TOSHIBA

Leading Innovation >>>

# Real-Time Task Partitioning using Cgroups

**Akihiro SUZUKI**

Advanced Software Technology Group
Corporate Software Engineering Center
TOSHIBA CORPORATION

2013/06/07

# Self-Introduction

- **Name**
  - Akihiro SUZUKI
- **Company**
  - TOSHIBA
    - Corporate Software Engineering Center
- **Job**
  - Embedded systems using Linux

# Contents

- Background
- Introduction to Cgroups
- Use cases
- Evaluation
- Discussion
- Conclusions

# Contents

- Background
- Introduction to Cgroups
- Use cases
- Evaluation
- Discussion
- Conclusions

# Background

- **Real-time tasks and general-purpose tasks running on the same system**
  - Real-time task
    - The task that must finish a specific processing within fixed time

- **Real-time tasks should be able to use resources anytime within strict time constraints**
  - Partition any resources and assign them to real-time tasks

- **Cgroups (Control Groups) can control several resources for groups of tasks**
  - Cgroups can partition several resources for real-time tasks

# Contents

- Background
- Introduction to Cgroups
- Use cases
- Evaluation
- Discussion
- Conclusions

# What are Cgroups?

- **Control Groups provide a mechanism for aggregating/partitioning sets of tasks, and all their future children, into hierarchical groups with specialized behavior. (Documentation/cgroups/cgroups.txt)**

# How to use Cgroups

- **Enable Cgroups in the kernel config file**

```
CONFIG_CGROUPS=y
CONFIG_CGROUP_FREEZER=y
CONFIG_CGROUP_DEVICE=y
CONFIG_CPUSETS=y
CONFIG_PROC_PID_CPUSET=y
CONFIG_CGROUP_SCHED=y
CONFIG_BLK_CGROUP=y
…
```

- **Mount the Cgroups filesystem**

```
# mount -t cgroup cgroup /cgroup
```

# How to use Cgroups

## How to make a group

```
# mkdir /cgroup/[GroupName]
```

## How to assign a task to a group

- Tasks are not only processes but also threads
- You have to set cpuset.cpus and cpuset.mems before moving tasks

```
# echo 0 > /cgroup/[GroupName]/cpuset.cpus
# echo 0 > /cgroup/[GroupName]/cpuset.mems
# echo [TID] > /cgroup/[GroupName]/tasks
```

# Subsystems

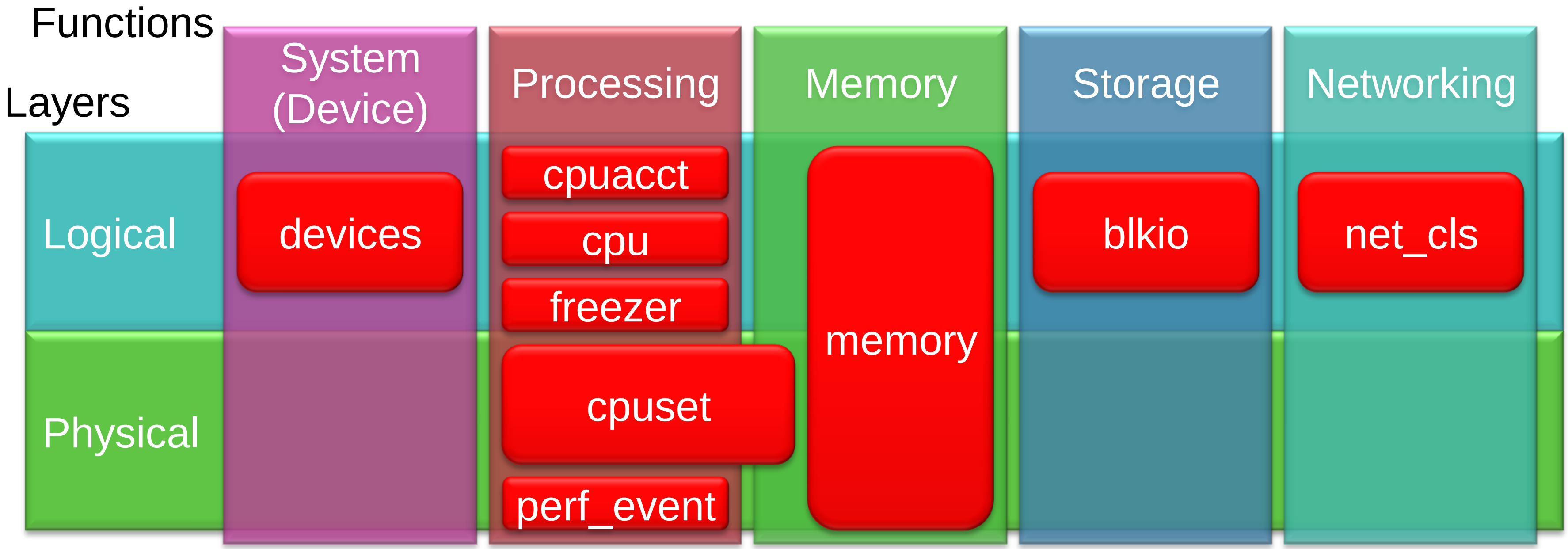
- **Cgroups have many subsystems**
  - Subsystems control several resources which can be used by tasks in groups
    - The number of physical CPU cores
    - CPU execution time
    - Physical memory limit
    - Block devices I/O bandwidth
    - …

- **How to enable a subsystem**
  - If you don’t add “-o [subsystem]”, all supported subsystems are enabled

```
# mount -t cgroup -o [subsytem] cgroup /cgroup
```

# Subsystems

- **What kind of subsystems are there?**
  - cpuset, cpu, cpuacct, memory, devices, blkio, net_cls, freezer, perf_event



- **How to check supported subsytems on your machine**

```
# cat /proc/cgroups
```

# Subsystem: cpuset

- **Assign physical CPU cores and memory node (e.g. on NUMA architecture) to a group**
  - Embedded systems usually don’t have more than 1 memory node
- **Useful parameters**
  - cpuset.cpus
    - Set of CPU cores that can be accessed by a group of tasks
  - cpuset.cpu_exclusive
    - A flag indicating if other groups can share the CPU core
- **Example**
  - “foo-group” uses CPU0, CPU1 and CPU2 exclusively

```
# echo 0-2 > /cgroup/foo-group/cpuset.cpus
# echo 1 > /cgroup/foo-group/cpuset.cpu_exclusive
```

# Subsystem: cpu

- **Schedule CPU access for a group by 2 schedulers**
  - CFS scheduler
    - Share CPU runtime between groups depending on a priority
  - RT scheduler
    - Assign fixed runtime to real-time tasks in a group
- **Useful parameters**
  - cpu.rt_period_us
    - Interval for reallocating CPU runtime for a group
  - cpu.rt_runtime_us
    - CPU runtime for a group in the period
- **Example**
  - Real-time tasks in “foo-group” run 0.95 sec in a period of 1 sec

```
# echo 1000000 > /cgroup/foo-group/cpu.rt_period_us
# echo 950000 > /cgroup/foo-group/cpu.rt_runtime_us
```

# Subsystem: cpuacct

- **Create a CPU resource usage report for each cgroups automatically**
- **Useful parameters**
  - cpuacct.usage
    - CPU runtime used by all tasks in a group
  - cpuacct.stat
    - Divided cpuacct.usage between user and system
  - cpuacct.usage_percpu
    - Divided cpuacct.usage per CPU
- **Example**
  - Show CPU runtime of “foo-group”

```
# cat /cgroup/foo-group/cpuacct.usage
13428211
```

# Subsystem: memory

- **Report memory usage and set physical memory limit for groups**
- **Useful parameters**
  - memory.limit_in_bytes
    - Set the maximum value of physical memory for a group
  - memory.oom_control
    - Flag of enable/disable oom-killer and notice
  - memory.stat
    - Report of memory statistics
- **Example**
  - Limit physical memory that can be used by “foo-group” to 100MB and disable oom-killer

```
# echo 104857600 > /cgroup/foo-group/memory.limit_in_byte
# echo 1 > /cgroup/foo-group/memory.oom_control
```

# Subsystem: devices

- **Limit access to device nodes from groups of tasks**
- **Useful parameters**
  - devices.allow
    - Set accessible devices from a group
  - devices.deny
    - Set non-accessible devices from a group
  - devices.list
    - Show accessible devices from a group
- **Example**
  - Show devices.list

```
# cat /cgroup/foo-group/devices.list
a *:* rwm
```

# Subsystem: blkio

- **Control accesses to block devices from a group**
- **There are 2 access control policies**
  - Share I/O bandwidth between groups
    - Set block I/O access ratio for each groups
  - I/O throttling
    - Set the limit for the number of I/O operation on a device node
- **Useful parameters**
  - blkio.weight
    - Set block I/O access ratio for each groups from 100 to 1000
- **Example**
  - The block I/O bandwidth of “foo-group” is 10 times larger than “bar-group”

```
# echo 1000 > /cgroup/foo-group/blkio.weight
# echo 100 > /cgroup/bar-group/blkio.weight
```

# Subsystem: net_cls, freezer, perf_event

- **net_cls**
  - Tag network packets sent by groups
    - Linux traffic controler (tc) can identify and assign a priority thanks to tagging by net_cls
  - tc can reserve network bandwidth
- **freezer**
  - Pause and resume all tasks in a group
  - Example: Freeze “foo-group”

```
# echo FROZEN > /cgroup/foo-group/freezer.state
```

