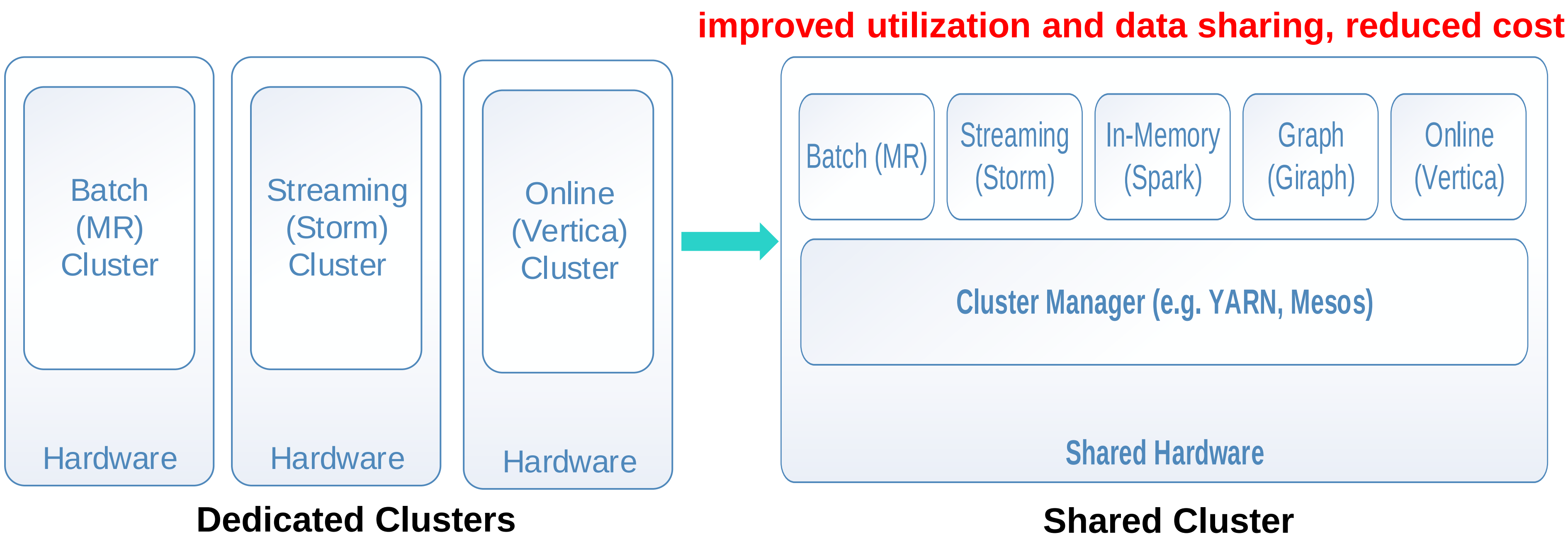
Improving Preemptive Scheduling with Application-Transparent Checkpointing in Shared Clusters

Jack Li, Calton Pu
Georgia Institute of Technology

Yuan Chen, Vanish Talwar, Dejan Milojicic
Hewlett Packard Labs

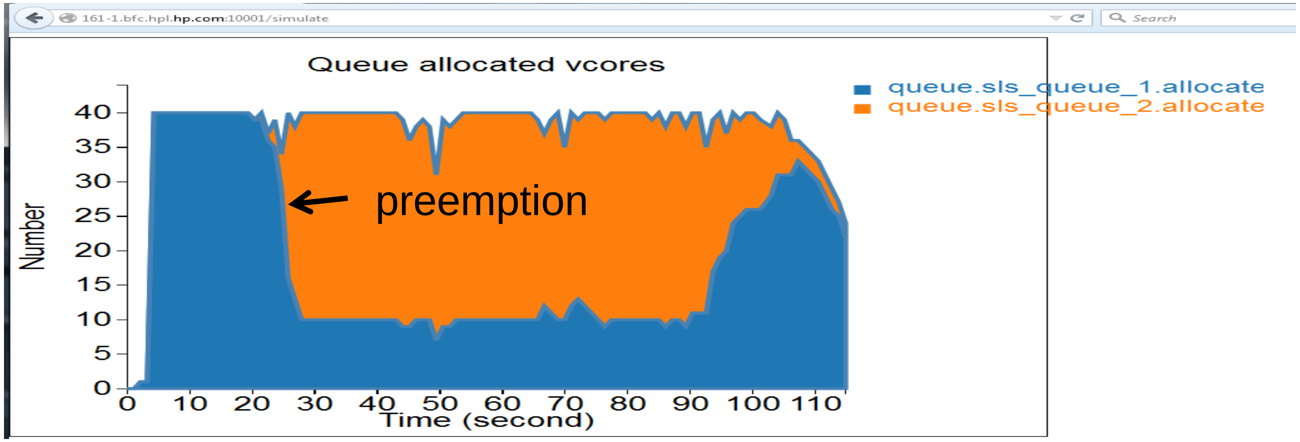
Shared Clusters for Big Data Systems

– Dynamic resource sharing across multiple frameworks, apps and users
  – Examples - Google cluster (Omega), Mesos, Hadoop YARN, Bing's Dryad



