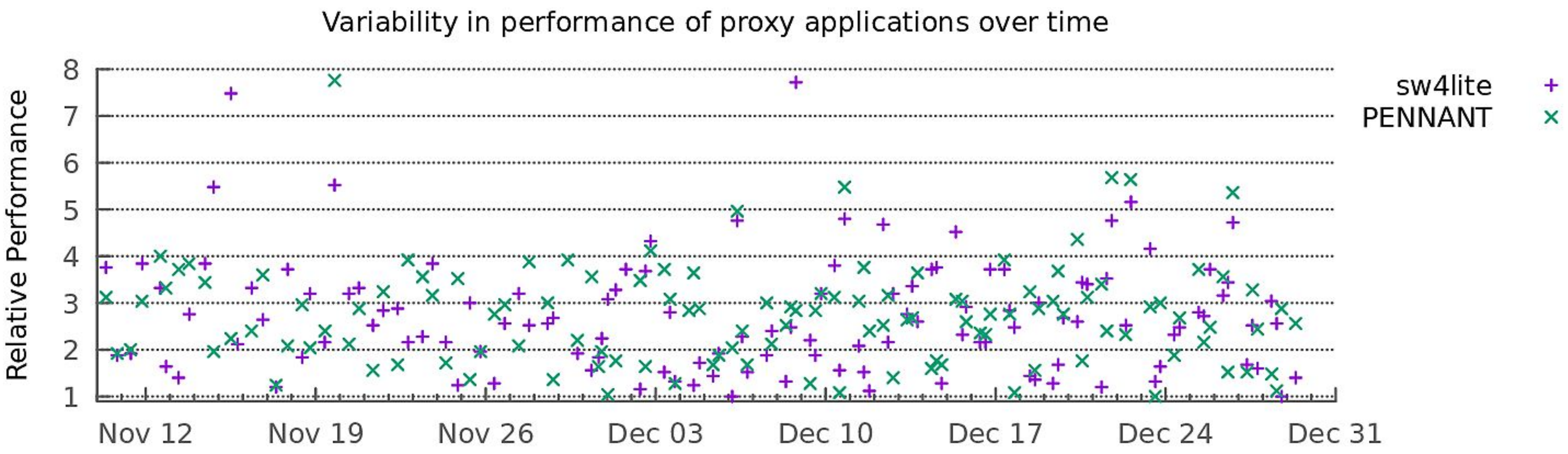
Resource Utilization Aware Job Scheduling to Mitigate Performance Variability

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

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

Performance Variation

- Same job can vary significantly in run time



Causes of Performance Variation

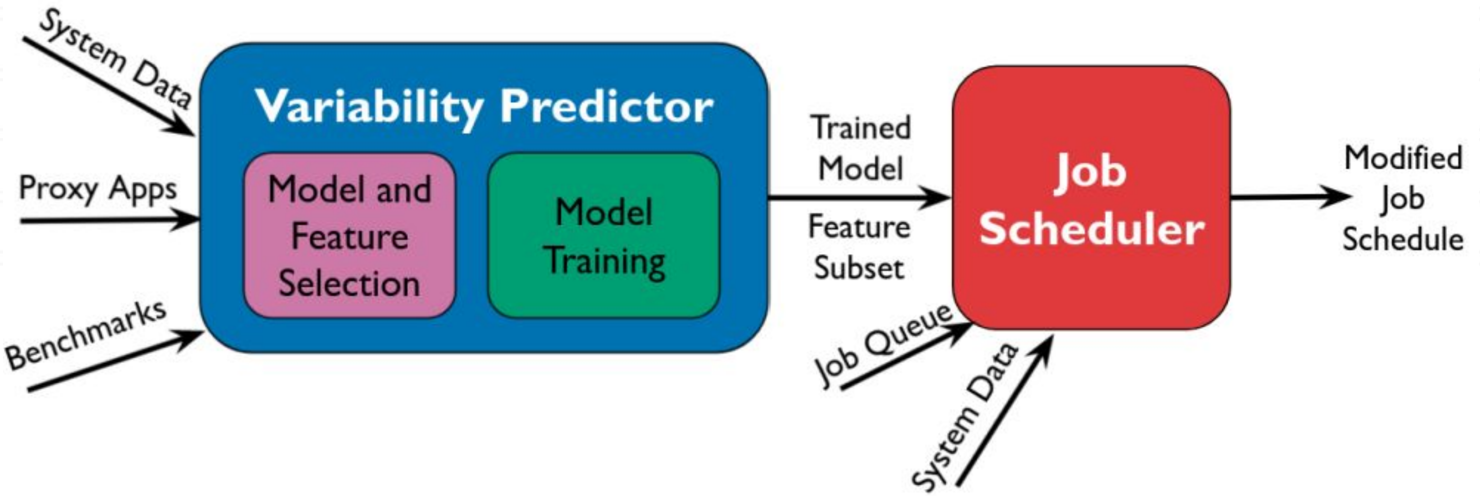
- System noise
- Software bugs
- Hardware performance degradation
- Shared resource contention

