IO latency tracking

Daniel Lezcano (dlezcano)
Linaro Power Management Team

CPUIDLE : the menu governor

- Tries to predict the next event on the system
- Does statistics on each CPU for the sleep duration
- Weighting the idle states regarding the pending IOs

CPUIDLE : the menu governor

- Source of wakeups:
  - IRQ
    - Timer, IOs, keyboard, ...
  - IPI
    - Mostly generated by the scheduler

But most of the interrupts are coming from:

- Timers, IOs and rescheduling IPIs

CPUIDLE : the menu governor

- It takes all the source of interrupts without distinction:
  - ➔ Is it up to the scheduler to predict the scheduler behavior (IPI reschedule) ?
- What about if a task is moved to another CPU ?
  - ➔ Can stay to a very shallow state for seconds for nothing because the statistics are different

How can we improve the situation ?

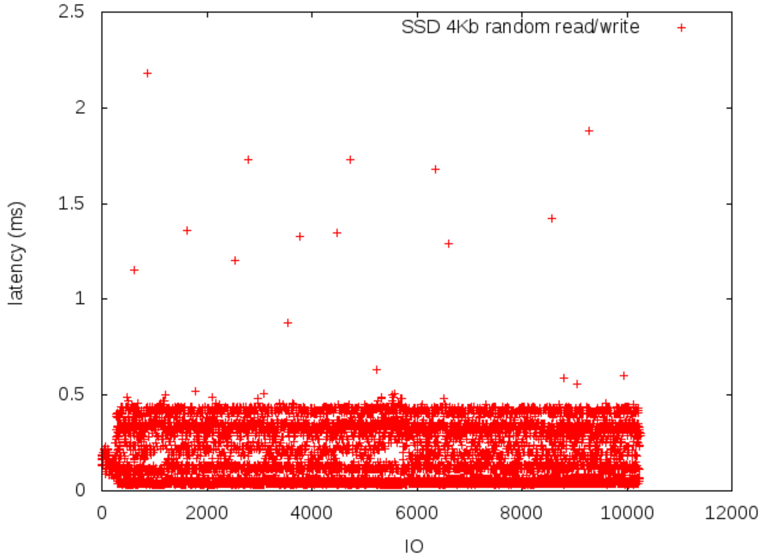
- Why not focus on event which are predictable ?
  - Timers
  - IO
- Let the scheduler to take the right decision with the IPI reschedule
- ... and consider the rest as noise preventing us to be accurate

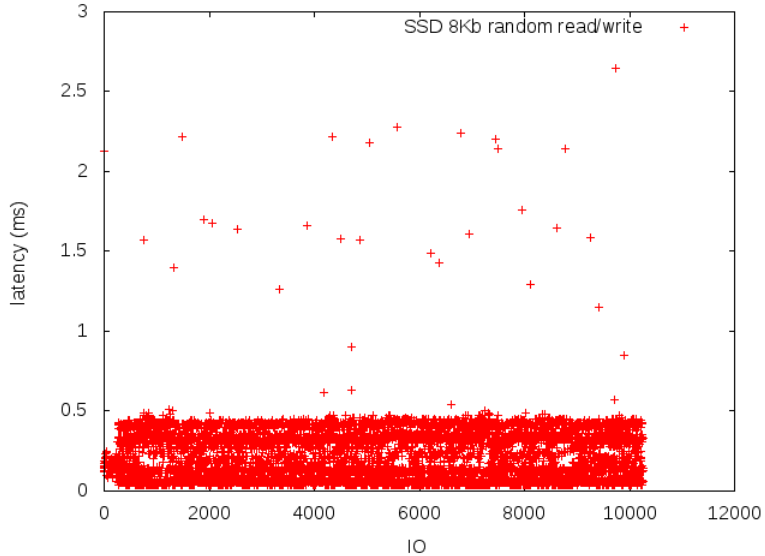
Why an IO could be predictable ?

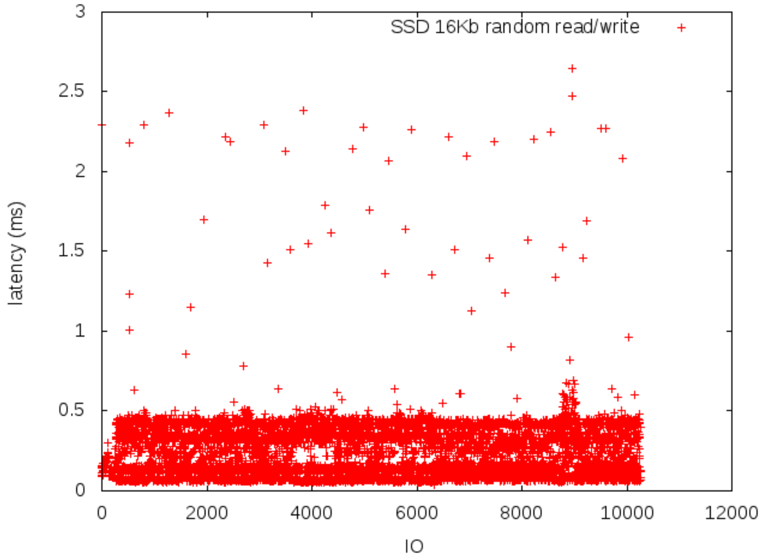
The duration of an IO is inside a reasonable interval, with repeating pattern

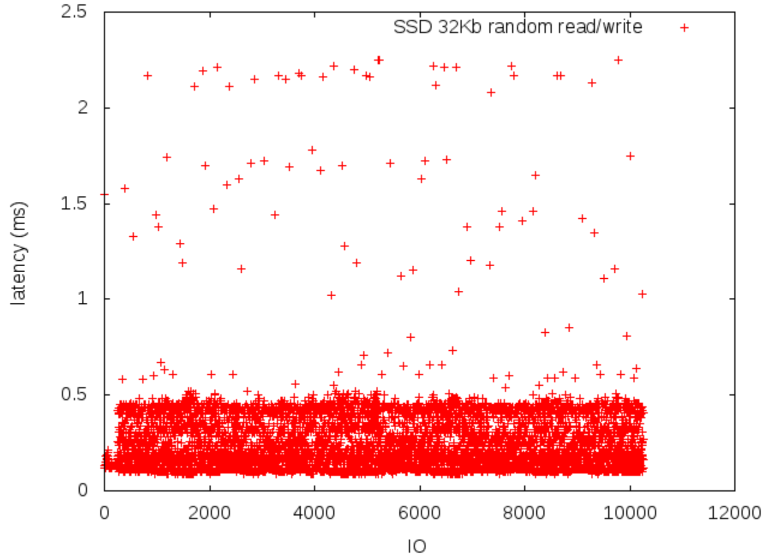
- The tasks are blocked on IO, hence these latencies information can be tied with it
  - The information can follow the task when this one is migrated

SSD behavior

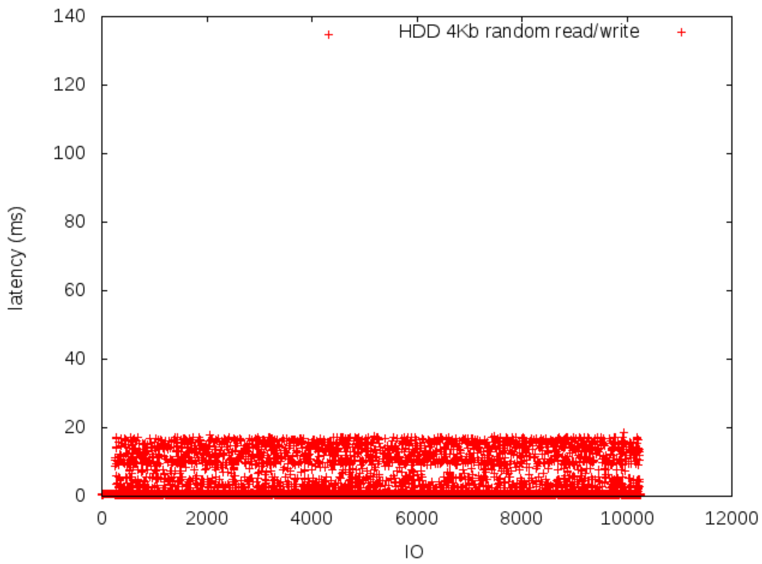


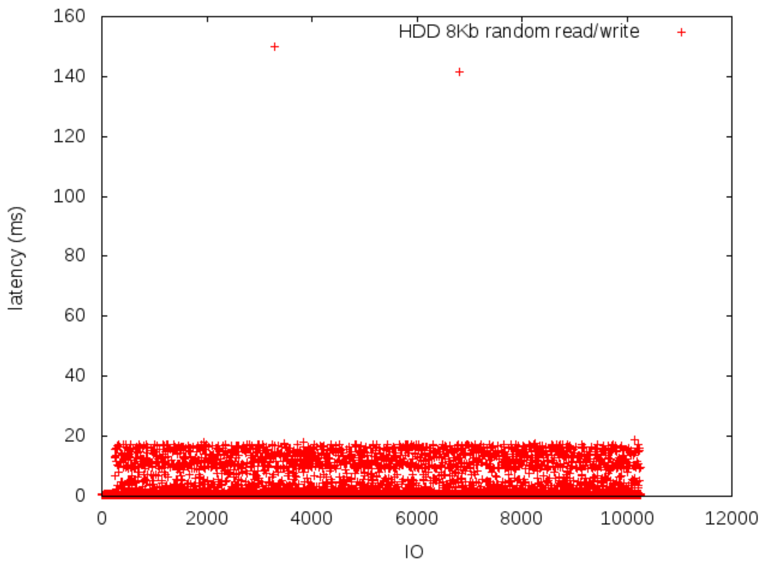


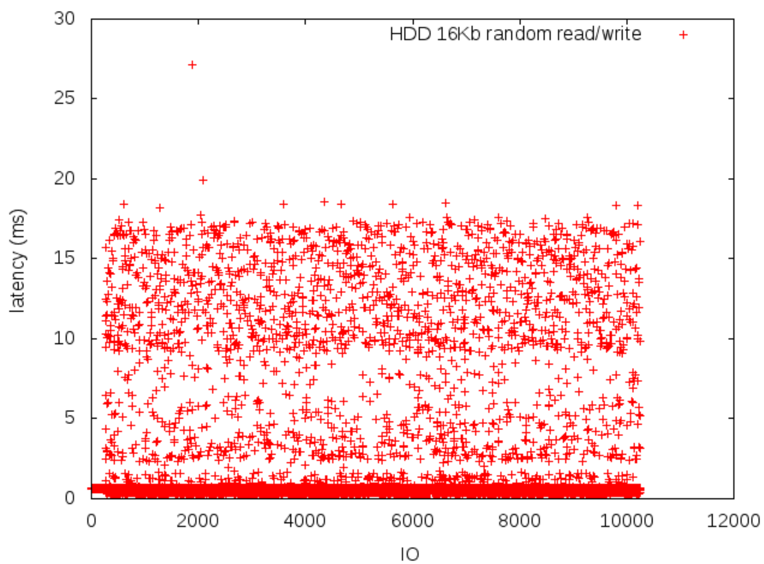


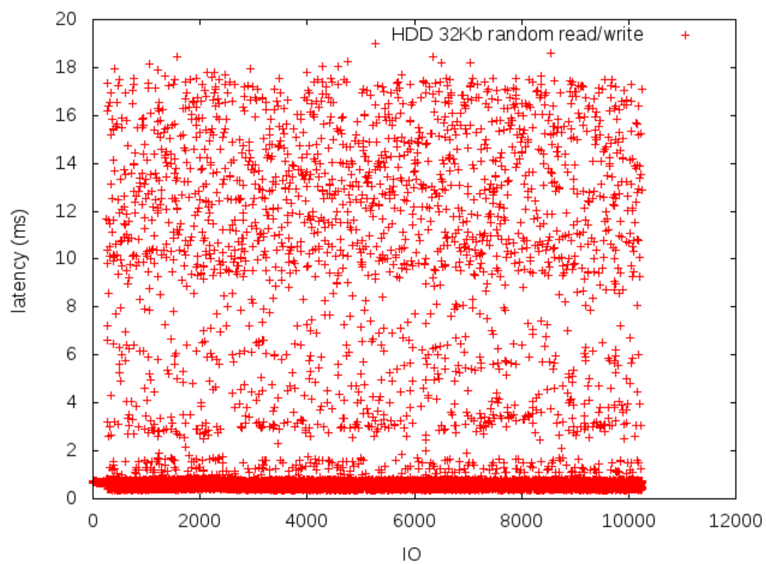


HDD behavior





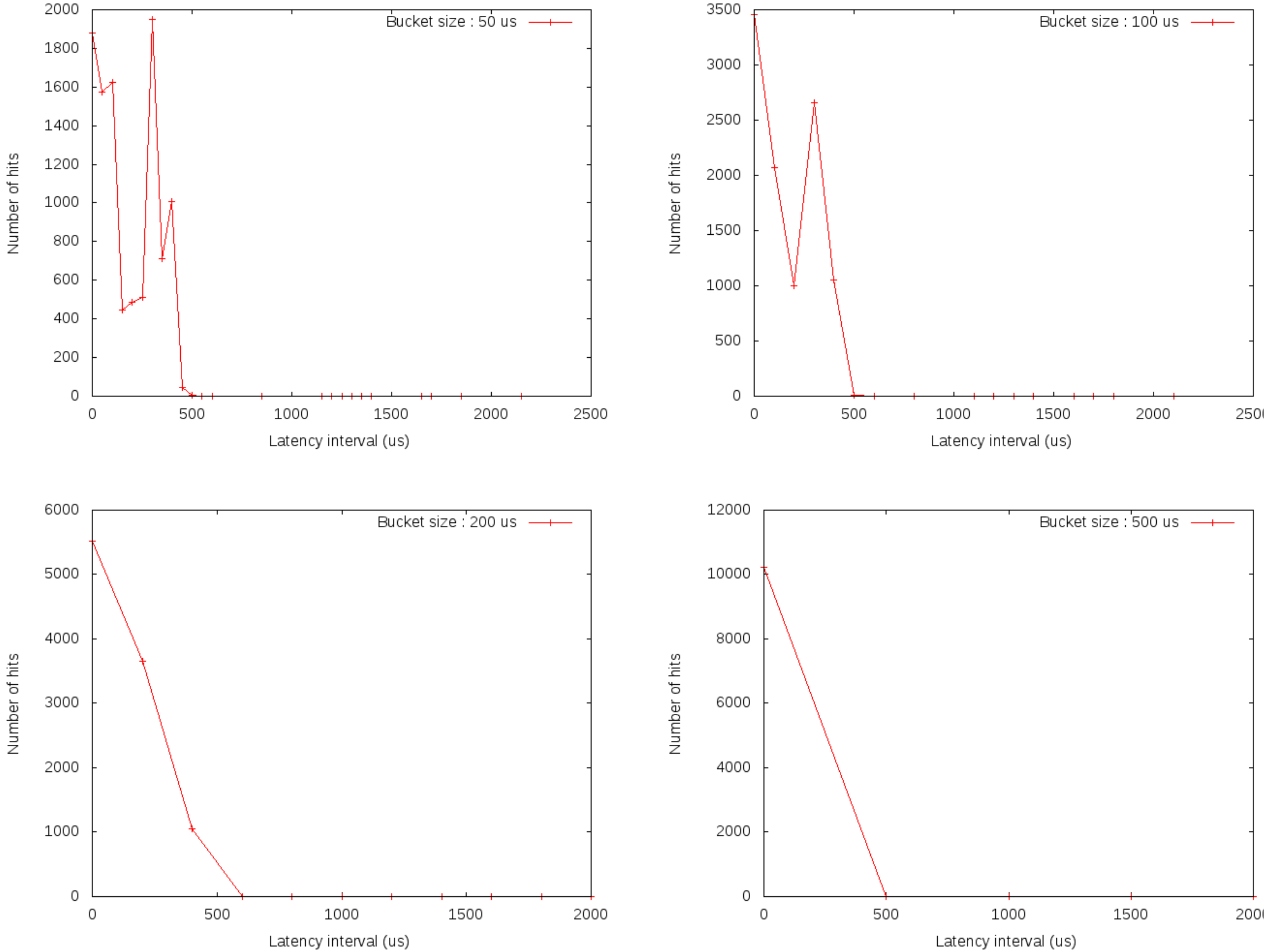




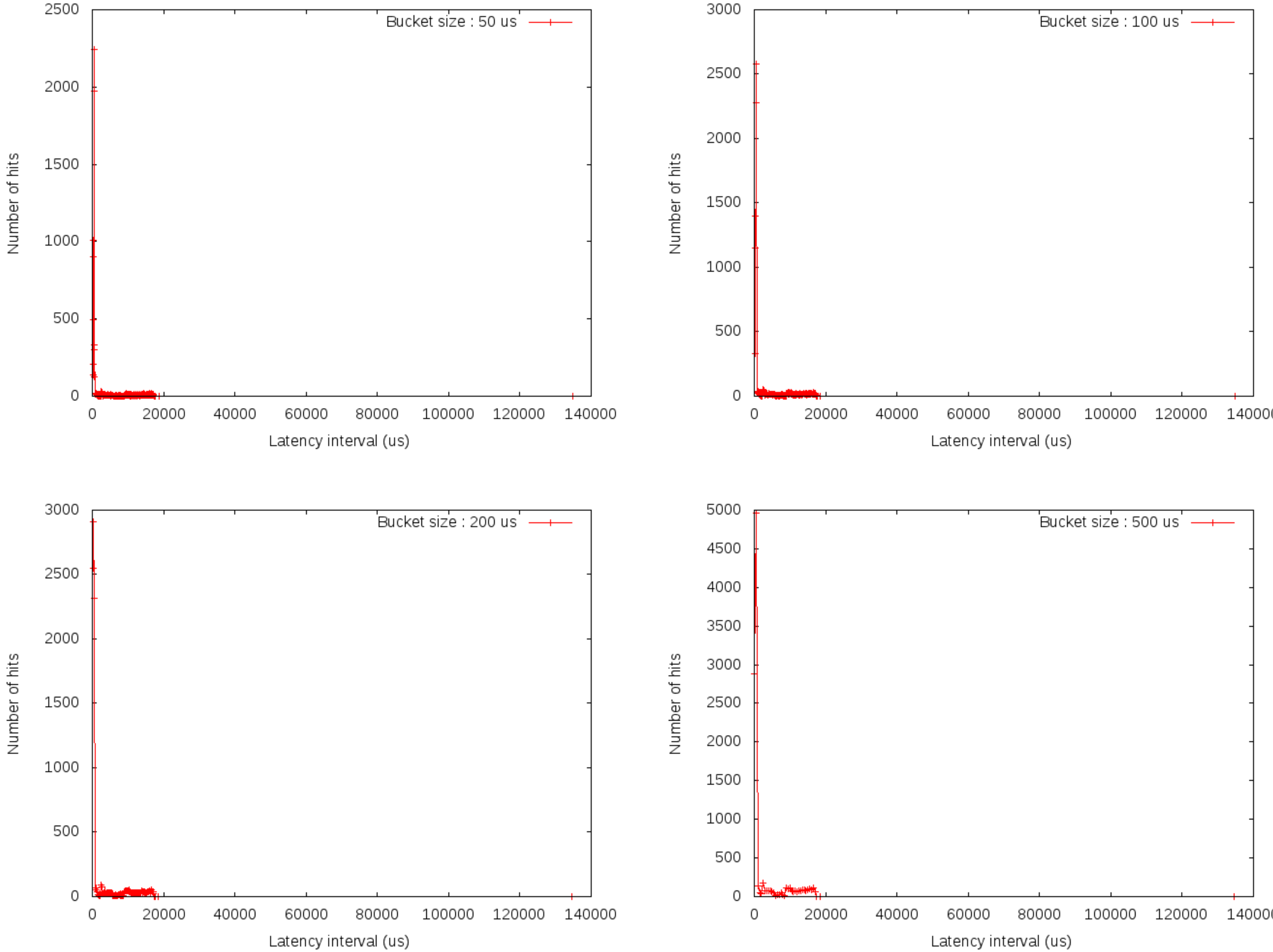
Observations

- Let's group the different latencies into buckets
- Experiment with different bucket size : 50us, 100us, 200us and 500us
- And observe the distribution : how many times a bucket is hit ?

