c0 192 "hatched addicere extrinsically p" (68617463 68656420 61646469 636570 60636074 70206660 70757320 70757320)
offset 0x00e0 224 "arlance fell aquiline passed ant" (61726c61 6e636520 66656c6c 206170 60636074 70206660 70757320 70757320)
offset 0x0100 256 "iquarian rot AGGRANDIZER AFTERBI" (69717561 7269616e 20726f74 206170 60636074 70206660 70757320 70757320)
offset 0x0120 288 "RTH ANALYTICS yearly occultation" (52544820 414e414c 595440 206170 60636074 70206660 70757320 70757320)
offset 0x0140 320 "nathra Notwithstanding discrimi" (206e6174 68726120 60636074 70206660 70757320 70757320 70757320 70757320)
offset 0x0160 352 "nate CHAMBER mongst tacitly prom" (6e617465 20426f74 424552 206d6f6e 67737420 74616369 746c7920 70726f6d)
offset 0x0180 384 "inences ambulacral designs avail" (696e656e 60636074 616d6275 6c616372 616c2064 65736967 6e732061 7661696c)
offset 0x01a0 416 "aberrating Argillaceous making " (20636074 72726174 696e6720 41726769 6c6c6163 656f7573 206d616b 696e6720)
offset 0x01c0 448 "prose apostele tro ALLODIARY bo" (70726f73 65206170 6f737465 6c652074 726f2041 4c4c4f44 49415259 20626f77)
offset 0x01e0 480 "lines malonyl exists Per bases a" (6c696e65 73206d61 6c6f6e79 6c206578 69737473 20506572 20626173 65732061)
offset 0x0200 512 "cuminat ciere legumes necessary" (63756d69 6e617465 20636965 7265206c 6567756d 6573206e 65636573 73617279)
offset 0x0220 544 "onward Bombax APPLY exploit sym" (206f6e77 61726420 426f6d62 61782041 50504c59 20657870 6c6f6974 2073796d)
offset 0x0240 576 "pathy ANGELICALNESS ALGARROBA gai" (70617468 7920414e 47454c49 43414c4e 45535320 414c4741 524f4241 20676169)
offset 0x0260 608 "ning Alpes ACTIVITY buoyancy wit" (6e696e67 20416c70 65732041 43544956 49545920 62756f79 616e6379 20776974)
offset 0x0280 640 "her filaments Blackfeet opponent" (68657220 66696c61 6d656e74 7320426c 61636b66 65657420 6f70706f 6e656e74)
offset 0x02a0 672 "s footing cannon anai APTLY arge" (7320666f 6f74696e 67206361 6e6e6f6e 20616e61 69204150 544c5920 61726765)
offset 0x02c0 704 "ntic ALLUSION appropinquatus apo" (6e746963 20414c4c 5553494f 4e206170 70726f70 696e7175 61747573 2061706f)
offset 0x02e0 736 "stel ARCHIEPISCOPATE AGGLOMERATE" (7374656c 20415243 48494550 4953434f 50415445 20414747 4c4f4d45 52415445)
offset 0x0300 768 "D nightingale aphol ACCUSTOMABLE" (44206e69 67687469 6e67616c 65206170 686f6c20 41434355 53544f4d 41424c45)
offset 0x0320 800 " law centripetal AMIABLE rin AMN" (206c6177 2063656e 74726970 6574616c 20414d49 41424c45 2072696e 20414d4e)
offset 0x0340 832 "ESIA yardarm deserving jure argu" (45534941 20796172 6461726d 20646573 65727669 6e67206a 75726520 61726775)
offset 0x0360 864 "mentatio Num flushed Abas APRICA" (6d656e74 6174696f 204e756d 20666c75 73686564 20416261 73204150 52494341)
offset 0x0380 896 "TION AFFLUENTLY old affected Hud" (54494f4e 20414646 4c55454e 544c5920 6f6c6420 61666665 63746564 20487564)
offset 0x03a0 928 "ibras ACROTISM ARAEOSYSTYLE Gall" (69627261 73204143 524f5449 534d2041 5241454f 53595354 594c4520 47616c6c)
offset 0x03c0 960 "icism PEAR ACHIEVEMENT reclining" (69636973 6d205045 41522041 43484945 56454d45 4e542072 65636c69 6e696e67)
offset 0x03e0 992 " mechanism Amalgamating cooperat" (206d6563 68616e69 736d2041 6d616c67 616d6174 696e6720 636f6f70 65726174)
```

Randomly selected words from
Webster's 1913 dictionary.
(fixed degree of compressibility)

Random workload "hot zone" – induce specified hit rate

- The `random_steady` and `random_independent` iosequencer types support optional "hot zone" parameter settings
- The random I/O "hot zone" receives a specified fraction of all I/Os.
 - The hot zone fraction of I/Os is a number from 0.0 to 1.0 or from 0% to 100%.
 - Use `hot_zone_IOPS_fraction` to control both reads & writes
 - Use `hot_zone_read_fraction` or `hot_zone_write_fraction` to control them separately
- Usually the default size `hot_zone_size_bytes = "1 MiB"` is fine.
 - KiB/MiB/GiB/TiB suffixes OK.
- This is an open-loop method of achieving a target subsystem cache hit ratio with what is otherwise a perfectly even random distribution.

"hot zone" notes

- The value the user specifies for `hot_zone_size_bytes` is rounded up to the next higher multiple of the `blocksize` parameter value.
- The "hot zone" starts at the beginning of the LUN coverage area specified by `RangeStart` (default 0% of the way through the LUN).
- The "hot zone" is designed to service hot zone hits as well as non-hot zone misses with the same set of tags, that is, within the same ivy workload.
- There is no separate reporting of the hot zone as the csv files are by workload.
- Hot zone I/Os go to random locations within the hot zone.

Statements - [DeleteWorkload]

- [DeleteWorkload] "r_steady" [select] "LDEV : 00:04";
- [DeleteWorkload] "r_steady" ;
 - Deletes all instances of the r_steady workload on all test hosts / all LUNS.
- [DeleteWorkload] ;
 - Deletes all workloads. (All workloads are automatically deleted when ivy ends.)

Workload parameter – “skew” or “skew_weight”

- The “skew” or “skew_weight” (use either name) parameter must be set to a non-zero number, defaulting to -1.0,
- Skew is used for two distinct purposes:
 1. With “Edit Rollup”, skew governs the distribution of a fixed `total_IOPS` value across LUNs and the workloads on those LUNs.
 - Works identically whether skew is positive or negative.
 - `Total_IOPS` value is first evenly distributed over LUNs, and then within the LUN proportional to each workload’s skew value.
 2. Where there are multiple IOPS=max workloads on a LUN, a positive skew value for a workload causes the ratio of the IOPS of each positive-skew with IOPS=max workload to be proportional to that workload’s positive skew value.
 - If an IOPS=max workload has a negative skew, it will run independently without reference to any other IOPS=max workloads.

Skew example with Edit Rollup & total_IOPS

```
[hosts] "sun159" [select] << { "serial_number" : 83011441 } >>;
```

```
[CreateWorkload] "r_steady"  
  [select] ""  
  [iogenerator] "random_steady"  
  [parameters] << blocksize=4KiB, maxtags=1, fractionread=0.5,  
                  RangeStart=0.0, RangeEnd=0.5 >>;
```

"skew" not specified,
defaults to -1.0

