Squeeze to Fit: Online Moldable Interval Scheduling for the Cloud

G. Diamantopoulos[1,2], P. Oikonomou[3], E. Archontis[3], P. Roumelis[3], T. Loukopoulos[3], S. Khan[4], G. Theodoropoulos[1,2], and N. Tziritas[3]

[1] Southern University of Science and Technology, Shenzhen, China
[2] Research Institute of Trustworthy Autonomous Systems, Shenzhen, China
[3] University of Thessaly, Lamia, Greece
[4] Mississippi State University, Mississippi, USA

Agenda

- ❑ Introduction
- ❑ Motivation
- ❑ System Model
- ❑ Problem Formulation
- ❑ The Squeezy Algorithm
- ❑ Experimental Setup
- ❑ Evaluation Results
- ❑ Conclusion and Future Work

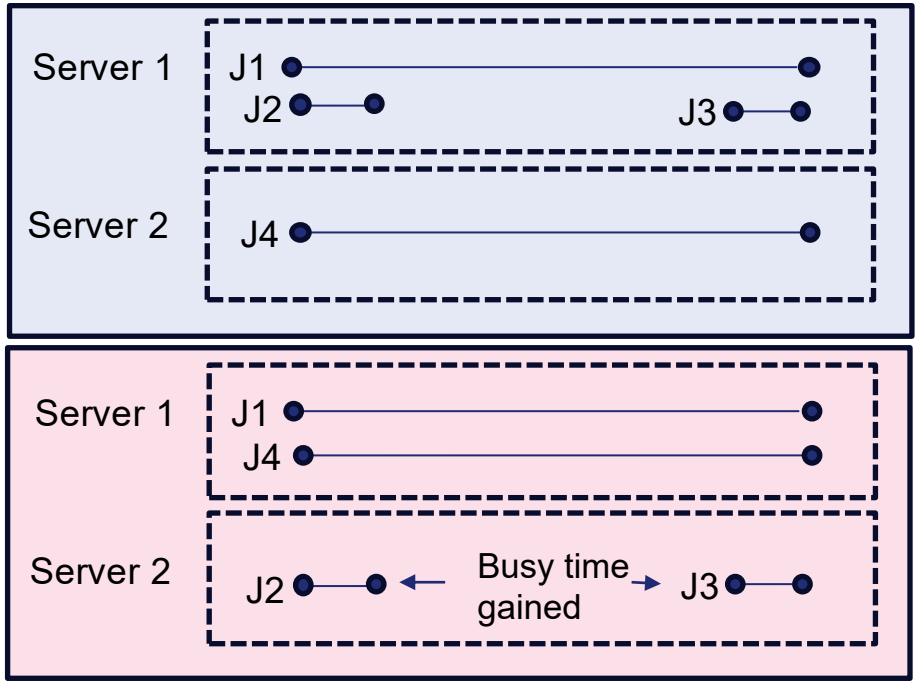
Introduction

What is the interval scheduling problem?

- Jobs are represented as time intervals.
- When a job arrives, the scheduler must immediately decide where to assign it.
- Future job arrivals are unknown.
- The problem is modelled as an online scheduling problem.
- Goal: Minimize the time for which resources remain busy.

Motivation

Online Scheduling Decisions



Our Approach

- **Similar jobs** are assigned to the same machine, decommissioning the machine whenever possible
- **Provision a new machine** when an incoming job is sufficiently dissimilar from the jobs already running.
- **Reduce a job's core allocation** when no suitable machine is available to avoid provisioning an unnecessary machine.

System Model

Machine Model

- The system has a pool of machines
  - e.g., VMs, containers, etc.
- Each machine has a fixed capacity (parallelism)
  - How many cores it offers
- New machines can be added on demand
- The number of active machines is unbounded
  - Resembling a cloud setting

Job Model

- Jobs arrive over time, each with its own release (start) time
- Each job requests a certain amount of computing resources to run
- A job's runtime depends on how many resources it's given
- Jobs are "moldable"
  - the number of cores assigned can be adjusted before execution
  - more cores speed it up, fewer cores slow it down

Downey Speed Up Model

Speedup Model

- ☐ The Downey speedup model is used to model how the execution interval length of a job changes w.r.t the number of allocated cores
- ☐ Uses parallelism variance which characterises the variability in the degree of parallelism across a job's execution
- ☐ It distinguishes between **low-variance and high-variance cases**:
  - **Low variance:** the degree of parallelism remains relatively stable.
  - **High variance:** the degree of parallelism varies substantially during execution.

Core-Speedup Knee Point

- ☐ The Downey Model includes a method to calculate the optimal number of cores for a job given its parallelism variance.
- ☐ The method is based on finding the knee-point of the curve between allocated cores and speedup

Optimisation Problem

Busy Time

- A machine's busy time is the total length of time it's active, treating any overlapping job intervals as one continuous stretch
- Cumulative busy time is thus the sum of the busy times of all machines used to processes a workload

Problem Formulation

Objective: Find an assignment ϕ that minimizes the cumulative busy time:

$$f = \sum_{i=1}^{|M|} B_i(\phi)$$

SUBJECT TO

C1) the demand of parallel executing jobs on any machine do not exceed its capacity at any point in time

C2) All jobs must be allocated at least one core

C3) Jobs are assigned for processing upon release (no queueing)

Control

The scheduler controls two aspects of the system for each arriving job

- the amount of cores allocated to the job
- Job to machine assignment (existing machine or newly provisioned)

