Resource Utilization Aware Job Scheduling to Mitigate Performance Variability

Daniel Nichols†, Aniruddha Marathe*, Kathleen Shoga*, Todd Gamblin*, Abhinav Bhave†

† University of Maryland, College Park
* Lawrence Livermore National Laboratory
Performance Variation

- Same job can vary significantly in run time
Causes of Performance Variation

- System noise
- Software bugs
- Hardware performance degradation
- Shared resource contention
Mitigating Variability from Shared Resource Contention

- Adaptive in-flight message rerouting
- More bandwidth
- Resource utilization aware job scheduling
RUSH: Resource Utilization-aware Scheduler for HPC

- Machine learning can predict future variation
- Schedule jobs with \textit{apriori} knowledge of variation
Predicting Variation

- **Model Input**
  - System state
  - Job description

- **Model Output**
  - 1 if job will experience variation; 0 otherwise
  - Variation: >1.5 st. devs. from average run time
Building a Dataset

- Proxy applications
  - Kripke, AMG, Laghos, SWFFT, sw4lite, LBANN, pennant
- Run each 3x a day from August 2020 - February 2021 on Quartz system at LLNL
  - Record performance (walltime)
  - Collect IO and Network counters with LDMS (5 mins. before job)
  - Collect network benchmarks
Model Selection

- Train AdaBoost, DecisionForest, ExtraTrees, kNN
  - Record F1-score using stratified k-fold cross validation
- Choose model with highest F1-score
Feature Selection

- Recursive feature elimination
- Select 20 best features
  - xmit_rate, recv_rate, xmit_discards, mpisend_time, mpirecv_time
- Reduces latency collecting features
Traditional Scheduling

Input $Q$ ← queue of jobs
   $M$ ← ML model
   $S$ ← current machine state
   $\text{SkipTable}$ ← Count of times skipped for each job
   $R_1$ ← Queue ordering policy
   $R_2$ ← Backfill ordering policy

1. sort $Q$ according to $R_1$
2. for job $j \in Q$ do
3.     if $j$ can be started currently then
4.         pop $j$ from $Q$
5.         $\text{Start}(j, Q, M, S, \text{SkipTable})$
6.     else
7.         Reserve $j$ at earliest possible time
8.         $L \leftarrow Q \setminus \{j\}$
9.         sort $L$ according to $R_2$
10.    for job $j' \in L$ do
11.        if $j'$ can be started currently without delaying reservation of $j$ then
12.            pop $j'$ from $Q$
13.            $\text{Start}(j', Q, M, S, \text{SkipTable})$
14.        end if
15.    end for
16.   end if
17. end for
Traditional Scheduling

Input $Q \leftarrow$ queue of jobs
   $M \leftarrow$ ML model
   $S \leftarrow$ current machine state
   $\text{SkipTable} \leftarrow$ Count of times skipped for each job
   $\mathcal{R}_1 \leftarrow$ Queue ordering policy
   $\mathcal{R}_2 \leftarrow$ Backfill ordering policy

1. sort $Q$ according to $\mathcal{R}_1$
2. for job $j \in Q$ do
3.   if $j$ can be started currently then
4.      pop $j$ from $Q$
5.      $\text{Start}(j, Q, M, S, \text{SkipTable})$
6.   else
7.      Reserve $j$ at earliest possible time
8.      $L \leftarrow Q \setminus \{j\}$
9.   sort $L$ according to $\mathcal{R}_2$
10. for job $j' \in L$ do
11.   if $j'$ can be started currently without delaying reservation of $j$ then
12.      pop $j'$ from $Q$
13.      $\text{Start}(j', Q, M, S, \text{SkipTable})$
14. end if
15. end for
16. break
17. end if
18. end for
Traditional Scheduling