Preemption in Shared Clusters

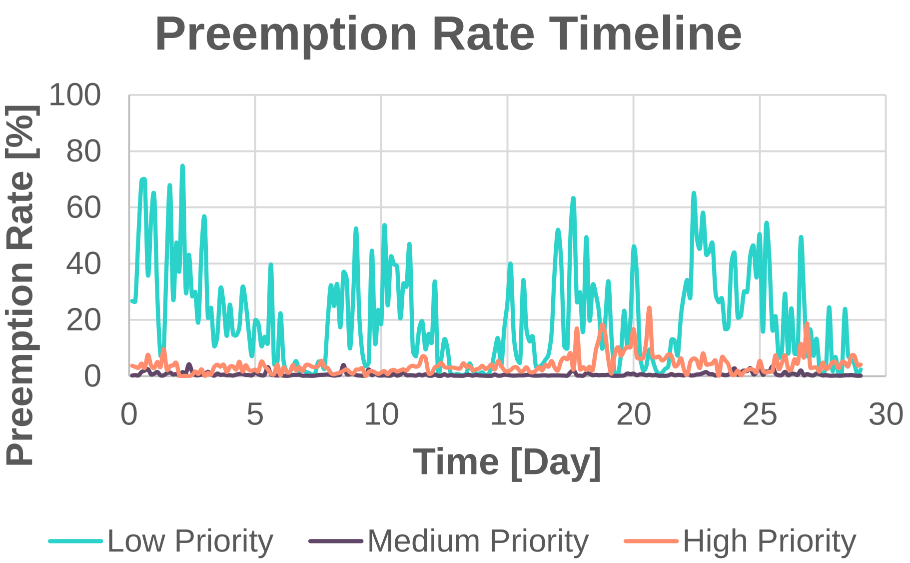
– **Coordinate resource sharing, guarantee QoS and enforce fairness**



– **Problem: preemption in shared clusters is expensive!**
  – Simply kill and restart jobs later
  – Significant resource waste
  – Delays completion time of long running or low priority jobs

Real World Examples

29-day trace from Google: 672,000 jobs on 12,500 machines



Task Priority	Num. of Tasks	Percent Evicted
Free (0-1)	28.4M	20.26%
Middle (2-8)	17.3M	0.55%
Production (9-11)	1.70M	1.02%

Many tasks preempted

Latency Sensitivity	Num. of Tasks	Percent Evicted
0 (lowest)	37.4M	11.76%
1	5.94M	18.87%
2	3.70M	8.14%
3 (highest)	0.28M	14.80%

Even latency-sensitive tasks are evicted

– Google Cluster: 12.4% of scheduled tasks preempted and up to 30k CPU-hours (35% of total capacity) wasted!

– Microsoft Dryad cluster[1]: ~21% jobs killed

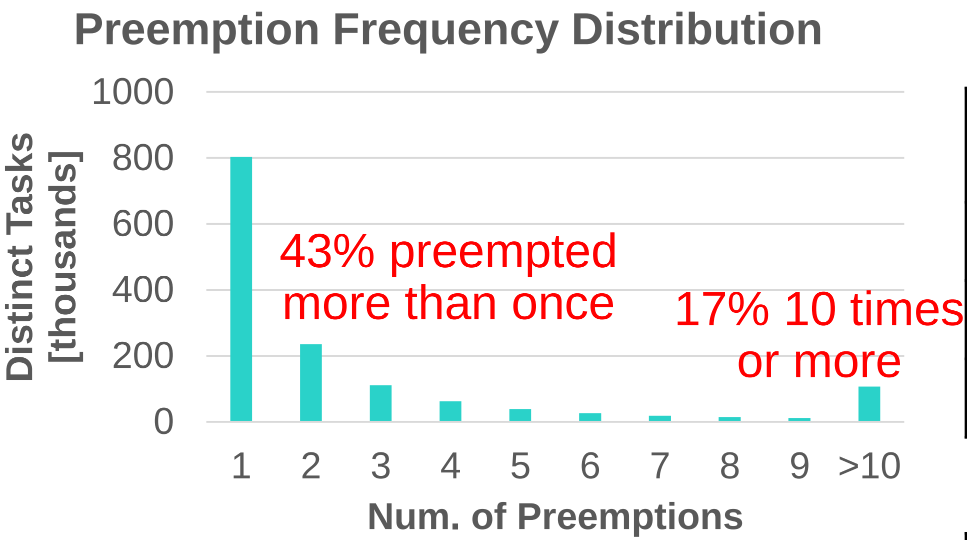
– Facebook Hadoop cluster[2]: repeatedly kill and restart long running jobs

Hewlett Packard Enterprise

[1] Scarlett: Coping with Skewed Popularity Content in MapReduce Clusters. Ananthanarayanan et. al. EuroSys 2011.
[2] Mitigating the Negative Impact of Preemption on Heterogeneous MapReduce Workloads. Cheng et. al. CNSM 2011.

Real World Examples

29-day trace from Google: 672,000 jobs on 12,500 machines



Task Priority	Num. of Tasks	Percent Evicted
Free (0-1)	28.4M	20.26%
Middle (2-8)	17.3M	0.55%
Production (9-11)	1.70M	1.02%

Many tasks preempted

Latency Sensitivity	Num. of Tasks	Percent Evicted
0 (lowest)	37.4M	11.76%
1	5.94M	18.87%
2	3.70M	8.14%
3 (highest)	0.28M	14.80%

Even latency-sensitive tasks are evicted

- Google Cluster: 12.4% of scheduled tasks preempted and up to 30k CPU-hours (35% of total capacity) wasted!
- Microsoft Dryad cluster[1]: ~21% jobs killed
- Facebook Hadoop cluster[2]: repeatedly kill and restart long running jobs

[1] Scarlett: Coping with Skewed Popularity Content in MapReduce Clusters. Ananthanarayanan et. al. EuroSys 2011.
[2] Mitigating the Negative Impact of Preemption on Heterogeneous MapReduce Workloads. Cheng et. al. CNSM 2011.