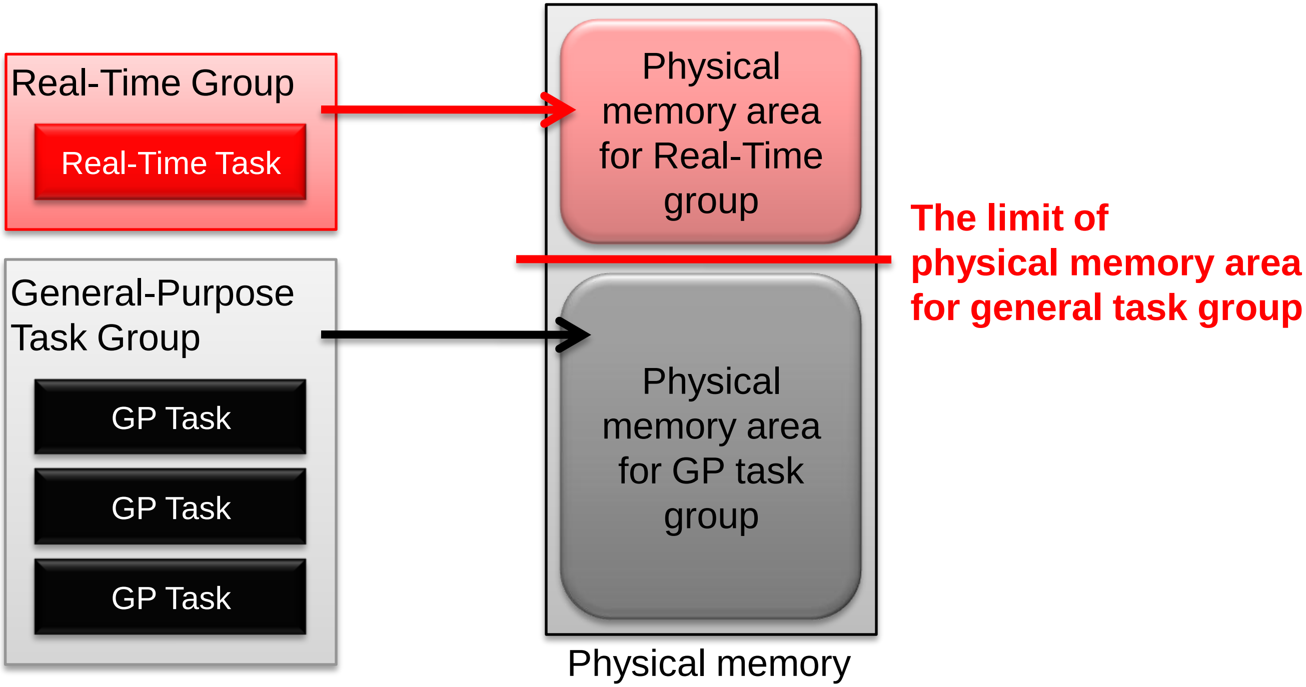
- **perf_event**
  - Enable monitoring using the “perf” tool
    - CPU cycles time, Executed instructions, Cache misses, Branch prediction misses, Page faults, Context switches, etc…

# Contents

- Background
- Introduction to Cgroups
- Use cases
- Evaluation
- Discussion
- Conclusions

# Reserving Physical Memory Space

- **Detail**
  - Reserve physical memory space to run a real-time task
- **Needed subsystem**
  - memory

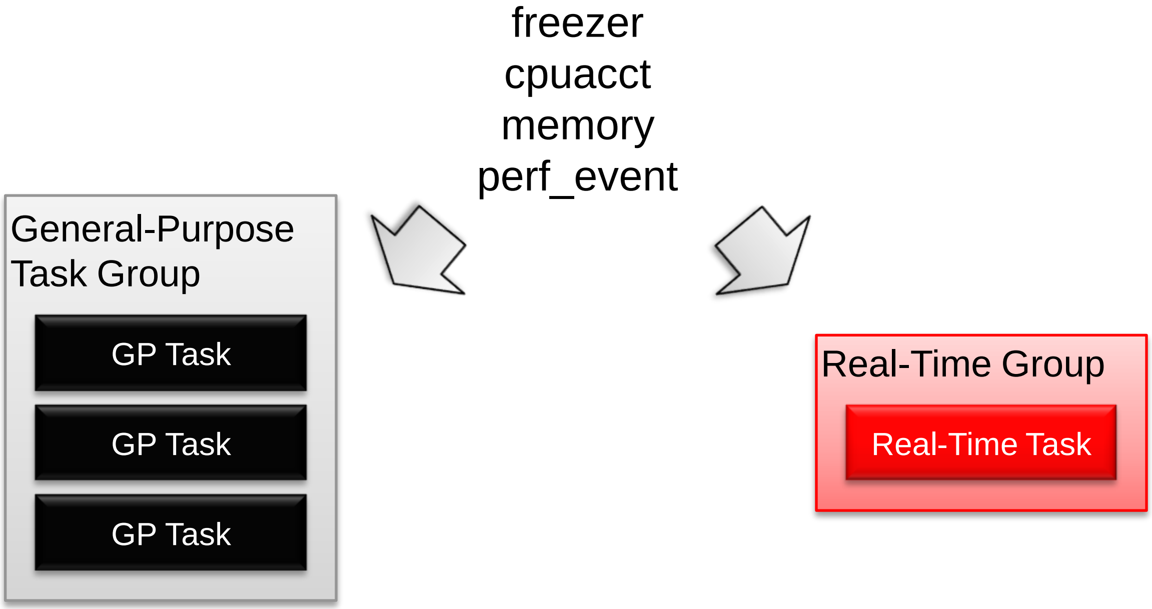


Real-Time Group
Real-Time Task
General-Purpose Task Group
GP Task
GP Task
GP Task
Physical memory area for Real-Time group
Physical memory area for GP task group
Physical memory

**The limit of physical memory area for general task group**

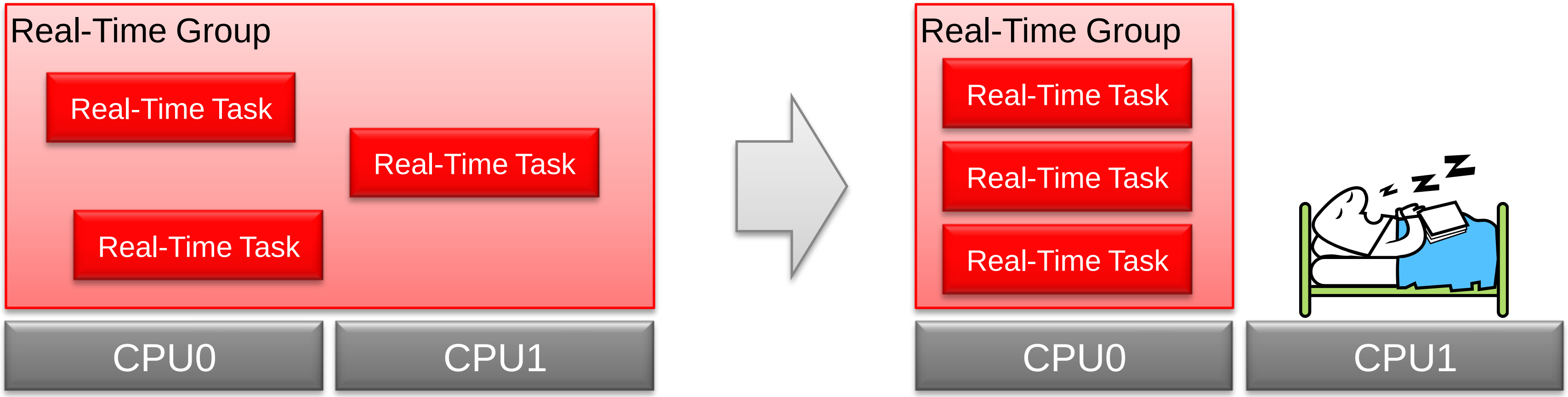
# Monitoring Groups

- **Detail**
  - Monitor some groups of general-purpose tasks and real-time tasks
- **Needed subsystems**
  - freezer, cpuacct, memory, perf_event



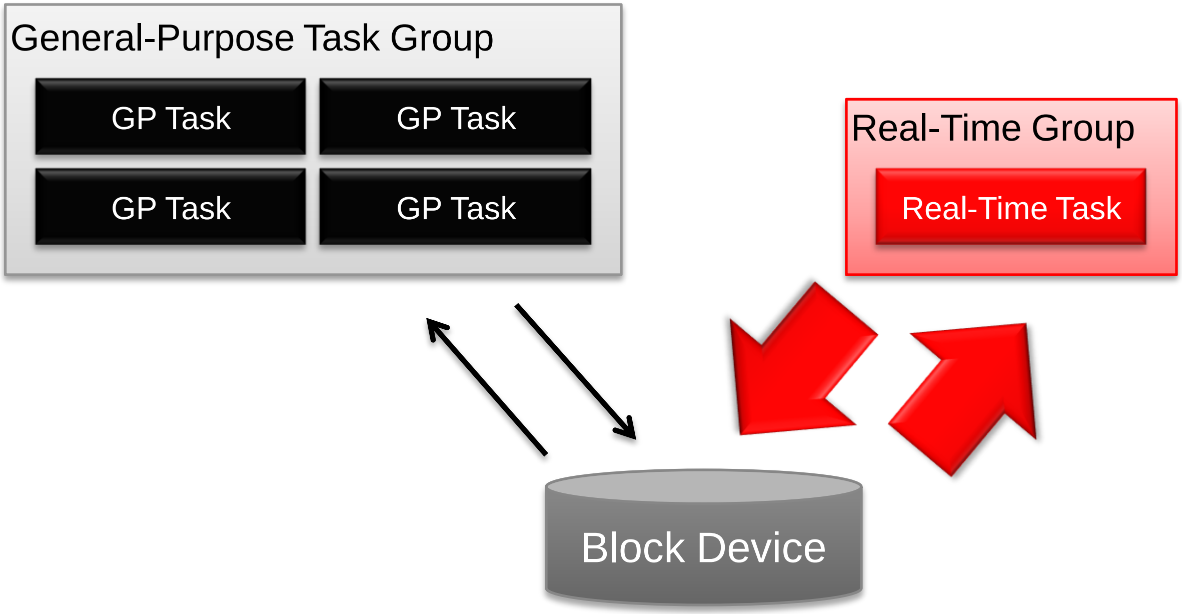
# Power saving

- **Detail**
  - When we detect, through cpuacct.usage, that the load of a CPU is not high, limit the number of physical CPUs using cpuset.cpus to achieve power saving
- **Needed subsystems**
  - cpuacct, cpuset



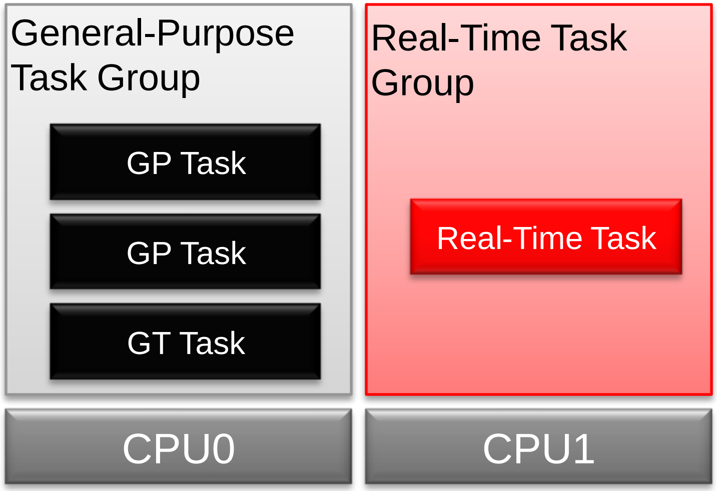
# Reserving Block Device I/O Bandwidth

- **Detail**
  - Assign needed I/O bandwidth to real-time tasks
  - Defend response time of real-time tasks against overloaded I/O requests by general-tasks [see evaluation]
- **Needed subsystem**
  - blkio