Run jobs that can be immediately started

```
Input Q ← queue of jobs
M ← ML model
S ← current machine state
SkipTable ← Count of times skipped for each job
R₁ ← Queue ordering policy
R₂ ← Backfill ordering policy

1 sort Q according to R₁
for job j ∈ Q do
  if j can be started currently then
    pop j from Q
    Start(j, Q, M, S, SkipTable)
  else
    Reserve j at earliest possible time
    L ← Q \ {j}
    sort L according to R₂
    for job j' ∈ L do
      if j' can be started currently without delaying reservation of j then
        pop j' from Q
        Start(j', Q, M, S, SkipTable)
      end if
    end for
  end if
end for
```
Traditional Scheduling

\[\text{Input } Q \leftarrow \text{queue of jobs}\]
\[M \leftarrow \text{ML model}\]
\[S \leftarrow \text{current machine state}\]
\[\text{SkipTable} \leftarrow \text{Count of times skipped for each job}\]
\[R_1 \leftarrow \text{Queue ordering policy}\]
\[R_2 \leftarrow \text{Backfill ordering policy}\]

1. sort \( Q \) according to \( R_1 \)
2. for job \( j \in Q \) do
3. if \( j \) can be started currently then
4. pop \( j \) from \( Q \)
5. \( \text{Start}(j, Q, M, S, \text{SkipTable}) \)
6. else
7. Reserve \( j \) at earliest possible time
8. \( L \leftarrow Q \setminus \{j\} \)
9. sort \( L \) according to \( R_2 \)
10. for job \( j' \in L \) do
11. if \( j' \) can be started currently without delaying reservation of \( j \) then
12. pop \( j' \) from \( Q \)
13. \( \text{Start}(j', Q, M, S, \text{SkipTable}) \)
14. end if
15. end for
16. break
17. end if
18. end for

Reserve jobs that cannot be started immediately
Traditional Scheduling

Input $Q \leftarrow$ queue of jobs
$M \leftarrow$ ML model
$S \leftarrow$ current machine state
SkipTable $\leftarrow$ Count of times skipped for each job
$R_1 \leftarrow$ Queue ordering policy
$R_2 \leftarrow$ Backfill ordering policy

1 sort $Q$ according to $R_1$
2 for job $j \in Q$ do
3     if $j$ can be started currently then
4         pop $j$ from $Q$
5         $Start(j, Q, M, S, \text{SkipTable})$
6     else
7         Reserve $j$ at earliest possible time
8         $L \leftarrow Q \setminus \{j\}$
9     sort $L$ according to $R_2$
10    for job $j' \in L$ do
11       if $j'$ can be started currently without delaying reservation of $j$ then
12          pop $j'$ from $Q$
13          $Start(j', Q, M, S, \text{SkipTable})$
14       end if
15     end for
16 break
17 end if
18 end for

Backfill remaining jobs
Traditional Scheduling

Input $Q$ ← queue of jobs
$M$ ← ML model
$S$ ← current machine state
SkipTable ← Count of times skipped for each job
$R_1$ ← Queue ordering policy
$R_2$ ← Backfill ordering policy

1. sort $Q$ according to $R_1$
2. for job $j \in Q$ do
3. if $j$ can be started currently then
4. remove $j$ from $Q$
5. $\text{Start}(j, Q, M, S, \text{SkipTable})$
6. else
7. Reserve $j$ at earliest possible time
8. $L \leftarrow Q \setminus \{j\}$
9. sort $L$ according to $R_2$
10. for job $j' \in L$ do
11. if $j'$ can be started currently without delay of $j$ then
12. remove $j'$ from $Q$
13. $\text{Start}(j', Q, M, S, \text{SkipTable})$
14. end if
15. end for
16. break
17. end if
18. end for

RUSH only modifies the start function
Variation-Aware Scheduling

**Start Function**

Input $j \leftarrow \text{job}$

- $Q \leftarrow \text{scheduler queue}$
- $M \leftarrow \text{ML model}$
- $S \leftarrow \text{current machine state}$
- SkipTable $\leftarrow \text{Count of times skipped for each job}$

1. if SkipTable[$j$] < $j\text{.skip\_threshold}$ and $M(j, S) \in \text{variation labels}$ then
2.SkipTable[$j$] $\leftarrow$ SkipTable[$j$] + 1
3. push $j$ after front of $Q$
4. else
5. launch job $j$
6. end if
Variation-Aware Scheduling