Mitigating Variability from Shared Resource Contention

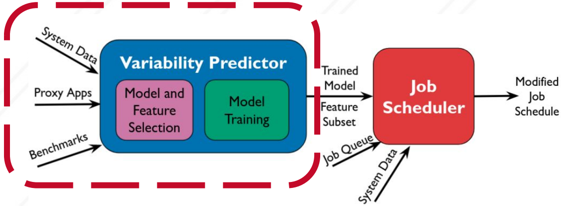
- Adaptive in-flight message rerouting
- More bandwidth
- Resource utilization aware job scheduling

RUSH: Resource Utilization-aware Scheduler for HPC

- Machine learning can predict future variation
- Schedule jobs with *apriori* knowledge of variation

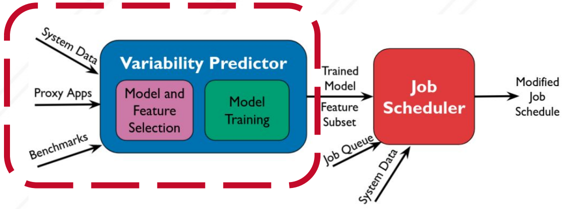


Predicting Variation



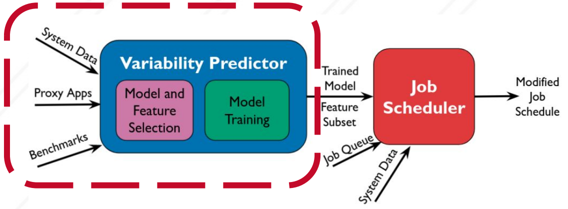
- Model Input
  - System state
  - Job description
- Model Output
  - 1 if job will experience variation; 0 otherwise
  - variation: >1.5 st. devs. from average run time

Building a Dataset

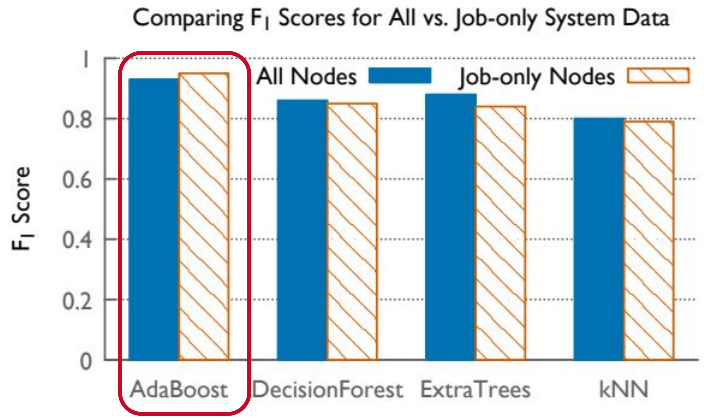


- Proxy applications
  - Kripke, AMG, Laghos, SWFFT, sw4lite, LBANN, pennant
- Run each 3x a day from August 2020 - February 2021 on Quartz system at LLNL
  - Record performance (walltime)
  - Collect IO and Network counters with LDMS (5 mins. before job)
  - Collect network benchmarks

