Google

Thunderbolt:
Throughput-Optimized, QoS-Aware Power Capping at Scale

Shaohong Li*, Xi Wang, Xiao Zhang, Vasileios Kontorinis, Sreekumar Kodakara, David Lo, Parthasarathy Ranganathan

* Presenter

Motivation: power oversubscription and capping

$200+B worldwide spend on data centers

Power oversubscription:
more capacity without construction

- Data center aggregated power usage is rarely close to the theoretical max
- But protective systems are needed to avoid overload due to rare power spikes

Power capping:
protective system that shaves power spikes

- Throttles running tasks without violating their SLOs

Google

Motivation: task QoS differentiation

Throughput-oriented tasks

- Completes on the order of hours or more
- Examples: web indexing, log processing
- Amenable to performance throttling (runs slower)
- Not amenable to disruption (wastes work)

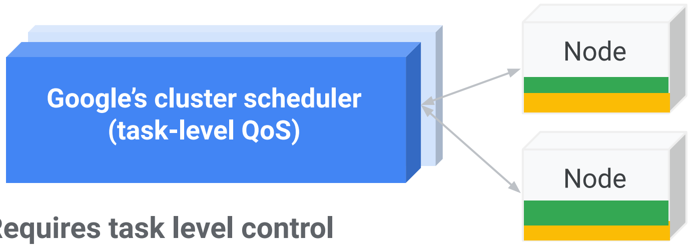
Latency-sensitive tasks

- Responds to requests on the order of milliseconds to seconds
- Examples: web front-end, search service
- Not amenable to performance throttling (becomes unresponsive)

Tasks of different QoS are co-located

- Google's cluster scheduler does not assign priority or QoS to machines. They are assigned to tasks
- Improves machine utilization

Goal: Task QoS-aware capping that gently throttles throughput-oriented tasks and exempts latency-sensitive tasks



Requires task level control

Google

Prior industry solutions did not meet our needs

Either task QoS-aware but has disruptive capping action...	...Or has gentle throttling but coarser-grained QoS differentiation
Example: Capping system for medium voltage power plane [1]	**Examples:** Dynamo [2], CapMaestro [3]
Appropriate for clusters with low-priority tasks that can tolerate disruption	Appropriate for clusters with coarser-grained QoS differentiation, such as machine level

[1] Sakalkar et al. Datacenter power oversubscription with a medium voltage power plane and priority-aware capping. ASPLOS 2020.
[2] Wu et al. Dynamo: Facebook's data center-wide power management system. ISCA 2016.
[3] Li et al. A scalable priority-aware approach to managing datacenter server power. HPCA 2019.

Google

Thunderbolt’s contributions

Simultaneously achieves the following:

01

Power safety with minimized performance degradation

Efficient use of power budget while being responsive and effective to reduce power.

02

Task-level QoS differentiation

Flexible to apply different throttling levels to tasks with different SLO, even on the same machine.

03

Hardware platform independence

Applicable to all platforms of all vendors. Accelerates new platform introduction.

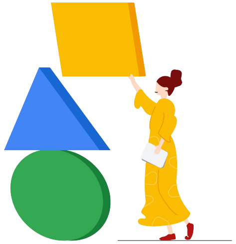
04

Tolerance of power telemetry unavailability

Failover subsystem that can safely operate without power telemetry.

Deployed at scale in Google data centers over years

Enables oversubscription in logs processing clusters from 0 to 9--25%



Google

Architecture & Implementation

Architecture

“Reactive capping” primary subsystem

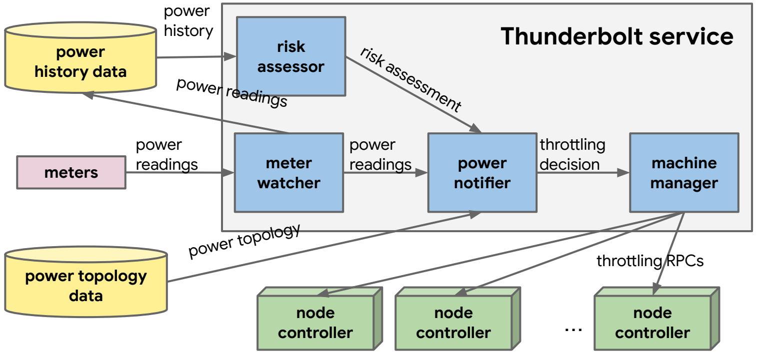
Operates when power signals are **available.**