# Exclusive Possession of Physical CPU Core

- **Detail**
  - Real-Time tasks use several physical CPU exclusively using cpuset.cpus and cpuset.cpu_exclusive to achieve short response time [see evaluation]
- **Needed subsystem**
  - cpuset



# Contents

- Background
- Introduction to Cgroups
- Use cases
- Evaluation
- Discussion
- Conclusions

# Evaluation Environment

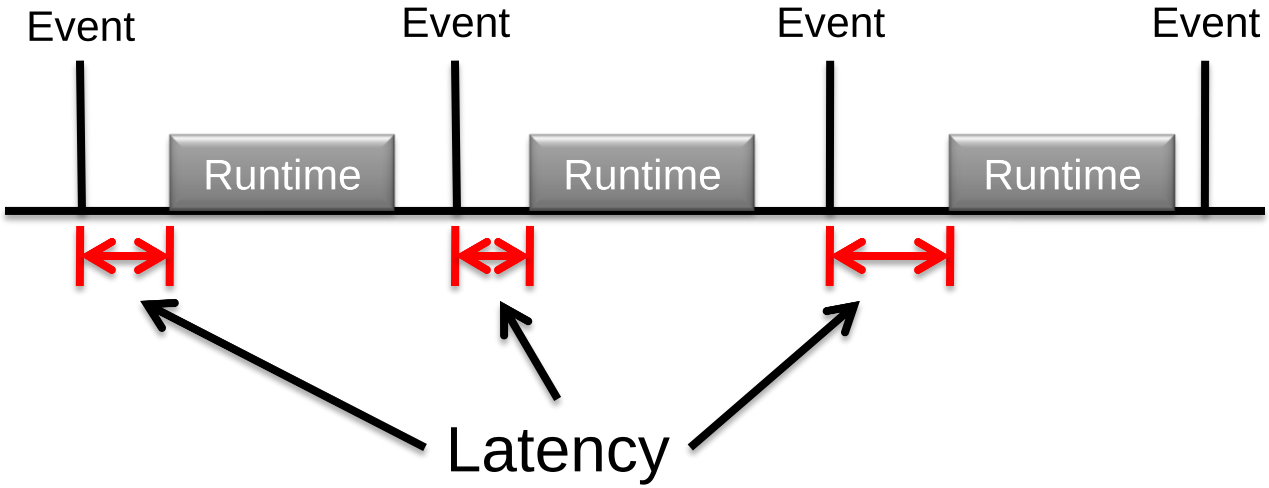
| | |
|---|---|
| Machine | HP Compaq 8200 Elite |
| CPU | Intel Core i7-2600 3.40GHz x 4 |
| Memory | 4GB |
| Kernel | v3.0.39-rt59 |
| Clock source | HPET |

```
# echo hpet >
/sys/device/system/clocksource/clocksource0/curren
t_clocksource
```

- **Disable power saving function of CPU cores**
  - idle=poll (at boot parameter)
- **Mount cpuset and blkio subsystems only**
  - Avoid overheads from other subsystems

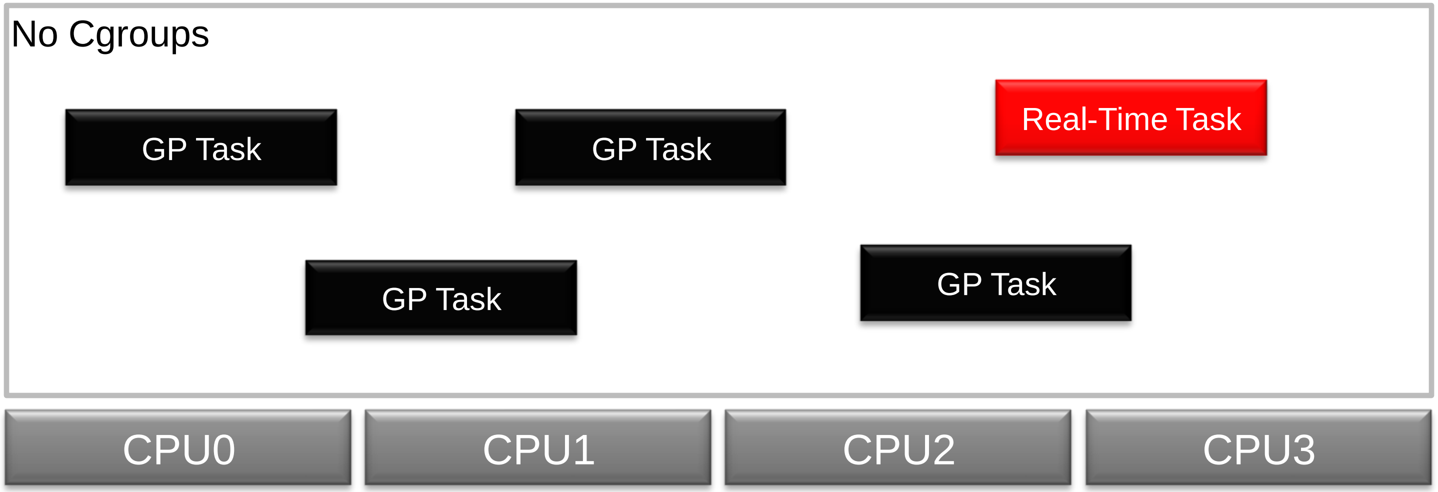
# How to Evaluate

- **Run cyclictest**
  - 4 conditions with 4 loads
  - 1,000,000 times
- **What is cyclictest?**
  - Run a real-time task that wakes up with a periodic time interval
  - Log response times, called “Latency”, of the real-time task



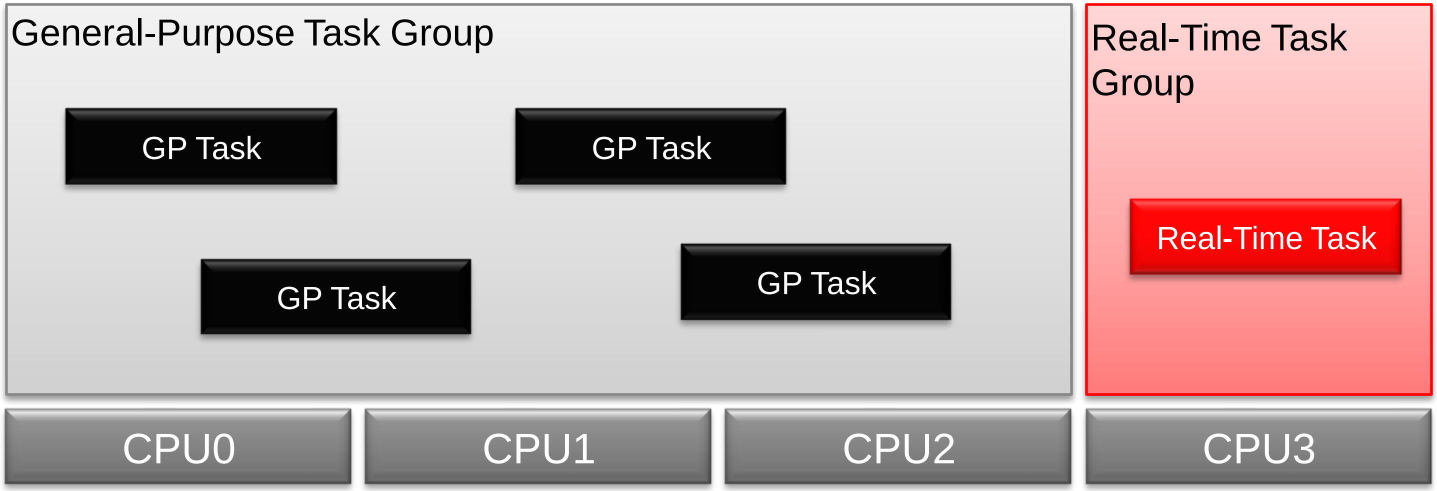
# Conditions

- **nocgroups**
  - Cgroups isn't used
  - 1 real-time tasks run with some general-purpose tasks



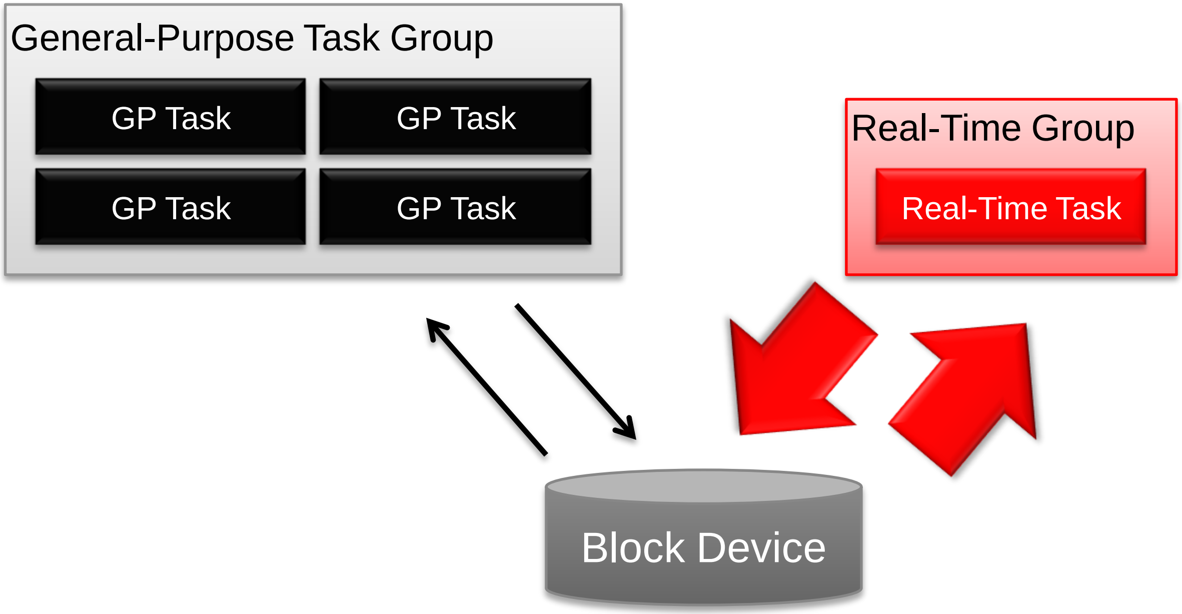
# Conditions

- **cpuset**
  - General-purpose tasks run in a general-purpose task group on 3 physical CPU core used exclusively
  - 1 real-time task runs in a real-time task group on 1 physical CPU core used exclusively