```
[CreateWorkload] "r_independent"  
  [select] ""  
  [iogenerator] "random_independent"  
  [parameters] << blocksize=4KiB, maxtags=1, fractionread=0.5,  
                  RangeStart=0.5, RangeEnd=1.0,  
                  skew = 2 >>;
```

"skew" specified
as 2.0

```
[edit rollout] "all=all" [parameters] "total_IOPS=1000";
```

```
[Go] "stepname=step_eh, measure_seconds = 30";
```

total_IOPS is first divided by the number of LUNs on all hosts behind the "all=all" rollout,
then the IOPS per LUN is divided up in the ratio to 1 part r_steady to 2 parts r_independent.

Skew example with IOPS=max

```
[hosts] "sun159" [select] << { "serial_number" : 83011441 } >>;
```

```
[CreateWorkload] "r_steady"  
  [select] ""  
  [iogenerator] "random_steady"  
  [parameters] << IOPS=max, skew = 1, blocksize=4KiB, maxtags=8, fractionread=0.5,  
                  RangeStart=0.0, RangeEnd=0.5 >>;
```

"skew" = 1

```
[CreateWorkload] "r_independent"  
  [select] ""  
  [iogenerator] "random_independent"  
  [parameters] << IOPS=max, skew = 2, blocksize=4KiB, maxtags=1, fractionread=0.5,  
                  RangeStart=0.5, RangeEnd=1.0 >>;
```

"skew" = 2


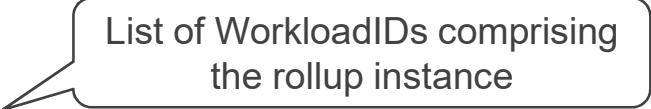

```
[Go] "stepname=step_eh, measure_seconds = 30";
```

One third of the IOPS will come from the workload with skew=1,
and two thirds will come from the workload with skew=2

What is an ivy "rollup"?

- A "rollup" is a grouping of workloads.
- Each workload is attached to a LUN, and this LUN has attributes like "Port", "LDEV", "LDEV type", "Pool ID", "Parity Group", "Consistency Group", "Serial Number", "Host", "Size GiB", "Vendor", "Product", "Port WWN", "CLPR", etc. These come from "showluns.sh".
- A "rollup" is a way of grouping the workloads by LUN attribute.
- No matter how you created the workloads, whether you selected by one of these attributes, or if you just created workloads on all **available test LUNs**, to get output data sliced and diced by port, say `[CreateRollup] "port";`
- The "port" rollup has instances like "1A", "2A", etc.
- Every workload (like `sun159+/dev/sdc+frantic`) belongs to exactly one instance in every rollup.
- You can make multiple rollups on different attributes to get data sliced & diced in different ways.

Rollups are key to how the ivy engine works

- Rollups are used to group workloads, to navigate between (in both directions), say, a port name 1A and those workloads on LUNs mapping to port 1A.
 - By-rollup csv files show data rolled up by rollup instance from results by individual `WorkloadID`.
 - `[EditRollup]` uses rollups in the other direction, to send, for example, `IOPS=1000` to the workloads on port 1A.
- When driving multiple subsystems: `[CreateRollup] "serial_number+port";`
 - `serial_number` and `port` must be valid LUN attribute names.
- In every rollup, each `WorkloadID` appears in exactly one rollup instance.
 - `"Serial_Number+Port"` 
 - `"410123+1A"` 
 - `"sun159+/dev/sdx+workload_name", "cb28+/dev/sdy+workload_name"`

To print this rollup structure out, say
`print(ivy_engine_get("rollup_structure"));`

You make rollups for four reasons

1. To get an output csv file with a csv folder by rollup type (e.g. `port`) and csv files by rollup instance (e.g. `1A`)
 - This is how you get custom "sliced & diced" data.
2. To perform IOPS dynamic feedback control (`dfc=PID`) at the granularity of the rollup instance.
 - One of the demos shows measuring IOPS at MP 50% busy at the granularity of the MPU, meaning to vary the IOPS up and down separately/independently for each MPU to achieve 50% busy MP cores in that MPU.
3. To identify a valid measurement period at the granularity of the rollup instance using `measure`.
 - For the valid period, when measuring at the granularity of the rollup instance, the data for each rollup instance individually met the +/- accuracy % criteria for a valid measurement. (For every `port`, the individual data for that port met the +/- accuracy criterion.)
4. To validate the test configuration as operating correctly
 - E.g. Validates that the number of ports reporting was what you expected
 - E.g. Validate that no one port had an IOPS too far below the highest IOPS seen on any port.

Statements – [CreateRollup]

- `[CreateRollup] "port" [nocsv] [quantity] 64 [MaxDroop] "20%";`
- `[nocsv]` – Optional - suppresses creation of `port` output csv files for this rollup.
- `[quantity] 64` – Optional - marks the test result invalid if there aren't 64 `port` instances reporting data .
- `[MaxDroop] "20%"`
 - Optional - marks the test result invalid if any one instance of the rollup has an IOPS more than 20% below that of the fastest instance.
 - Useful to catch the situation where, say, one port is running slowly compared to the others because it's in error recovery.

The "all" rollup

- There is always a special "all" rollup which only has one instance "all".
- The "all=all" instance contains all workloads.
- The "all" rollup is automatically created, and you cannot delete it.
- For example `[EditRollup] "all=all" [parameters] "IOPS=max";` will set IOPS=max in every workload.
- Every rollup has its summary csv file folder in the `[OutputFolderRoot]` directory.
 - Look at the `"xxx.all.summary.csv"` file in the "all" subfolder of the ivy output folder to get the overall summary data.

[CreateRollup] combines LUN attribute names

- If you would like to see a rollup instance for each unique LDEV across two or more subsystems, make "serial number+LDEV".
- [nocsv] – prevents csv files from being created.
- The [quantity] <int expression> clause can enforce that the right number of distinct rollup instances are reporting in this rollup.
 - If not, even if the DFC reports "success" and designates a subinterval subsequence representing a successful measurement, no measurement rollup csv data will be produced – instead error msg.
 - Make a rollup by "port" and use the [quantity] rollup to validate you have the number of paths you think you have.
- [MaxDroop] <double expression>
 - "25%" means invalidate test if any one rollup instance has an average IOPS more then 25% below the highest average IOPS over all rollup instances.

[DeleteRollup]

- [DeleteRollup] "serial number+Port";
- [DeleteRollup] ;
 - Deletes all rollups except the "all" rollup.

[EditRollup]

- [EditRollup]
 "serial_number+Port = { 410123+1A, 410123+2A }"
[parameters] "fractionRead = 100%";
- The [EditRollup] statement operates in between test steps while the workload threads are in "waiting for command" state.
- It gives you access to the same mechanism used by Dynamic Feedback Control to send out real time parameter updates to running workload threads.
- You specify a set of rollup instances to send to, such as "all=all", or "Port={1A, 3A, 5A, 7A}" and you specify the text [parameters] string to send to the remote iosequencer to parse and apply.

[EditRollup]

- [EditRollup] is typically used at the top of a do-loop, to change whatever parameters vary by loop pass.
- Use [EditRollup] "all=all" to send to all workload threads.
- There is a special parameter name that is only recognized by [EditRollup] - `total_IOPS` - where the numeric value you specify is first evenly divided amongst all test LUNs, and then within each LUN, proportional to the absolute value of each workload's skew parameter before being sent out as `IOPS=xxx` to each workload.
 - [EditRollup] "all=all" [parameters] "total_IOPS = 1000000";

[EditRollup] attributes names containing spaces

- Many attributes names contain spaces, like "serial number".
- Inside the ivy engine, when making rollups, these attribute names are “normalized”
 - All alphabetic characters A–Z are converted to lower case a–z.
 - All sequences of one or more non-alphanumeric characters (not A–Z, not a–z, not 0–9) are replaced with a single underscore _.
- When creating, deleting, or editing rollups with attribute names containing spaces, you can either put quotes around the attribute combo name, like