Checkpointing-based Preemptive Scheduling

Our solution: use checkpoint/restore for preemption instead of kill/restart

Use system level, application-transparent checkpointing mechanism

– Linux CRIU (**C**heckpoint-**R**estore **I**n **U**serspace)

– Distributed and remote checkpoint/restart

Leverage fast storage such as NVM for efficient checkpointing

– Store checkpoints on NVM (NVMFS or NVRAM)

Adaptive preemption policies and optimization techniques

– Combine checkpoint and kill, local and remote checkpointing/resumption

– Incremental checkpointing with memory trackers

Application-transparent Suspend-Resume

Checkpointing using CRIU (Checkpoint/Restore In Userspace)

– Freeze a running program and suspend it in memory or output to disk
– Saves sockets, threads, namespaces, memory mappings, pipes

Dump

– Build process tree from /proc/$pid/task/$tid/children and seize them with ptrace
– Collect VMA areas, file descriptor numbers, registers, etc… of each process

Restore

– Read process tree from file and start saved processes with clone() call
– New memory map created filled with checkpointed data

Suspend-Resume with DFS and NVM

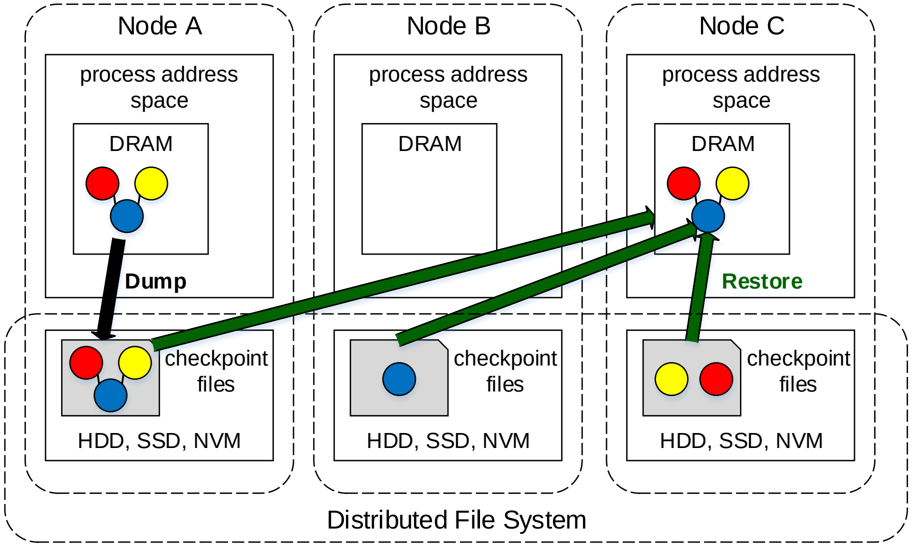
Support distributed and remote checkpoint-resume

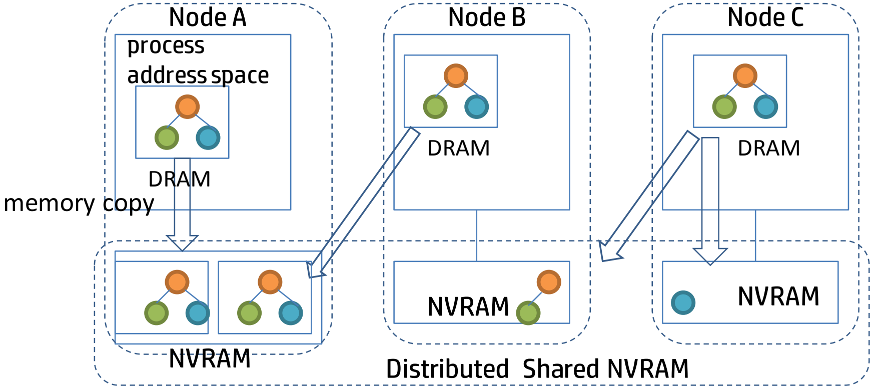
– Save checkpoints on HDFS

Checkpoint with NVM

– Use NVM as fast disk
  – Save CRIU checkpoints in NVM-based file systems (e.g, PMFS)
– Use NVM as virtual memory (NVRAM)
  – Copy checkpoints from DRAM to NVM using memory operations
  – Shadow buffer