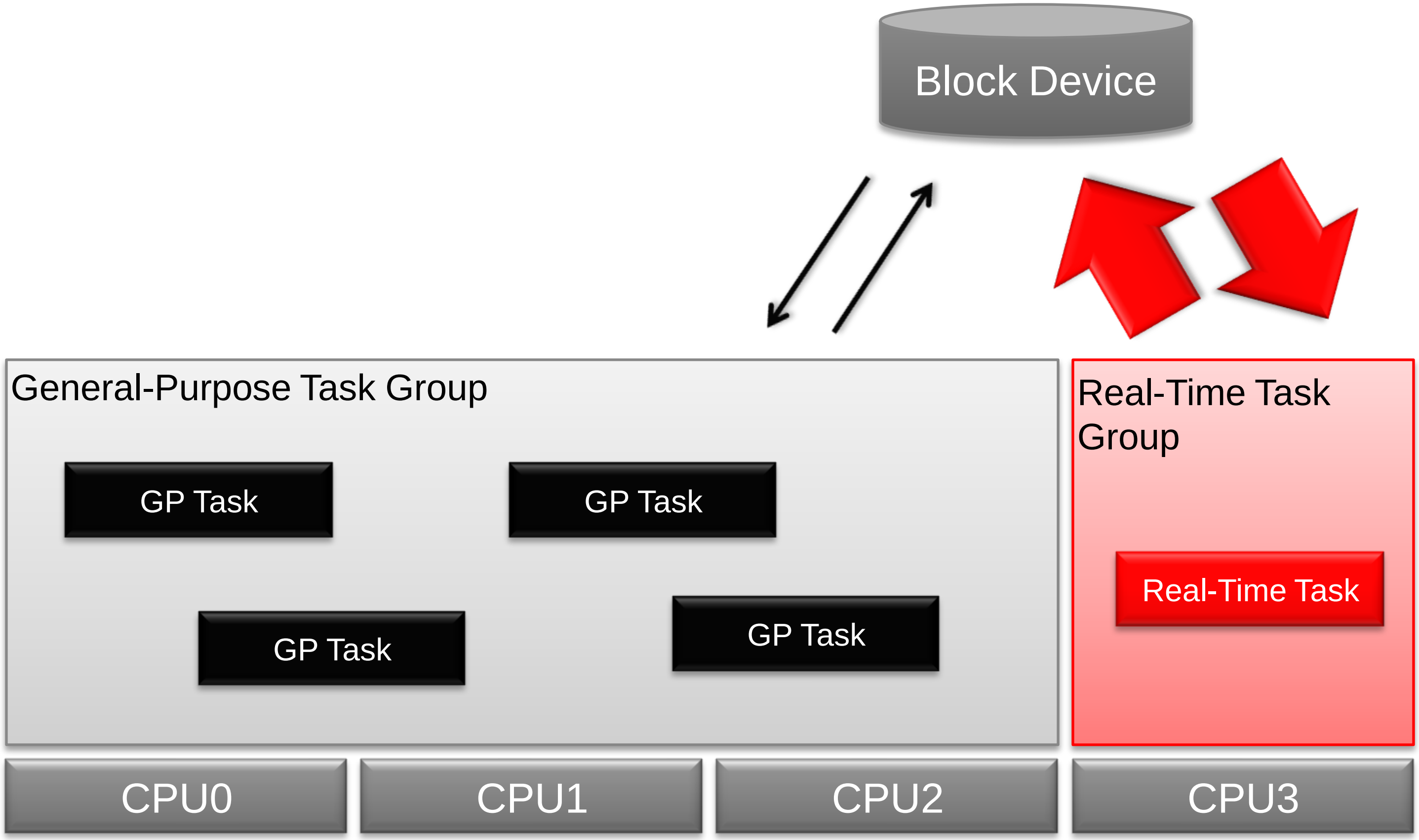
# Conditions

- **blkio**
  - General-purpose tasks run in a general-purpose task group
  - 1 real-time task runs in a real-time task group with 10 times larger bandwidth than a general-purpose task group



# Conditions

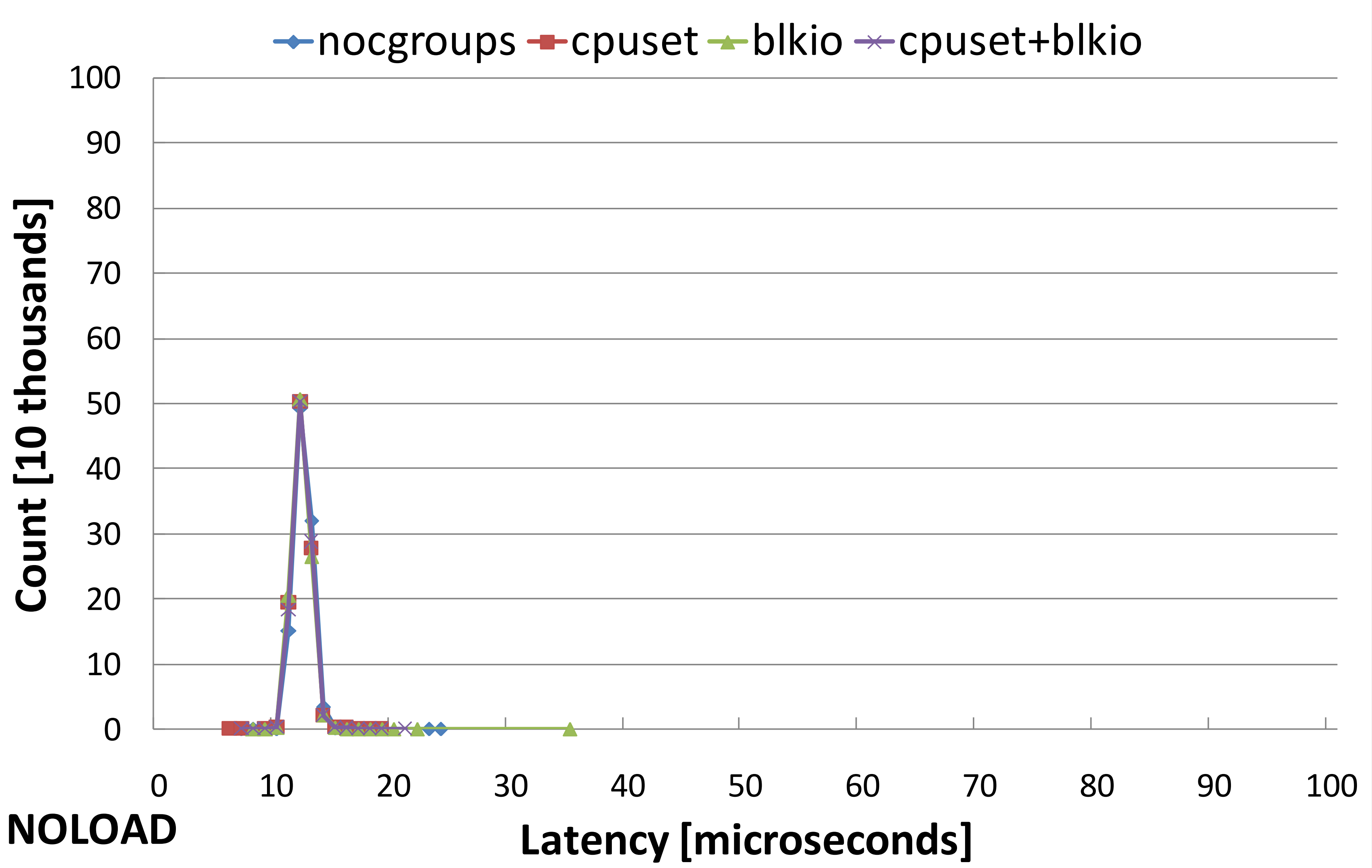
- **cpuset + blkio**
  - Both of cpuset and blkio



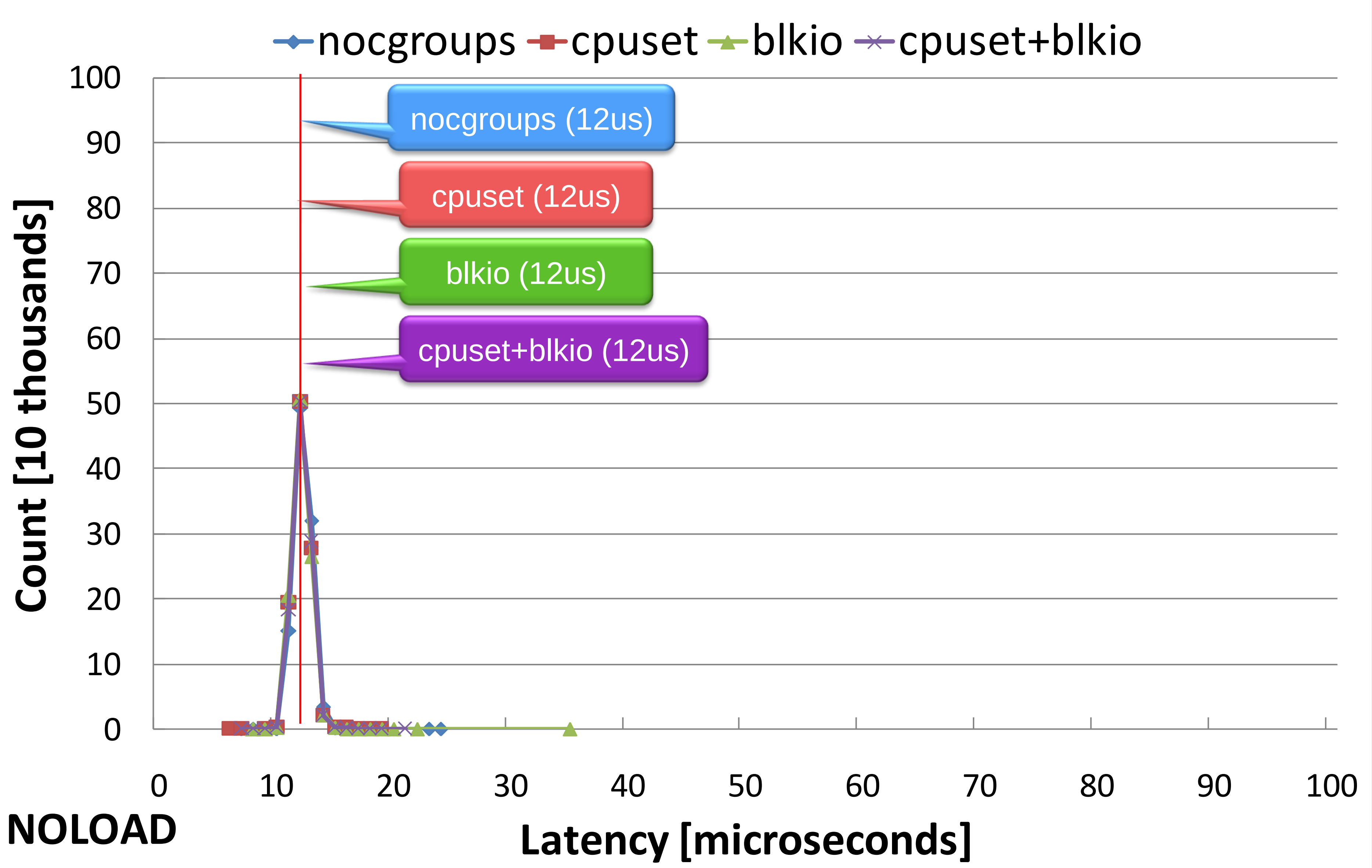
# Loads

- **NOLOAD**
  - No any loads
- **CPU-LOAD**
  - Set CPU usage rate to 100%
    - Running 4 busy loop threads
- **SCHED-LOAD**
  - Generate many context switches
    - Running 270 busy loop threads that sleep 1us during each loop
    - CPU usage rate is 100%
- **IO-LOAD**
  - Generate many disk I/O requests
    - Running 50 busy loop threads that open a file, write 4KB data to it, synchronize it and sleep 1us during each loop
    - Average 47-50 kernel threads wait for I/O request