```
[EditRollup] << "Serial Number+Port" = { 410123+1A, 410123+2A } >> ...
```

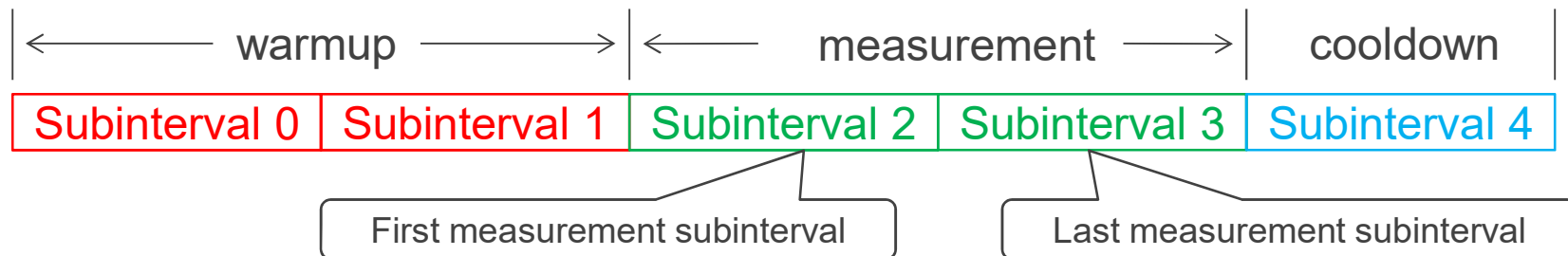
or else to avoid using quotes, you can spell the attribute names using underscores instead of spaces and special characters. Once the attribute names have been normalized, these are equivalent.

```
[EditRollup] << Serial_Number+Port = { 410123+1A, 410123+2A } >> ...
```


Statement – [Go]

- [Go] "stepname = random4K, **subinterval_seconds** = 5, ..."
- The [Go] statement starts the workload threads running a "test step", which is a sequence of "subintervals" each of a duration specified in the `subinterval_seconds` parameter, defaulting to 5 seconds.
 - If you have a case for using ivy to measure a restricted set of things much more frequently, we can talk about putting in support.
 - Most of the time 5 seconds is plenty short and if you are going to be doing any tests that will run for hours you may want to consider a longer subinterval just to mercifully cut down on the size of the csv files by subinterval.
 - Sometimes when you say you want an answer to +/- 1% and the behaviour is a bit noisy, it can take time to see enough to say you are sufficiently confident statistically. (Did you say you wanted "valid" data?)

Test step = warmup, measure, cooldown



- There must be at least one measurement subinterval. Warmup & cooldown are optional.
- Parameter defaults
 - `warmup_seconds` – defaults to the value of `subinterval_seconds`, which itself defaults to 5 seconds. `warmup_seconds` is divided by `subinterval_seconds`, and rounded up to get the (minimum) number of warmup subintervals.
 - `measure_seconds` defaults to 60 - also rounded up to the minimum number of measurement subintervals.
 - `cooldown_seconds` defaults to 0.
 - `cooldown_by_wp = on` - If a command device is available for the subsystem under test, the cooldown period is extended until write pending is below `subsystem_WP_threshold` which defaults to 1.5%.
 - `cooldown_by_MP_busy = on` - If a command device is available for the subsystem under test, the cooldown period is extended until average subsystem MP_core % busy is below `subsystem_busy_threshold` which defaults to 5%.

MM:SS or HH:MM:SS are OK

- For `warmup_seconds`, `measure_seconds`, `cooldown_seconds`, and also for a parameter we haven't discussed yet, for `timeout_seconds`, it's OK to specify a value like `10:00` meaning 10 minutes, or `10:00:00` meaning 10 hours.

Sequential fill

- To ensure that a test step runs at least until all sequential write threads have wrapped around and have written to all possible blocks:

```
[Go!] " ... sequential_fill = on ... ";
```

- At the point where a successful measurement (of either fixed or variable duration) would have been declared, with `sequential_fill=on` the measurement period is automatically extended from then until sequential fill is complete.
- Sequential fill progress is 40.46% after 0:02:51. Estimated remaining 0:04:11 to complete fill at 2017-04-22 11:24:19.
- Even when `sequential_fill` is not on, when writing sequentially you'll get this same sequential fill progress message on the console.

For each test step (for each [go]) you get:

- A subfolder of the overall test output folder that contains the csv files with one line for each subinterval in that test step. (suppress this with `-no_stepcsv.`)
 - Nested subfolders for each workload data rollup
 - Containing a csv file for each rollup instance, with one line per subinterval.
 - For internal Hitachi lab use with command device connector and license key, a nested subfolder with raw RAID_subsystem RMLIB API data. (Suppress with `-no_storcsv.`)
 - Collected time-synchronized "just before" the end of each subinterval.
- A single line in the overall test results "summary.csv" files.
 - In ivy terminology, this is called a "measurement" line, which represents the rollup from the first to last measurement subintervals.
 - Unless "measure=on" with specified accuracy timed out – then you get an error message line
 - Well, actually, with `DFC=IOPS_staircase`, you get one line per staircase step.

Provisional by-subinterval csv files

- Unless suppressed with `-no_stepcsv`, a subfolder is made for each test step with by-subinterval csv files for each rollup instance.
- Normally csv files are written at the end of a test step when the subinterval sequence is finished. (Data are kept in memory until then.)
- The by-subinterval csv files lines have a `warmup / measure / cooldown` column. This column can't be determined until after the subinterval sequence is done, and the "measure" feature has decided.
- To get provisional by-subinterval csv files with a blank `warmup / measure / cooldown` column written while subintervals are running, say
`ivy_engine_set("provisional_csv_lines", "on");` in your ivyscript program before `[go]`.
- This way you can kill ivy with ctrl-C while it's running, and still see the by-subinterval data up until then. Of course you won't get a summary csv line for that step that was running.

Looping within a [Go] statement

- `[go] << blocksize = ("2 KiB", "4 KiB"),
maxTags = (1, 2, 4, 8), measure_seconds = 1:00 >>`
- The syntax is `<workload parameter name> = (value1, value2, ...)`
- The use of parenthesis () is mandatory, and this is what tells the ivy engine that the parameter name is a workload [parameters] name, not a [go] parameter name.
- The first `<workload parameter name> = (value1, value2, ...)` becomes the outer loop, and so forth. Thus you can control the order of the nesting of the loops.
- A test step is run for each loop pass, that is, for each combination of parameter values.

Looping within a [Go] statement – IOPS_curve

- `IOPS_curve = (10%, 20%, 0.8, 0.9)`
- This is a special kind of looping within a [Go] statement which prepares for the first loop pass by doing:
`[EditRollup] "all=all" [Parameters] "IOPS=max";`
- The achieved IOPS measured on the `IOPS=max` loop pass is then multiplied by the specified fractions to prepare for subsequent passes using:
`[EditRollup] "all=all" [Parameters] "Total_IOPS=xxx";`
- When combined with looping over other workload parameters, normally the `IOPS_curve` loop is specified last so it becomes the innermost loop.
 - This is because changing other parameters within an `IOPS_curve` loop may affect the measured max IOPS.

`cooldown_seconds`

-
- Cooldown runs at IOPS=0.
 - Use `cooldown_seconds` to run at IOPS=0 for a fixed time period to allow for the test configuration to recover from the ivy test step that's ending, such as to give time for dirty data to destage from cache.
 - Here there's no feedback from the storage to know when the recovery period is complete; cooldown just runs for a fixed period.

`cooldown_by_wp` & `cooldown_by_MP_busy`

- These are available for Hitachi internal lab use with a command device connector and license key. The command device connector is not part of the ivy open source project.
- When `cooldown_by_wp` and/or `cooldown_by_MP_busy` are being used, additional cooldown subintervals run at `IOPS=0` until Write Pending is empty, and/or subsystem MP % busy has dropped below the threshold, respectively.
- Defaults: `cooldown_by_wp = on` and `subsystem_WP_threshold = 1.5%`
 - Set `cooldown_by_wp = off`
 - When it is valid to carry forward dirty data in cache (Write Pending) from one test step to the next.
 - This can speed up the next test step if the next step doesn't stabilize until WP is full, AND if both steps place the SAME things into WP that destage to the SAME place.
- Defaults: `cooldown_by_MP_busy = on` and `subsystem_busy_threshold = 5%`
 - This extends cooldown until any residual subsystem microprocessor activity has ended.
 - `cooldown_by_MP_busy_stay_down_time_seconds` (default one subinterval) specifies how long MP busy must remain below the threshold. Can be specified as "1:00" meaning a minute.

`no_perf = on`

- With a command device (Hitachi lab use only), this suppresses collecting performance data from the subsystem *only while* ivy is driving I/O.
- This can be useful if every subsystem port is being driven intensively, and thus would interfere with communicating with the command device, which is on one of the ports.
- When used with `cooldown_by_wp` or `cooldown_by_MP_busy`, after the first cooldown subinterval while I/O is still being driven, during subsequent subintervals at IOPS=0, even with `suppress_perf=on`, collection of performance data from the subsystem resumes to provide the necessary Write Pending % and subsystem MP % busy information to support the `cooldown_by_wp` and `cooldown_by_MP_busy` features.
- The `-no_perf ivy` command line option sets the default [Go] statement `no_perf` parameter to “on” instead of “off”.

`skip_LDEV = on`

- The collection of `LDEV` data (which is also the source for `PG` data) is the most resource intensive part of collecting real time subsystem performance data.
- For Hitachi internal lab use with a command device, `skip_LDEV=on` is a `[Go]` statement parameter that will suppress collection of `LDEV` and `PG` performance data.
- This can reduce the amount of communication with the subsystem required, if the command device LUN is on the same port as very-heavily driven data LUNs.
- This still lets you do dynamic feedback control on other metrics such as subsystem MP % busy.
- The `-skip_LDEV` ivy command line option sets the `[Go]` statement default `skip_LDEV` parameter value to “on” instead of “off”.

`check_failed_component = on / off`

- For Hitachi internal lab use with a command device, this `[Go]` statement parameter controls whether ivy checks the subsystem for a failed component.
- With `check_failed_component = on`, as each `[Go]` statement is executed, a check for a failed subsystem component is made immediately before I/O starts to be driven, and also immediately after the subinterval sequence is done and before csv files are printed. Ivy aborts when one of these checks detects a failed subsystem component.
- The default is `check_failed_component = on`, except if the command line parameter `-no_check_failed_component` is used, which sets the default for `check_failed_component` to `off`.

The default `[Go]` statement

- `[go] ;` Useful when you are developing an ivyscript workflow and you just want to see quick sample csv files.
 - `Default subinterval_seconds = 5`
 - `warmup_seconds` defaults to the value of `subinterval_seconds`.
 - `Default measure_seconds = 60`
 - Runs at least one cooldown subinterval.
 - Defaults `cooldown_by_wp = "on"` and `subsystem_WP_threshold=1.5%`
 - If you have a command device and the proprietary command device connector software, continuing more cooldown subintervals until WP is below threshold.
 - Defaults `cooldown_by_MP_busy="on"` and `subsystem_busy_threshold=5%`
 - If you have a command device and the proprietary command device connector software, continuing more cooldown subintervals until MP_core % busy is below the threshold.

stepname

- On the [Go] statement to start a test step, you can optionally specify "stepname=", which defaults to "step" followed by a four digit step number starting with 0000, so the default name for the first step is `step0000`.
- Giving a test step a meaningful name is useful when looking at overall measurement summary csv files, where you get one csv line for each test step.
- Those labels are handy when making Excel charts, as you can use the stepname column as the series name on a chart.

measure shorthand

- `measure = MB_per_second` is short for
 - `measure=on, focus_rollup=all, source=workload, category=overall, accumulator_type=bytes_transferred, accessor=sum`