**Start Function**

```
Input $j \leftarrow \text{job}$

$Q \leftarrow \text{scheduler queue}$
$M \leftarrow \text{ML model}$
$S \leftarrow \text{current machine state}$
$\text{SkipTable} \leftarrow \text{count of times skipped for } j$

1. **if** $\text{SkipTable}[j] < j\.skip\_threshold \text{ and } M(j, S) \in \text{variation labels} \text{ then}$
2. $\text{SkipTable}[j] \leftarrow \text{SkipTable}[j] + 1$
3. $\text{push } j \text{ after front of } Q$
4. **else**
5. $\text{launch job } j$
6. **end if**
```

If model predicts variation, then put job back on top of queue.
Variation-Aware Scheduling

Start Function

\textbf{Input} \ j \leftarrow \text{job}  \\
\quad \text{Q} \leftarrow \text{scheduler queue}  \\
\quad \text{M} \leftarrow \text{ML model}  \\
\quad \text{S} \leftarrow \text{current machine state}  \\
\quad \text{SkipTable} \leftarrow \text{Count of times skipped for each job}  \\

1. \textbf{if} \ \text{SkipTable}[j] < j.\text{skip\_threshold} \ \textbf{and} \ \text{M}(j, S) \in \text{variation labels} \ \textbf{then}  \\
2. \quad \text{SkipTable}[j] \leftarrow \text{SkipTable}[j] + 1  \\
3. \quad \text{push} \ j \ \text{after front of} \ \text{Q}  \\
4. \quad \textbf{else}  \\
5. \quad \textbf{end if}  \\
6. \quad \text{launch job} \ j  \\

Otherwise run job as normal
Variation-Aware Scheduling

**Start Function**

```
Input: j \leftarrow job
Q \leftarrow \text{scheduler queue}
M \leftarrow \text{ML model}
S \leftarrow \text{current machine state}
SkipTable \leftarrow \text{Count of times skipped for each job}

1. if SkipTable[j] < j.skip\_threshold and M(j, S) \in \text{variation labels} then
2. \quad \text{SkipTable}[j] \leftarrow \text{SkipTable}[j] + 1
3. \quad \text{push } j \text{ after front of } Q
4. else
5. \quad \text{launch job } j
6. end if
```

Limit skips to prevent job starvation
Implementation

- Machine learning trained and exported with SciKit
- Extend Flux\(^1\) to implement RUSH

\(^1\) https://flux-framework.org/
Experiments

- Run simulated workload on Quartz
  - 512 node allocation
  - ~190 jobs with 1 hour makespan
  - Run FCFS+EASY (5x) and RUSH (5x)
  - Record makespan, average wait time, and # jobs experiencing variation
Results: All Data All Applications

- Model trained on entire dataset, running all apps
- Variation drops significantly

RUSH reduces # jobs with variation
Results: All Data All Applications

- Model trained on entire dataset, running all apps

  RUSH reduces max run time

  RUSH reduces range of run times
Results: Partial Data Partial Applications

- Test generalizability
- Train model on AMG, Kripke, sw4lite, and SWFFT data
Results: Partial Data Partial Applications

- Model trained on some apps, while running other apps

RUSH reduces # jobs with variation

![Number of Occurrences of Variation (PDPA) Graph]

Number of jobs with variation:
- **Laghos**: FCFS, RUSH + PDPA
- **LBANN**: FCFS, RUSH + PDPA
Results: Partial Data Partial Applications

- Model trained on some apps, while running other apps

**RUSH reduces max run time**

**RUSH reduces range of run times**
Results: Partial Data Partial Applications

- Model trained on some apps, while running other apps

RUSH generalizes to apps it has not seen
Results: Throughput

All 5 experiments in paper had an improvement in makespan
Conclusion

- Collect historical performance data
- Train machine learning models to predict variation
- Use variation prediction to schedule jobs
- Reduce max run time by up to 5.8% and average number of runs with variation from 17 to 4