# NOLOAD


nocgroups
cpuset
blkio
cpuset+blkio
Count [10 thousands]
100
90
80
70
60
50
40
30
20
10
0
0
10
20
30
40
50
60
70
80
90
100
Latency [microseconds]
NOLOAD

# NOLOAD Average



nocgroups (12us)

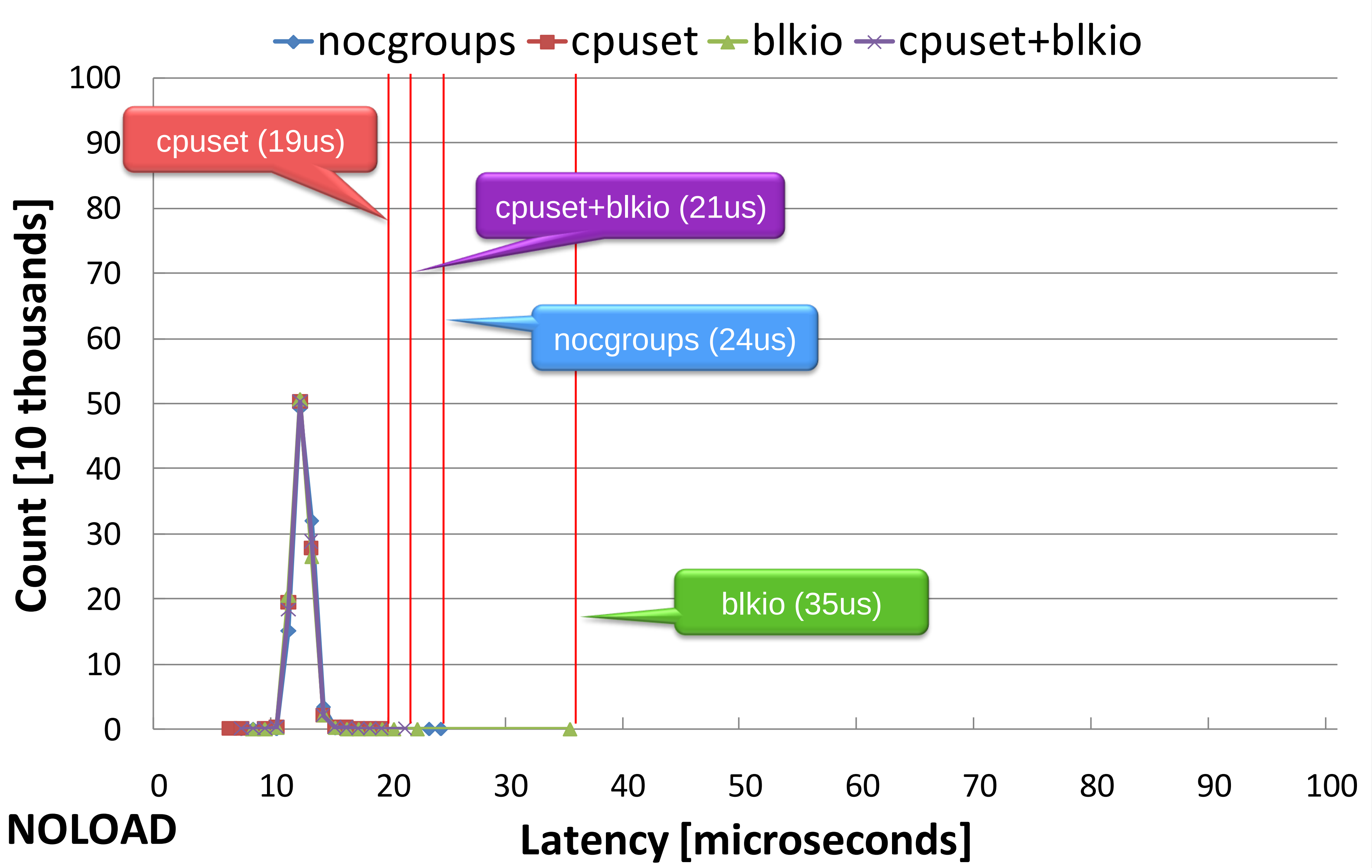
cpuset (12us)

blkio (12us)

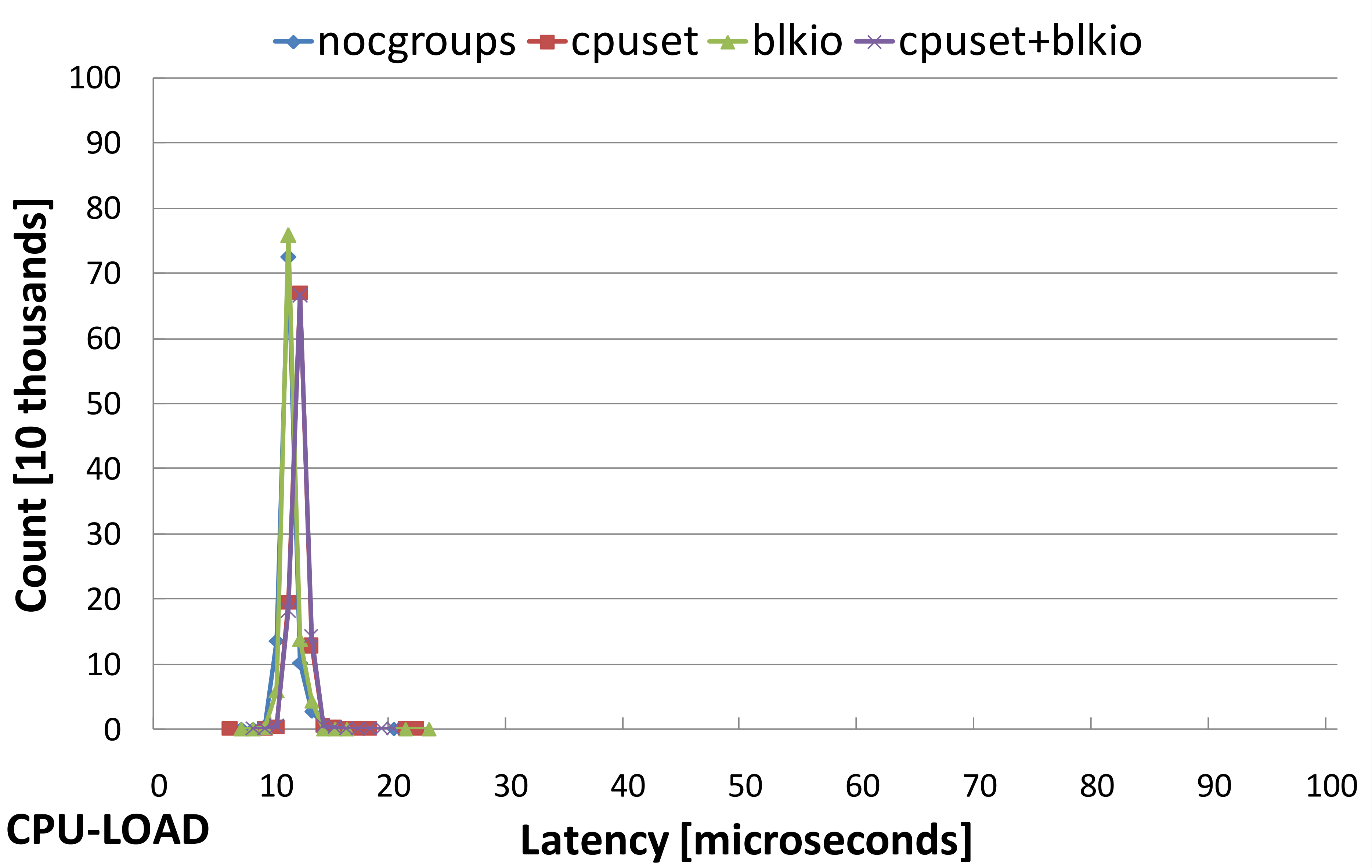
cpuset+blkio (12us)