- `measure = IOPS` is short for
 - `measure=on, focus_rollup=all, source=workload, category=overall, accumulator_type=bytes_transferred, accessor=count`
- `measure = service_time_seconds` is short for
 - `measure=on, focus_rollup=all, source=workload, category=overall, accumulator_type=service_time, accessor=avg`
- `measure = response_time_seconds` is short for
 - `measure=on, focus_rollup=all, source=workload, category=overall, accumulator_type=response_time, accessor=avg`

- Advanced things for the ivy power user.
- These give access to internal ivy mechanisms.
- Explained in a separate section.

measure shorthand – with command device

- `measure = MP_core_busy_percent` is short for
 - `measure=on, focus_rollup=all, source=RAID_subsystem, subsystem_element=MP_core, element_metric=busy_percent`
- `measure = PG_busy_percent` is short for
 - `measure=on, focus_rollup=all, source=RAID_subsystem, subsystem_element=PG, element_metric=busy_percent`
- `measure = CLPR_WP_percent` is short for
 - `measure=on, focus_rollup=all, source=RAID_subsystem, subsystem_element=CLPR, element_metric=WP_percent`

- Advanced things for the ivy power user.
- These give access to internal ivy mechanisms.
- Explained in a separate section.

[Go] "focus metric"

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

- When `measure` and/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`, 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`.

Subsystem data filtered by rollup instance

- 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.

Subsystem data by rollup instance – csv columns

We know how many drives underlie each workload rollup

Shows you if the OS / device driver are breaking up your large block application-level I/O into smaller pieces

Matching subsystem vs. application data validates that both host-workload rollups and subsystem data rollups are working correctly

Shows you if there is delay between when the application issues the I/O and when the device driver issues the I/O.

Shows you the amount of that delay in ms.

subsystem avg LDEV sequential_blocks size_KiB	subsystem avg LDEV read_service_time_ms	subsystem avg LDEV write_service_time_ms	host IOPS per drive	host decimal MB/s per drive	Subsystem IOPS as % of application IOPS	Subsystem MB/s as % of application MB/s	Subsystem service time as % of application service time	Path latency = application minus subsystem service time (ms)	Overall IOPS	Overall Decimal MB/s	Overall Average Blocksize (KiB)	Overall Little's Law Avg Q
0	6.82666	0	1.90	-	98.22%	98.22%	98.69%	0.091	30	0.12288	4	0.207528
4	0	60.5501	98.30	0.40	99.99%	99.99%	99.08%	0.563	1572.9	6.4426	4	96.1246
4	50.0142	49.0471	120.70	0.50	100.43%	100.43%	99.27%	0.364	1931.98	7.91339	4	95.9223
4	41.4719	32.018	162.00	0.70	99.99%	99.99%	99.24%	0.283	2592.74	10.6199	4	95.9846
4	39.3228	3.90259	194.90	0.80	99.68%	99.68%	99.02%	0.303	3118.22	12.7722	4	96.0234
0	20.9364	0	283.10	1.20	100.51%	100.51%	98.79%	0.257	4529.76	18.5539	4	95.9988

RMLIB API candidates flagged to display

maxTags	IOPS input parameter setting	fraction read	test host cores	test host avg core CPU % busy	subsystem MP_core count	subsystem m avg MP_core busy_per cent	subsystem m CLPR count	subsystem m avg CLPR WP_per cent	subsystem m PG count	subsystem m avg PG busy_per cent	subsystem m LDEV count	subsystem m avg LDEV random_ read_IOP S	subsystem m avg LDEV random_ write_IO PS	subsystem m avg LDEV random_ blocksize _KiB
16	5	1	16	0.10%	12	2.52%	1	0%	4	1.12%	6	4.91	-	4.00
16	max	0	16	0.00%	12	5.39%	1	68.78%	4	99.83%	6	-	261.42	4.00
16	max	0.25	16	0.00%	12	5.63%	1	68.84%	4	99.82%	6	80.11	242.82	4.00
16	max	0.5	16	0.00%	12	6.06%	1	68.27%	4	99.85%	6	215.46	216.43	4.00
16	max	0.75	16	0.10%	12	6.23%	1	62.59%	4	99.80%	6	388.87	128.99	4.00
16	max	1	16	0.10%	12	4.34%	1	0.20%	4	95.16%	6	758.77	-	4.00

- The “subsystem” columns are automatically generated according to the focus metric RMLIB API candidate table.
- As raw data comes in for each MP_core, CLPR, PG, LDEV, etc., ivy filters the data to aggregate for each rollup only the data for the MP_cores, etc. that map to LDEVs/LUNs underlying workloads in that rollup.
- Make a rollup by MPU, and each MPU rollup instance will show data for 4 MP_cores.

Examples of data for each rollup – drive / PG type

Rollup Type	Rollup Instance	drive type	drive quantity	RAID level	PG layout	iogenerator type	blocksize	maxTags	IOPS input parameter setting	fraction Read
all	all	DKR2E-H4R0SS+DKR5D-J600SS	8+8=16	RAID-5	3+1	random_steady	4 KiB	16	5	1
all	all	DKR2E-H4R0SS+DKR5D-J600SS	8+8=16	RAID-5	3+1	random_steady	4 KiB	16	max	0
all	all	DKR2E-H4R0SS+DKR5D-J600SS	8+8=16	RAID-5	3+1	random_steady	4 KiB	16	max	0.25
all	all	DKR2E-H4R0SS+DKR5D-J600SS	8+8=16	RAID-5	3+1	random_steady	4 KiB	16	max	0.5
all	all	DKR2E-H4R0SS+DKR5D-J600SS	8+8=16	RAID-5	3+1	random_steady	4 KiB	16	max	0.75
all	all	DKR2E-H4R0SS+DKR5D-J600SS	8+8=16	RAID-5	3+1	random_steady	4 KiB	16	max	1

- Information comes from RMLIB API configuration data, filtered / aggregated for each rollup instance.
- (There are also dedicated csv folders that contain detailed RMLIB API subsystem configuration and performance data csv files.)

Now that we know how to specify the focus metric

- We will look at
 - The `[go]` statement `measure=on` option and its subparameters
 - Specifying `measure=on` on a Go statement means "watch the focus metric and when you have seen enough to make a measurement of the specified accuracy, stop. Timeout if it takes too long."
 - The `[go]` statement `dfc=pid` option and its subparameters
 - If you don't specify a `dfc`, the workload settings remain constant through the test.
 - If you specify a `dfc` (dynamic feedback controller), it gets called at the end of every subinterval once all the rollups are done.
 - The DFC looks at what has happened so far, looking at all workload data and all subsystem data, and then may use the ivy engine real time edit rollup mechanism to send out parameter updates to rollup instances (to the workload threads belonging to the rollup instance).

measure = on

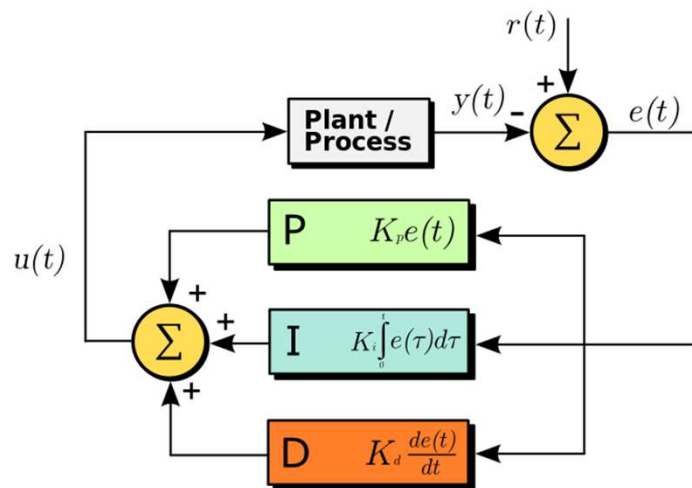
-
- `accuracy_plus_minus = 5%`
 - Any numeric value with an optional trailing % sign maybe specified.
 - `confidence = 95%`
 - How confident you need to be that your measurement falls within the specified plus or minus range around the long term average that you would get measuring forever.
 - Default is 95%
 - Ivy has a menu of 11 specific pre-loaded confidence values that you pick from.
 - 50%, 60%, 70%, 80%, 90%, 95%, 98%, 99%, 99.5%, 99.8%, and 99.9%
 - http://en.wikipedia.org/wiki/Student%27s_t-distribution

measure **Write Pending-based stability criteria**

- `max_wp = 2%` - default `100%`
 - A subinterval sequence will be rejected if WP is above the limit at any point in the sequence.
 - Set this to "1%" or so for read tests to ensure WP is empty during the test.
- `min_wp = 67%` - default `0%`
 - A subinterval sequence will be rejected if WP is below the limit at any point in the sequence.
 - Use this for write tests to ensure WP is full during the test.
- `max_wp_change = 3%` - default `3%`
 - A subinterval sequence will be rejected if WP varies up and down by more than the specified (absolute) amount at any point in the sequence. `max_wp_range="3%"` matches from 0% to 3% Write Pending, as well as from 67% to 70% Write Pending. (not a percent OF the WP value)
 - Use this in general all the time so you reject periods with major movement in Write Pending.

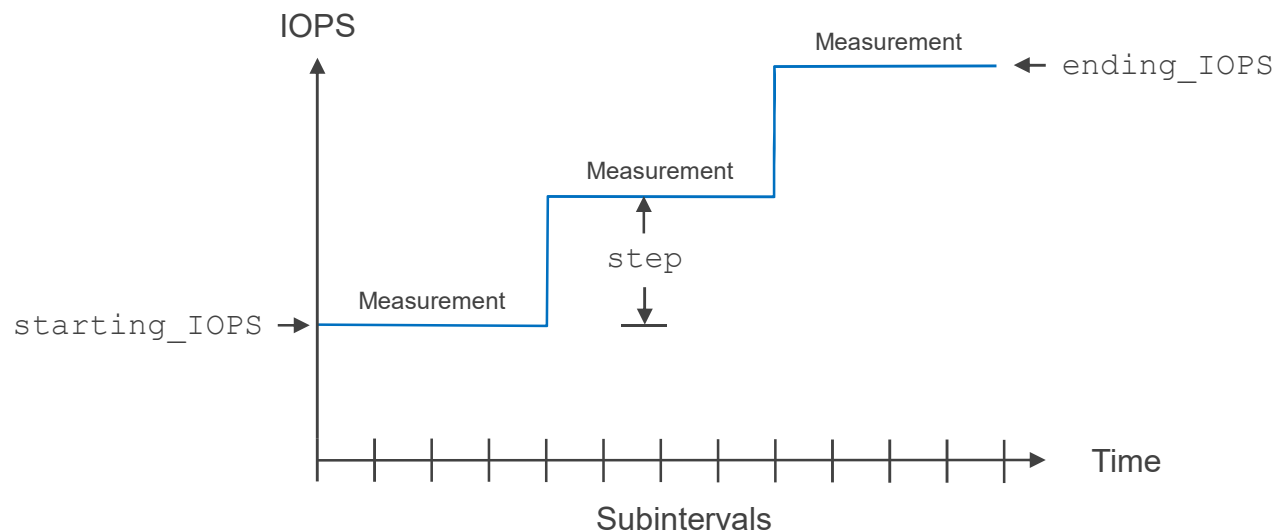
dfc=pid dynamically adjusts total IOPS

- See separate presentation "ivy adaptive PID".



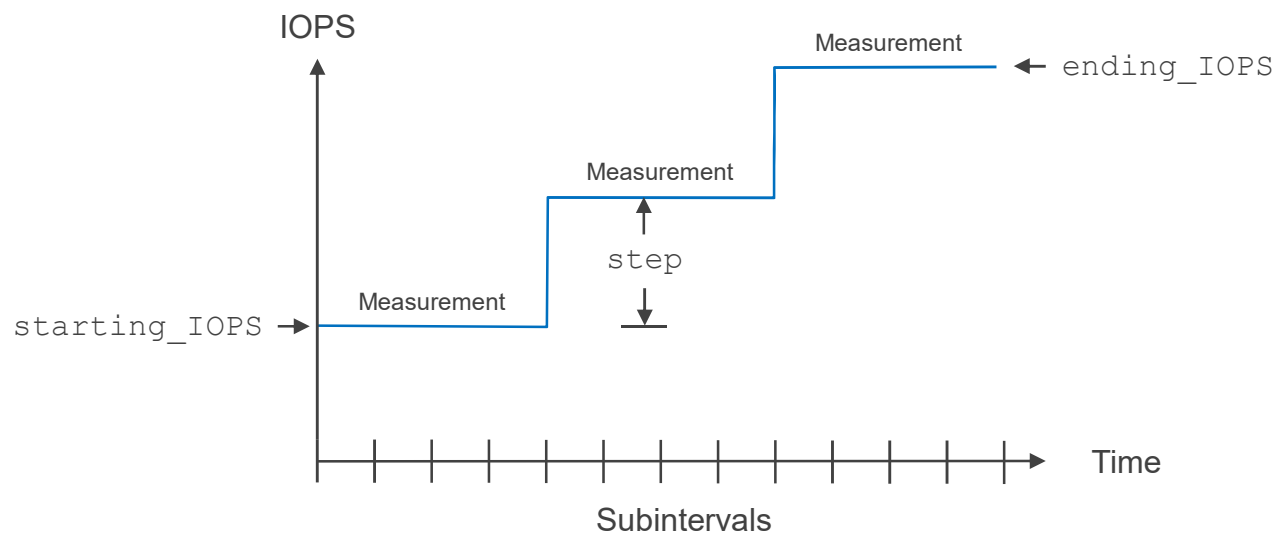
DFC = IOPS_staircase

- Normally, each ivy “step” runs a subinterval sequence to make one measurement, and then pauses for a very brief time to write csv files before the next step.
- `DFC = IOPS_staircase` is used to make a series of “IOPS staircase step” measurements in one unbroken series of subintervals, continuously driving I/O.



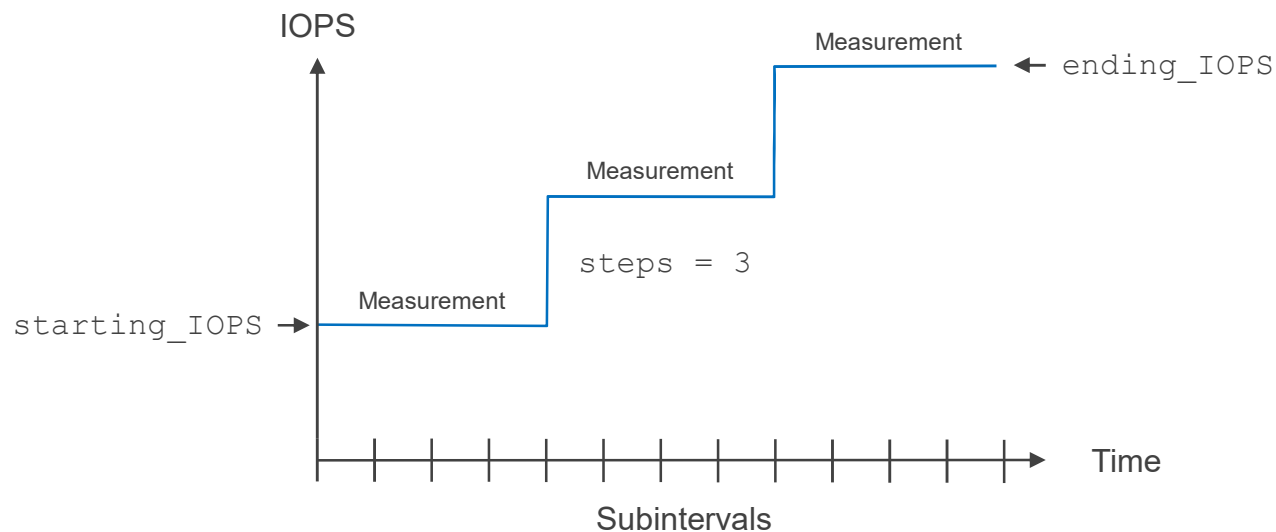
DFC = IOPS_staircase

- Each measurement is comprised of zero or more warmup subintervals (`warmup_seconds`) and one or more measurement subintervals (`measure_seconds`).
- The `measure` feature can be used with `DFC=IOPS_staircase`, in which case the duration of each step on the staircase is variable, and `warmup_seconds` & `measure_seconds` become minimums.



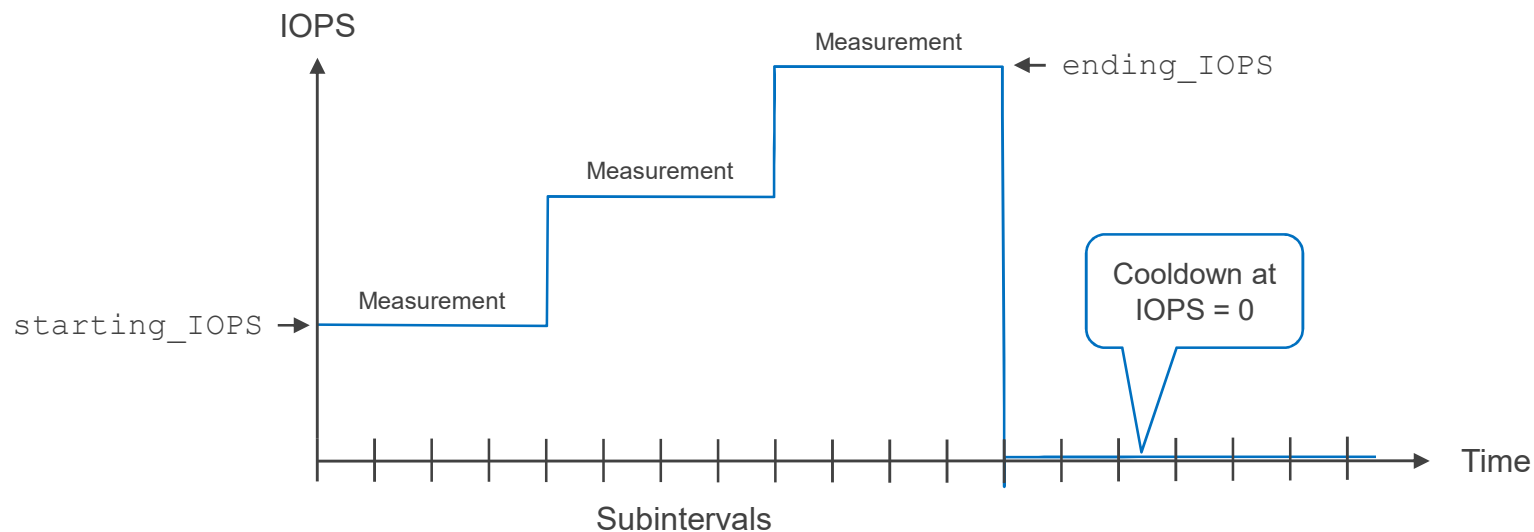
DFC = IOPS_staircase

- Use `starting_IOPS`, `ending_IOPS` and `steps`, and that many equal height steps are made (shown below).
- Use `step="+50%"` to increase the IOPS by 50% on each step.
- You can provide both `step` (size of step) and `steps` (number of steps) instead of `ending_IOPS`.
- If “`ending_IOPS`” is not provided, staircase steps continue until the requested IOPS can no longer be attained.



DFC = IOPS_staircase

- `cooldown_by_WP` and `cooldown_by_MP_busy` only apply at the end of the test step, that is, after the last measurement in the staircase.



DFC = IOPS_staircase

- starting_IOPS = nnn, etc. are values used with