SSD – 4KB Random RW



HDD – 4KB Random RW



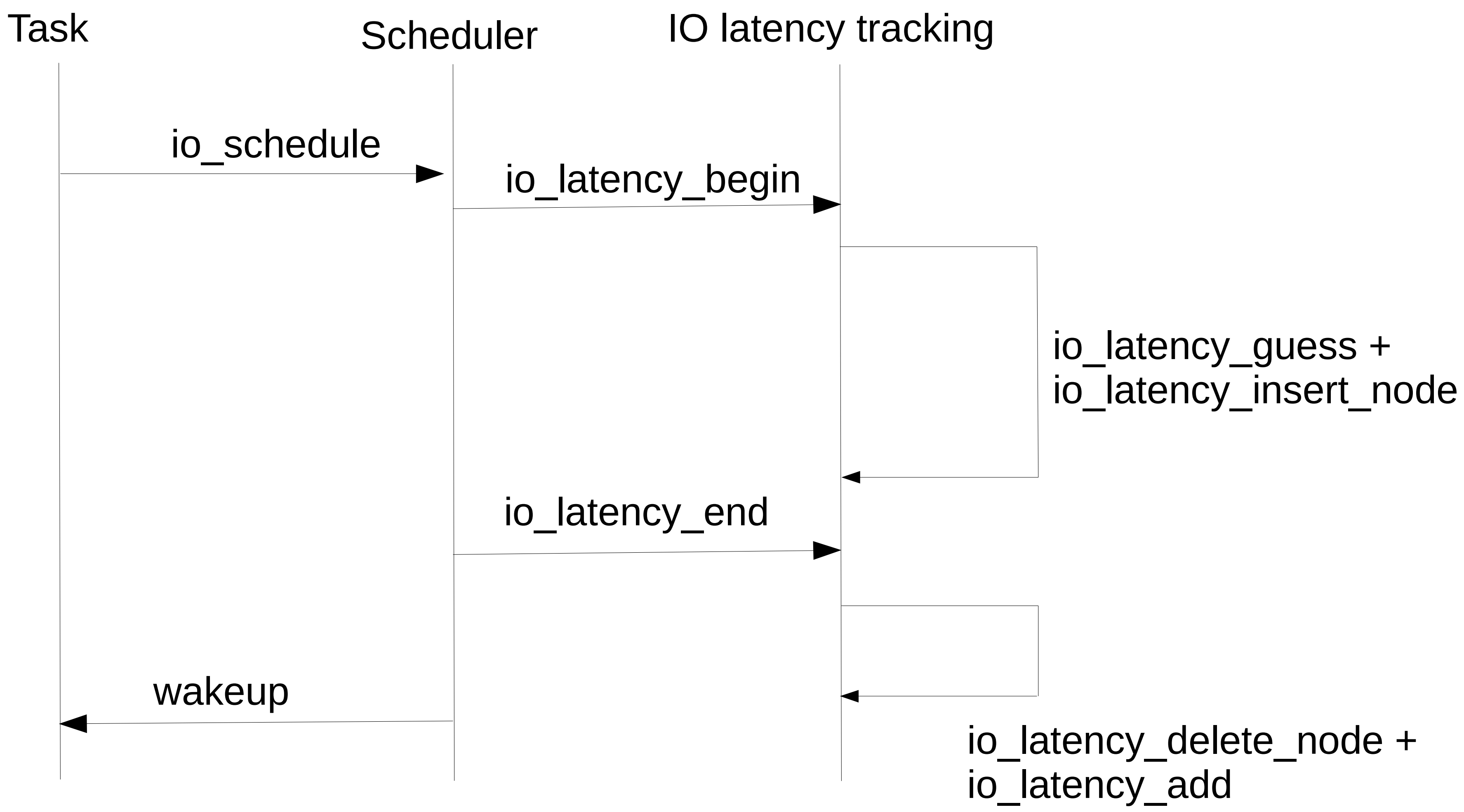
Observations - Conclusions

- The smaller the bucket is, the more there are buckets
  - ➔ Reduce probability to estimate the right bucket
- On slow media, there is an important deviation from the average latency
- The 200us bucket size shows a interesting trade-off between the number of buckets and the accuracy
- We can rely on these observations to build a model to predict the next IO

IO latency tracking infrastructure

- Each time a task is unblocked after an IO completion, measure the duration
- Add this latency to the IO latency tracking infrastructure
- Next time the task is blocked on a IO, ask the IO latency tracking infrastructure the guessed blocking time
- When going to idle, take the remaining time to be blocked on an IO as part of the equation to compute the sleep duration

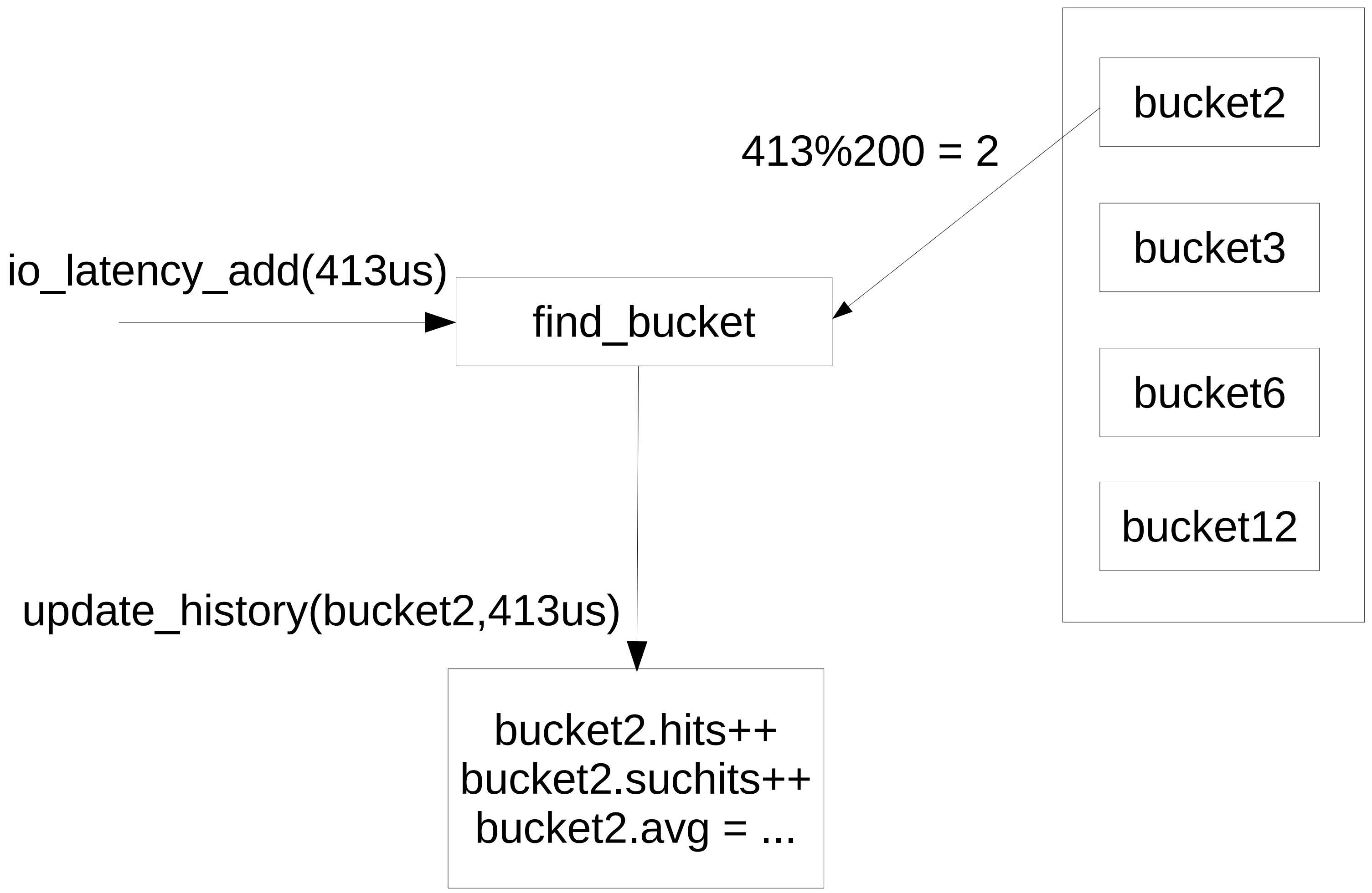
IO latency tracking infrastructure



IO latency tracking infrastructure

- Group the latencies into buckets, representing an interval (200us)
- Each bucket has an index
- Each index gives the bucket's interval
  - index : 0   => [0, 199]
  - index : 10 => [2000, 2199]
  - Index : 5   => [1000, 1199]

IO latency tracking infrastructure



IO latency tracking infrastructure

- How do we guess the next blocking time ?
  - Buckets are sorted
  - Position in the list shows the history (first happening more, last happening less)
  - Buckets have the number of hits and the position in the list weight these numbers
  - Compute a score with the position in the list and the number of hits with a decaying function

IO latency tracking infrastructure

<u>Score = nrhits / (pos + 1)²</u>

first

bucket2.hits = 123	Score = 123 / $(0+1)^2$ = **123**
bucket3.hits = 321	Score = 321 / $(1+1)^2$ = **80**
bucket6.hits = 12	Score = 12 / $(2+1)^2$ = **1**
bucket12.hits = 32	Score = 12 / $(3+1)^2$ = **0**

last

Bucket 2 is the next expected IO latency

IO latency tracking infrastructure

- What happens when there are several tasks blocked on an IO ?
  - A red-black tree per cpu
  - The number of elements of this tree is the number of tasks blocked on a IO
  - Each node is the guessed IO duration for the corresponding task
  - Left most node is the smaller guessed IO duration

IO latency tracking infrastructure + CPUidle

- When entering idle we have now:
  - The next timer event which is reliable (next_timer_event)
  - The next IO completion (next_io_event)

- The next event is:

next_event = min(next_timer_event, next_io_event)

IO latency tracking + CPUidle

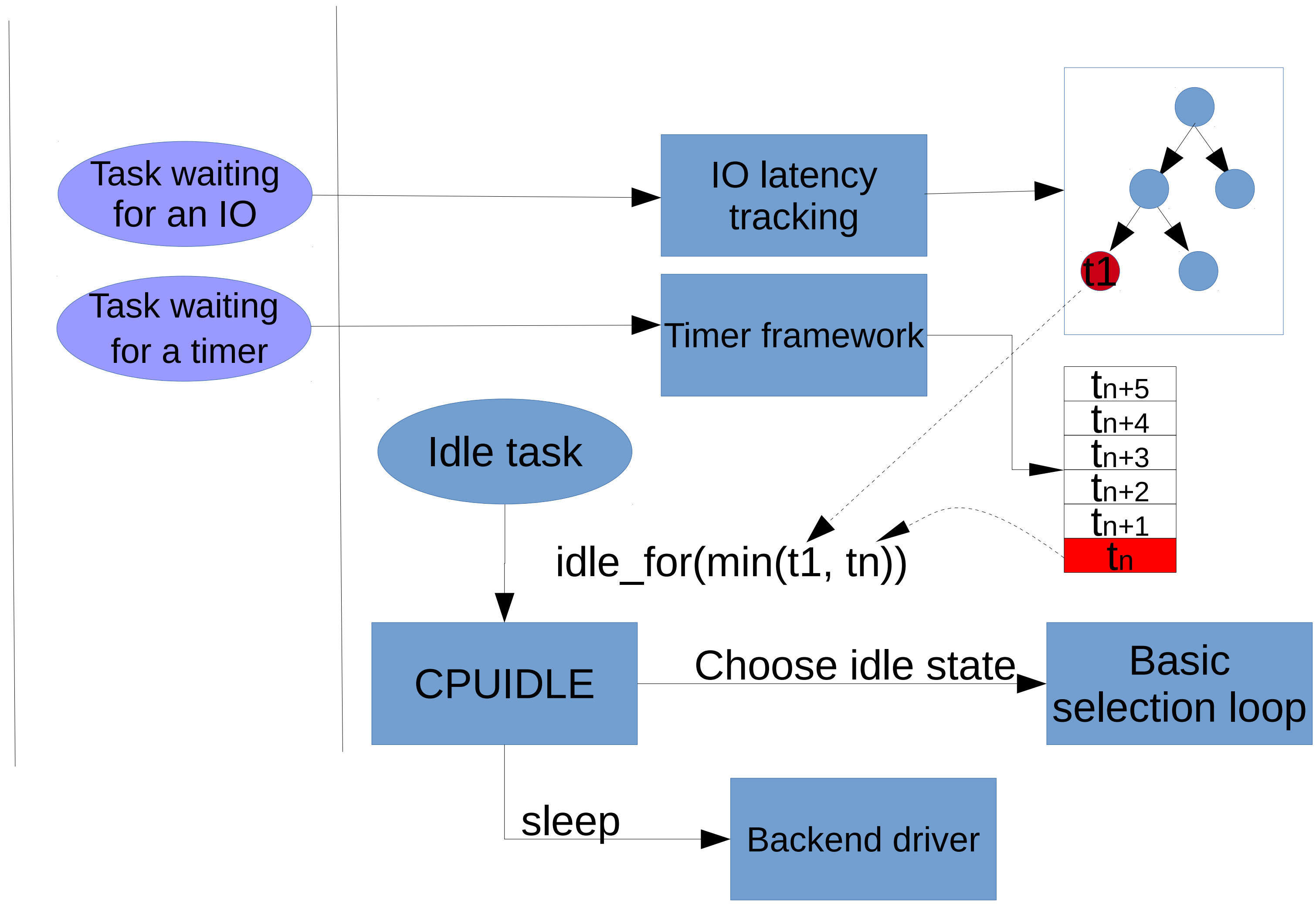
- Choosing the idle state is straighforward:

```
for (i = 0; i < drv->state_count; i++) {
	struct cpuidle_state *s = &drv->states[i];
	struct cpuidle_state_usage *su = &dev->states_usage[i];

	if (s->disabled || su->disable)
		continue;
	if (s->target_residency > next_event)
		continue;
	if (s->exit_latency > latency_req)
		continue;

	idle_state = i;
}
```

The big picture



Some results

- Tools used to measure:
  - Some extra infos added to sysfs for each cpu
    - Over predicted : the sleep duration was shorter
    - Under predicted : the sleep duration was longer
  - Idlestat : a tool computing the idle state statistics
    - http://git.linaro.org/power/idlestat.git
  - Iolatsimu: a tool implementing the prediction algorithm and randomly read/write a file
    - http://git.linaro.org/people/daniel.lezcano/iolatsimu.git

Some results

- Tests done under certain circumstances
  - Process pinned on one cpu in order to prevent migration
  - Tried on the media described at the beginning of the documentation

- For each block size measure the prediction regarding the idle state which has been choose

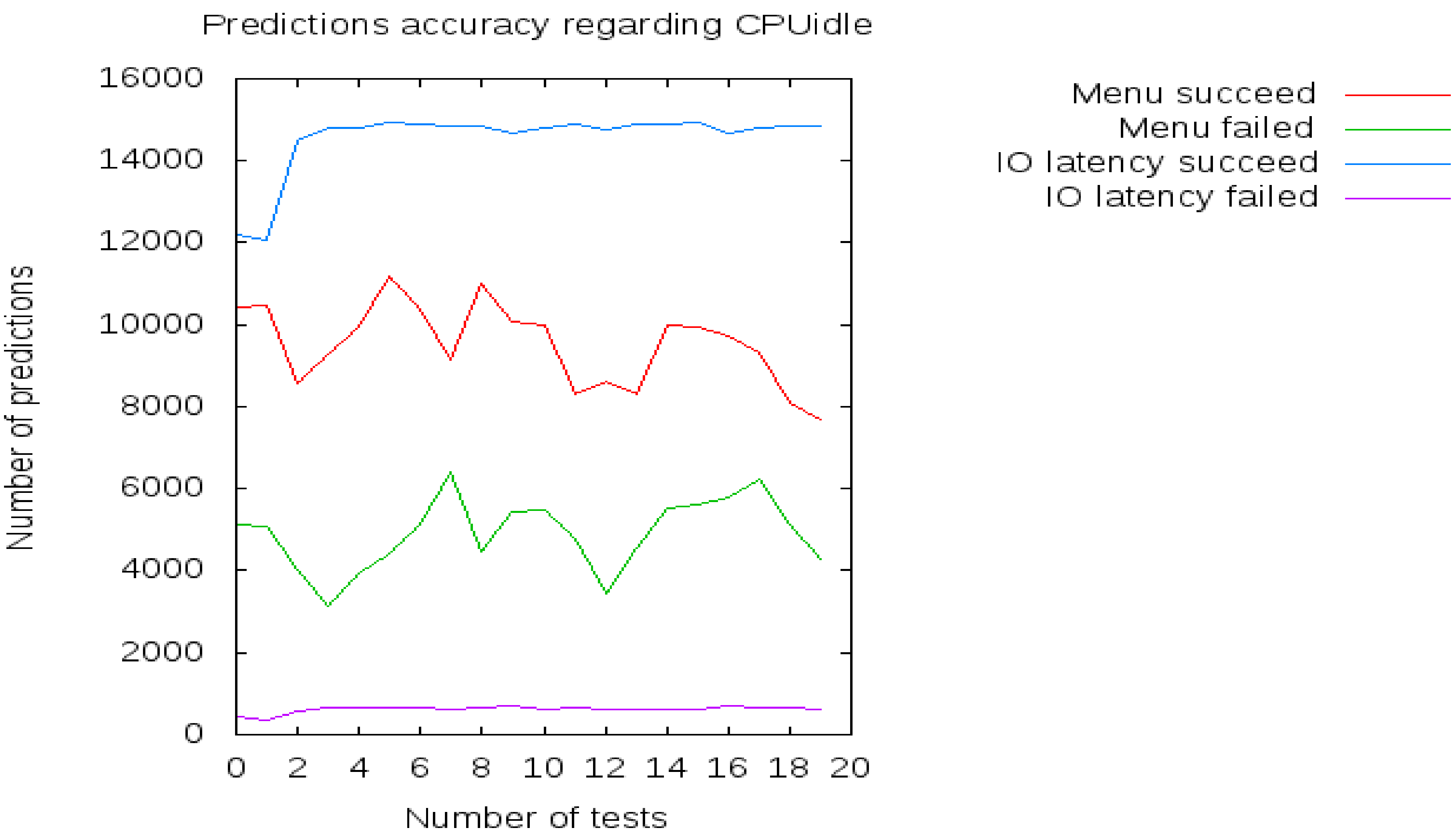
Some results

IO latency framework + timer give an expected sleep time of 30us

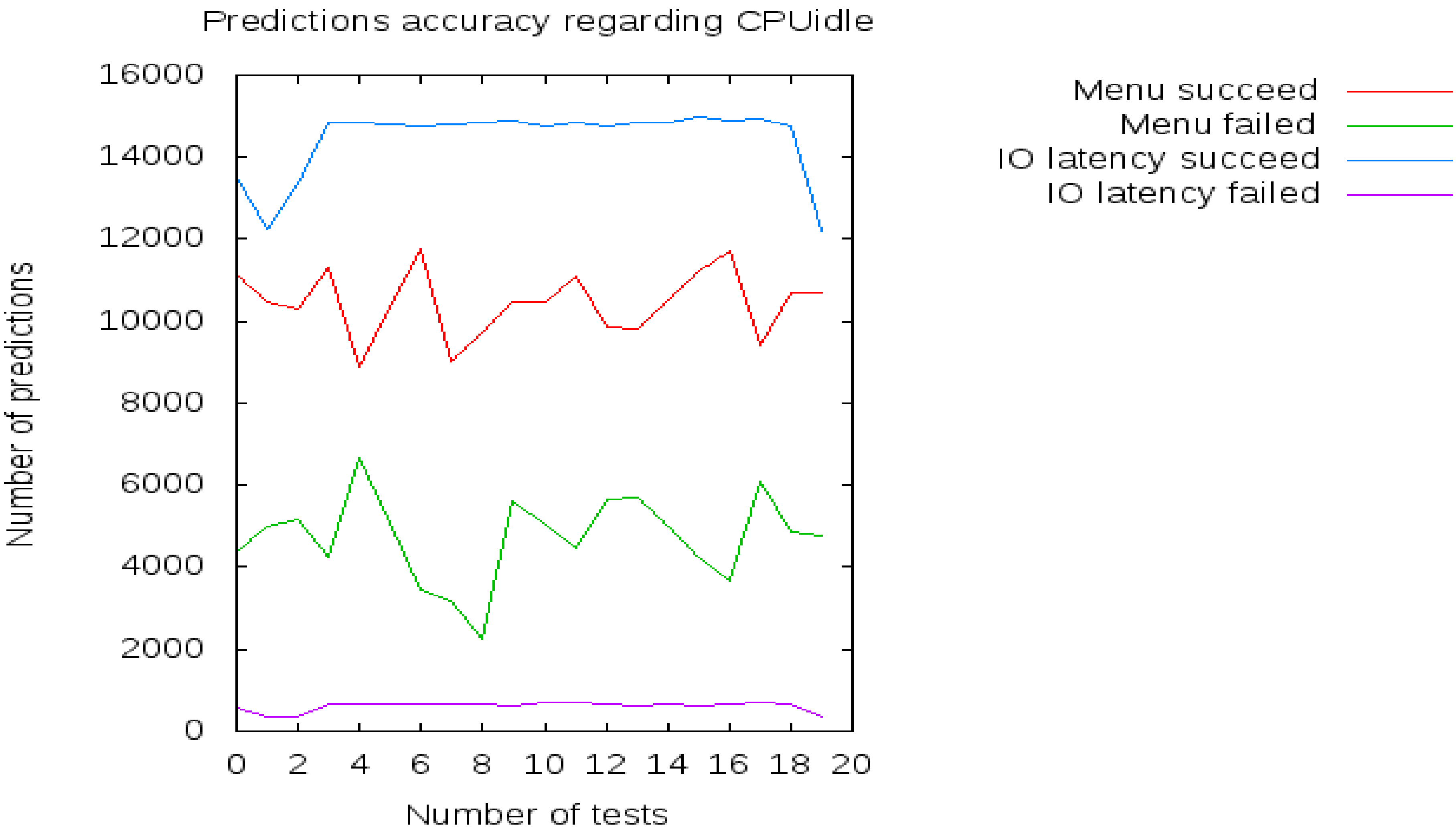
1. Effective sleep time = 15
2. Effective sleep time = 35
3. Effective sleep time = 45

1. and 3. are failed predictions, 2. is a right prediction

SDD results - 4KB



HDD results - 4KB



Conclusion

- A noticeable improvement in terms of prediction, more than 30% under some circumstances
- The resulting design makes the idle state predictions within the scheduler
  - ➔ Idling decision could be integrated in a smarter way

Next steps

- Improve the IO latency tracking framework algorithm, detect repeating patterns, ...
- Investigate pointless IPI with IO completion
- Increase the test coverage with all benchmarks
- Fix the IO completions measurement probe points
- Make the latency per device
- Improve the scheduler to take smarter decision regarding idle

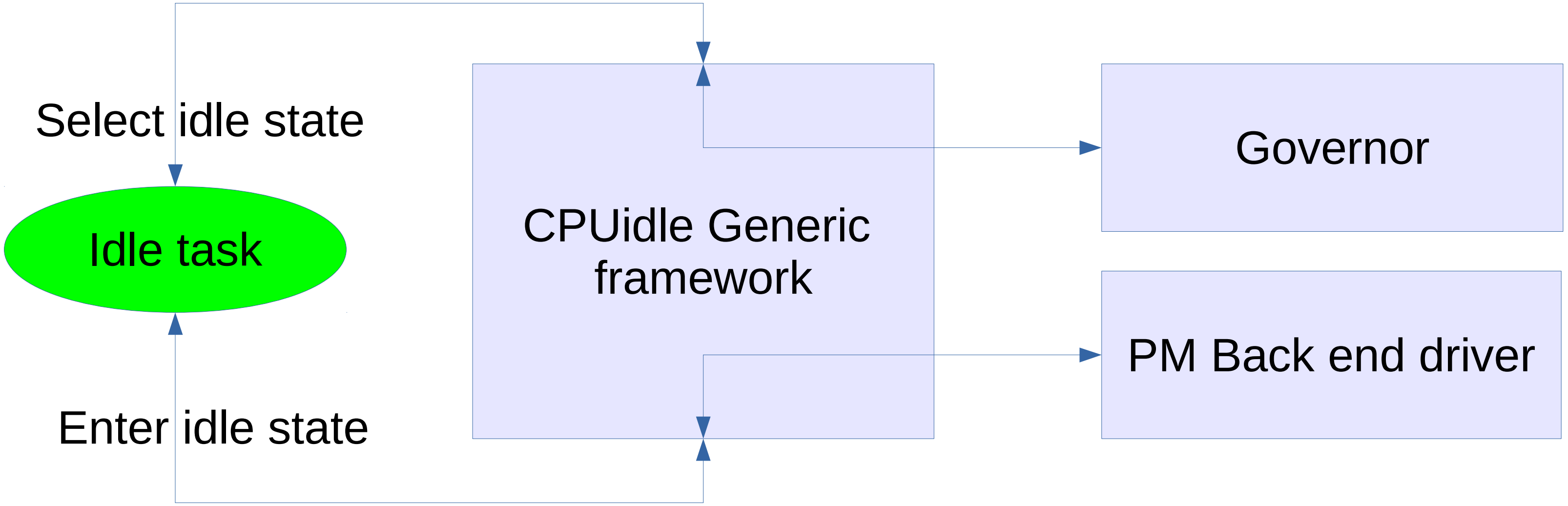
THANKS !