TOSHIBA
Leading Innovation >>>

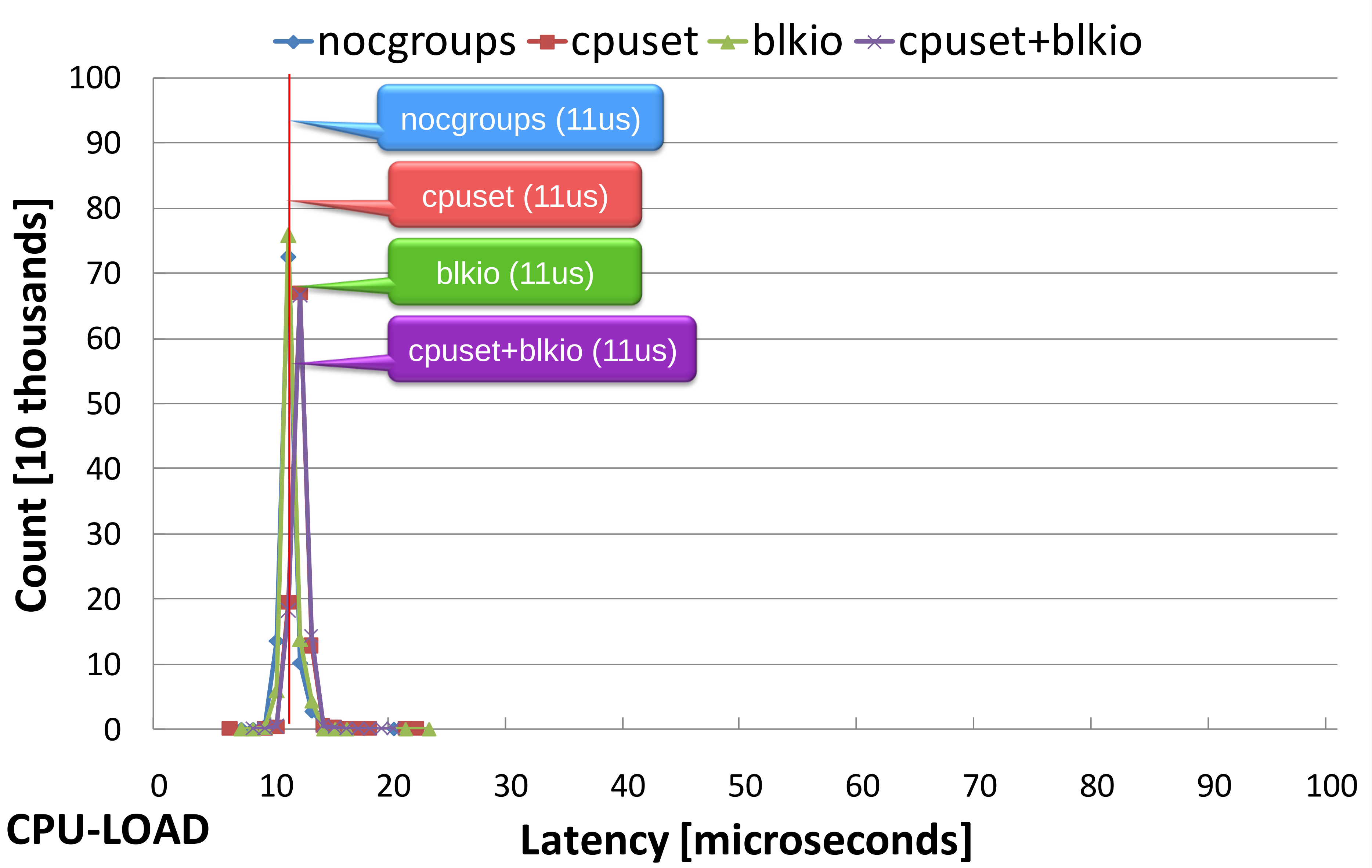
# NOLOAD Max


nocgroups
cpuset
blkio
cpuset+blkio
Count [10 thousands]
100
90
80
70
60
50
40
30
20
10
0
0
10
20
30
40
50
60
70
80
90
100
Latency [microseconds]
cpuset (19us)
cpuset+blkio (21us)
nocgroups (24us)
blkio (35us)
NOLOAD

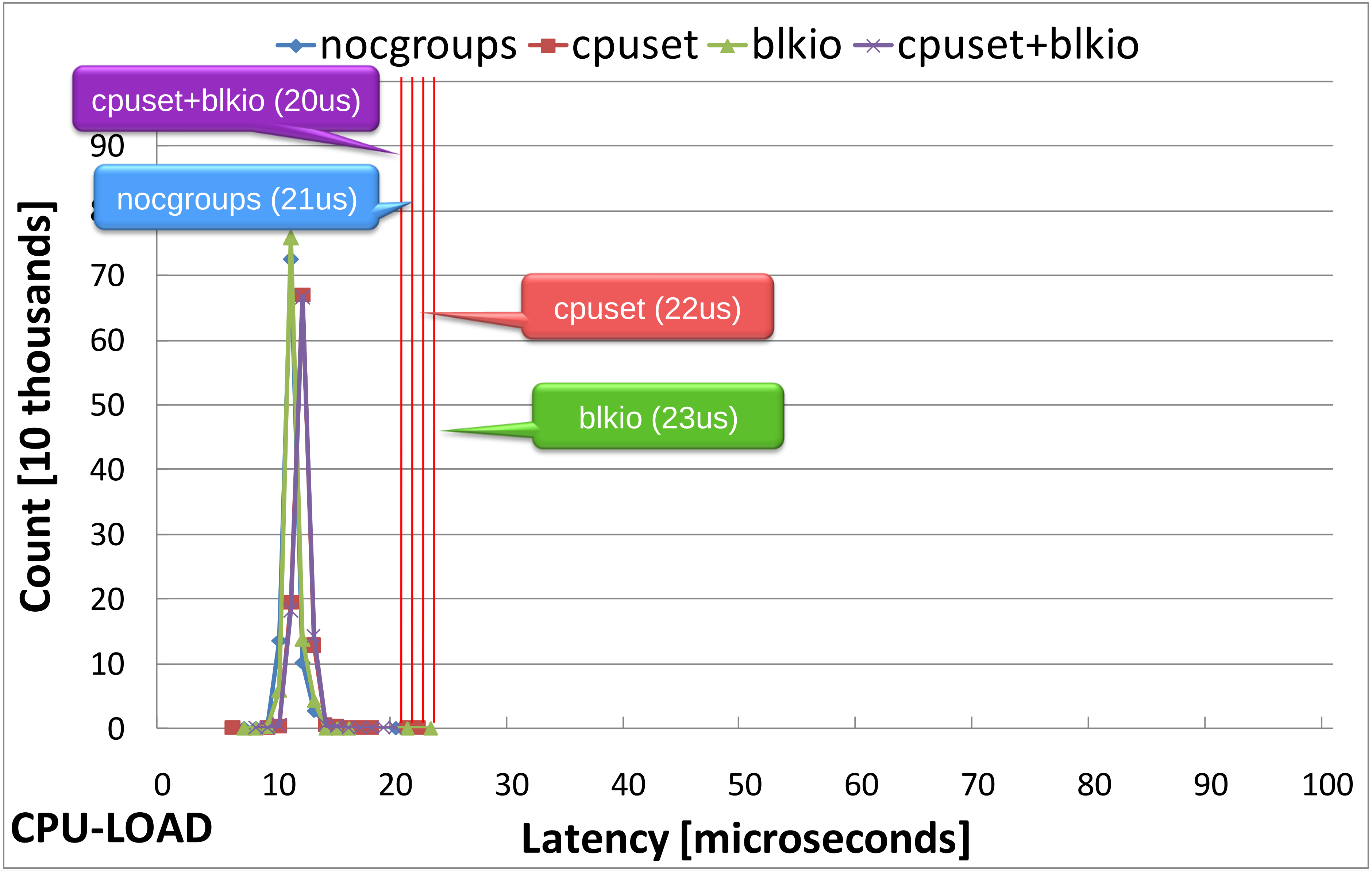
# CPU-LOAD


nocgroups
cpuset
blkio
cpuset+blkio
Count [10 thousands]
100
90
80
70
60
50
40
30
20
10
0
0
10
20
30
40
50
60
70
80
90
100
CPU-LOAD
Latency [microseconds]