```
[EditRollup] "all=all" [parameters] "total_IOPS = nnn";
```

- With edit rollup “total_IOPS”, the total IOPS value is first distributed evenly over all test host LUNs that have workloads defined. These are called “test LUNs”.
- Then within each test LUN, its share of total_IOPS is divided up between the workloads on that LUN proportional to the absolute value of the skew parameter of each workload. The resulting value is set as the IOPS for the workload.
 - skew defaults to -1.0. The negative value means “don’t apply this skew with IOPS=max”.
- One of the implications of this is that with DFC=IOPS_staircase, there can be no workloads running at IOPS=max. Every workload gets a share of the total_IOPS setting.

DFC=IOPS_staircase and testing to saturation

- **Note:** The IOPS steps up to the next level up a little bit (less than a second, more typically 1/4 second) after the end of the last subinterval in the previous step.
- This means that the very first subinterval in each following step averages a little bit lower than the IOPS setting for that step.
- Thus if you have `warmup_seconds = 0` with `DFC=IOPS_staircase`, the steps will tend to be marked "invalid" because the IOPS setting wasn't achieved.
- When running with at least one warmup subinterval, this won't happen.
- This becomes important with stepping to saturation, because the staircase ends when the IOPS setting hasn't been achieved.
- So use at least one warmup subinterval when testing to saturation.

achieved_IOPS_tolerance

- `ivy_engine_get("achieved_IOPS_tolerance")`
`ivy_engine_set("achieved_IOPS_tolerance", "0.25%")`
- When running at a specific IOPS numeric value (that is, not running `IOPS=max`), the measurement will be marked "invalid" if the achieved IOPS is more than `achieved_IOPS_tolerance` different from the (rollup instance's) `Total_IOPS` setting.
- This is also the criterion that determines when to stop a `DFC=IOPS_staircase` run to saturation that doesn't specify an `ending_IOPS` or equivalent. The staircase will stop after a step that fails to achieve its IOPS setting.
- The default `achieved_IOPS_tolerance` is 0.25%. This is good for the vast majority of cases.

[Go] parameter summary with defaults 1/2

■ Overall

- stepname = **stepNNNN**
- subinterval_seconds = 5
- warmup_seconds = **same as** subinterval_seconds
- measure_seconds = 60
- cooldown_seconds = 0
- cooldown_by_wp = on
- subsystem_WP_threshold = 1.5%
- cooldown_by_MP_busy = on
- subsystem_busy_threshold = 5%

■ For dfc = pid

- low_IOPS, low_target,
high_IOPS, high_target,
target_value
- max_ripple, gain_step,
max_monotone, ballpark_seconds

■ For DFC = IOPS_staircase

- starting_IOPS
- ending_IOPS
- step
- steps

■ For measure = on

- accuracy_plus_minus = 2%
- confidence = 95%
 - 50%, 60%, 70%, 80%, 90%, 95%,
98%, 99%, 99.5%, 99.8%, or
99.9%
- max_wp = 100%
- min_wp = 0%
- max_wp_change = 3%

[Go] parameter summary with defaults 2/2

■ Focus metric

- focus_rollup = all
- source = ""
 - or workload / RAID_subsystem
- subsystem_element = ""
- element_metric = ""
- category = overall
 - or read, write, random, sequential, random_read, random_write, sequential_read, sequential_write
- accumulator_type = ""
 - or bytes_transferred, service_time, response_time
- accessor = ""
 - avg, count, min, max, sum, variance, standardDeviation

A general note on ivy parameter names

- Ivy "normalizes" all user-specified parameter names by converting to lower case and removing underscore `_` characters
 - For example, `maxTags` may also be written `max_tags` or `MAXTAGS`.
- You can also say `[create workload]` instead of `[CreateWorkload]`.

When a test host gets too hot

- `ivydriver` checks the CPU temperature once per subinterval.
- When the hottest core gets within 5 degrees Celsius of the critical limit, `ivydriver` starts issuing `<Warning>` messages.
 - NOTE: We have seen CPU throttling and even machine checks appear in this range. Check system logs.
- When the hottest core actually hits the critical temperature and the CPU is definitely throttled or temporarily paused, the default behaviour is to issue an `<Error>` message and abort.
 - Default is `ivy_engine_set("critical_temp", "error")`
 - Alternative is `ivy_engine_set("critical_temp", "warn")`

NOTE: this `ivy_engine_set` must come before the `[hosts]` statement to be effective.
- With `"warn"`, if the CPU stops for several seconds, you may need to run with a longer `subinterval_seconds` on the `[go]` statement.