Introduction

- New architecture : HMP or big.Little
- The scheduler is only aware of SMP
- We need to integrate the different power management subsystems into the scheduler
  - Cpufreq, pm qos, cpuidle, ...
- This presentation is about integrating cpuidle with the scheduler

CPUIDLE

- CPUidle is a framework divided into three parts



CPUIDLE : the generic framework

- Provides an API:
  - To register a driver and a governor
  - To ask for the governor suggestion
  - To enter idle with the state abstraction
- No algorithm

CPUIDLE : the back end drivers

- It gives the abstraction for the complex PM code
- Very generic for some hardware based on firmware abstraction (eg. ACPI, PSCI, ...)
- Very hardware dependent and complex if directly handled in the kernel (eg. ARM SoC specific drivers)

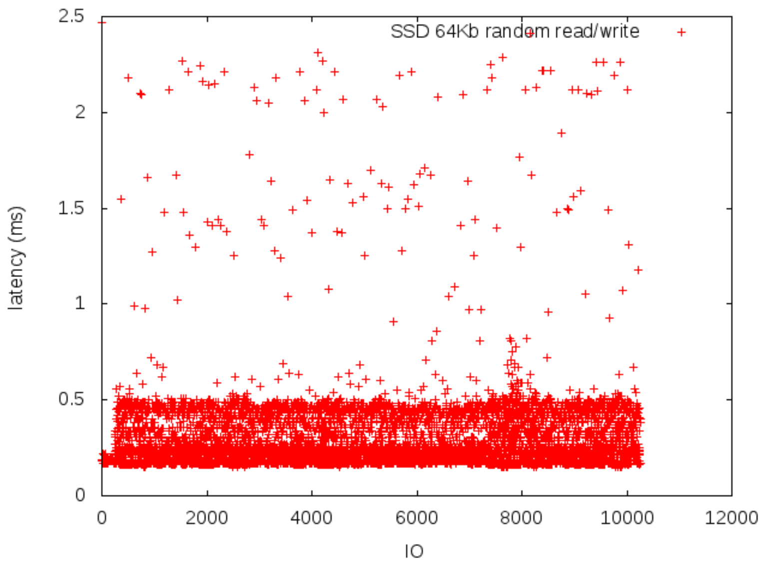
CPUIDLE : the governors

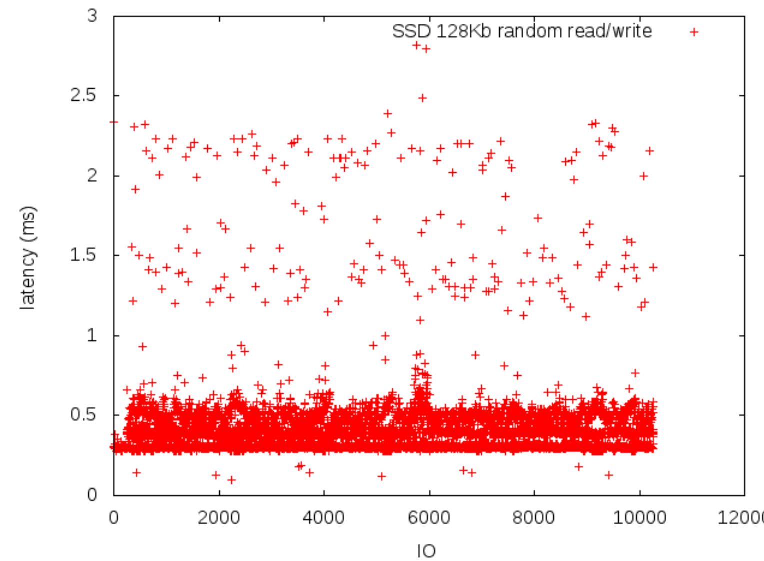
- Two governors used today:
  - Ladder with periodic tick configuration:
    - for server focused on performance
  - Menu with nohz configuration:
    - for most of the platforms trying to provide the best trade-off between performance and energy saving

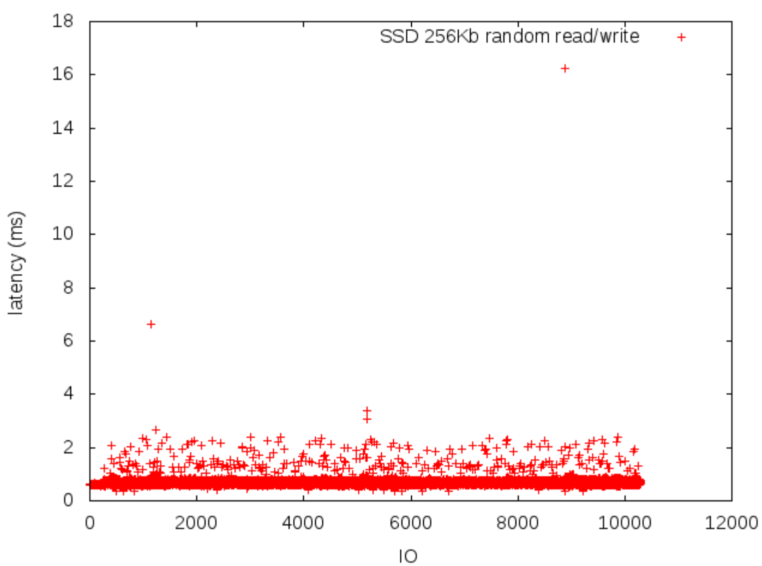
Experimentation

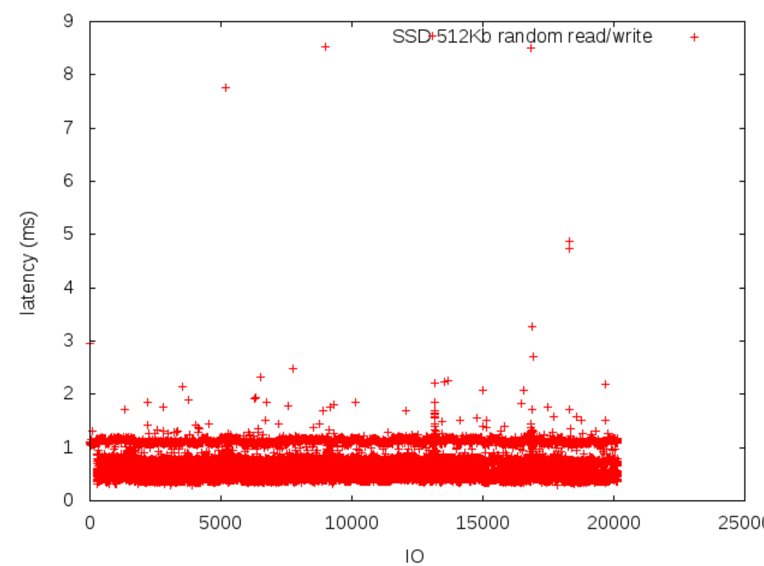
- Latency measurements for:
  - SSD 6Gb/s
  - HDD 6Gb/s
- ... with different block sizes
  - From 4KB to 512KB