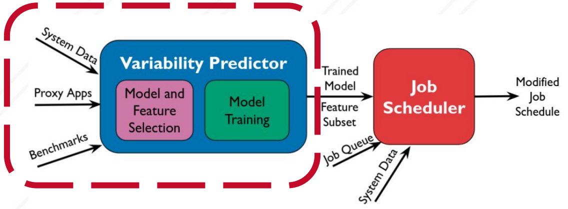
Model Selection



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

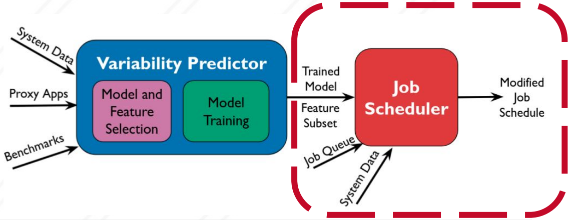


Feature Selection



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

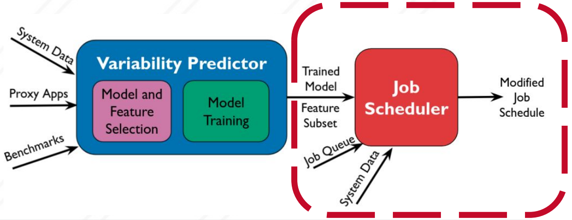
Traditional Scheduling



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

sort Q according to $\mathcal{R}_1$
for job $j \in Q$ **do**
    if j can be started currently **then**
        pop j from Q
        $Start(j, Q, M, S, \mathsf{SkipTable})$
    else
        Reserve j at earliest possible time
        $L \leftarrow Q \setminus \{j\}$
        sort L according to $\mathcal{R}_2$
        for job $j' \in L$ **do**
            if j' can be started currently without delaying reservation of j **then**
                pop j' from Q
                $Start(j', Q, M, S, \mathsf{SkipTable})$
            end if
        end for
        break
    end if
end for

Traditional Scheduling

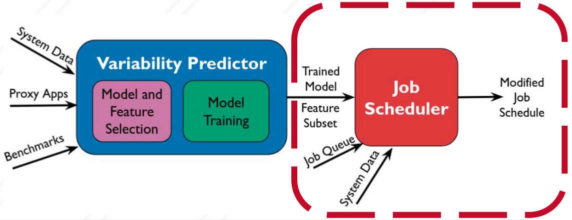


Input $Q \leftarrow$ queue of jobs
$M \leftarrow$ ML model
$S \leftarrow$ current machine state
SkipTable $\leftarrow$ Count of times skipped for each job
$\mathcal{R}_1 \leftarrow$ Queue ordering policy
$\mathcal{R}_2 \leftarrow$ Backfill ordering policy

sort Q according to $\mathcal{R}_1$
for job $j \in Q$ **do**
  if j can be started currently **then**
    pop j from Q
    $Start(j, Q, M, S, \text{SkipTable})$
  else
    Reserve j at earliest possible time
    $L \leftarrow Q \setminus \{j\}$
    sort L according to $\mathcal{R}_2$
    for job $j' \in L$ **do**
      if j' can be started currently without delaying reservation of j **then**
        pop j' from Q
        $Start(j', Q, M, S, \text{SkipTable})$
      end if
    end for
    break
  end if
end for

Sort queue Q with policy R_1

Traditional Scheduling

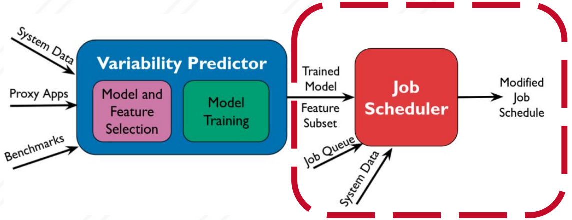



```
Input Q ← queue of jobs
      M ← ML model
      S ← current machine state
      SkipTable ← Count of times skipped for each job
      R1 ← Queue ordering policy
      R2 ← Backfill ordering policy

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
        break
    end if
end for
```

Run jobs that can be immediately started

Traditional Scheduling

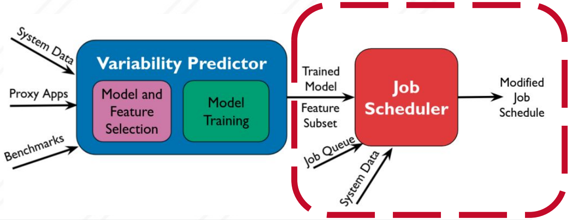



```
Input Q ← queue of jobs
      M ← ML model
      S ← current machine state
      SkipTable ← Count of times skipped for each job
      R₁ ← Queue ordering policy
      R₂ ← Backfill ordering policy

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
        break
    end if
end for
```