```
ivy_engine_set("spinloop", "off");
```

- By default, in ivy version 4 with `io_uring`, I/O driving workload threads operate in a "spin loop" with the CPU 100% busy testing for I/O completion events.
 - This optimizes IOPS and service time.
- If you turn off "`spinloop`", ivy 4 will wait for I/O completion events or until it's time to start the next I/O if there's nothing else to do.
 - This reduces CPU % busy most of the time, but at the expense of some potential IOPS loss.
 - You can also turn `spinloop` off to see if you are running out of CPU.

`ivy_engine_set("generate_at_a_time", 4) ;`

- If there are no I/O completion events to harvest, and no new I/Os to start at this time, ivy will "pre-generate" one or more I/Os ahead of time.
- The "generate_at_a_time" parameter (default 4) specifies the limit for the number of I/Os to pre-generate before checking for I/O completions again.
- This is a tradeoff between responsiveness and CPU busy.


```
ivy_engine_set("fruitless_passes_before_wait",1);
```

- If you turn off "spinloop", ivy will wait for I/O completion events or until it's time to start the next I/O.
- This parameter can be used to specify a number of times that ivy's I/O driving engine checks for any work to do before waiting.
- The default is 1.
- Setting this higher with spinloop turned off will compromise between spinloop performance and non-spinloop CPU busy / heat generation.