1. Meter watcher reads power from meters
2. Power notifier determines whether and how to throttle based on the “*load shaping*” policy
3. Machine manager sends throttling (load shaping) RPCs to node controllers
4. Node controller throttles tasks using “*CPU bandwidth control*”

“Proactive capping” failover subsystem

Operates when power signals are **unavailable.**

1. Risk assessor reads power history data and assesses risk of power overload
2. If risk is high, machine manager sends throttling (“*CPU jailing*”) RPCs to node controllers
3. Node controller throttles tasks using CPU jailing



Architecture

“Reactive capping” primary subsystem

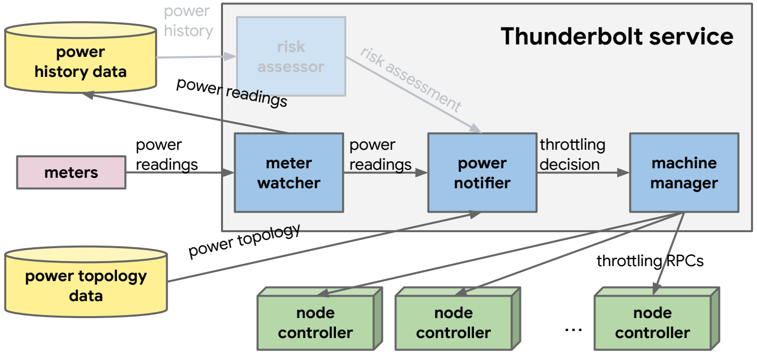
Operates when power signals are **available.**

1. Meter watcher reads power from meters
2. Power notifier determines whether and how to throttle based on the “***load shaping***” policy
3. Machine manager sends throttling (load shaping) RPCs to node controllers
4. Node controller throttles tasks using “***CPU bandwidth control”***

“Proactive capping” failover subsystem

Operates when power signals are unavailable

1. Risk assessor reads power history data and assesses risk of power overload
2. If risk is high, machine manager sends throttling (“*CPU jailing*”) RPCs to node controllers
3. Node controller throttles tasks using CPU jailing



Architecture

“Reactive capping” primary subsystem

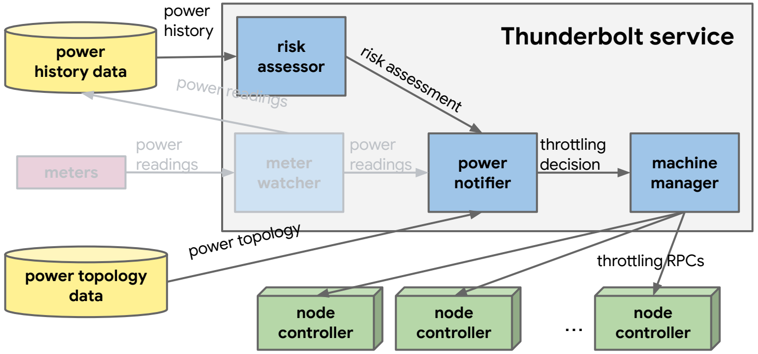
Operates when power signals are available.

1. Meter watcher reads power from meters
2. Power notifier determines whether and how to throttle based on the “*load shaping*” policy
3. Machine manager sends throttling (load shaping) RPCs to node controllers
4. Node controller throttles tasks using “*CPU bandwidth control*”

“Proactive capping” failover subsystem

Operates when power signals are **unavailable.**

1. Risk assessor reads power history data and assesses risk of power overload
2. If risk is high, machine manager sends throttling (“***CPU jailing”***) RPCs to node controllers
3. Node controller throttles tasks using *CPU jailing*



Google

Mechanism and policy details

Reactive capping

Node mechanism:

CPU bandwidth control

Control policy:

Load shaping

Proactive capping

Node mechanism:

CPU jailing

Control policy:

Risk assessment

Mechanism and policy details

Reactive capping

Node mechanism:

CPU bandwidth control

Control policy:

Load shaping

Proactive capping

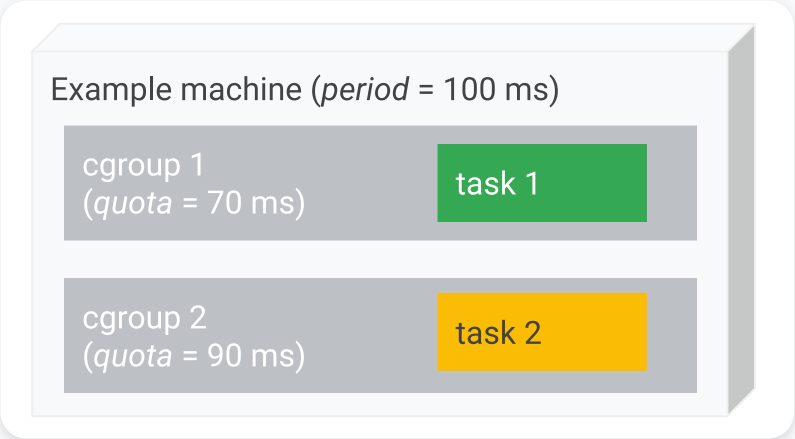
Node mechanism:

CPU jailing

Control policy:

Risk assessment

Reactive capping mechanism: CPU bandwidth control



If the machine has 2 logical CPUs, then its CPU utilization is capped at (70 + 90) / (100 * 2) = 80%

Linux kernel feature

- Platform independent

Task-level CPU cap

- A task is assigned to a control group (cgroup)
- A CPU cap is assigned to each cgroup by setting its *quota*

Google

Why not RAPL or DVFS?

RAPL

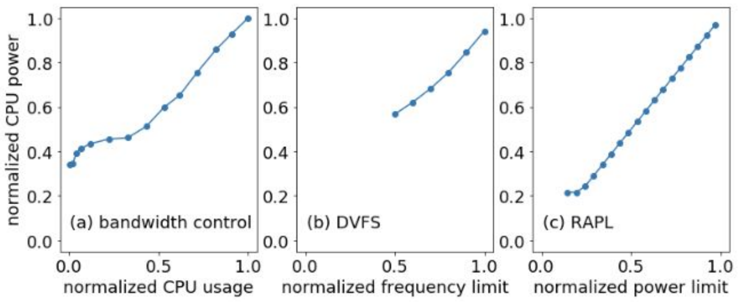
- Per-socket control. Cannot achieve task-level control without additional scheduling constraint.
- Intel specific. Not desirable for our clusters with diverse platforms.
- Precise power control and wide control dynamic range.

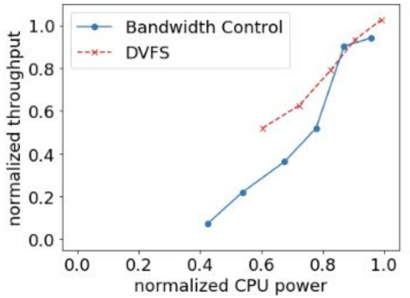
DVFS

- Supported by most modern platforms.
- Per-core control not supported by Intel pre-Haswell and some non-x86 platforms.
- Narrower power control dynamic range than bandwidth control.
- Higher throughput than bandwidth control under the same power budget.

CPU bandwidth control's native **task-level control** and **platform independence** is vital for scalability (DVFS may be added for future efficiency optimization where per-core control is supported)

CPU bandwidth control, DVFS, RAPL on Intel Skylake CPU





CPU power and set point

- Workload: power virus
- Power dynamic range: RAPL > BW control > DVFS

CPU power and throughput

- Workload: video transcoding
- Power dynamic range: BW control > DVFS
- Efficiency: DVFS > BW control

Google

Mechanism and policy details

Reactive capping

Node mechanism:

CPU bandwidth control

Control policy:

Load shaping

Proactive capping

Node mechanism:

CPU jailing

Control policy:

Risk assessment

Reactive capping policy: load shaping

Randomized unthrottling, multiplicative decrease

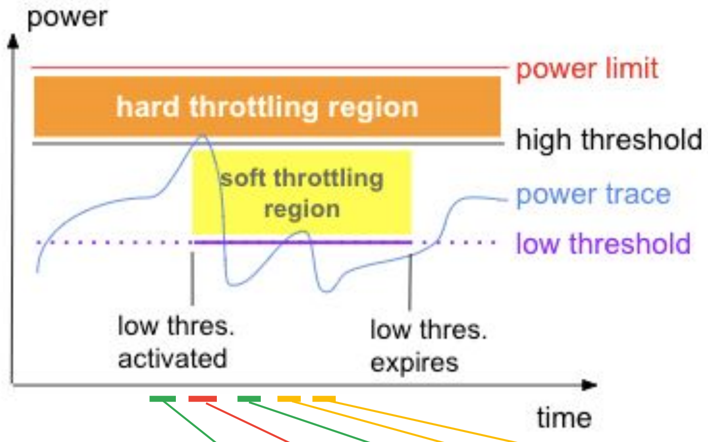
- Machines unthrottled randomly when throttling ends
- Task CPU cap decreases multiplicatively when throttling is active

Two thresholds with two multipliers

- Balances safety and efficiency

QoS differentiation: exempting latency tasks

- Continuous monitoring ensures safe power with exempt tasks



quota (ms)	-	2	-	80	56
usage (ms)	200	2	100	70	50

hard multiplier = 0.01, soft multiplier = 0.8

Google

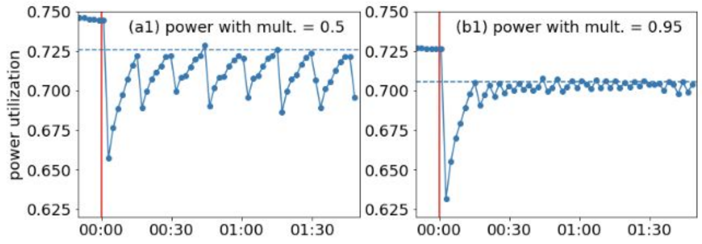
Load shaping on a production cluster

Production cluster

- Tens of thousands of machines
- Diverse workloads, both throughput-oriented and latency-sensitive

Power utilization pattern

- Sawtooth-like pattern
- Power reduced in 2s
- Smaller multiplier ⇒ oscillation with greater magnitude



Google

Load shaping on a production cluster

Failure of affected tasks

- Load shaping does not notably increase failures

99%-ile read latency of exempt storage service

- Load shaping does not notably increase latency

	Duration	Failure fraction	Latency
Baseline	25 min.	0.00002	79 ms
0.95 soft mult.	5 min.	0.00000	79 ms
0.75 soft mult.	10 min.	0.00003	80 ms
0.5 soft mult.	5 min.	0.00007	78 ms

Mechanism and policy details

Reactive capping

Node mechanism:

CPU bandwidth control

Control policy:

Load shaping

Proactive capping

Node mechanism:

CPU jailing

Control policy:

Risk assessment

Proactive capping mechanism: CPU jailing

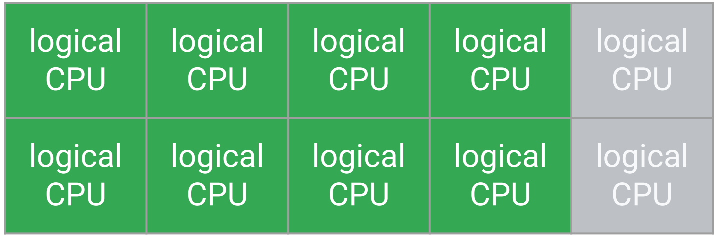
Deterministic machine CPU cap

- Deterministic cap is important for power safety
- A fraction, *J*, of logical CPUs on each machine are made unavailable to tasks
- Each machine's CPU utilization is capped at (1 - *J*)
- *J* can be determined by translating power budget to CPU budget using a model of power-CPU relation

Relaxed QoS differentiation

- Jailed CPUs are inaccessible to all tasks
- Required for strong power guarantee
- Higher priority tasks have more access to remaining CPUs

CPU mask of tasks with 20% CPU jailing (*J* = 0.2)



green = accessible; **gray** = jailed

Google

20% CPU jailing on a production cluster

Production cluster

- Tens of thousands of machines
- Diverse workloads, both throughput-oriented and latency-sensitive