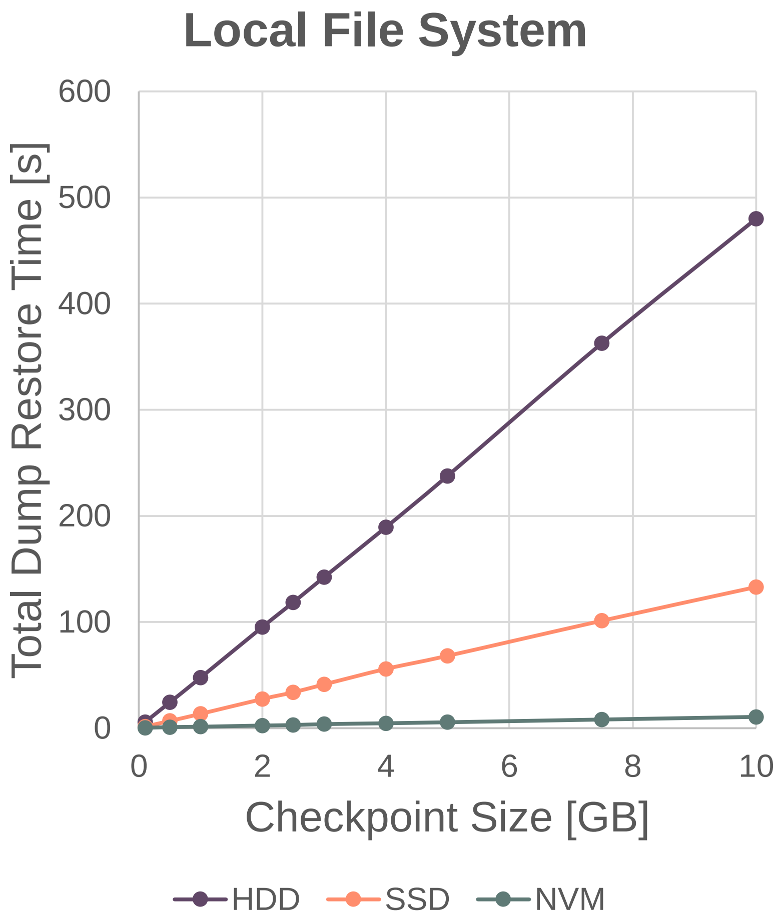
Incremental checkpointing

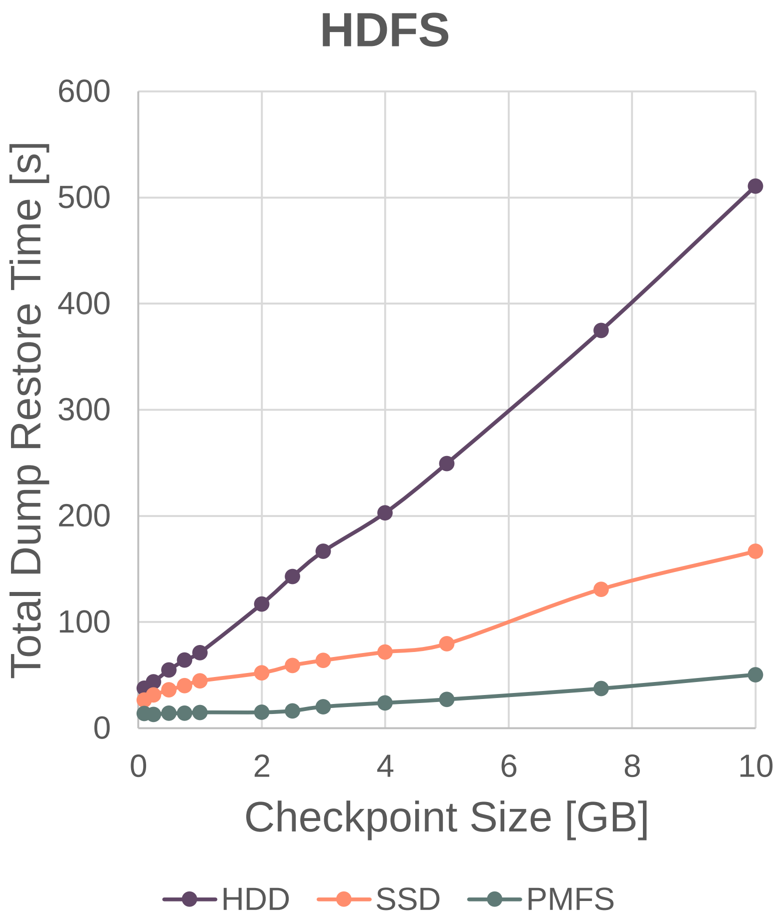




Hewlett Packard Enterprise

Suspend and Restore Performance





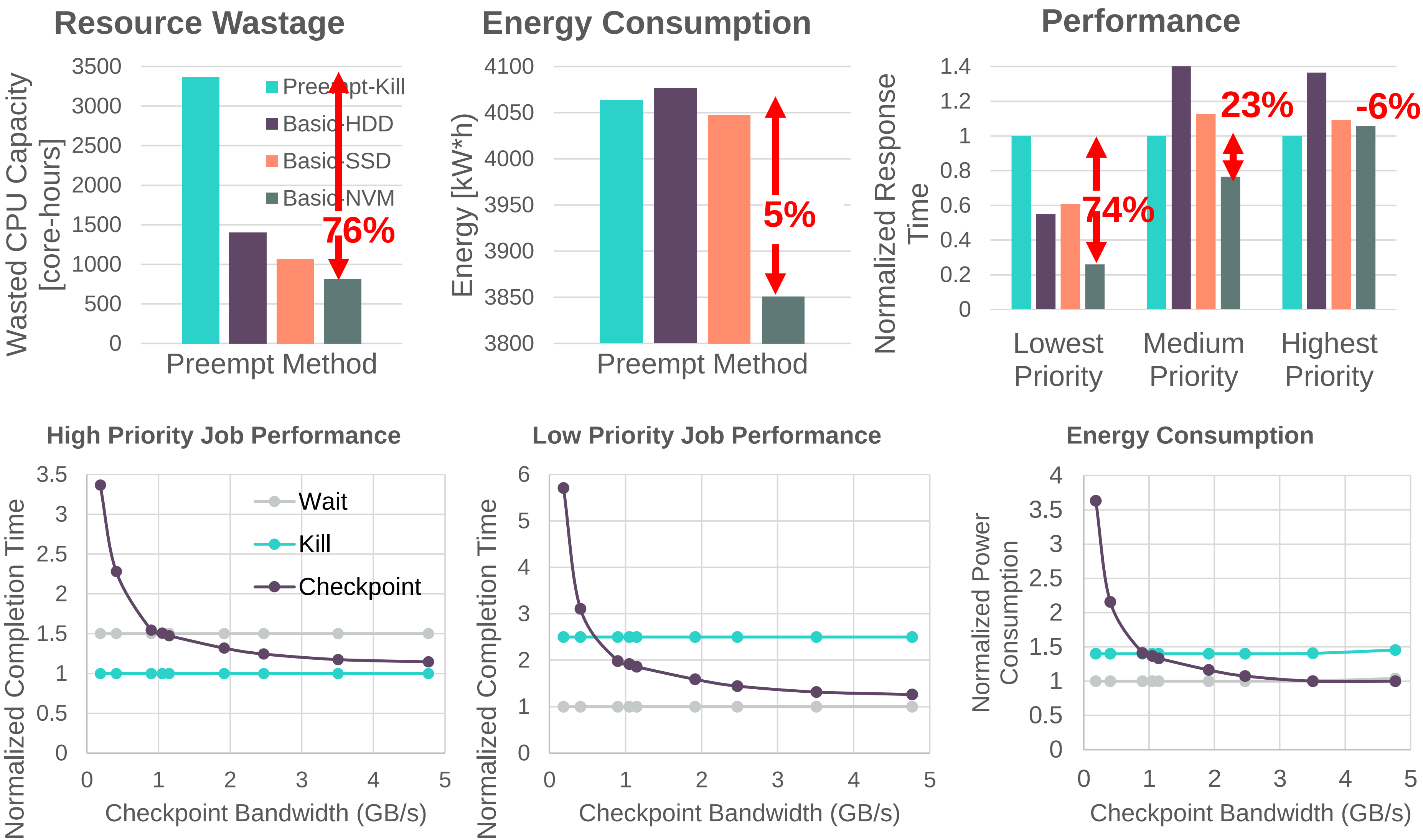
Hewlett Packard Enterprise

Benefits of Incremental Checkpointing

5GB initial dump size, change 10% of the memory and dump again

Storage	First Checkpoint	Second Checkpoint
HDD	169.18s	15.34s
SSD	43.73s	4.08s
PMFS	2.92s	0.28s

Google Trace-driven Simulation



Adaptive Policies and Optimization

Adaptive preemption dynamically selects victim tasks and preemption mechanisms (checkpoint or kill) based on the progress of each task and its checkpoint/restore overhead.