```
ivy_engine_set("track_long_running_IOs", "off");
```

-
- Turning this off reduces CPU use a bit.

Advanced topics

 Hitachi Data Systems

© Hitachi Vantara 2020. All rights reserved.

source **of the** focus_metric (without shorthand)

- 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 (*more next page*)
- source = RAID_subsystem
 - Requires the proprietary command device connector that is not part of the ivy open source project.
 - 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. (*more even later after we explain source=workload*)

Selecting a "source=workload" metric

- 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" (without shorthand)

- 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:

- 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.
 - If IOPS is set to a specific number, but that IOPS is not achieved, response time is suppressed.

Ivy uses the Linux
nanosecond
resolution clock
for all timing

How ivy posts results of each I/O

- 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

- 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

- For the `all=all` rollup 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**

- overall
read, write
random, sequential
random_read, random_write, sequential_read, sequential_write
- These are actually the virtual categories, representing the rollup over the underlying service time / response time bucket arrays (histograms).
 - 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

- Category buckets have 3 accumulators
- `accumulator_type = 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_type = 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_type = 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

- 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

- 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_engine.h” 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

- 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,

**The remainder of this
presentation is a draft
section describing
`dedupe=target_spread`**

Original serpentine dedupe method

- The serpentine method writes an average of “dedupe” copies of each new unique data pattern, writes these at *the same time* regardless of the location being written to.
- When writing / filling sequentially, the achieved dedupe ratio is “dedupe”.
- When subsequently writing new patterns randomly, each copy goes to a new random location, possibly over-writing one copy of a previous pattern, causing the achieved dedupe ratio to fall over time to an asymptotic limit.

serpentine **method, continued**

- The *serpentine asymptotic dedupe ratio* (*target dedupe ratio = R*) at equilibrium is given by the formula (*R / HarmonicNumber (R)*).
- $\text{Asymptotic Dedupe_ratio}(R) = R / (1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{R})$
- The approximation for this is $\sim R / \log_e(2R + 1)$

- Pattern Generation changed to achieve the target dedupe ratio with a mix of unique patterns and patterns with higher number of duplicates (target + spread) following a predefined distribution.
- For example: Generated pattern sequence for target dedupe ratio of 2.0
- P1, P2, P3, P4, ***P11, P11, P11, P11, P11, P11, P11***, P12
- Dedupe ratio = (total # of blocks = 12) / (unique blocks = 6) = 2.0

Pattern generation at the “dedupe unit” size

- Hitachi Advanced Data Reduction deduplicates at the fixed 8 KiB block size. i.e., at a conceptual “dedupe unit”.
- `target_spread` pattern generation is at the 8 KiB dedupe unit level.
- The `target_spread` pattern sequence generated is based on the distribution (mix of unique blocks and duplicate blocks repeating in a sequence) corresponding to a target dedupe ratio and at the dedupe unit size. The pattern sequence is the same for Sequential or Random writes.
- For larger block sizes, the larger block will be filled with blocks of dedupe unit size with the distribution based pattern sequence.

target_spread **(continued)**

- For example, for blocksize = 256 KiB, 32 dedupe unit size blocks are generated and filled – the 32 dedupe unit sized blocks will follow the pattern sequence.
- Pattern generation is to be consistent at each LUN and for any reasonably sized (≥ 1 GiB) sample set of contiguous blocks.
- Unique starting seed for each LUN based off of a universal fixed seed i.e., no deduplication across LUNs (unlike serpentine sequence).
- Probabilistic Pattern reuse for Random writes.


Pattern sequence generation algorithm

- Each block of data starts with the block_seed and the Eyeo data generator repeatedly runs xorshift64 to generate a pseudo-random noise as data.
- $\text{block_seed} = \text{pattern_seed} \wedge \text{pattern_number}$.
- pattern_seed and pattern_number is changed based on a serpentine sequence (old method) or using a distribution (new method) when a new pattern is generated.

Pattern reuse in random writes

- Use of universal starter seed
 - `#define universal_seed 1234567` in `include/ivydefines.h`
- Using a generated pod of seeds with the universal seed as a starting seed using the distribution for the target dedupe ratio.
- Reuse patterns by readjusting starting `pattern_seed` and `pattern_number`.
- Pattern seeds are reused with probabilities based on the target dedupe ratio and expected dedupe ratio steady state after multiple random write workloads.

pattern_number_reuse_threshold heuristics

- Dedupe ratio dependent threshold to keep pattern numbers in the range: [0..pattern_number_reuse_threshold]
- The numeric threshold values in italics were determined based on running modified 6D2P_debup.ivyscript,  and dedupe ratios measured using DRE tool.

6D2P_dedup.ivyscript

```
if (target_dedupe < 2.0)
```

```
    pattern_number_reuse_threshold =  
        (1.0 - reuse_probability) * 100000;
```

```
else if (target_dedupe > 10.0)
```

```
    pattern_number_reuse_threshold =  
        (1.0 - reuse_probability) * 65000;
```

```
else pattern_number_reuse_threshold = (1.0 - reuse_probability) * 50000;
```


- DedupePatternRegulator.{h, cpp}
 - Responsible for the pattern distribution for target dedupe ratio
 - Generates a pod of starting seeds on instantiation for reuse in the case of random write workloads
 - Probabilistic random or reuse of starting seed decision (decide_reuse() method)
- Eyeo.cpp
 - pattern filling with dedupe unit size based blocks from an array of block seeds spanning the xfer size block size
- Workload.cpp
 - Build an array of block seeds spanning an xfer block at dedupe unit boundary for consumption by Eyeo



HITACHI
Inspire the Next

Thank You

 **Hitachi Data Systems**

© Hitachi Vantara 2019. All rights reserved.