Reserve jobs that cannot be started immediately

Traditional Scheduling

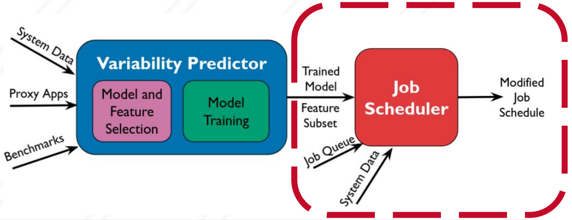



```
Input Q ← queue of jobs
      M ← ML model
      S ← current machine state
      SkipTable ← Count of times skipped for each job
      R1 ← Queue ordering policy
      R2 ← Backfill ordering policy

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
        break
    end if
end for
```

Backfill remaining jobs

Traditional Scheduling

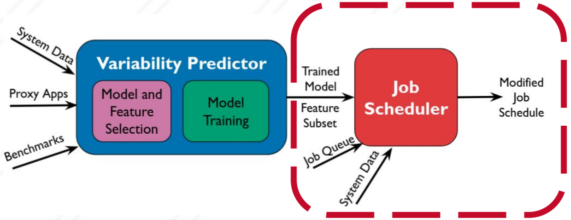


Input $Q \leftarrow$ queue of jobs
$M \leftarrow$ ML model
$S \leftarrow$ current machine state
SkipTable $\leftarrow$ Count of times skipped for each job
$\mathcal{R}_1 \leftarrow$ Queue ordering policy
$\mathcal{R}_2 \leftarrow$ Backfill ordering policy

sort Q according to $\mathcal{R}_1$
for job $j \in Q$ **do**
  if j can be started currently **then**
    pop j from Q
    $\boxed{Start(j, Q, M, S, \mathsf{SkipTable})}$
  else
    Reserve j at earliest possible time
    $L \leftarrow Q \setminus \{j\}$
    sort L according to $\mathcal{R}_2$
    for job $j' \in L$ **do**
      if j' can be started currently without del[ay]
      of j **then**
        pop j' from Q
        $\boxed{Start(j', Q, M, S, \mathsf{SkipTable})}$
      end if
    end for
    break
  end if
end for

RUSH only modifies the *start* function

Variation-Aware Scheduling

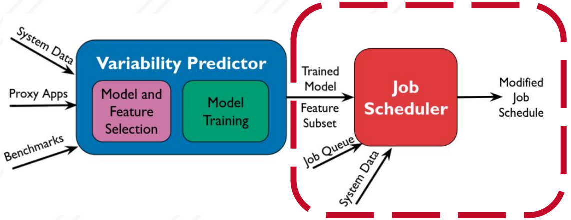


Start Function

Input $j \leftarrow$ job
$Q \leftarrow$ scheduler queue
$M \leftarrow$ ML model
$S \leftarrow$ current machine state
SkipTable $\leftarrow$ Count of times skipped for each job

if SkipTable$[j] < j.skip_threshold$ **and** $M(j, S) \in$ variation labels **then**
    SkipTable$[j] \leftarrow$ SkipTable$[j] + 1$
    push j after front of Q
else
    launch job j
end if

Variation-Aware Scheduling



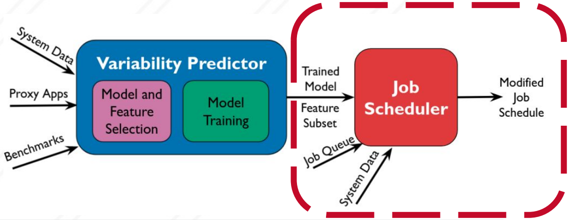
Start Function

Input $j \leftarrow$ job
$Q \leftarrow$ scheduler queue
$M \leftarrow$ ML model
$S \leftarrow$ current machine state
$\mathsf{SkipTable} \leftarrow$ Count of times skipped fc