Adaptive resumption restores preempted jobs/tasks locally or remotely according to their overheads and available resources.

Incremental checkpointing with memory trackers

Adaptive Preemption Algorithms

Algorithm 1: Preemption Algorithm

$$overhead_{chkpt} = \frac{size}{bw_{write}} + \frac{size}{bw_{read}} + queue\_time_{dump}$$

```
candidate_victims = get_candidate_victims();
sort(candidate_victims);
for Task t in candidate_victims do
    if t.progress > t.checkpoint_overhead then
        if t.previous_checkpoint != null then
            do_incremental_checkpoint(t);
        else
            do_normal_checkpoint(t);
        end
    else
        kill(t);
    end
end
```

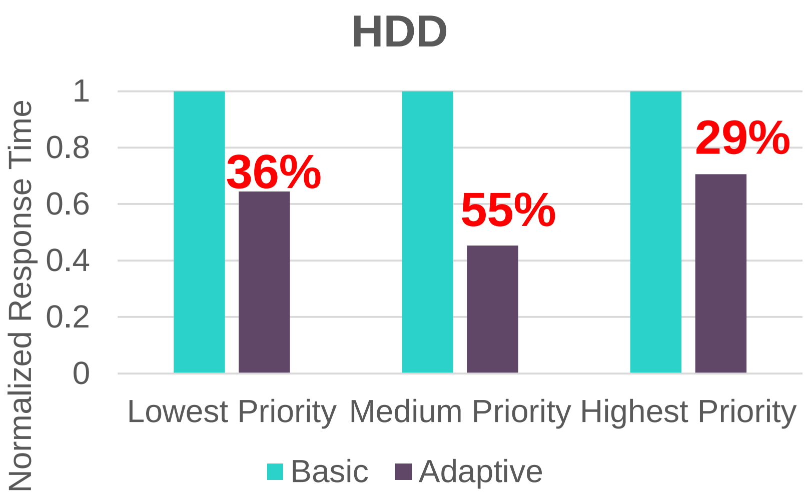
Algorithm 2: Resumption Algorithm

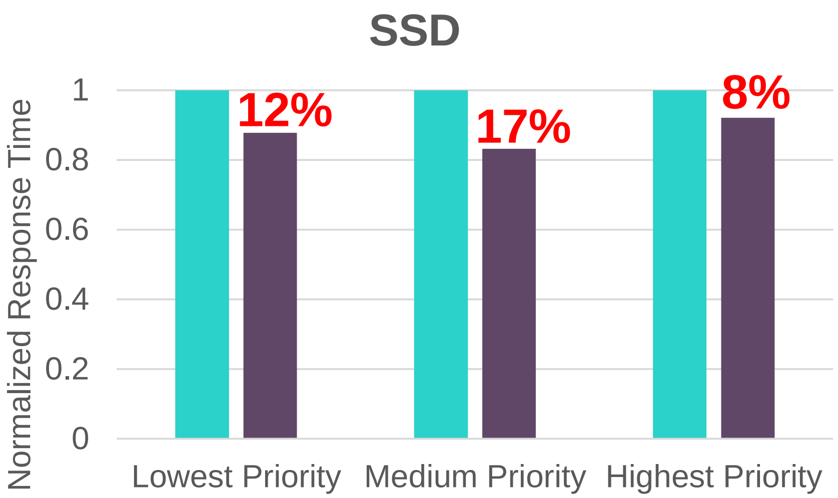
$$overhead_{local} = \frac{size}{bw_{read}} + queue\_time_{local}$$

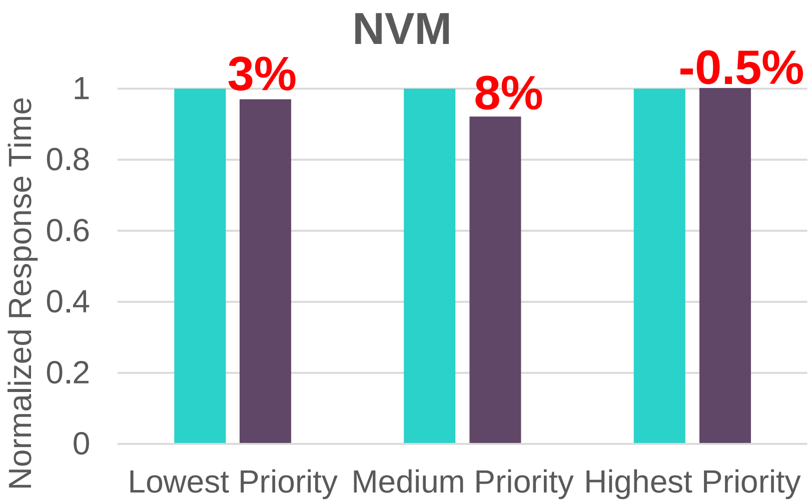
$$overhead_{remote} = \frac{size}{bw_{net}} + \frac{size}{bw_{read}} + queue\_time_{remote}$$

```
preempted_tasks = get_preempted_tasks();
for Task t in preempted_tasks do
    if t.previous_checkpoint == null then
        restart_task(t);
    else
        if t.local_resume_overhead <=
        t.remote_resume_overhead then
            do_local_resume(t);
        else
            do_remote_resume(t);
        end
    end
end
```