SSD behavior

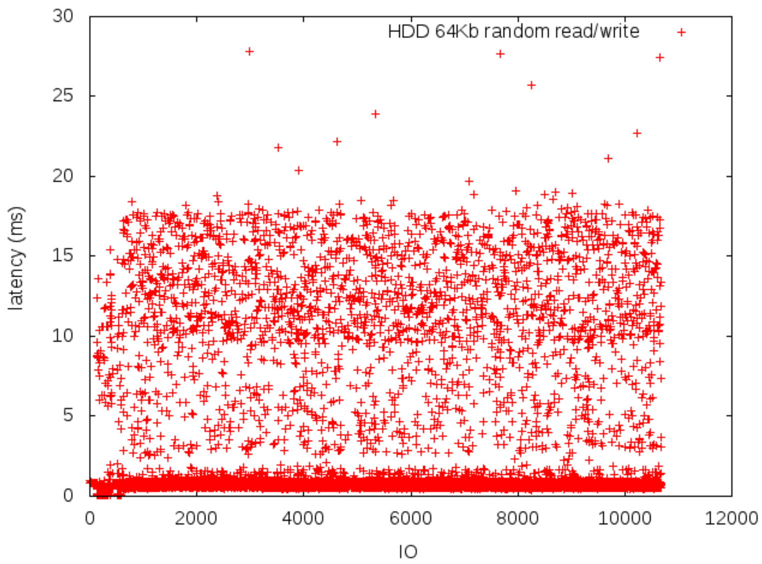


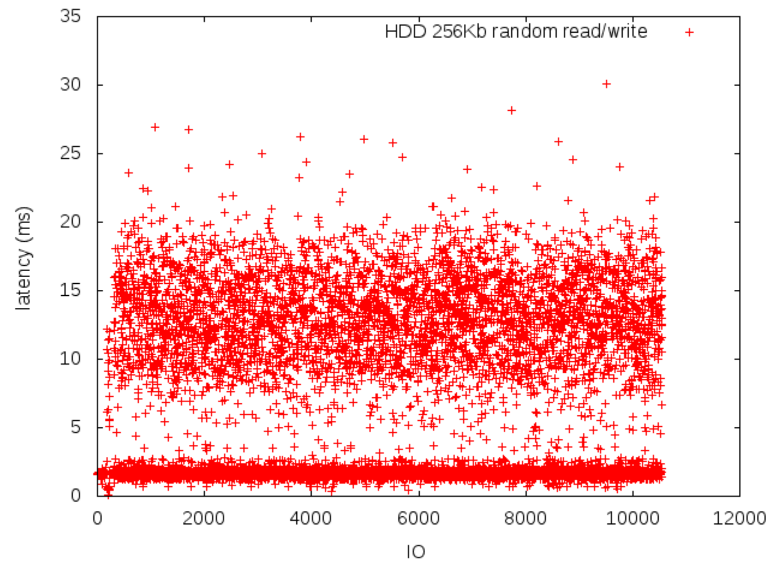


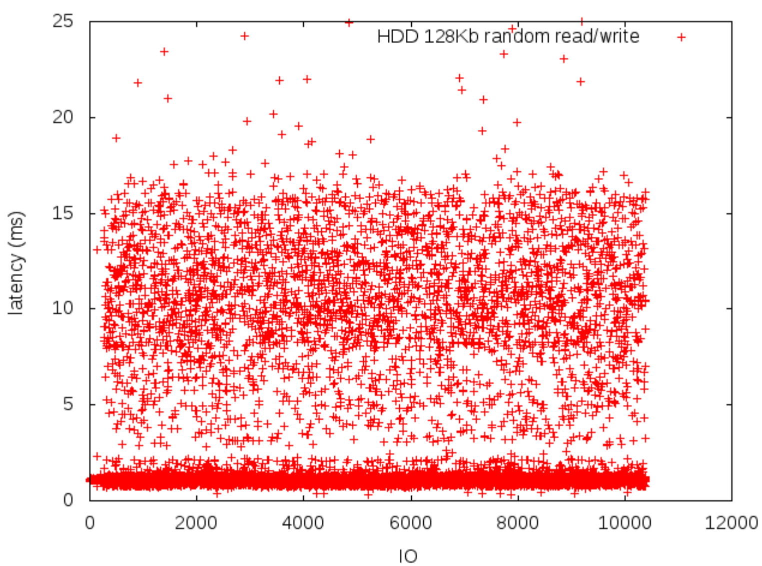


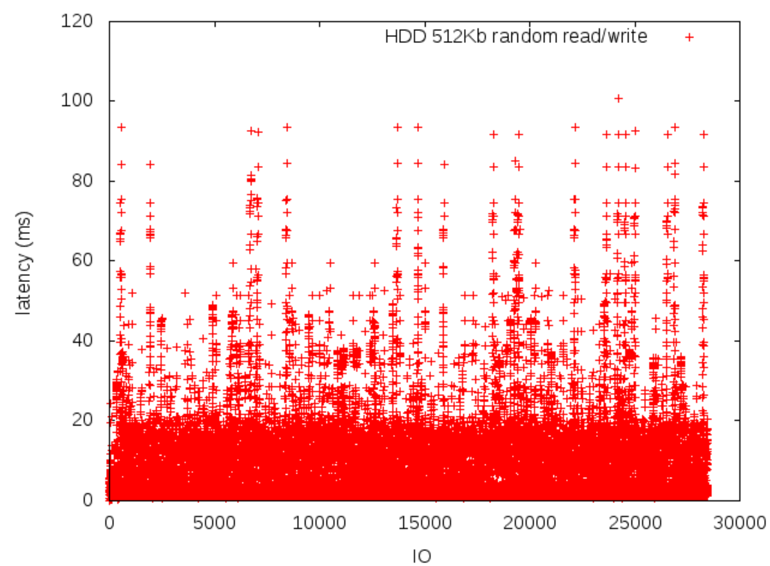


HDD behavior

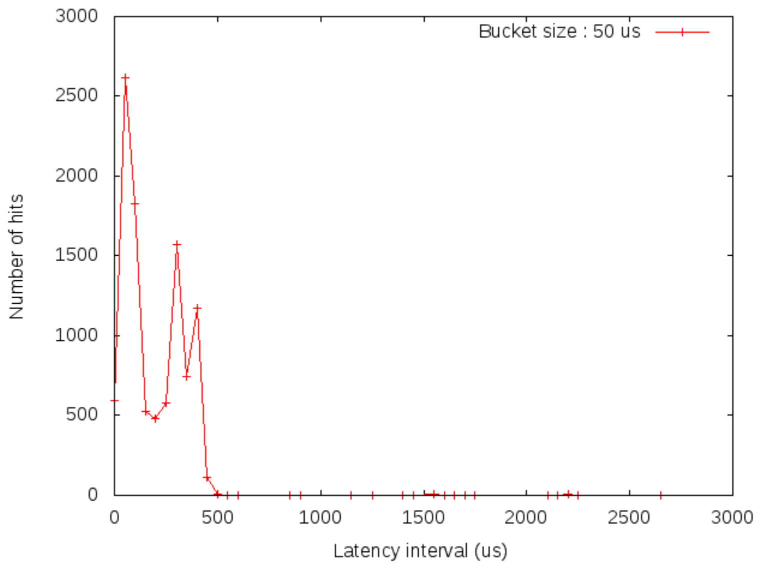


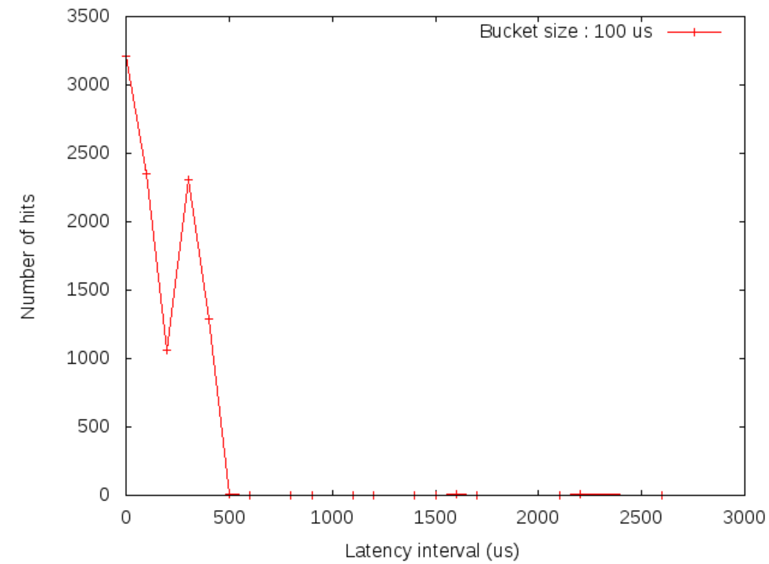


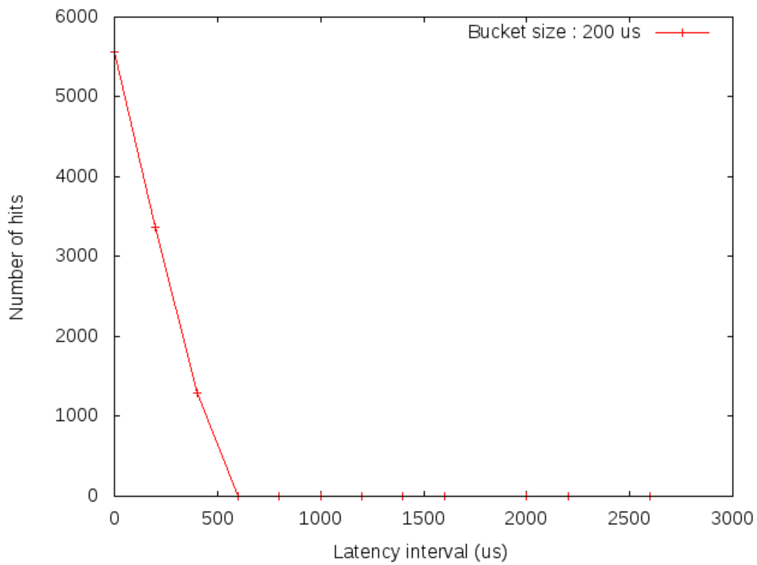


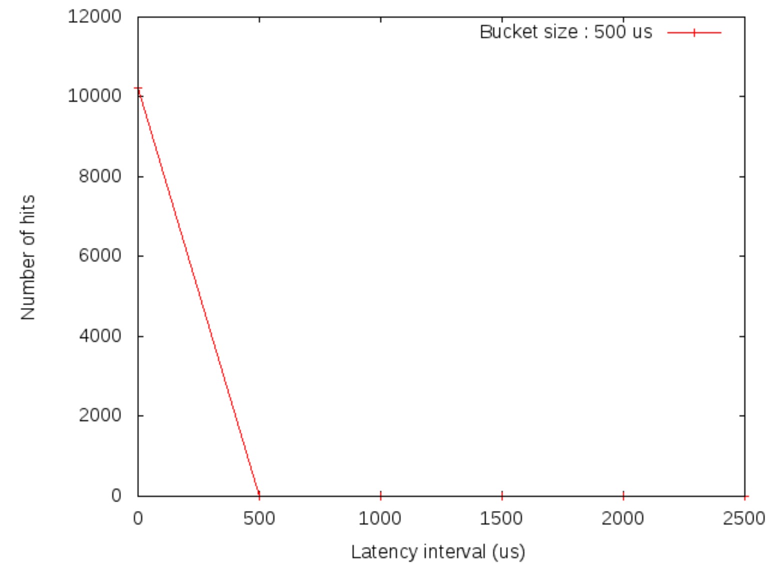


SSD – 8KB Random RW

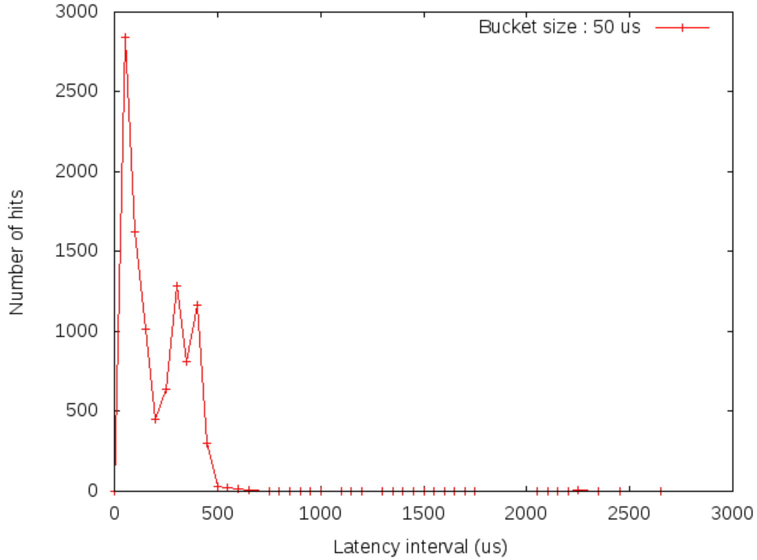


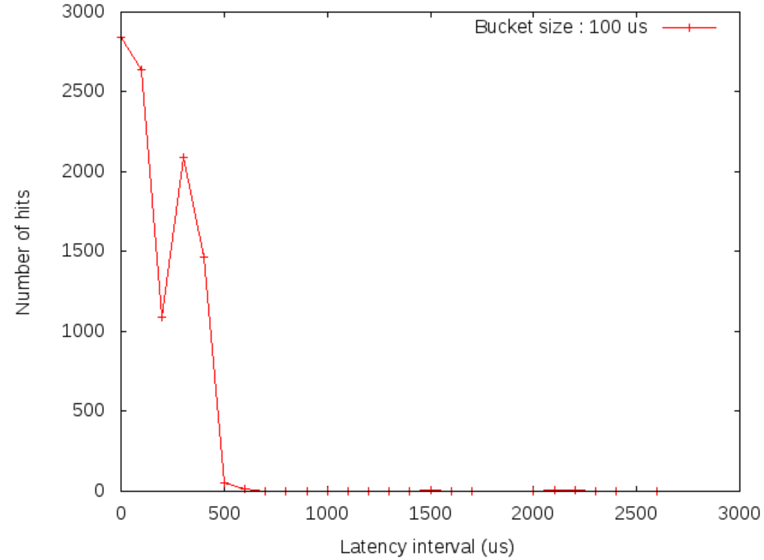


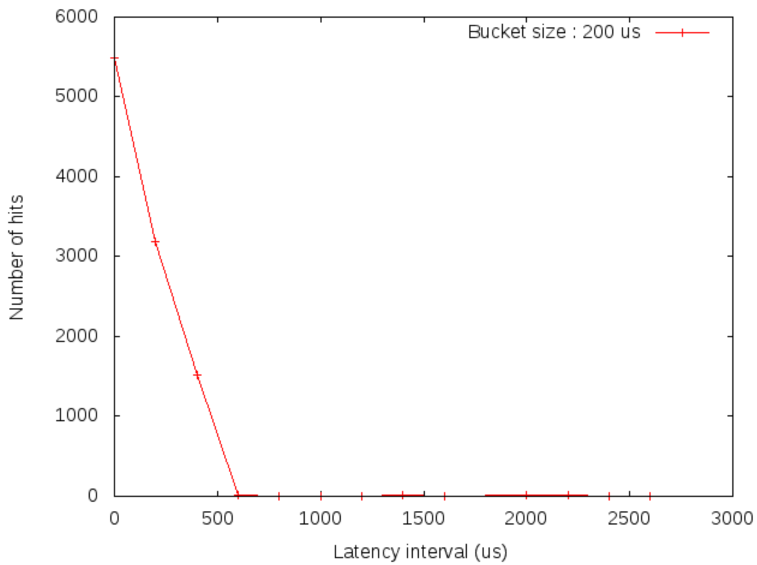


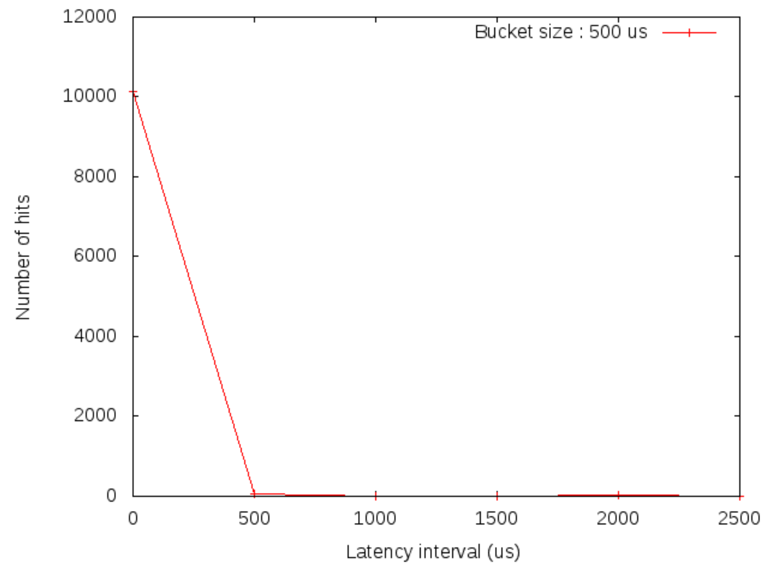