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

If model predicts variation, then put job back on top of queue

Variation-Aware Scheduling



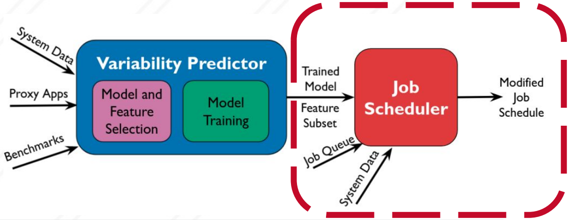
Start Function

Input $j \leftarrow$ job
$Q \leftarrow$ scheduler queue
$M \leftarrow$ ML model
$S \leftarrow$ current machine state
SkipTable $\leftarrow$ Count of times skipped for each job

if SkipTable$[j] < j.skip_threshold$ **and** $M(j, S) \in$ variation labels **then**
  SkipTable$[j] \leftarrow$ SkipTable$[j] + 1$
  push j after front of Q
else
  launch job j
end if

Otherwise run job as normal

Variation-Aware Scheduling



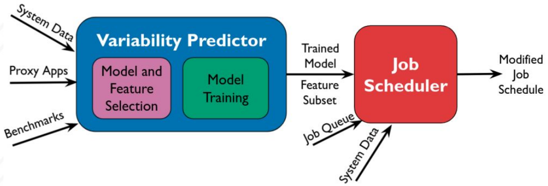
Start Function

Input $j \leftarrow$ job
$Q \leftarrow$ scheduler queue
$M \leftarrow$ ML model
$S \leftarrow$ current machine state
SkipTable $\leftarrow$ Count of times skipped for each job