Performance Improvement with Adaptive Policies







Hewlett Packard Enterprise

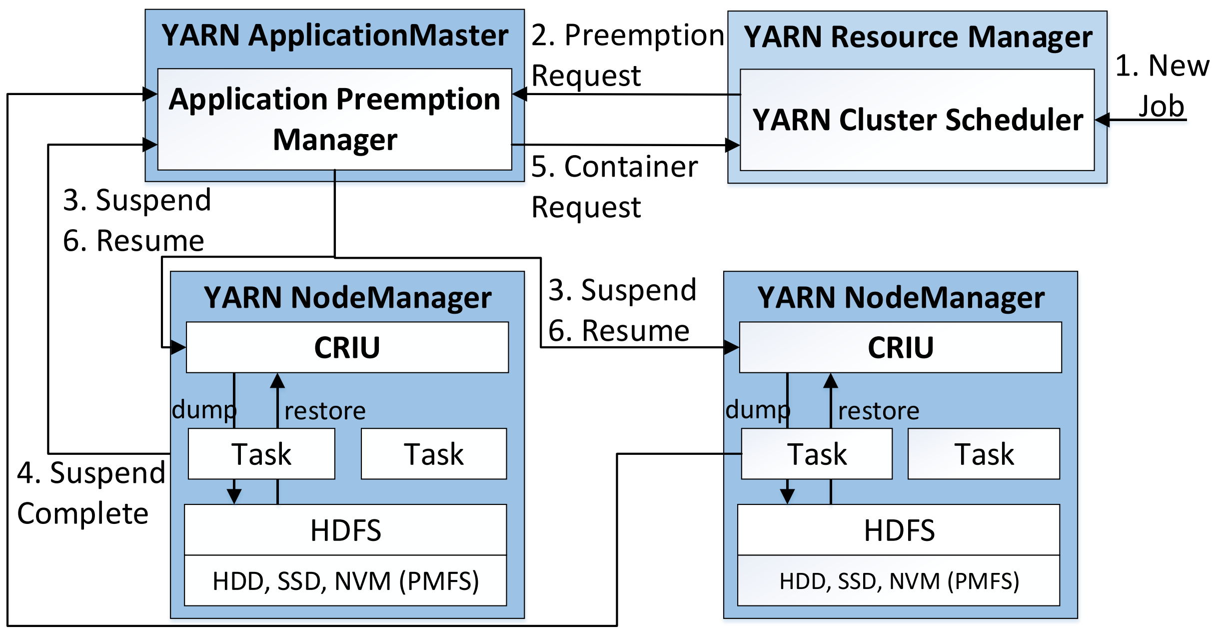
Implementation with Hadoop YARN

YARN – cluster resource manager

- Global resource scheduler (ResourceManager)
- Submit ApplicationMasters (jobs) to RM
- Supports capacity and fair scheduling

DistributedShell

- Comes standard with YARN
- Runs a shell command in a set of containers in a distributed and parallel manner



Hewlett Packard Enterprise

Testbed and Experiment Setup

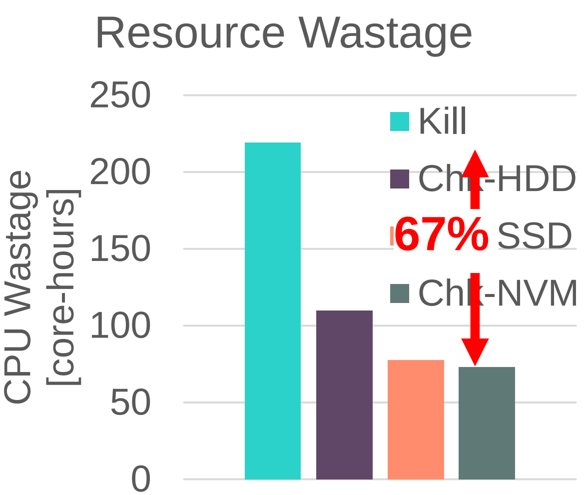
- 8 node Hadoop YARN cluster
  - Dual socket Xeon 5650 CPU (6 cores/each)
  - 96GB memory (48GB emulated NVM using PMFS)
  - 500GB HDD (un-optimized)
  - 120GB SSD
  - 24 concurrent containers (1 CPU/2 GB memory)

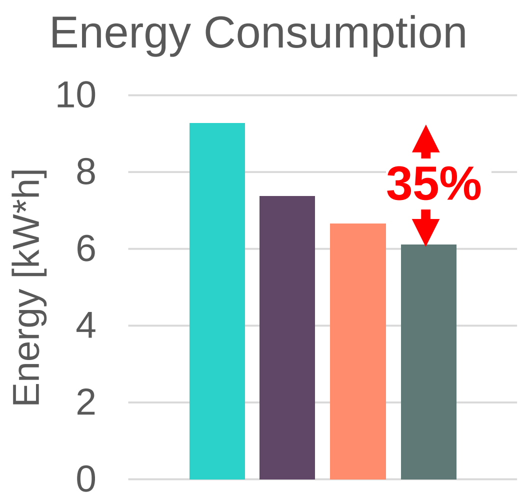
- Workload
  - Modeled after Facebook workload[1]
  - Mix of high/low priority jobs (7,000+ tasks)

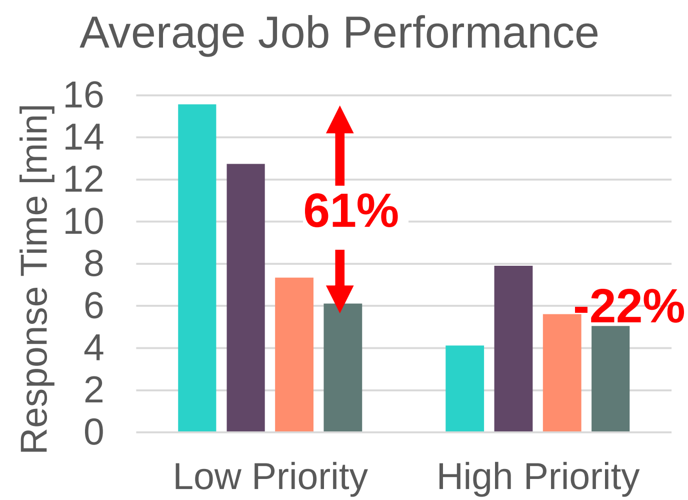
[1] Mitigating the Negative Impact of Preemption on Heterogeneous MapReduce Workloads. Cheng et. al. CNSM 2011.