SSD – 16KB Random RW

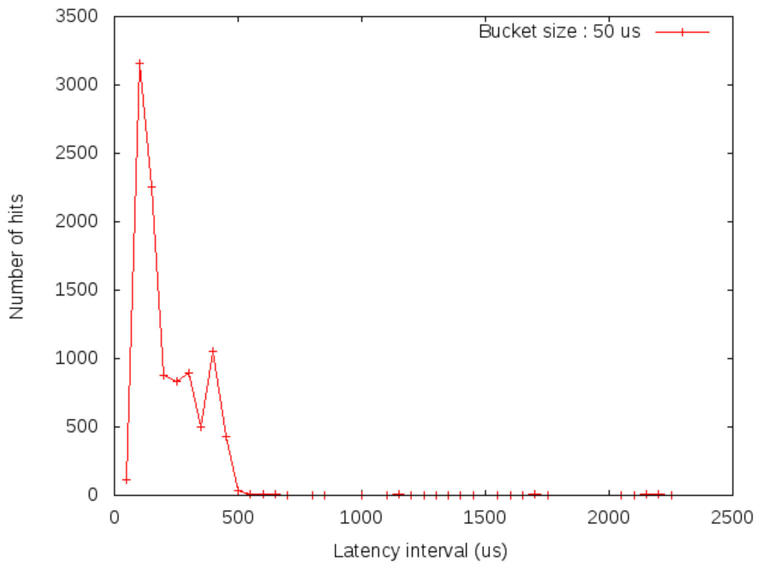


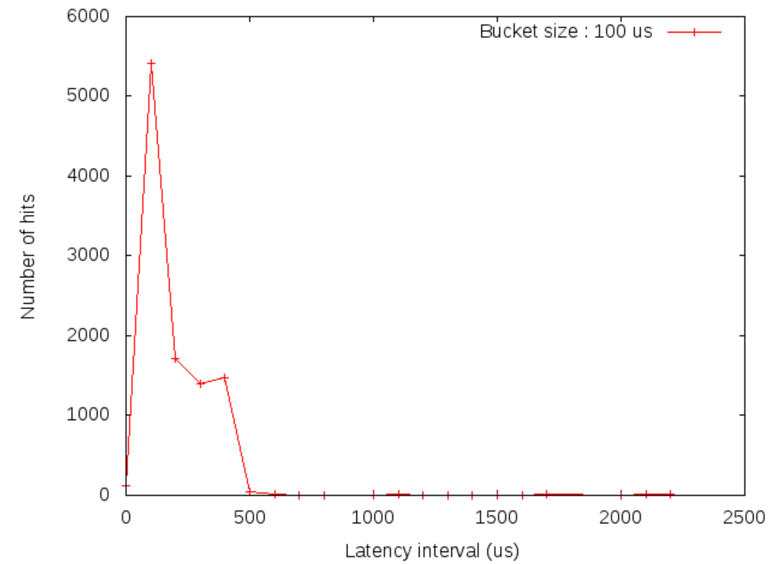


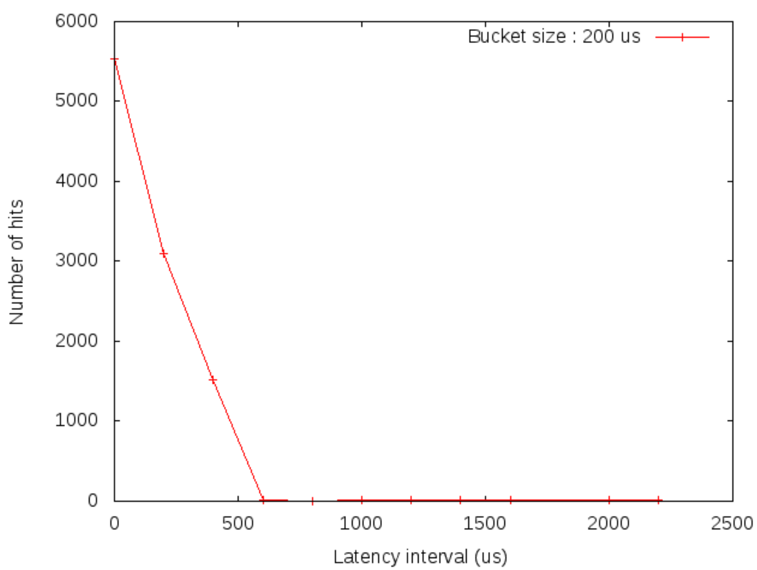


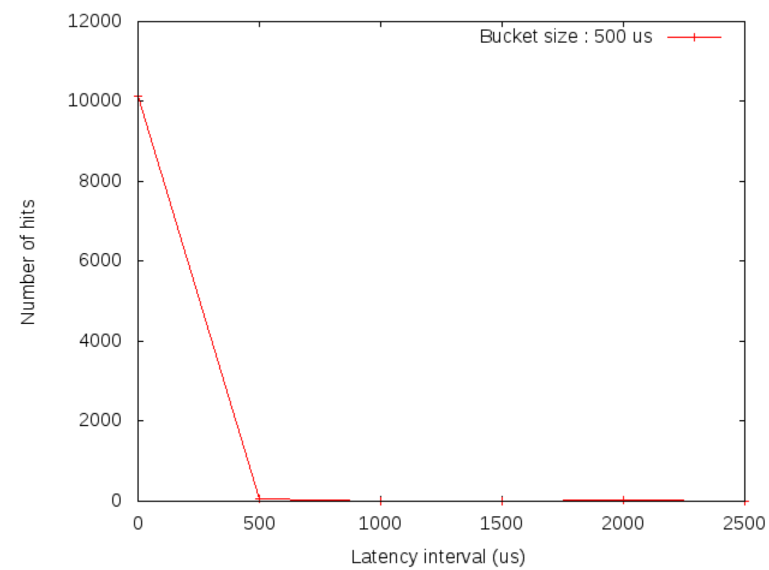


SSD – 32KB Random RW

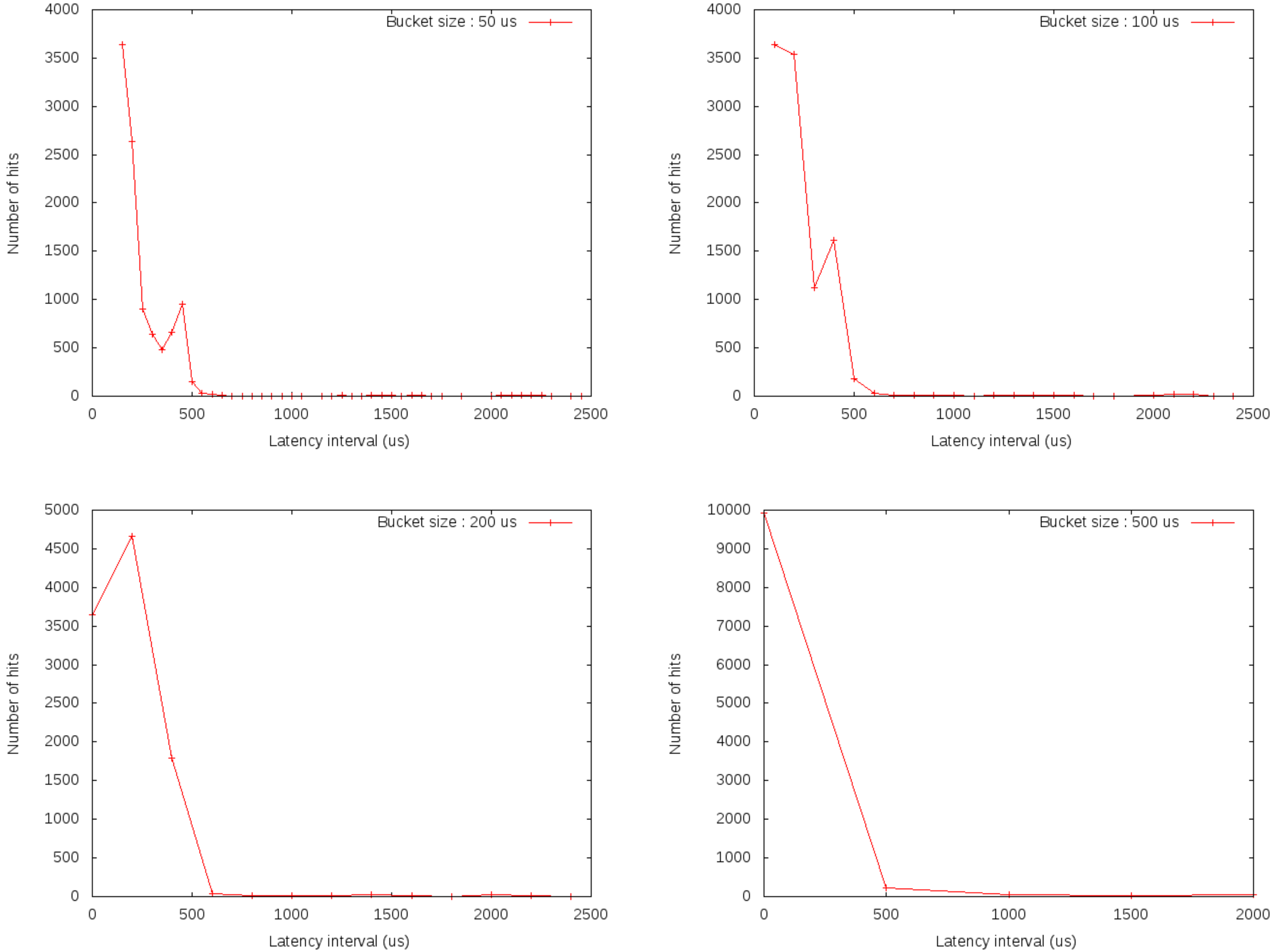




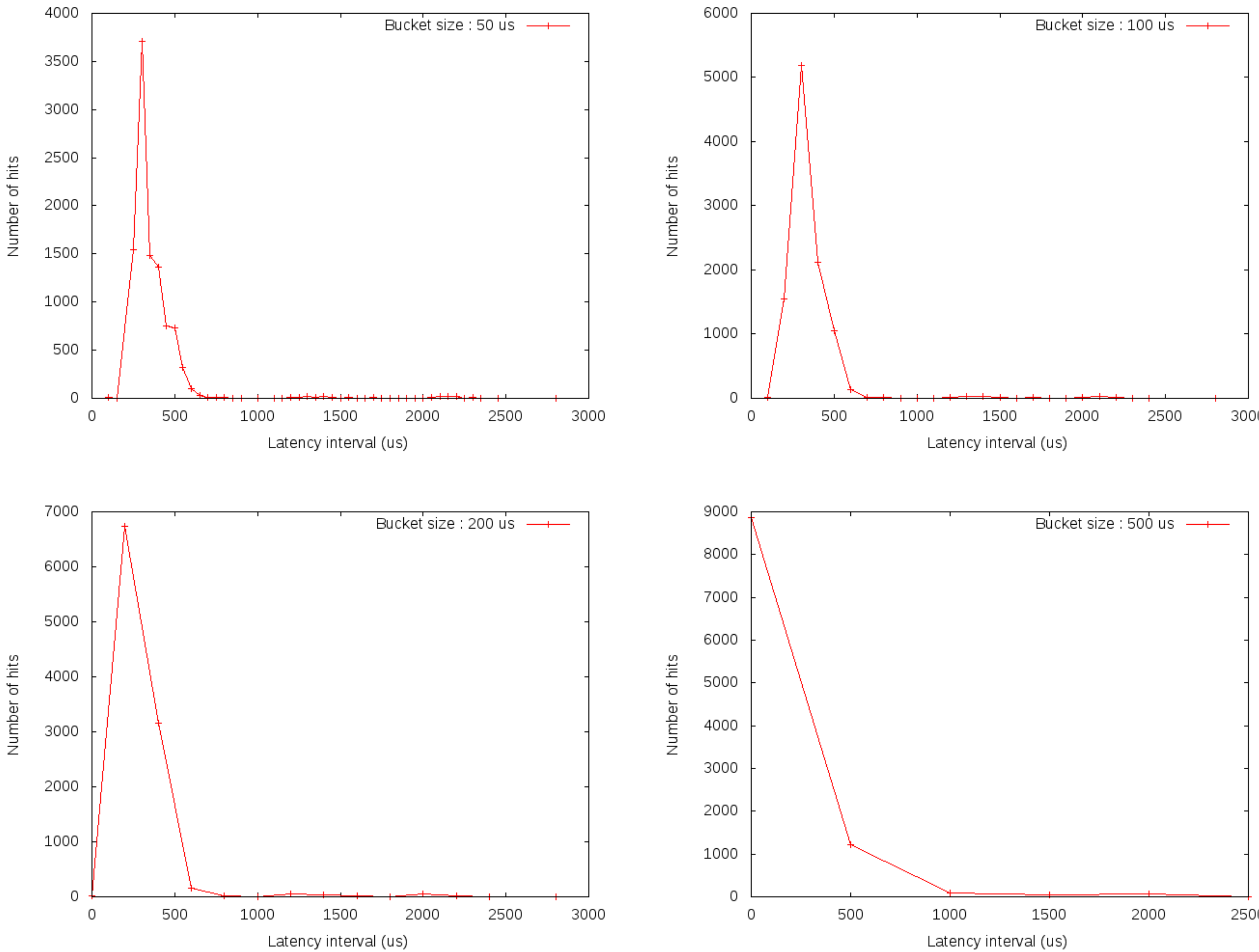




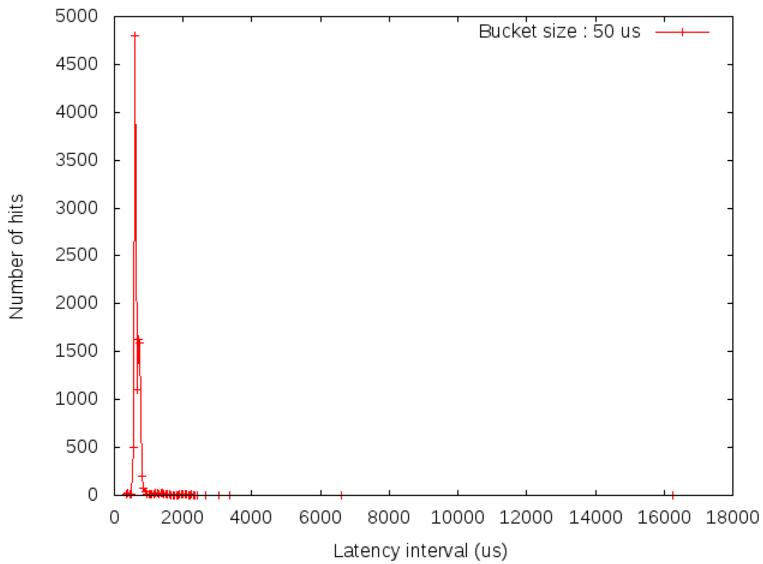
SSD – 64KB Random RW

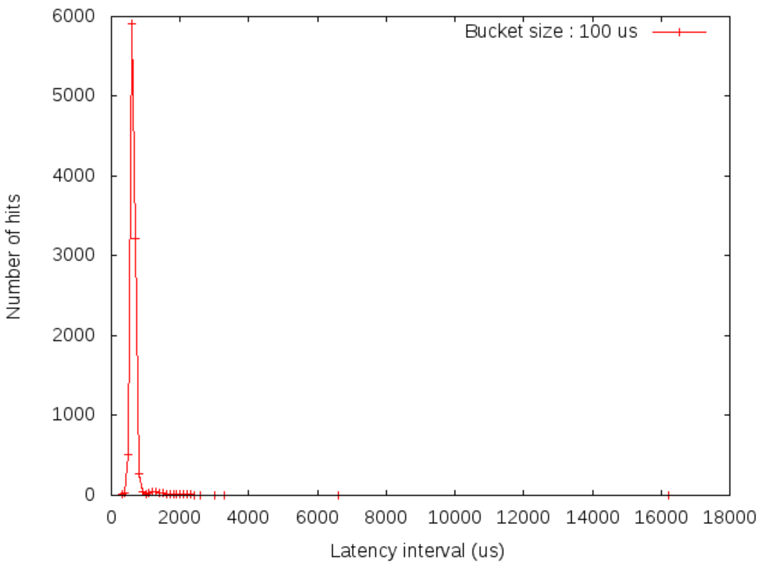


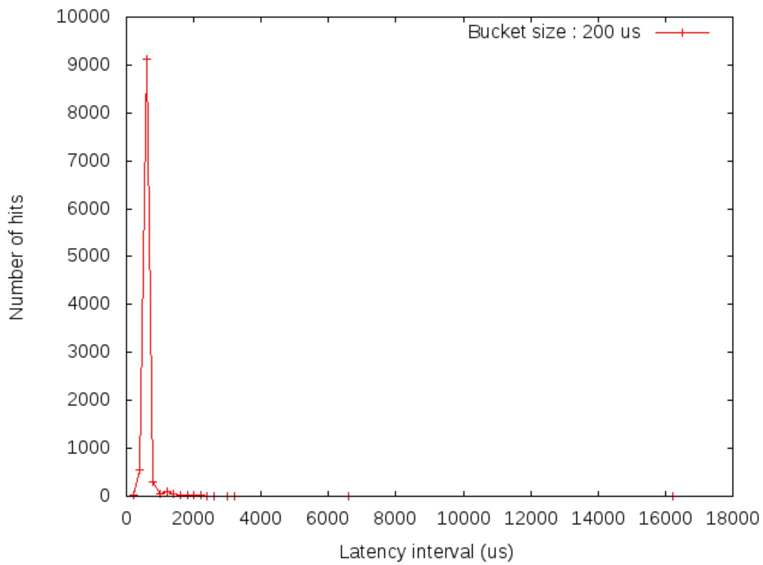
SSD – 128KB Randow RW

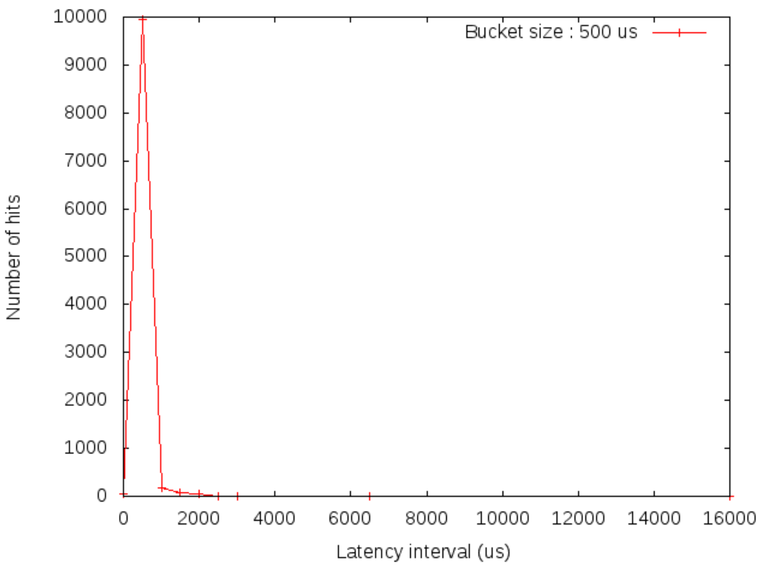


SSD – 256KB Random RW

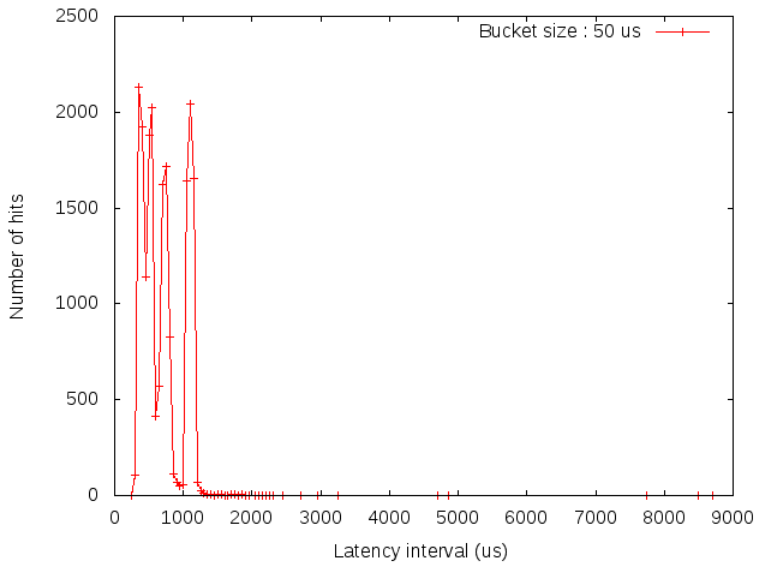