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

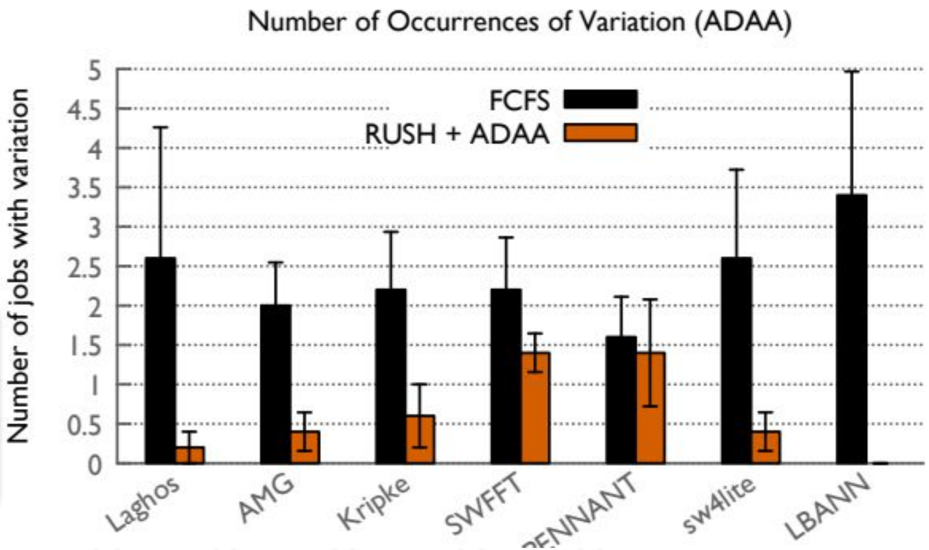
Experiments

- 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

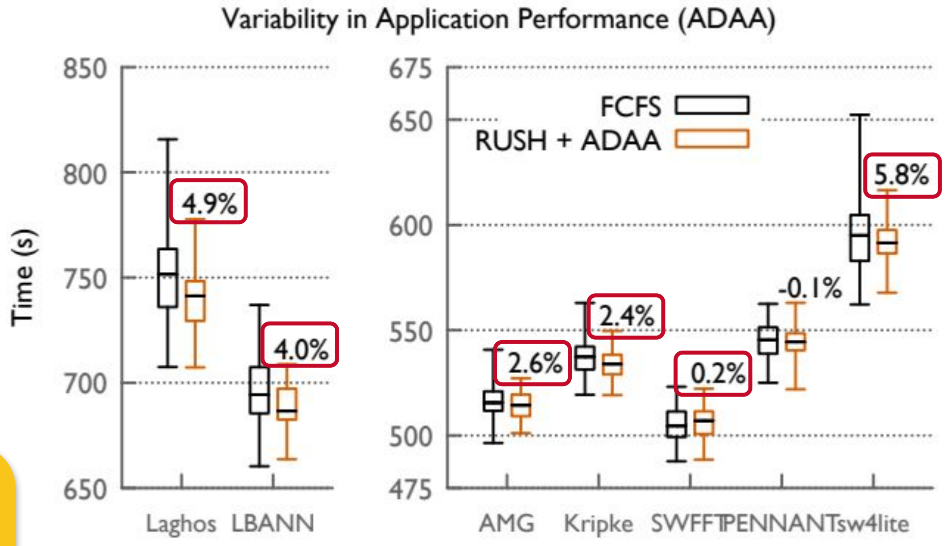


Results: All Data All Applications

- Model trained on entire dataset, running all apps

RUSH reduces max run time

RUSH reduces range of run times



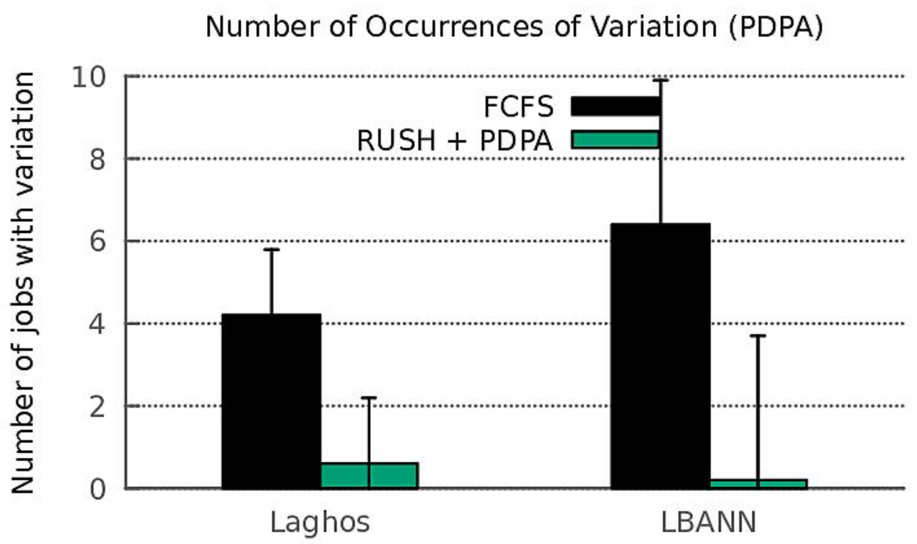
Results: Partial Data Partial Applications