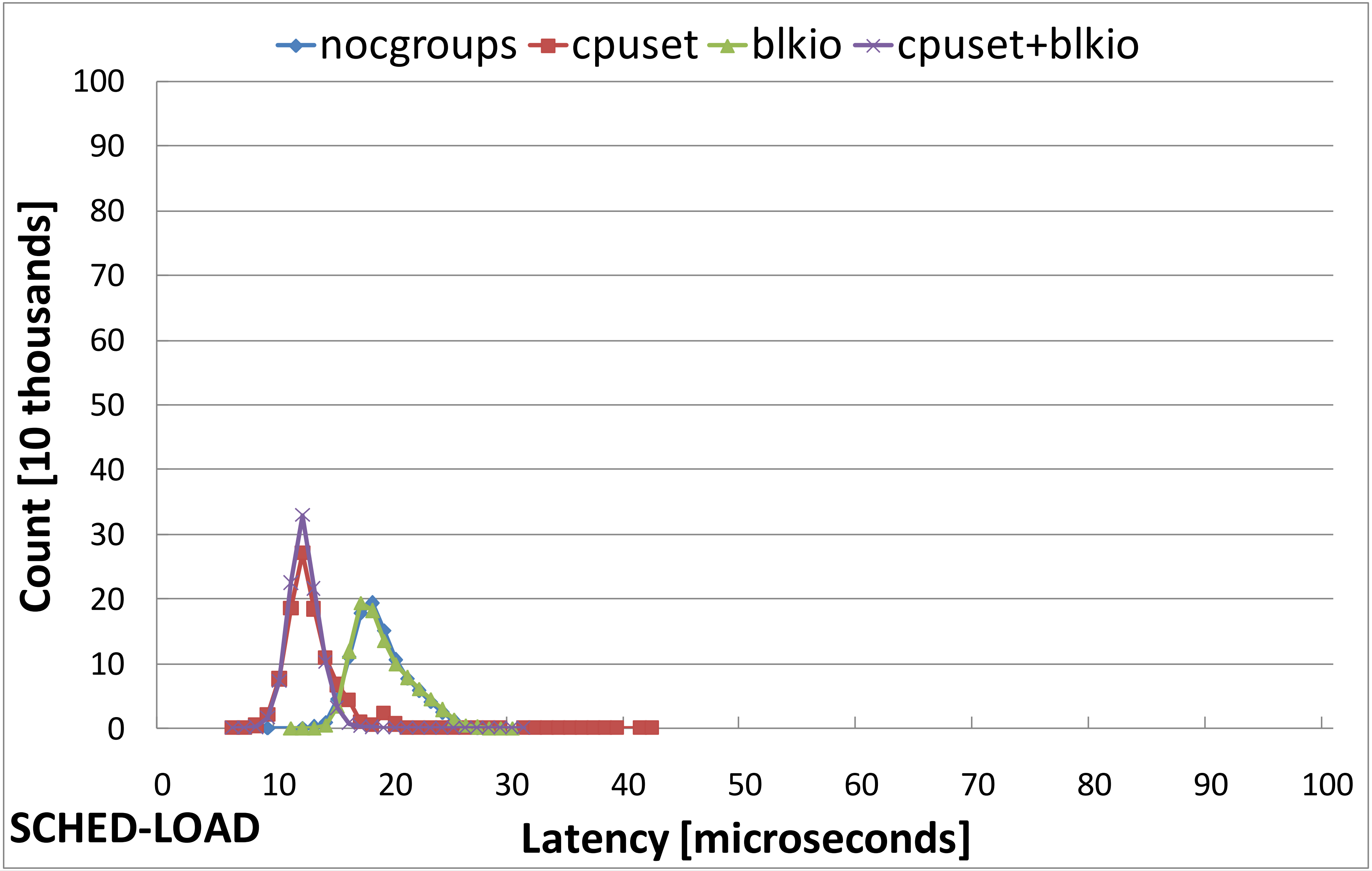
# CPU-LOAD Average



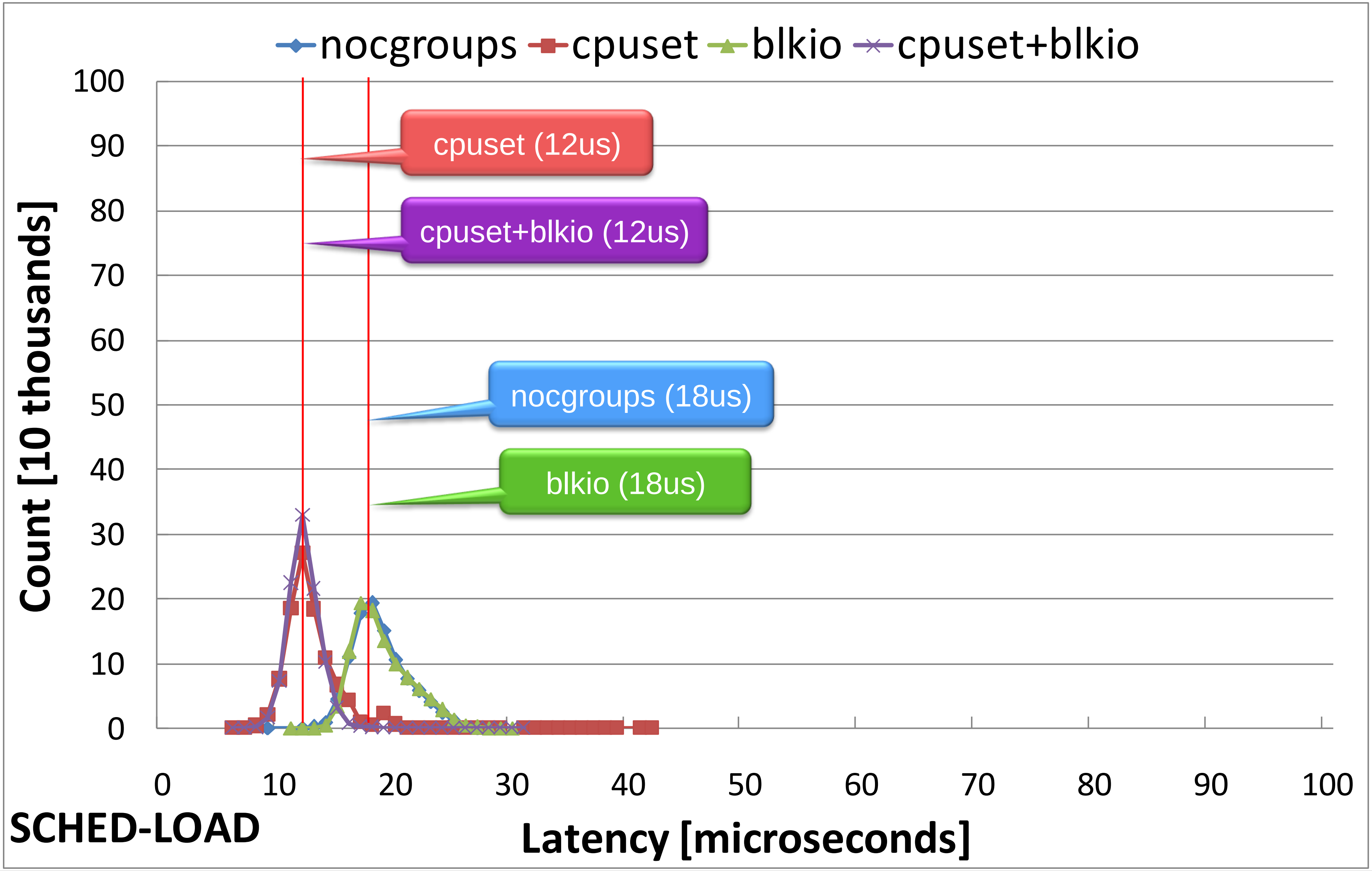
# CPU-LOAD Max


nocgroups
cpuset
blkio
cpuset+blkio
cpuset+blkio (20us)
nocgroups (21us)
cpuset (22us)
blkio (23us)
Count [10 thousands]
0
10
20
30
40
50
60
70
90
0
10
20
30
40
50
60
70
80
90
100
CPU-LOAD
Latency [microseconds]

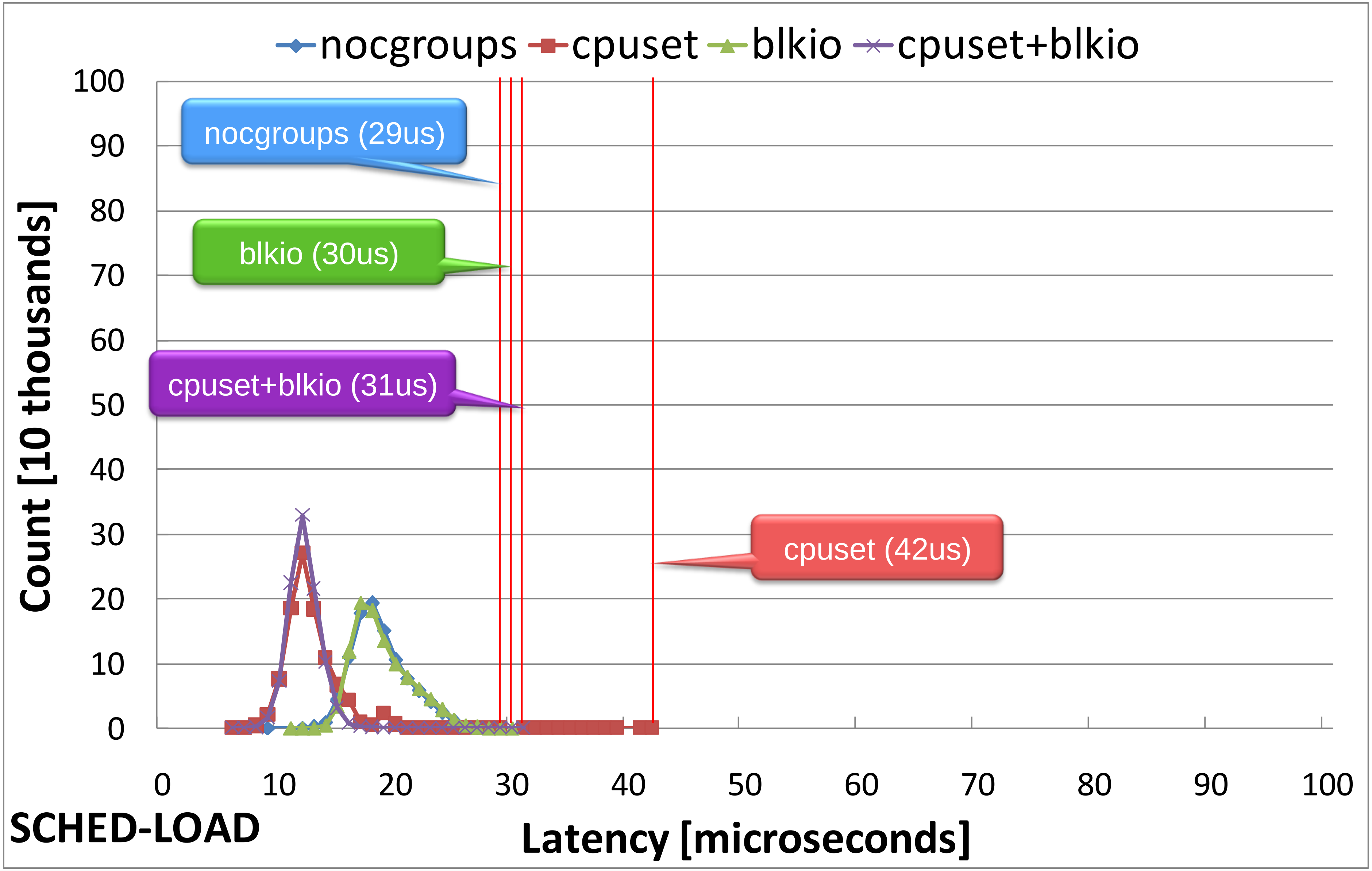
# SCHED-LOAD


nocgroups
cpuset
blkio
cpuset+blkio
Count [10 thousands]
100
90
80
70
60
50
40
30
20
10
0
0
10
20
30
40
50
60
70
80
90
100
SCHED-LOAD
Latency [microseconds]