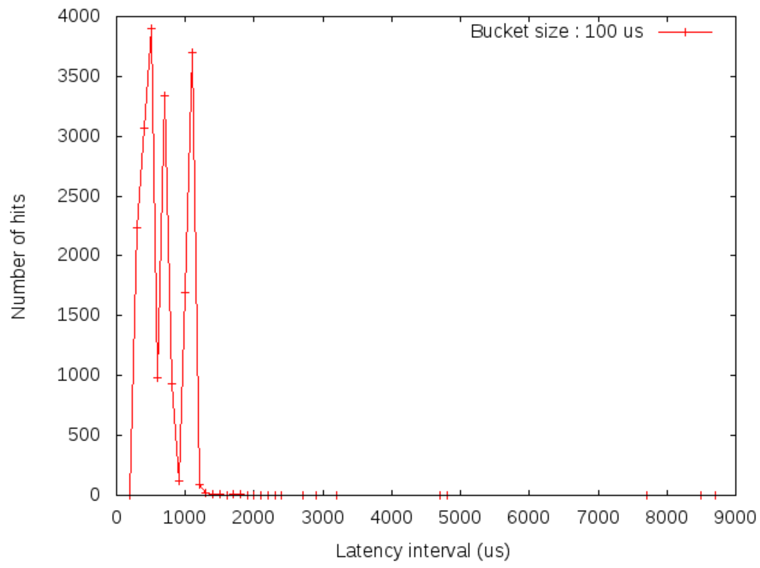


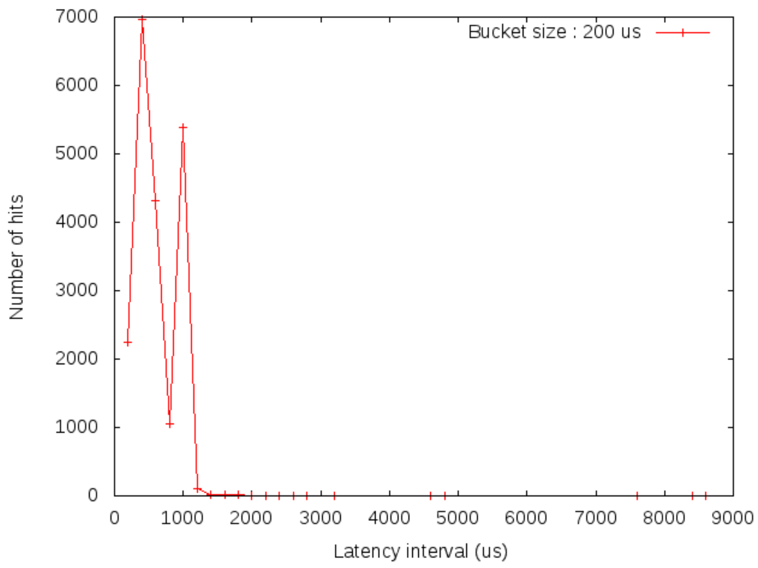


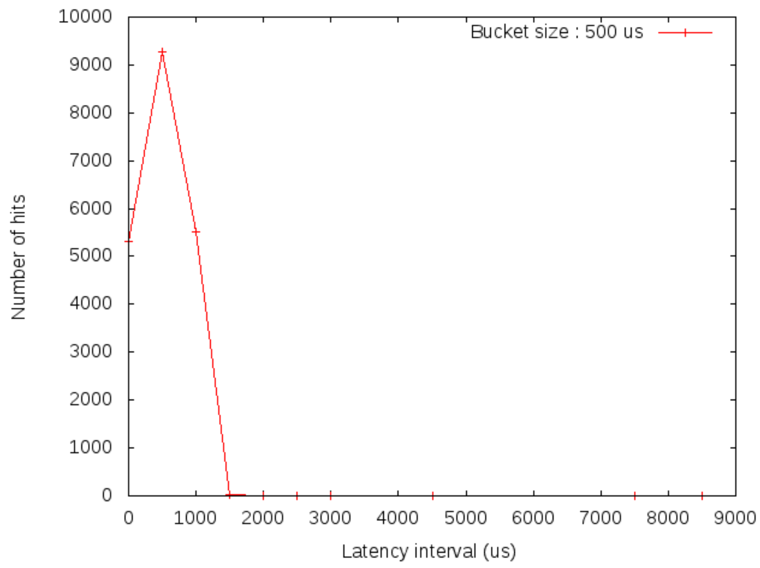


SSD – 512KB Randow RW

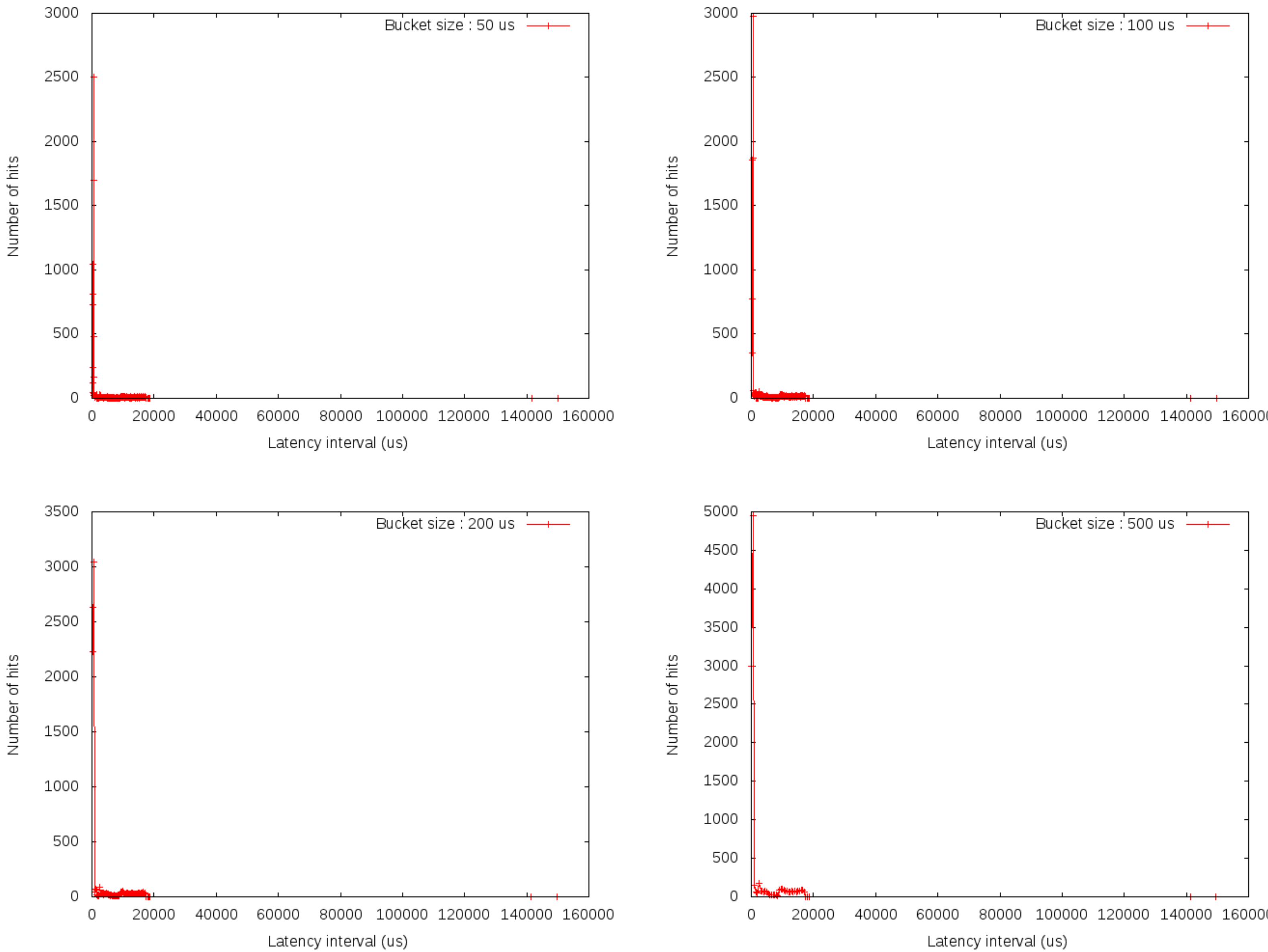




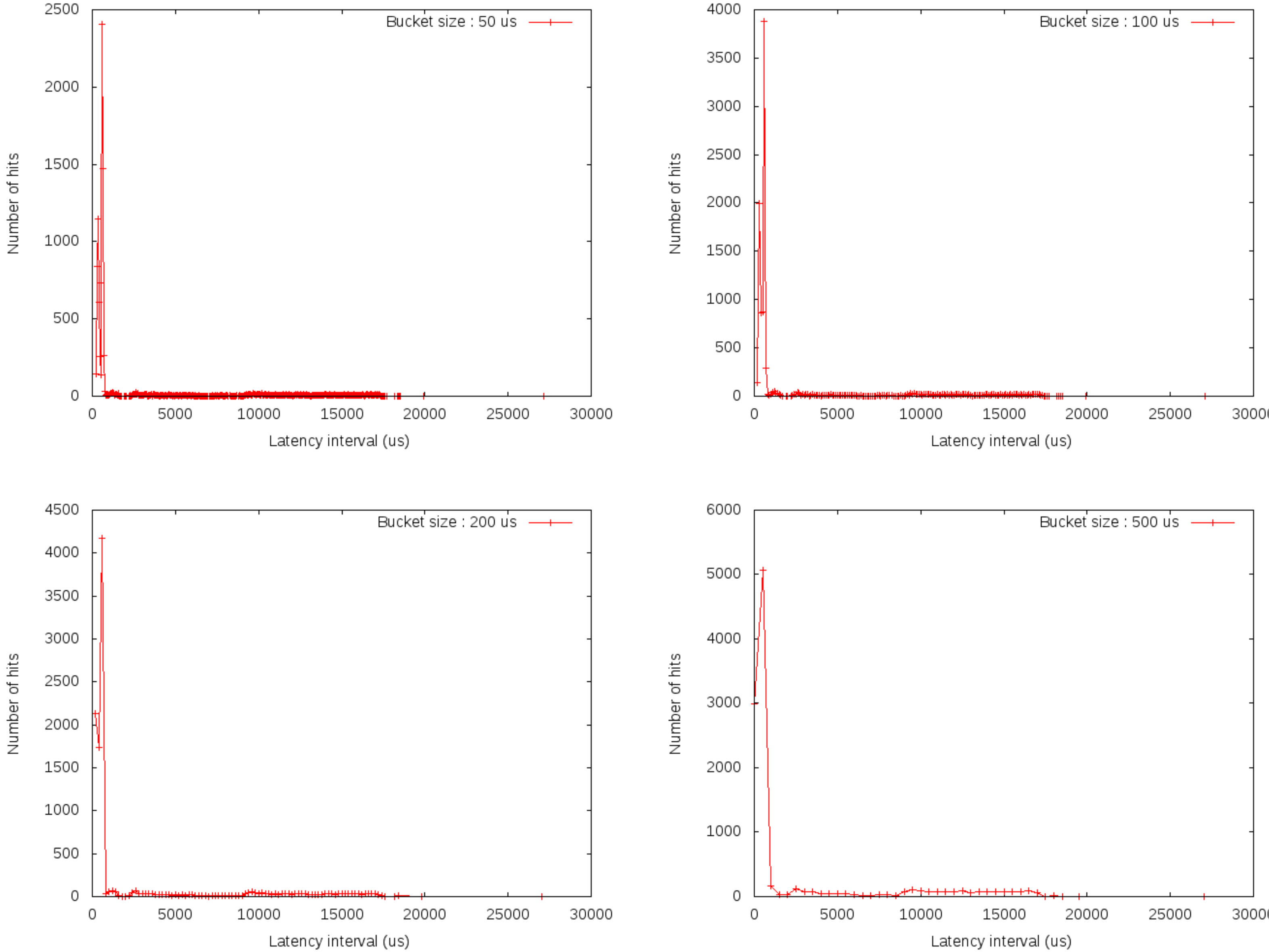




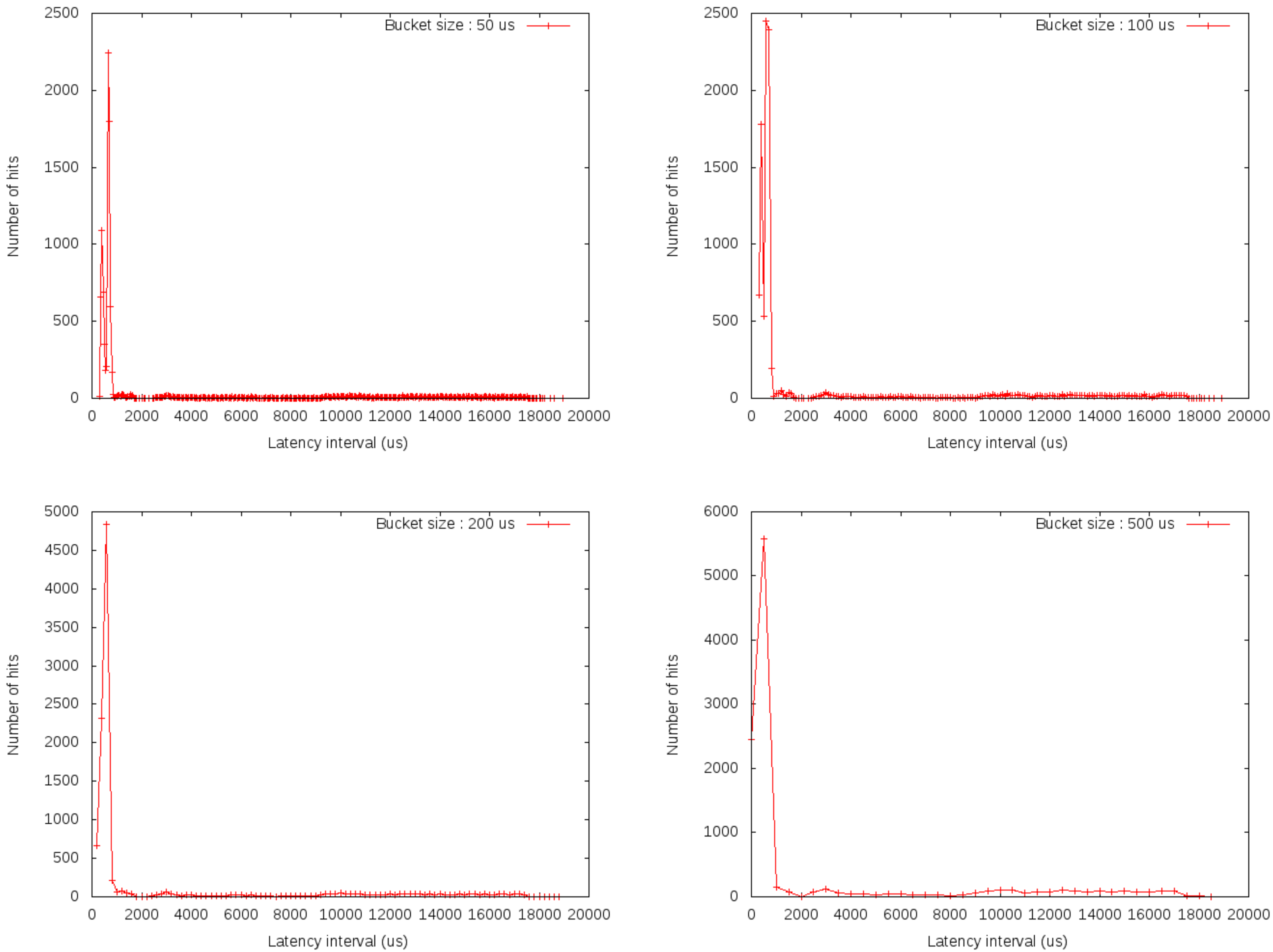
HDD – 8KB Random RW



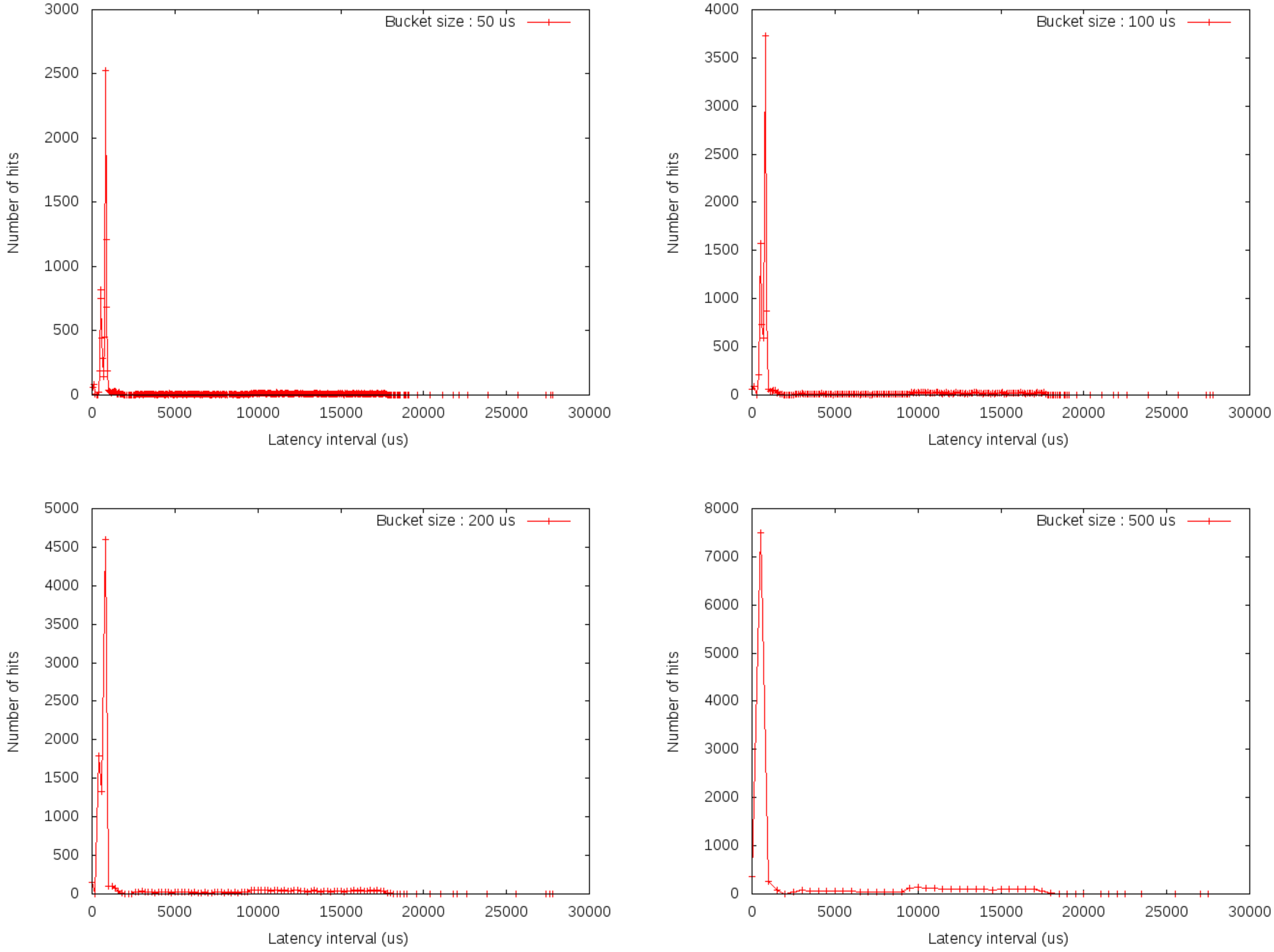
HDD – 16KB Random RW



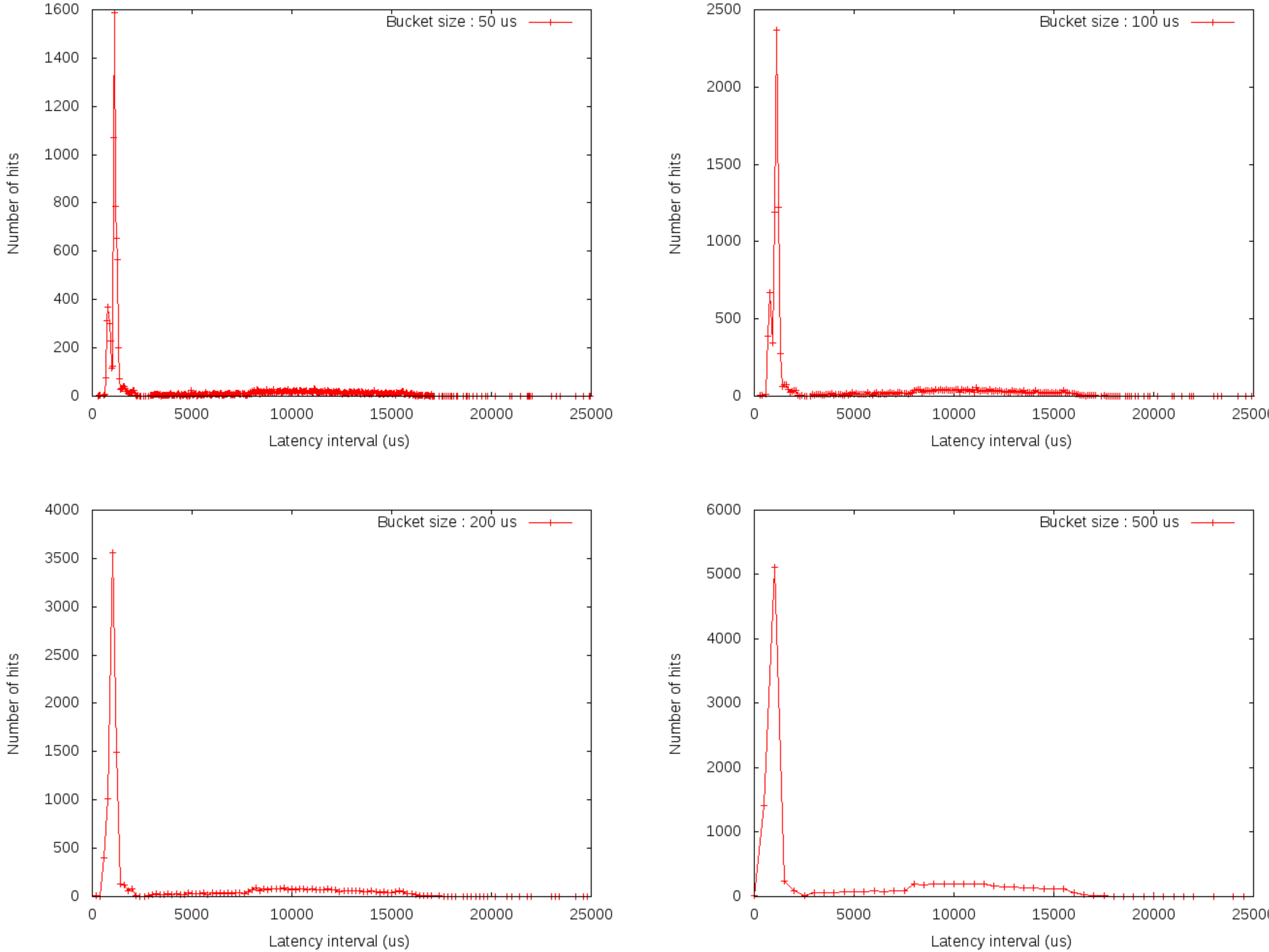
HDD – 32KB Random RW



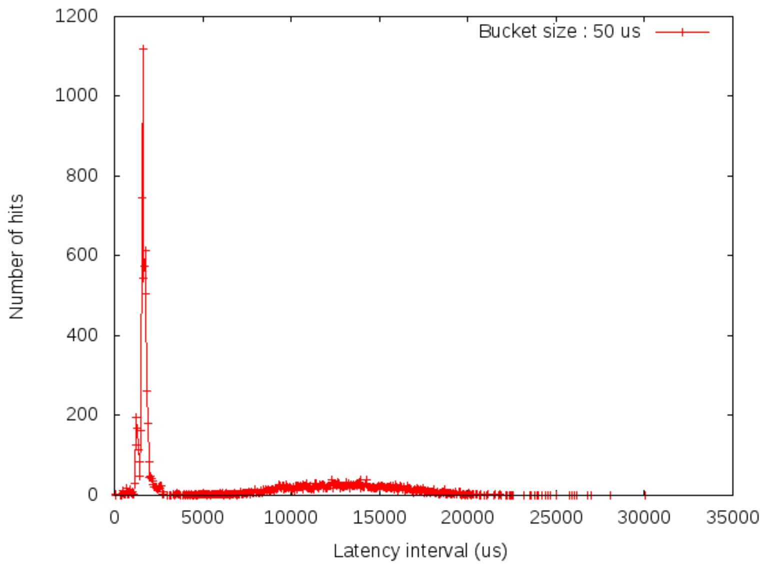
HDD – 64KB Randow RW

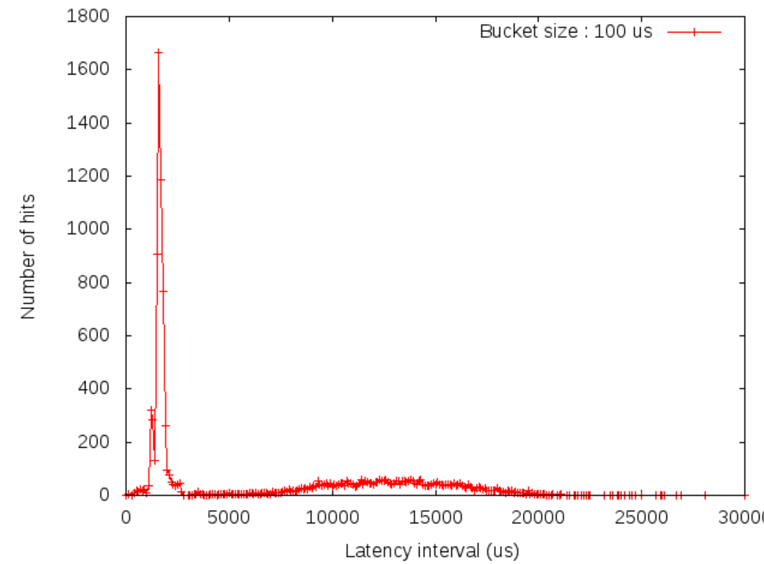


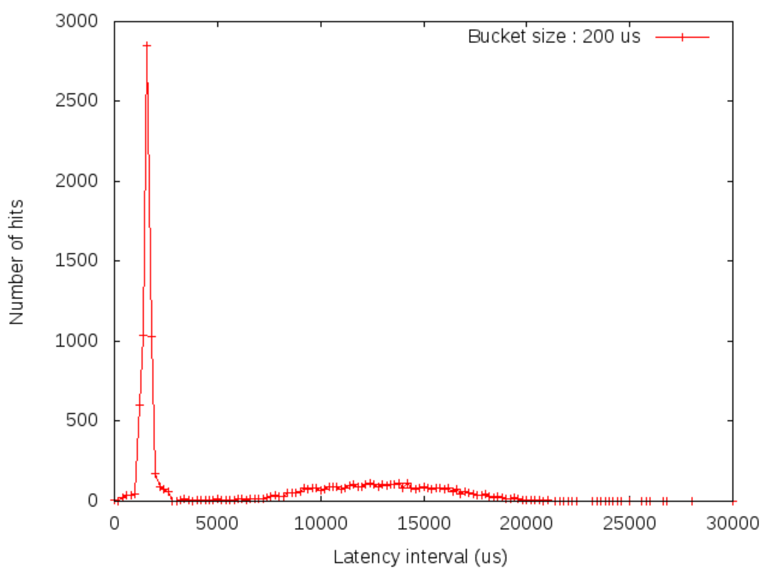