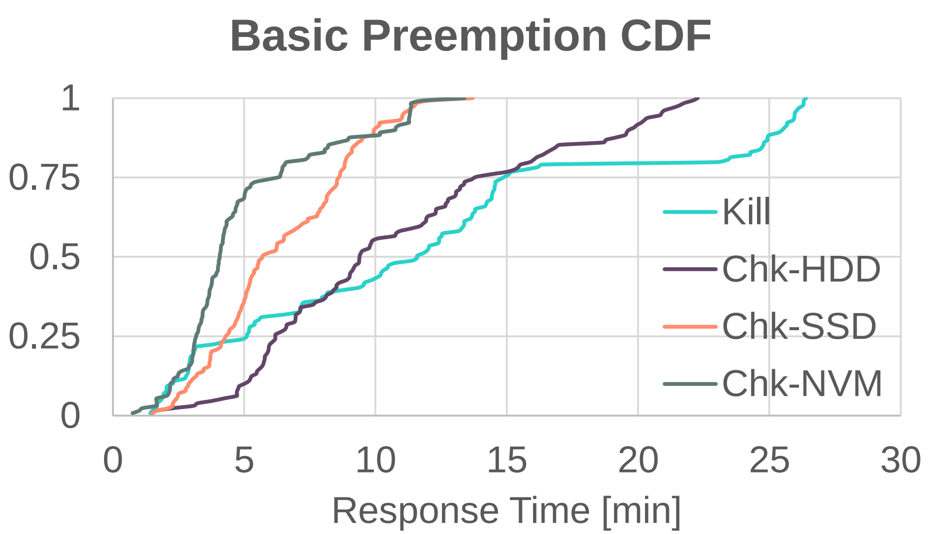
Hewlett Packard Enterprise

Comparison of Different Preemption Policies on YARN

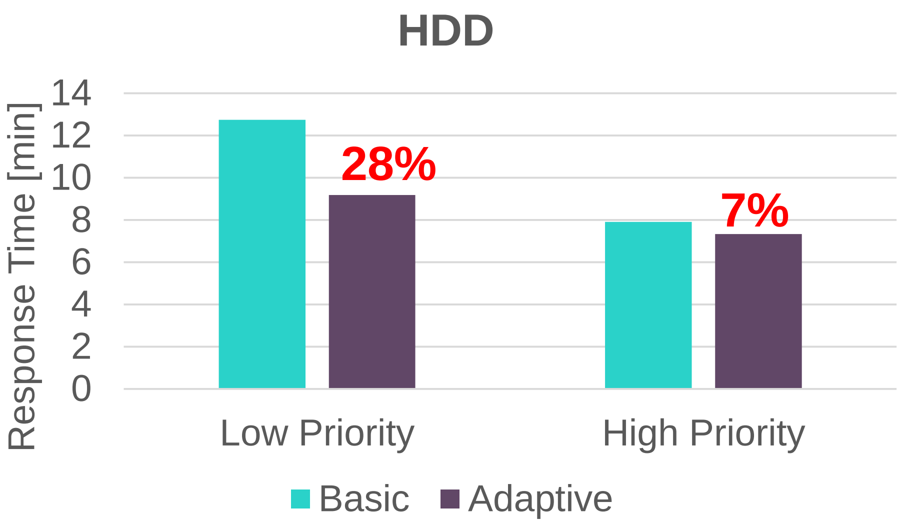


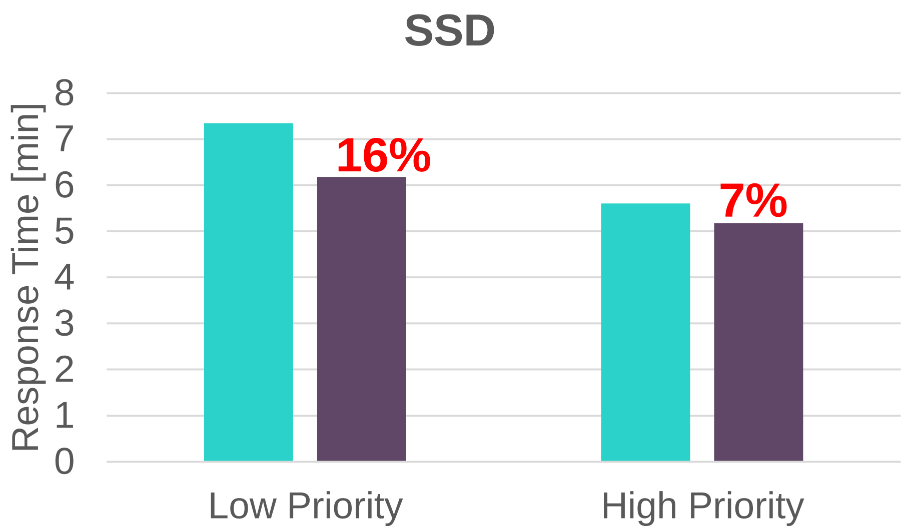


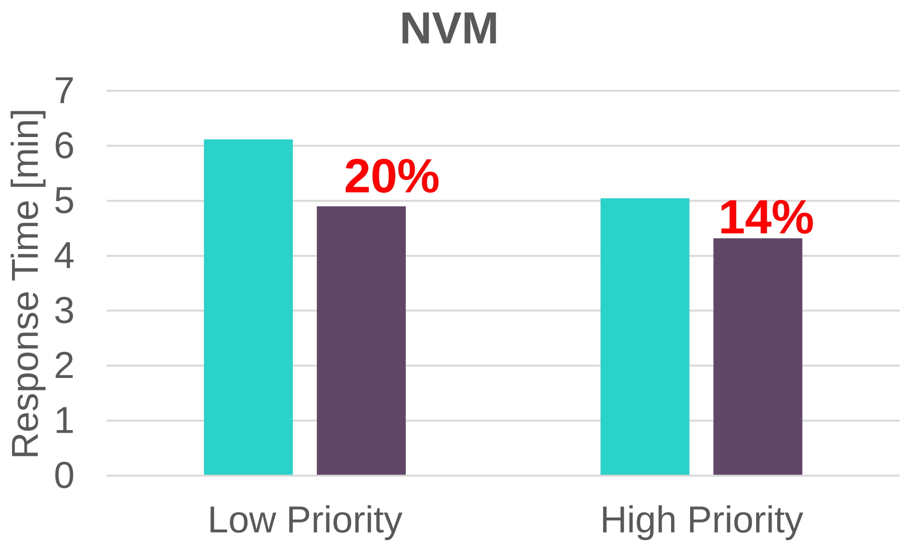




Benefits of Adaptive Preemption



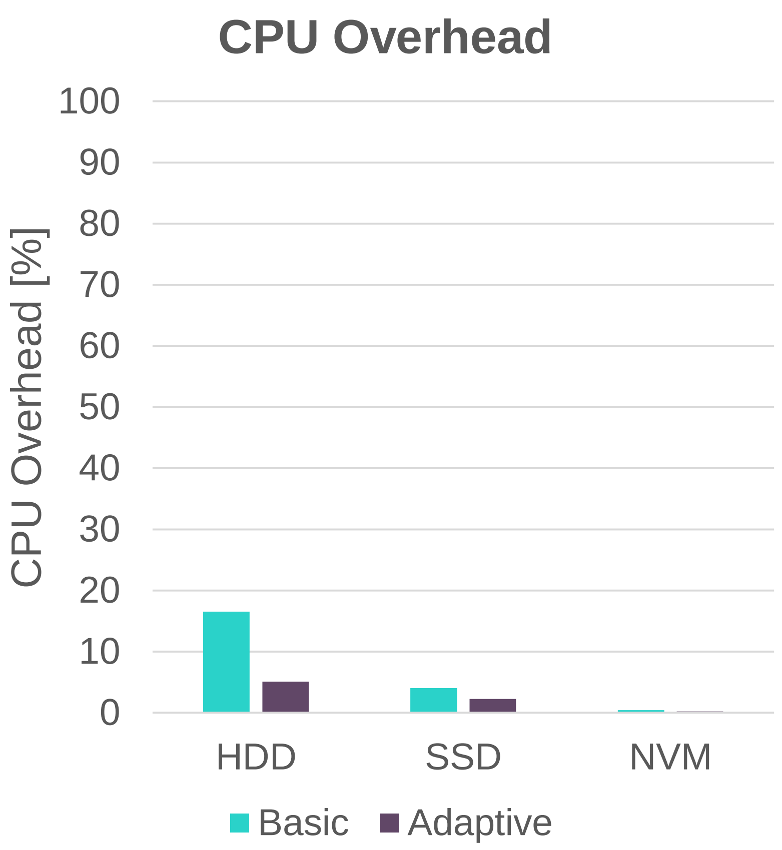


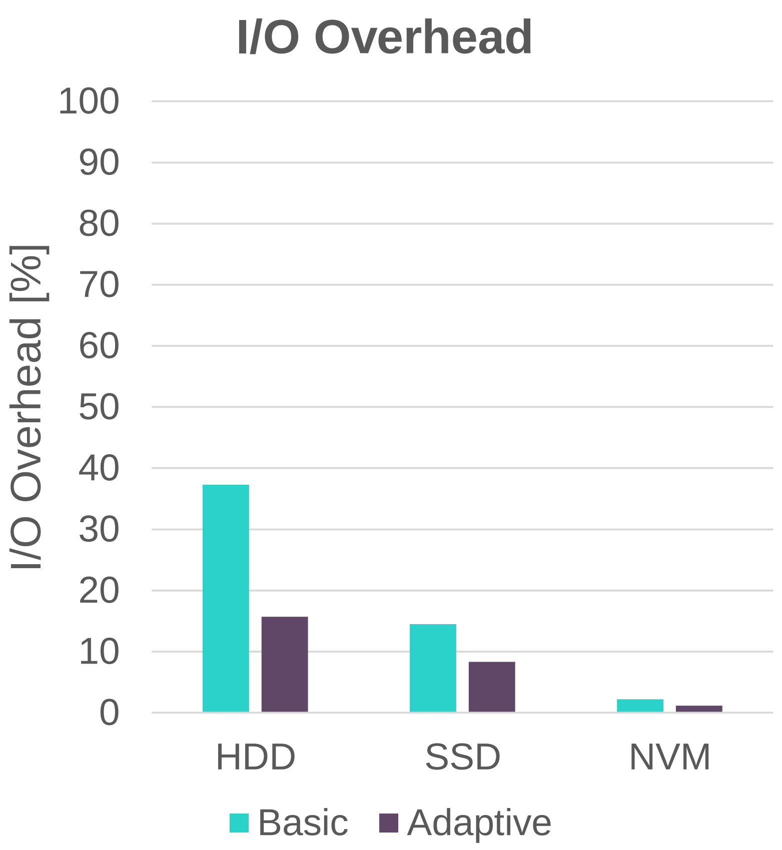


Hewlett Packard Enterprise

Overhead of Checkpoint-based Preemption

CPU overhead is negligible, but I/O overhead is significant on slow storage





Hewlett Packard Enterprise

Conclusion and Future Work

- Preemption in shared clusters is expensive and preemption using application-transparent checkpointing is able to improve resource efficiency and overall application performance.
- Adaptive preemption that combines checkpoint and kill can further improve the performance and reduce the preemption cost.
- By leveraging emerging fast storage technologies such as NVM, even more savings can be achieved.

Future Work

- A wide range of applications
- Checkpointing with NVRAM
- Integration with other cluster scheduling policies

Thank you

Contact: jack.li@cc.gatech.edu
yuan.chen@hpe.com