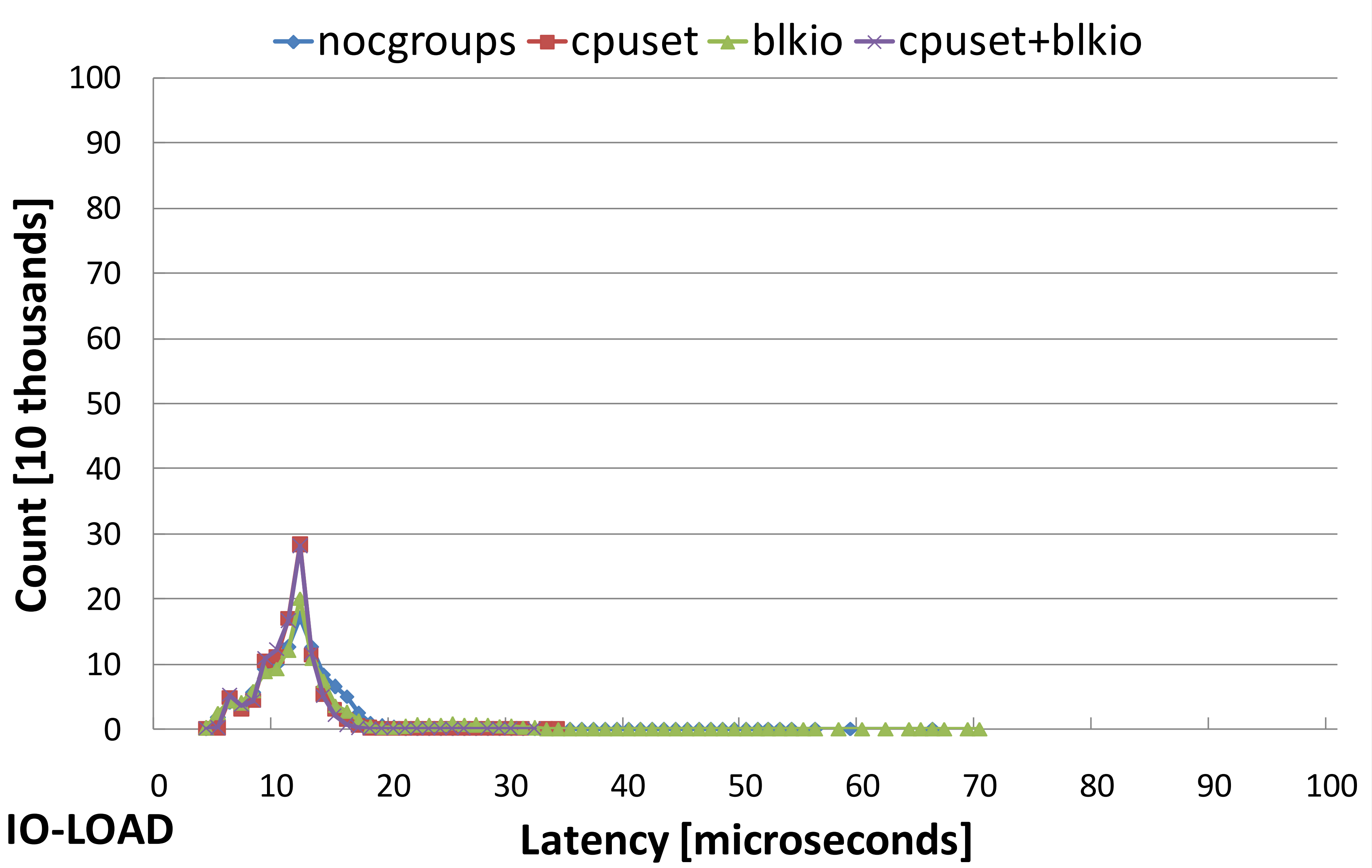
# SCHED-LOAD Average



# SCHED-LOAD Max

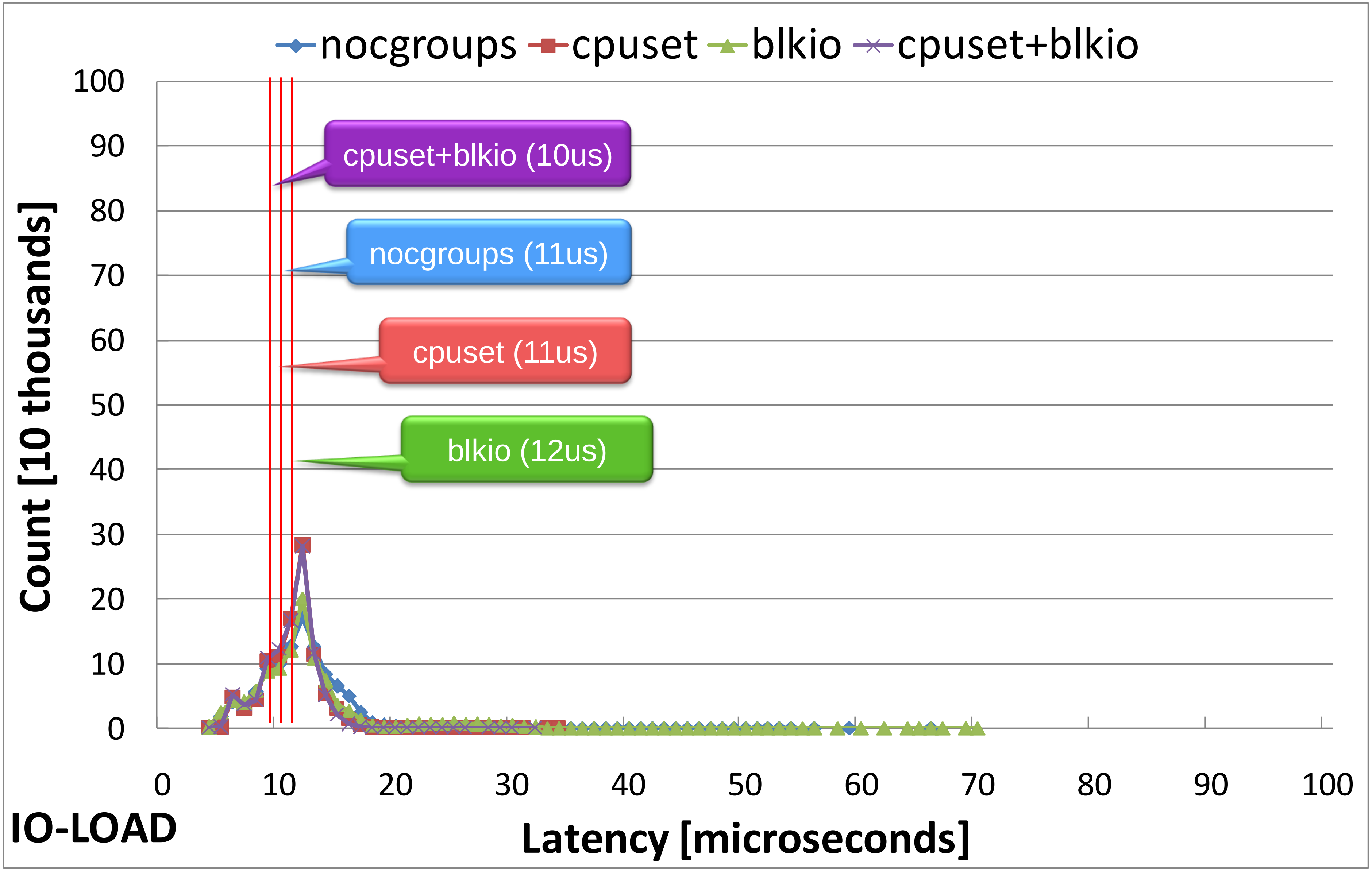


# IO-LOAD

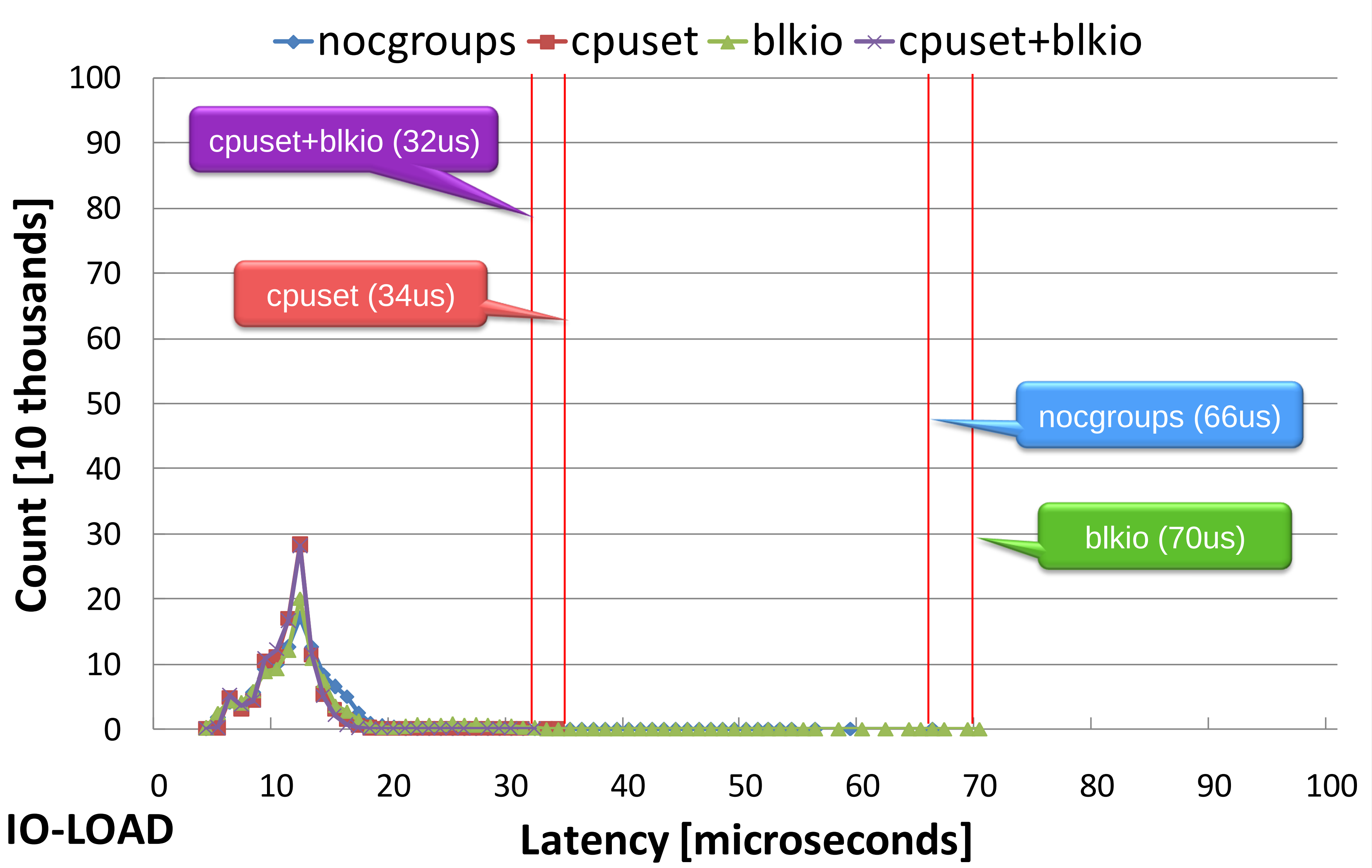


nocgroups
cpuset
blkio
cpuset+blkio
Count [10 thousands]
100
90
80
70
60
50
40
30
20
10
0
0
10
20
30
40
50
60
70
80
90
100
IO-LOAD
Latency [microseconds]

# IO-LOAD Average



# IO-LOAD Max



IO-LOAD

# Contents

- Background
- Introduction to Cgroups
- Use cases
- Evaluation
- Discussion
- Conclusions

# Discussion

- **cpuset**
  - Advantages
    - Contributed to shorten average response time with SCHED-LOAD
    - Contributed to shorten max response time with IO-LOAD
  - Disadvantage
    - Max response time with SCHED-LOAD is longer than nocgroups
- **blkio**
  - There are no advantages
  - Disadvantage
    - Max response time with NOLOAD is longer than nocgroups
- **cpuset + blkio**
  - Advantages are same as cpuset
  - There are no disadvantages

# Contents

- **Background**
- **Introduction to Cgroups**
- **Use cases**
- **Evaluation**
- **Discussion**
- **Conclusions**

# Conclusions

- **Cgroups can supply a mechanism of resource partitioning**
  - Real-time tasks can use partitioned resources and achieve many advantage against general-purpose tasks
  - cpuset and blkio subsystems contributes to shorten response time for a real-time task

- **We want to partition more resources for real-time tasks**
  - Not only short response time but also management, control and protection
  - Do you have other ideas and use cases for partitioning of real-time tasks?

# References

- **Resource Management Guide - Red Hat Customer Portal**
  - https://access.redhat.com/site/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Resource_Management_Guide/