HDD – 128KB Randow RW

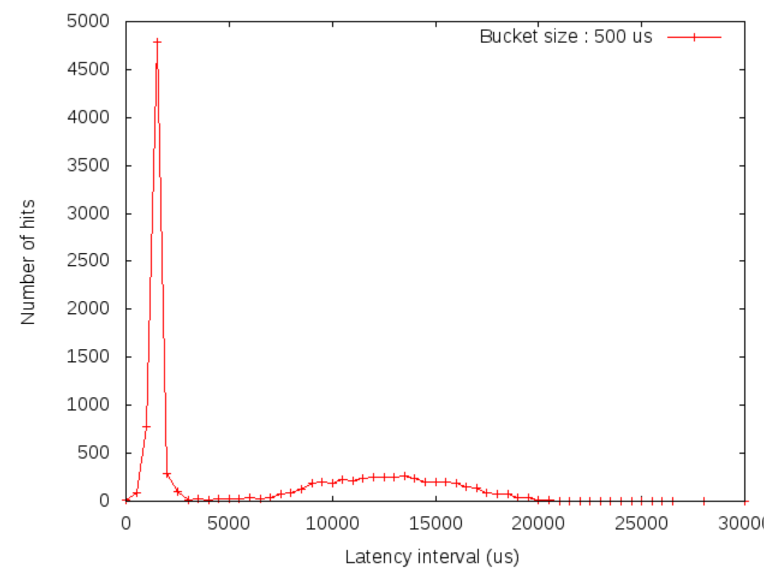


HDD – 256KB Randow RW

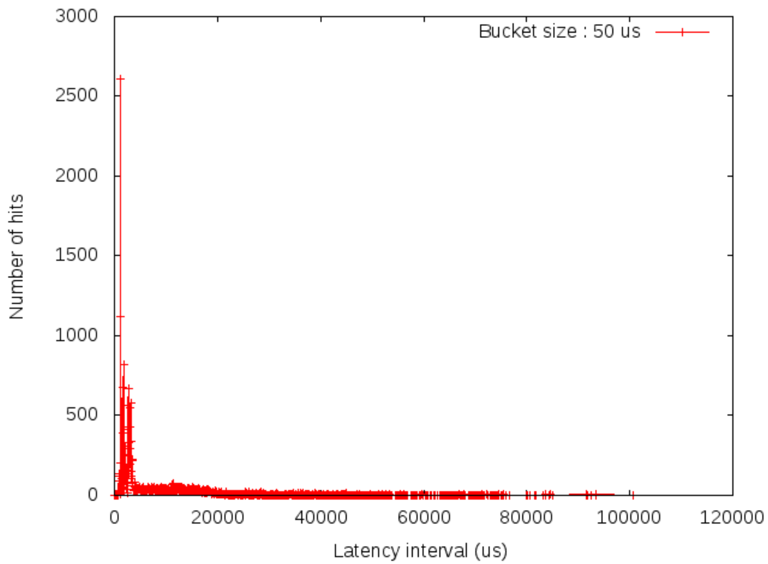


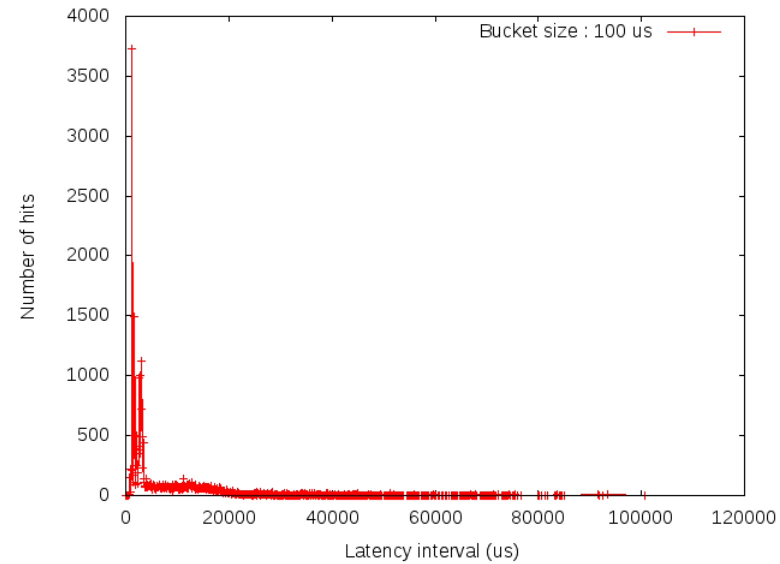


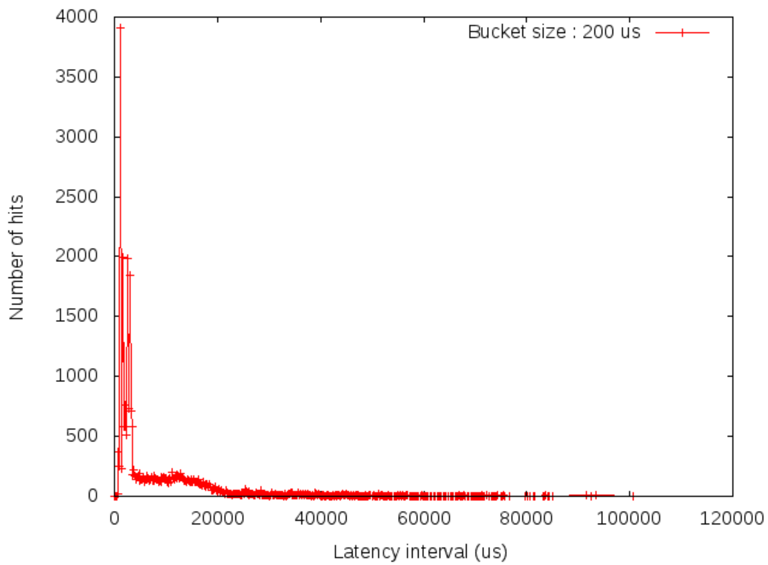


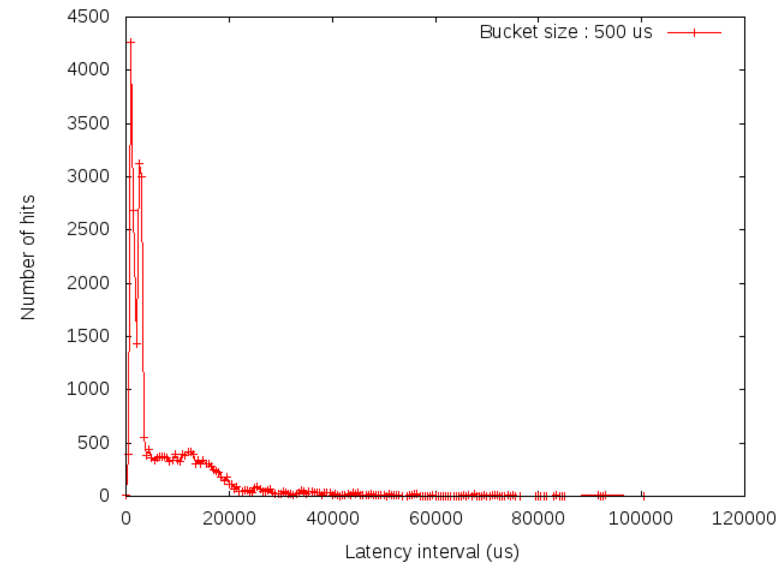


HDD – 512KB Random RW

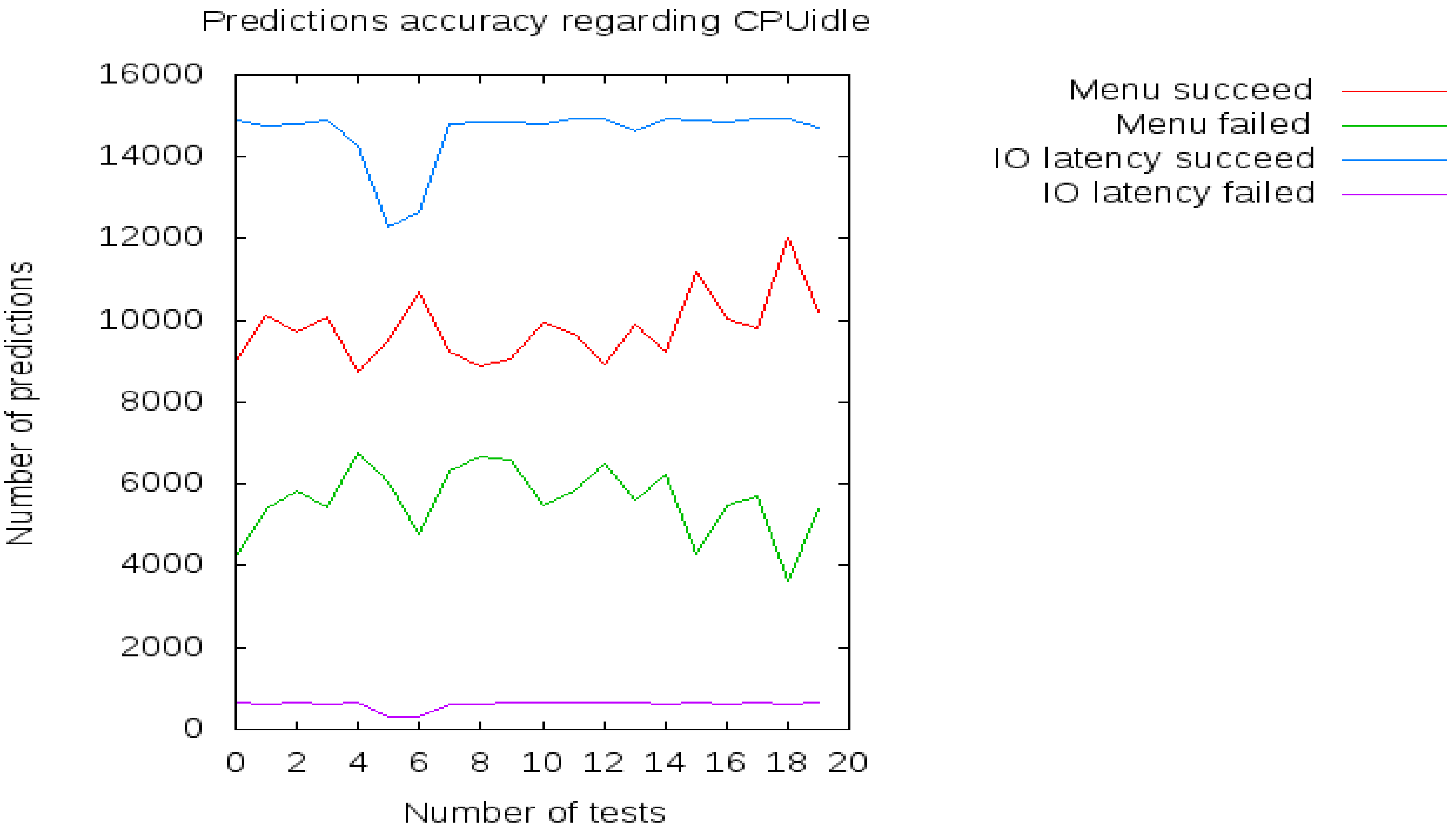




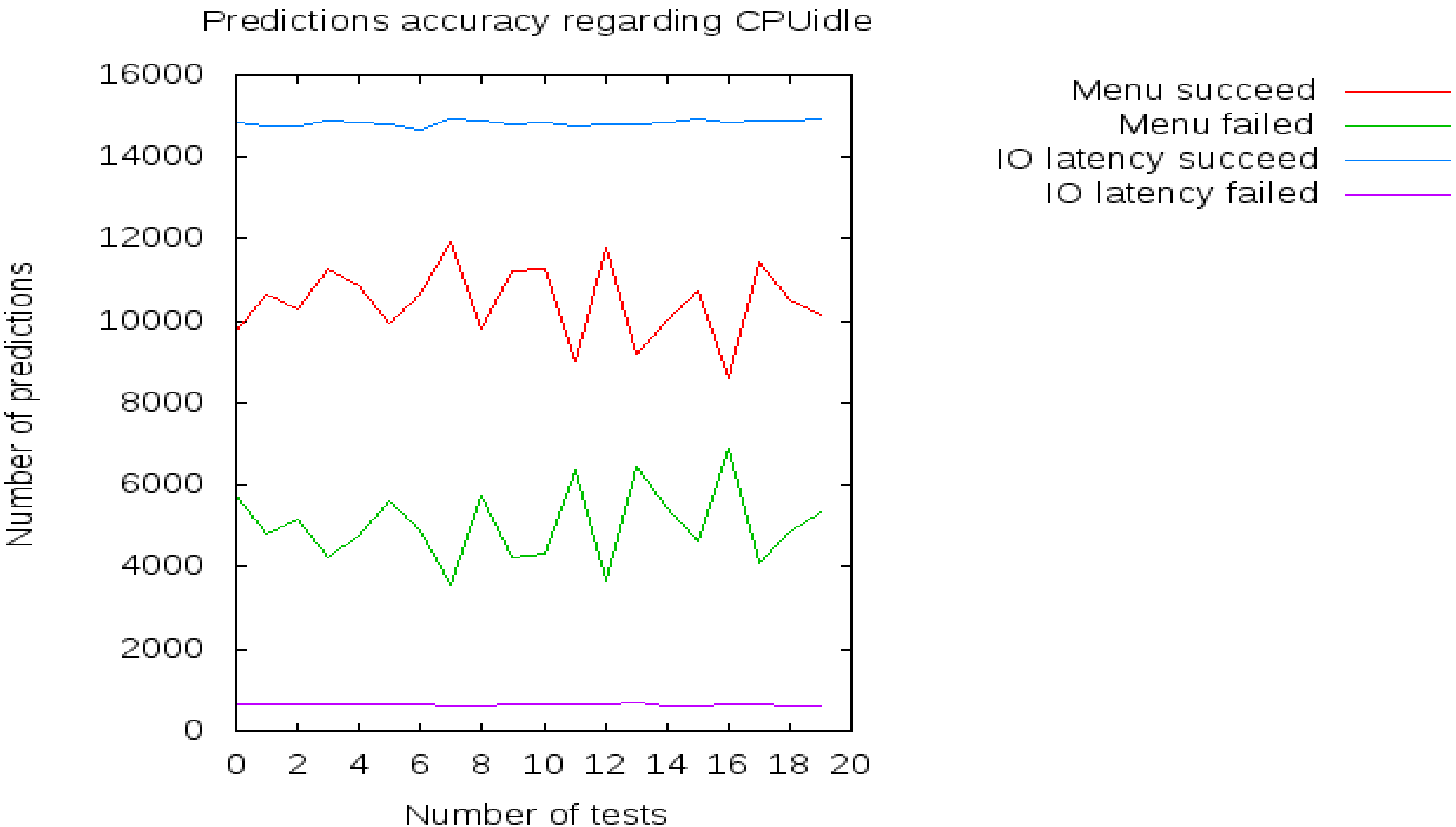




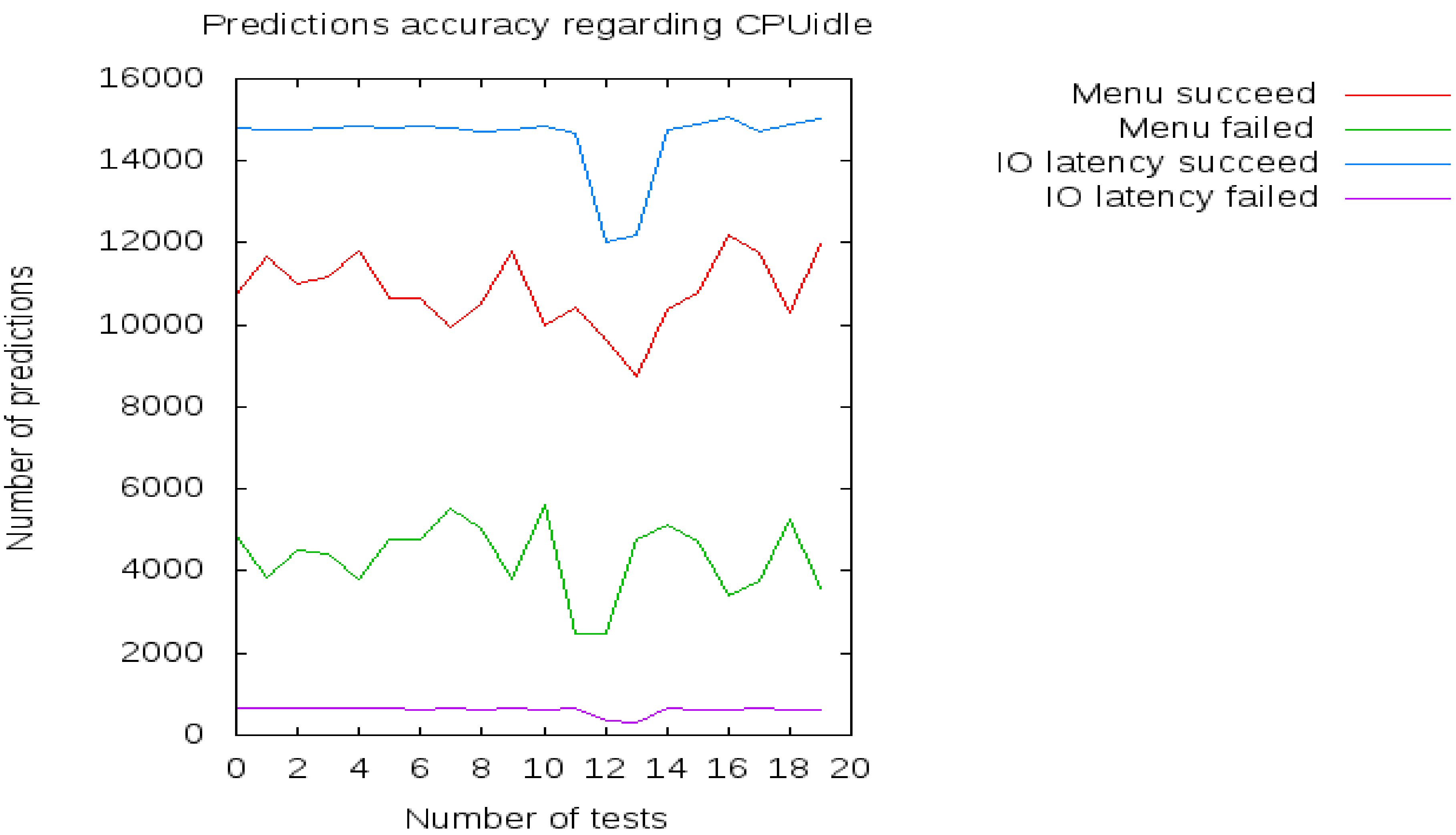
SDD results – 8KB



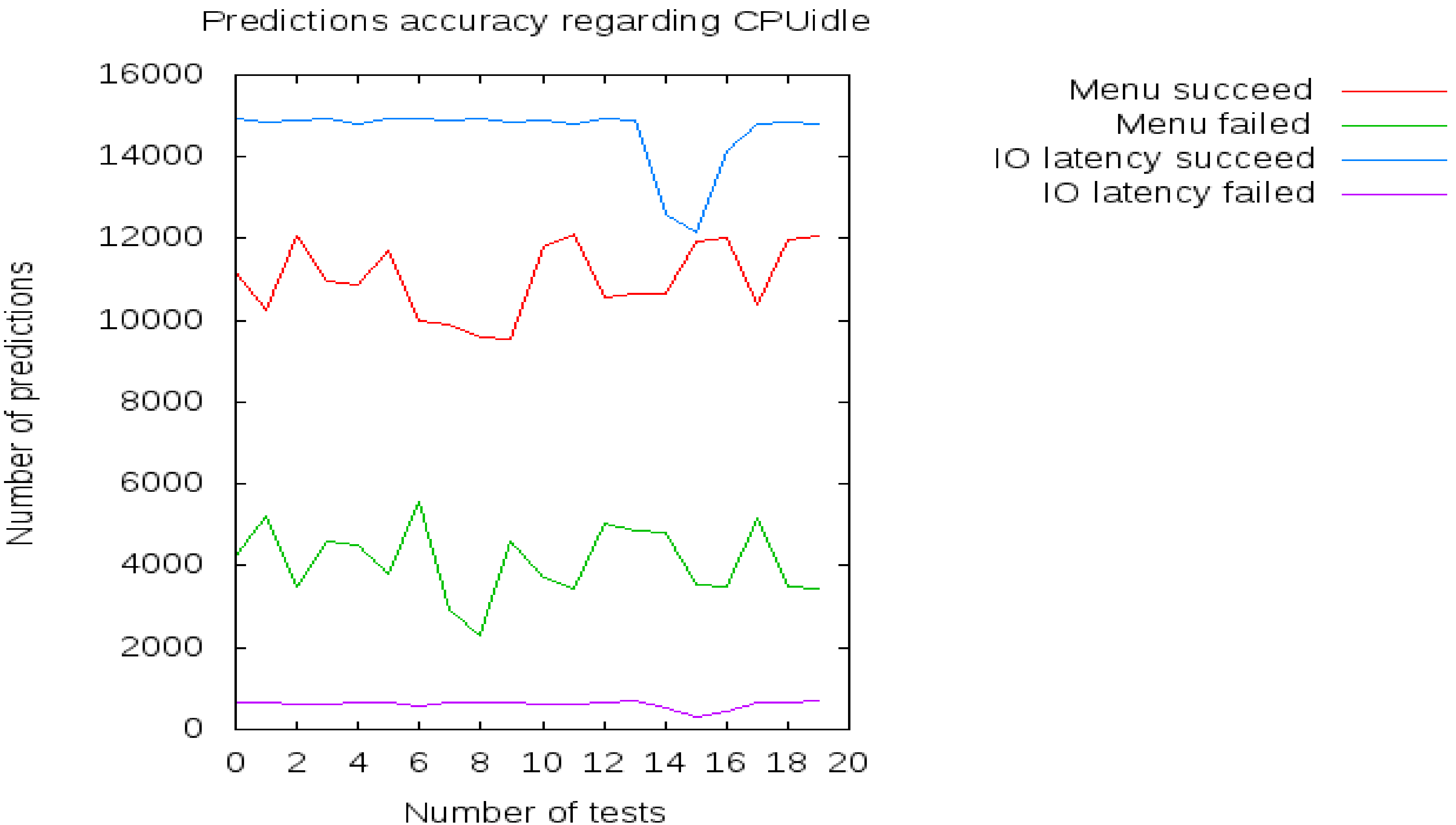
SSD results - 16KB



SSD results - 32KB

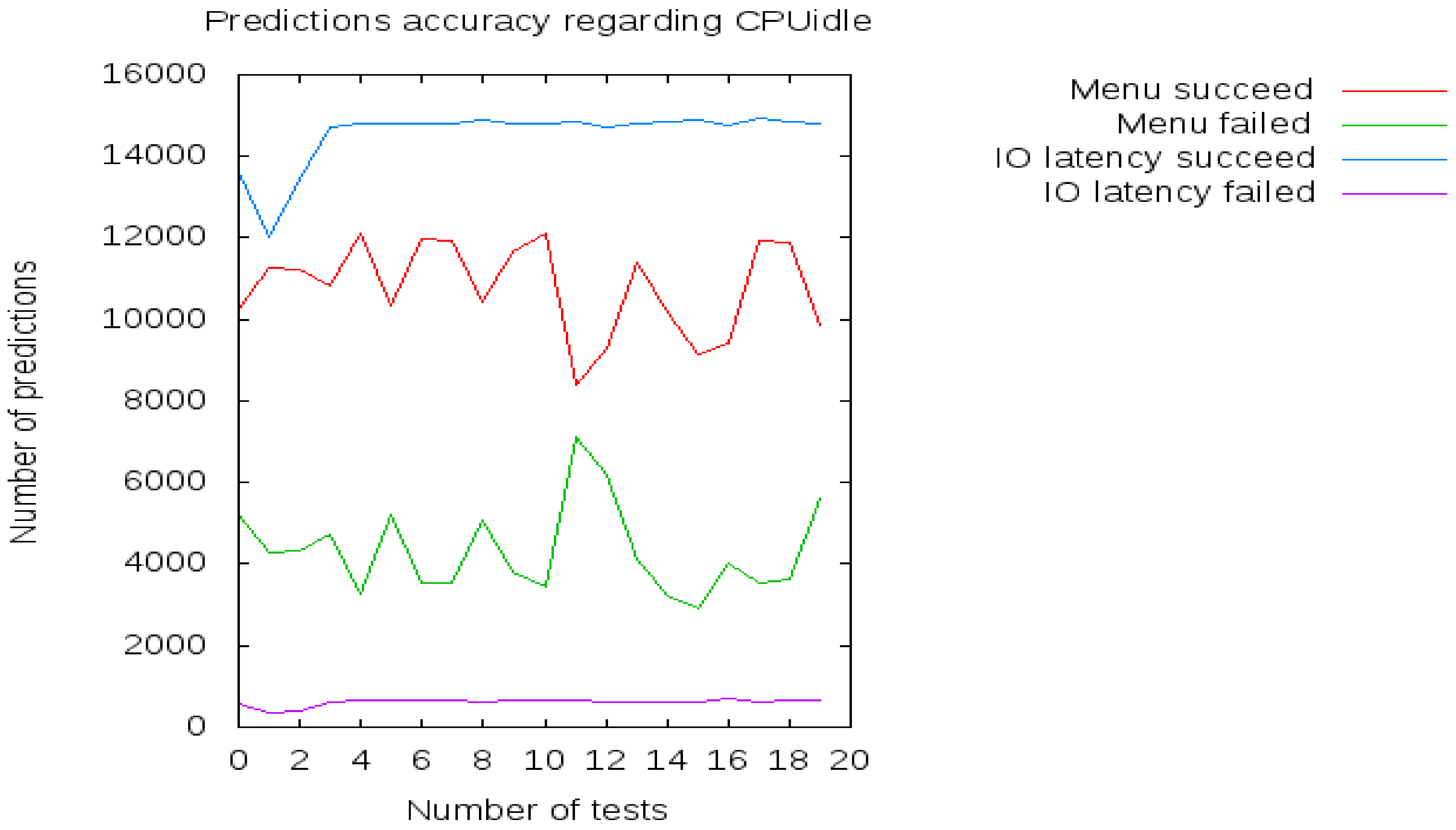