- 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

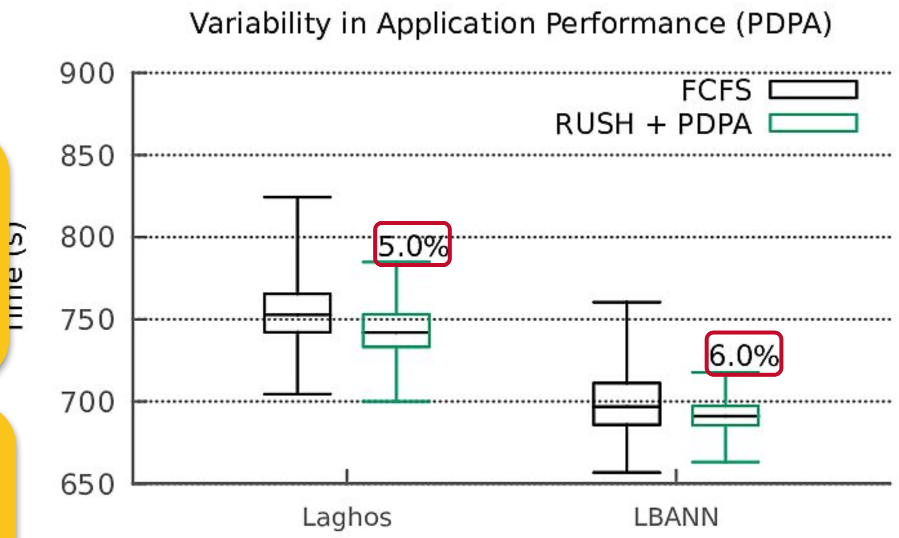


Results: Partial Data Partial Applications

- Model trained on some apps, while running other apps

RUSH reduces max run time

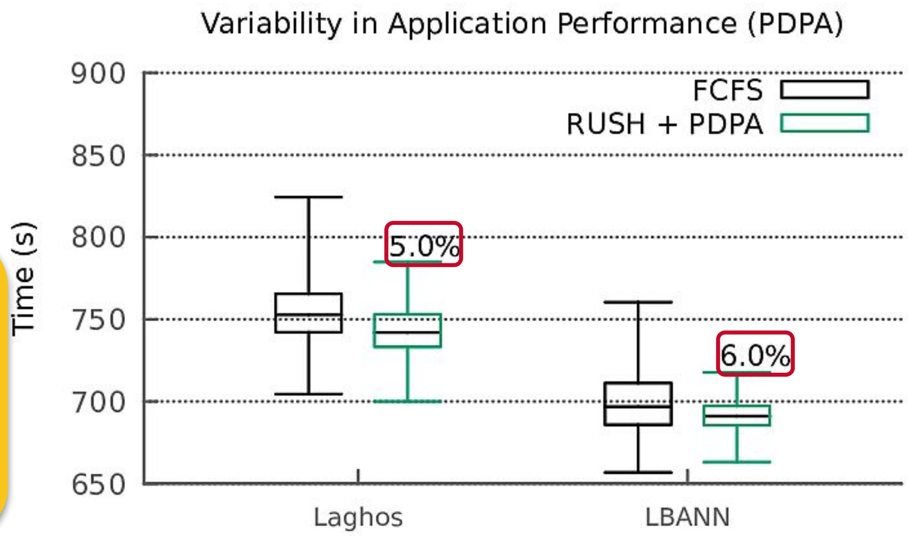
RUSH reduces range of run times



Results: Partial Data Partial Applications

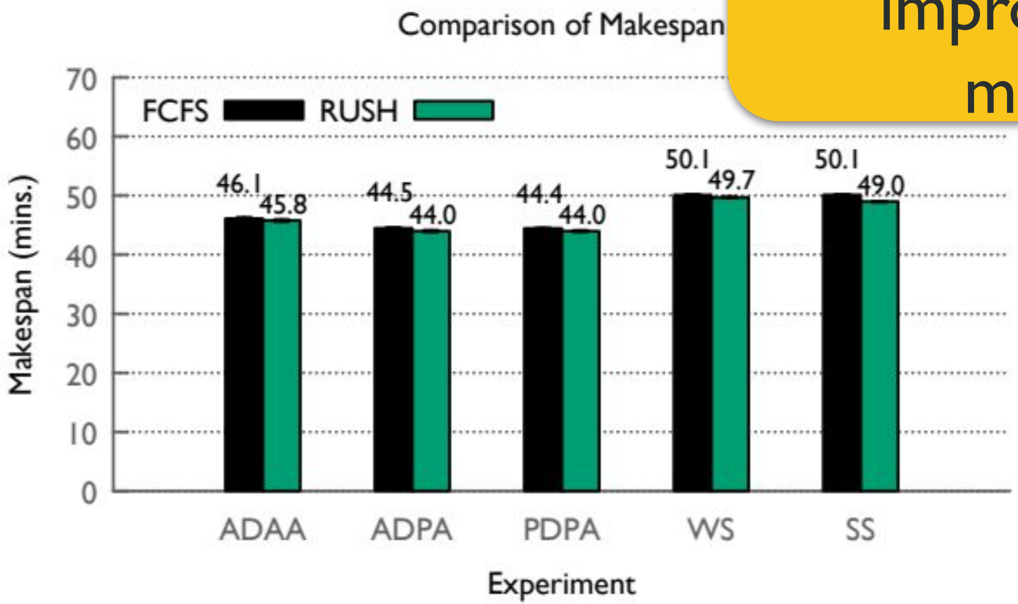
- Model trained on some apps, while running other apps

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