Task failures

- 20% jailing does not notably affect failures

99%-ile read latency of storage service

- 20% jailing does not notably affect latency

	Duration	Failure fraction	Latency
Baseline	60 min.	0.00003	79 ms
CPU jailing	55 min.	0.00002	86 ms

Mechanism and policy details

Reactive capping

Node mechanism:

CPU bandwidth control

Control policy:

Load shaping

Proactive capping

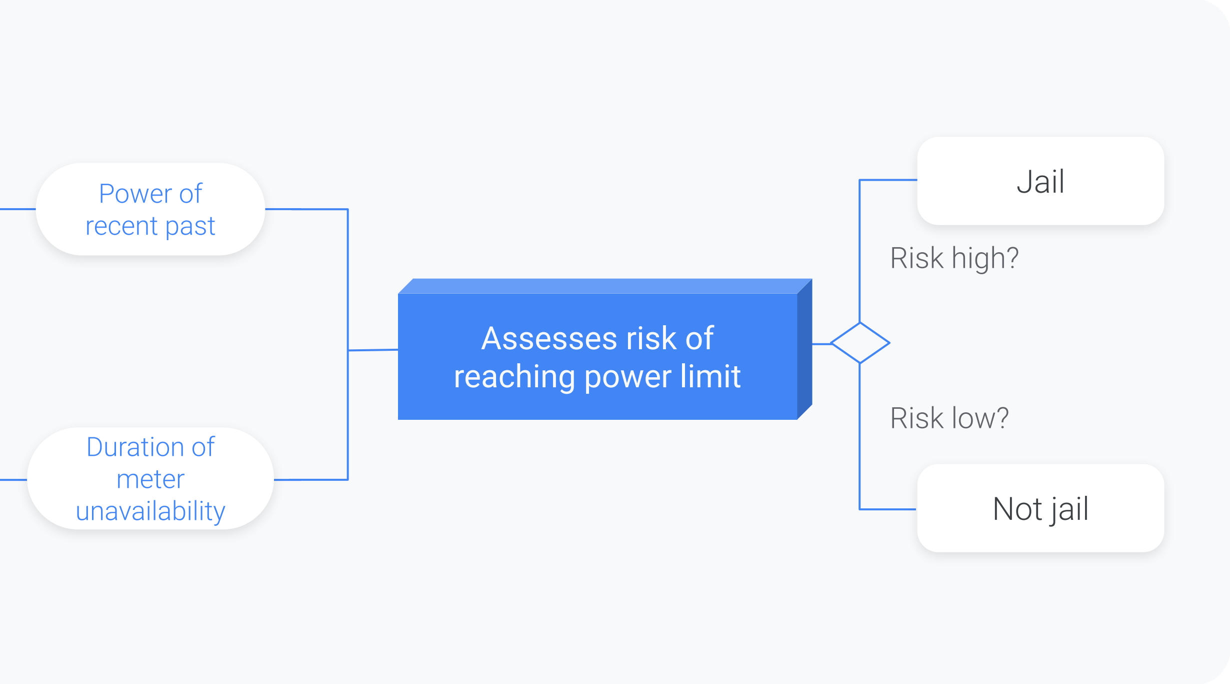
Node mechanism:

CPU jailing

Control policy:

Risk assessment

Proactive capping policy: risk assessment



Risk assessment using a probabilistic model

- Conservative model that has low false negative rate
- Details out of scope for confidentiality; any conservative model that assesses risk and makes a binary decision can be used

Deployment at Scale in Google

Deployed in logs processing clusters

Workloads in logs processing clusters are mostly throughput-oriented but there are also critical latency-sensitive ones

Enabled 9%--25% oversubscription

Throttling has triggered 5 times in 5 clusters in 130 days with negligible performance impact

Summary

1 **Power oversubscription with power capping**	Effective for reducing data center costs Throttling needs to be compatible with SLOs
2 **Thunderbolt power capping system**	Power safety with minimized performance degradation Task-level QoS differentiation Hardware platform independence Tolerance of power telemetry unavailability
3 **Deployment at scale in Google logs processing clusters over years**	9%--25% oversubscription achieved Triggered in production with negligible performance degradation

Thank you