SDD results - 64KB

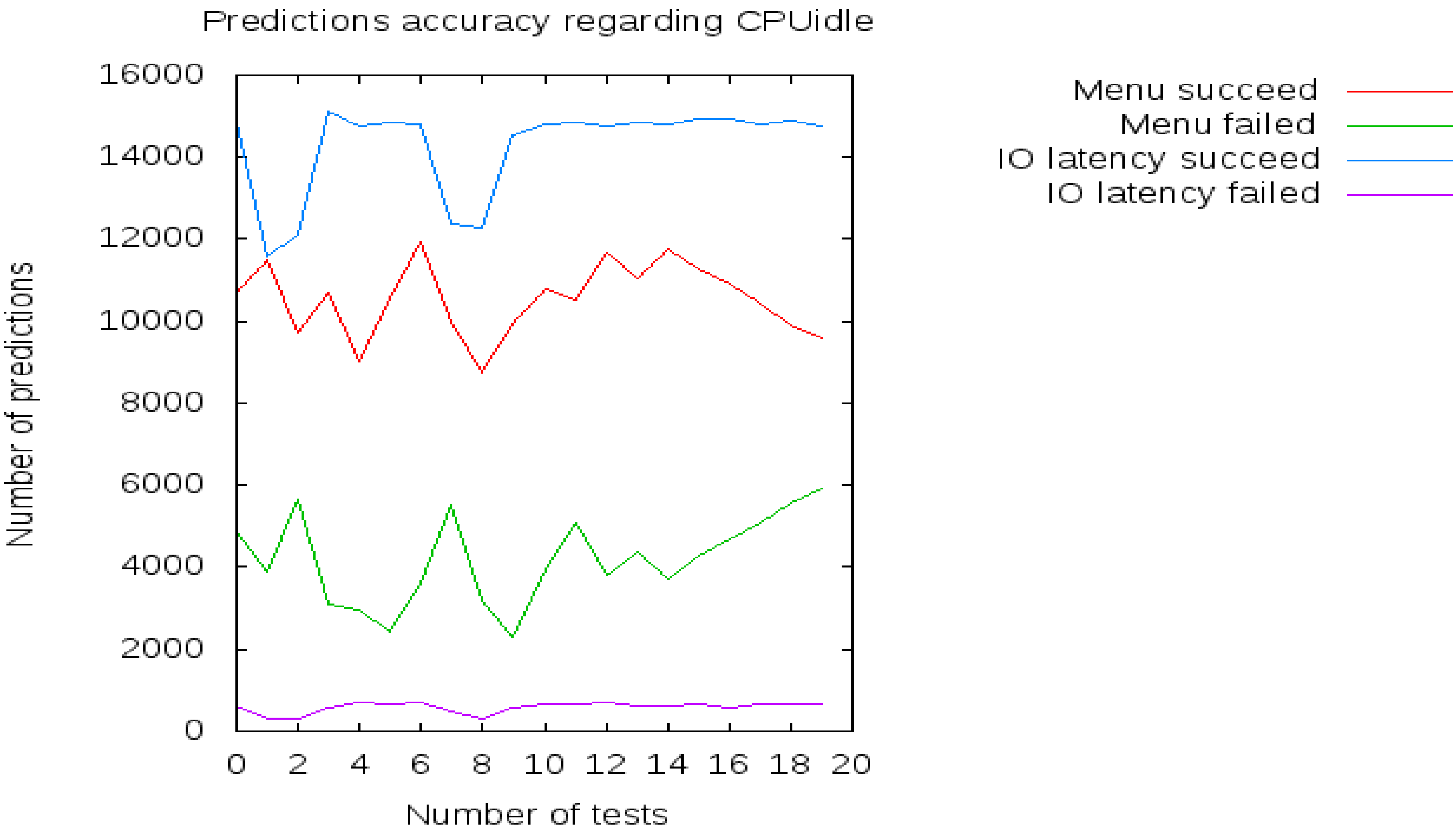


Linaro

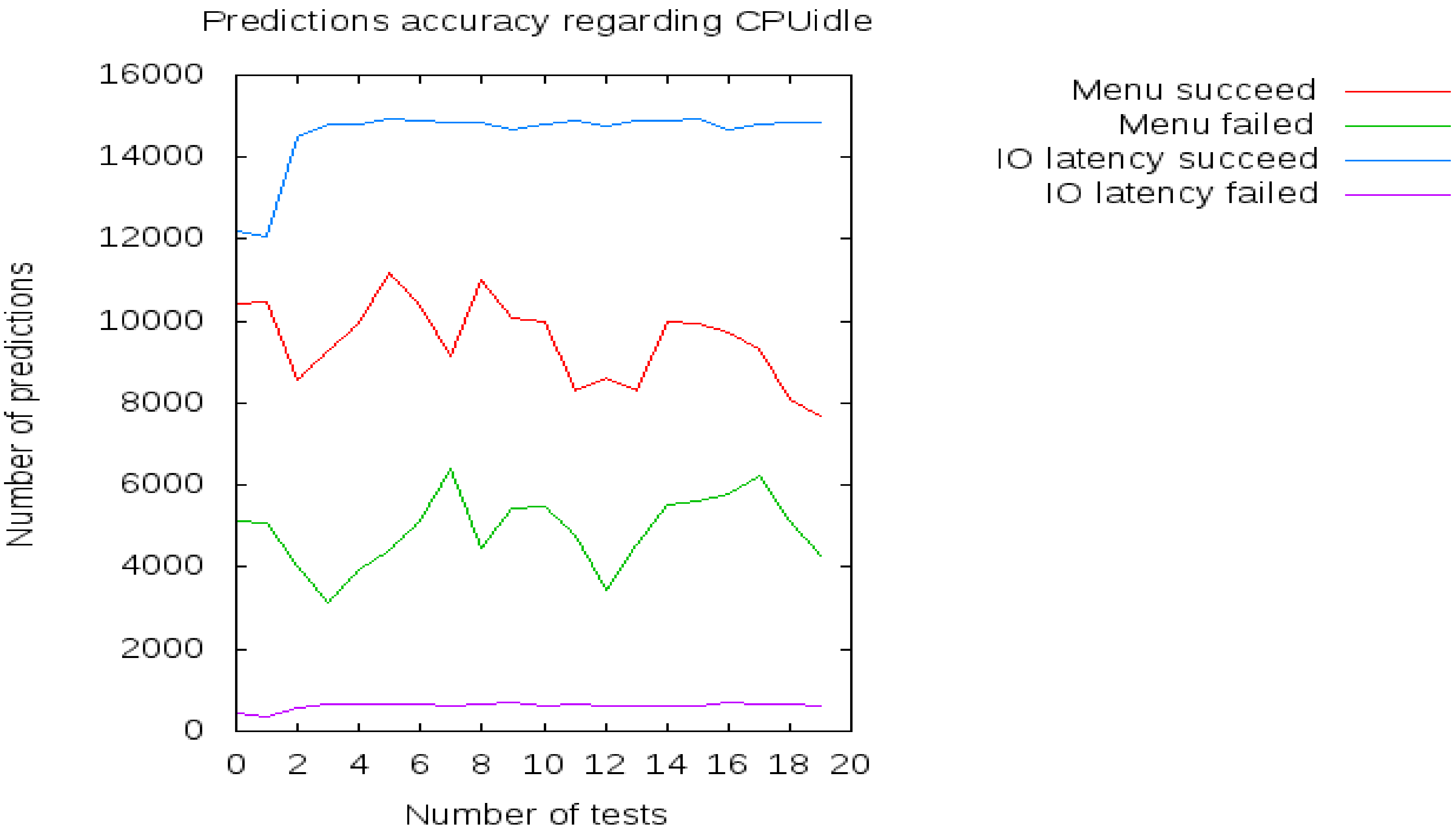
SDD results - 128KB



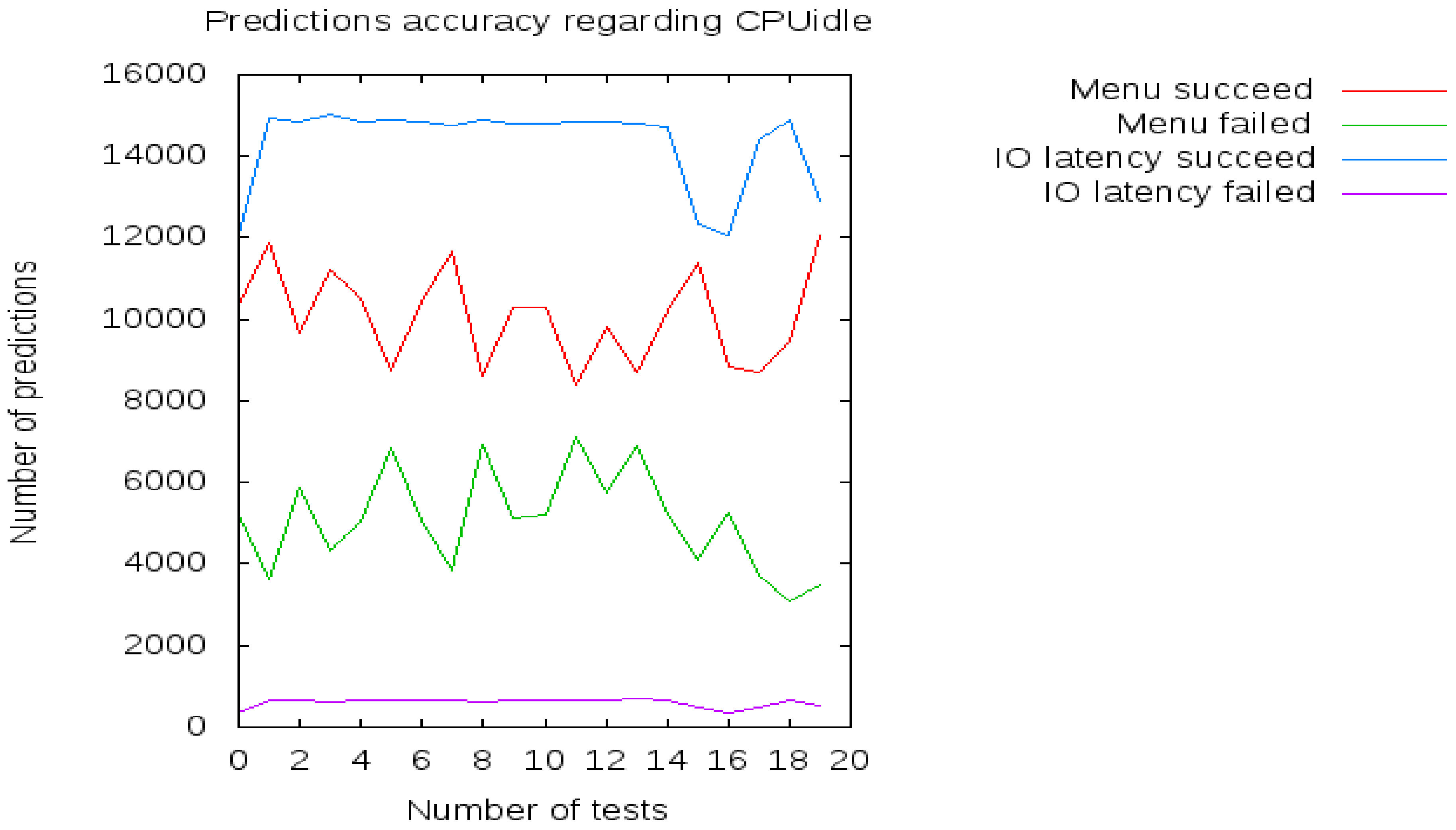
SDD results - 256KB



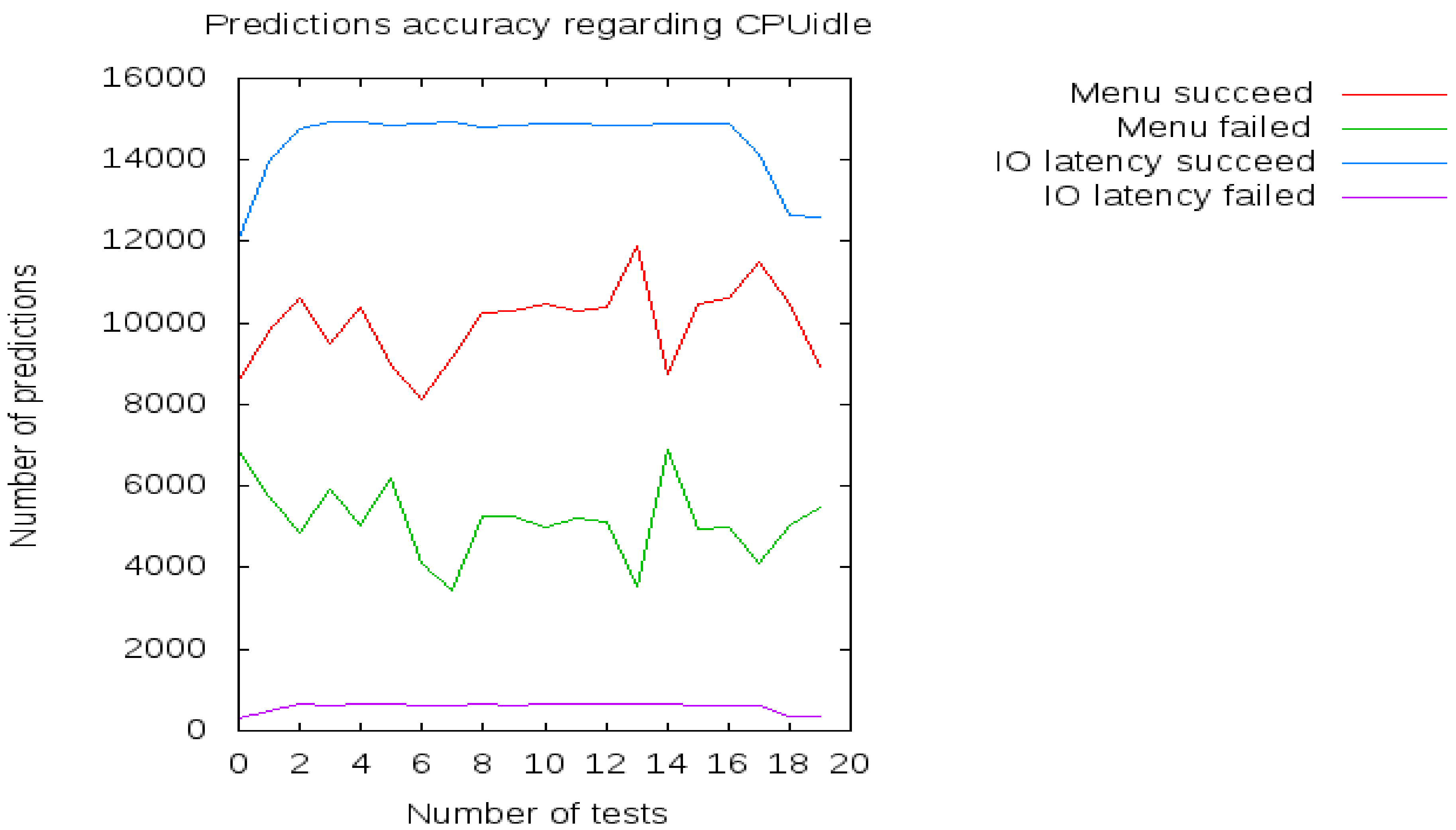
SDD results - 512KB



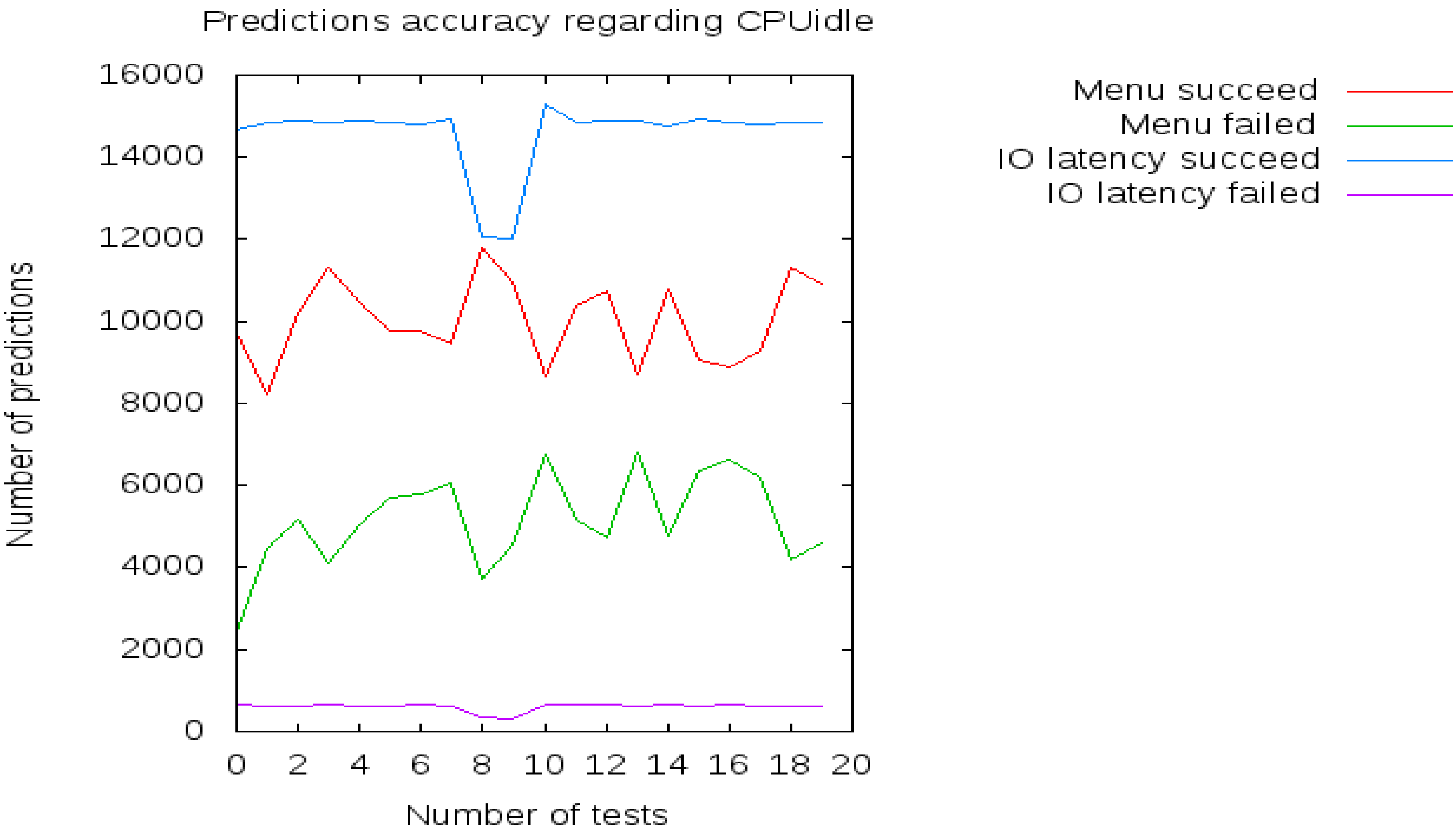
HDD results - 8KB



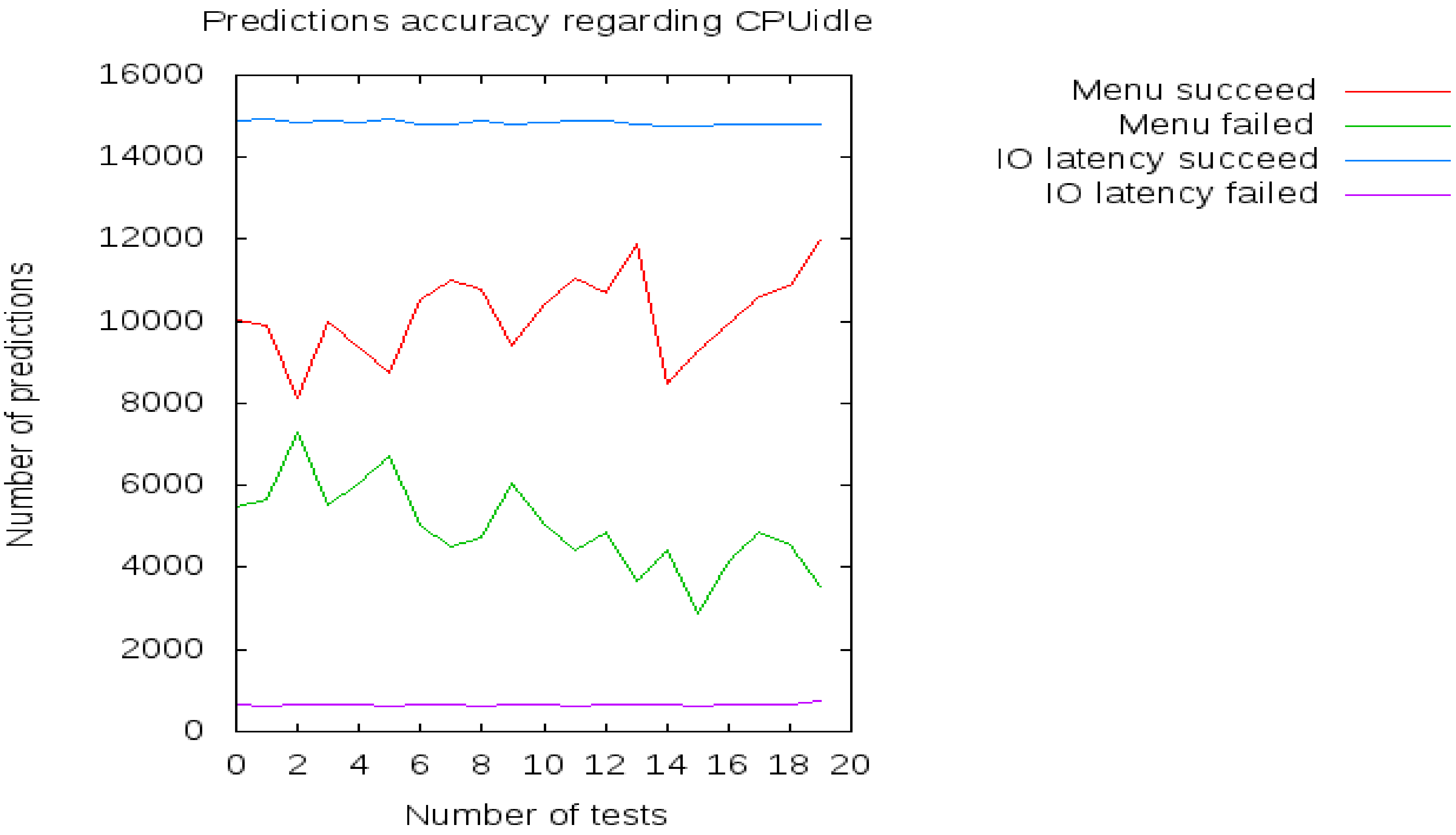
HDD results - 16KB



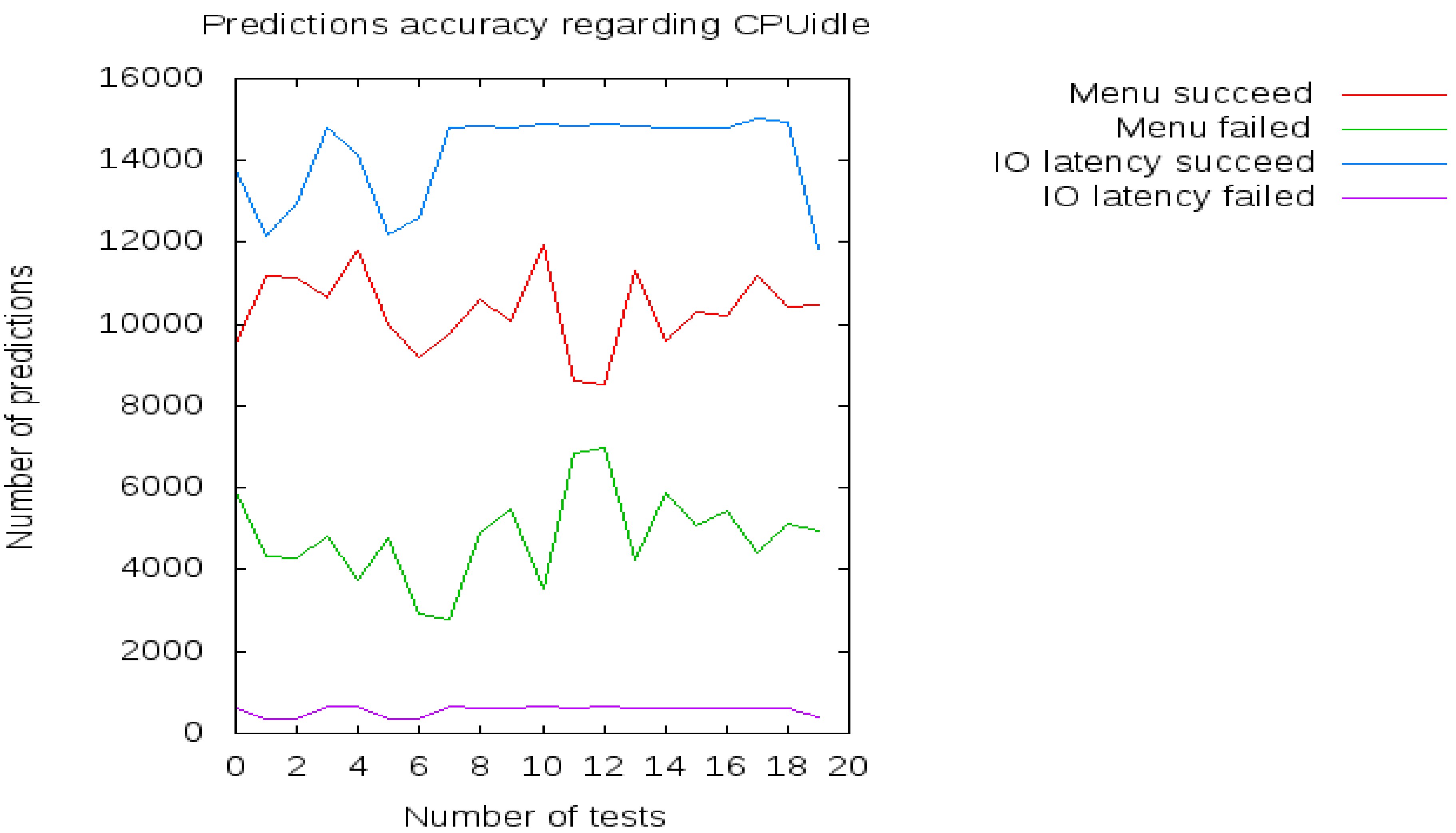
HDD results - 32KB



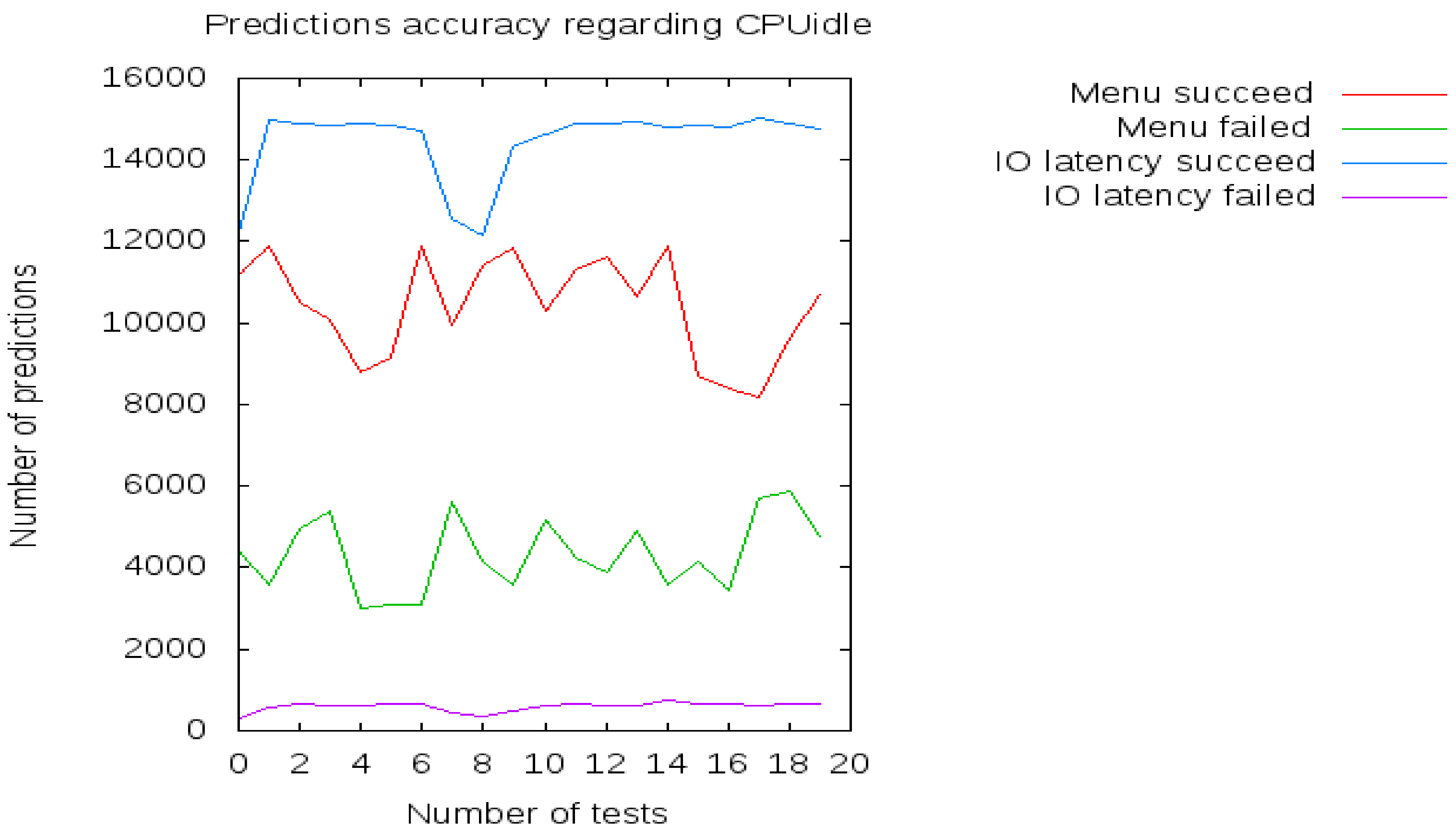
HDD results - 64KB



HDD results - 128KB



HDD results - 256KB



HDD results - 512KB