The Squeezy Algorithm

Algorithm Intuition

- Squeezy attempts to iteratively reduce the number of cores allocated to a job to allow assignment to the active set of machines
- Initially, molds jobs to their knee point (Downey Model) and attempts First Fit allocation
- If no available machines can accommodate the job, squeezy iteratively reduces the number of cores allocated to to the job and tried again
  - Instead of immediately provisioning a new machine

Algorithm 1 The *Squeezy* Algorithm

Input: job j_x, interval $L(I_x)$ for d_x, variance σ_x, active machines M, threshold θ
Output: No. CPUs for j_x, an assignment $j_x \rightarrow m$ (where $m \in M$ or newly provisioned)

1: Calculate optimal cores for j_x (d_x^*) using Eq. 4
2: Calculate $S_1 = S(d_x)$ and $S_2 = S(d_x^*)$ using Eq. 1 ($\sigma_x < 1$) or Eq. 2 ($\sigma_x \geq 1$)
3: $L'(I_x) \leftarrow L(I_x) \times \frac{S_1}{S_2}$ (Eq. 3)
4: Attempt First Fit assignment $j_x \rightarrow M$ with d_x^*; **return on success**
5: **for** $d = d_x^* - 1$ **to** 1 **step** -1 **do**
6: Calculate $S_1 = S(d_x)$ and $S_2 = S(d)$ using Eq. 1 ($\sigma_x < 1$) or Eq. 2 ($\sigma_x \geq 1$)
7: $L'(I_x) \leftarrow L(I_x) \times \frac{S_1}{S_2}$ (Eq. 3)
8: $L^{max} \leftarrow \{\max(L(j_y) \mid j_y \in m_i) \mid m_i \in M\}$
9: $L^{min} \leftarrow \{\min(L(j_y) \mid j_y \in m_i) \mid m_i \in M\}$
10: $M' \leftarrow \emptyset$
11: **for** each $m \in M$ **do**
12: **if** $L'(I_x) \leq (l_m^{max} \times \theta)$ **AND** $L'(I_x) \geq (\frac{l_m^{min}}{\theta})$ **then**
13: $M' \leftarrow M' \cup \{m\}$
14: **end if**
15: **end for**
16: Attempt First Fit assignment $j_x \rightarrow M'$ with d; **return on success**
17: **end for**
18: open new machine m_{new} and append it into M
19: assign $j_x \rightarrow m_{new}$ with d_x^*

The Squeezy Algorithm

The Threshold θ

- Squeezy utilises an exclusivity threshold to determine if the execution interval of a job is compatible with the existing intervals in a machine m before assignment
- Ensures allocated job intervals are not θ times larger or smaller than intervals of executing jobs
- **The motivation is twofold:**
  - Prevents excessive resizing of jobs
  - Allocating similar sized jobs together improves resource utilisation

Algorithm 1 The *Squeezy* Algorithm

Input: job j_x, interval $L(I_x)$ for d_x, variance σ_x, active machines M, threshold θ
Output: No. CPUs for j_x, an assignment $j_x \rightarrow m$ (where $m \in M$ or newly provisioned)

Calculate optimal cores for j_x (d_x^*) using Eq. 4
Calculate $S_1 = S(d_x)$ and $S_2 = S(d_x^*)$ using Eq. 1 ($\sigma_x < 1$) or Eq. 2 ($\sigma_x \geq 1$)
$L'(I_x) \leftarrow L(I_x) \times \frac{S_1}{S_2}$ (Eq. 3)
Attempt First Fit assignment $j_x \rightarrow M$ with d_x^*; **return on success**
for $d = d_x^* - 1$ **to** 1 **step** -1 **do**
  Calculate $S_1 = S(d_x)$ and $S_2 = S(d)$ using Eq. 1 ($\sigma_x < 1$) or Eq. 2 ($\sigma_x \geq 1$)
  $L'(I_x) \leftarrow L(I_x) \times \frac{S_1}{S_2}$ (Eq. 3)
  $L^{max} \leftarrow \{\max(L(j_y) \mid j_y \in m_i) \mid m_i \in M\}$
  $L^{min} \leftarrow \{\min(L(j_y) \mid j_y \in m_i) \mid m_i \in M\}$
  $M' \leftarrow \emptyset$
  for each $m \in M$ **do**
    if $L'(I_x) \leq (l_m^{max} \times \theta)$ **AND** $L'(I_x) \geq (\frac{l_m^{min}}{\theta})$ **then**
      $M' \leftarrow M' \cup \{m\}$
    end if
  end for
  Attempt First Fit assignment $j_x \rightarrow M'$ with d; **return on success**
end for
open new machine m_{new} and append it into M
assign $j_x \rightarrow m_{new}$ with d_x^*

If the job is still not allocated after its cores are reduced to 1 a new machine is provisioned and the job is allocated to it at its knee point cores

Experimental Setup

Simulation Environment

- A moldable interval scheduling simulator was developed to allow for experimental evaluation
- The simulator implements the job and machine logic and allows injecting and simulating the allocation and processing of job traces
- The mutual exclusivity threshold for Squeezy was set at $\theta = 2$
  - I.e., jobs with intervals that differ by more than a factor of 2 cannot be packed in the same machine

Workloads

Name	Num. Jobs	Avg. CPUs	Avg. Duration
1 SDSC Par96 [6]	19058	4.99	4456
2 LLNL Atlas [6]	10080	8	1118
3 Sandia Ros [6]	21828	4.60	27831
4 SDSC DataStar [6]	32391	8	6464
5 MetaCentrum [14]	148548	2.81	18459
6 Eucalyptus [27]	9173	1.67	25368
7 KIT FH2 2016 [6]	17203	1.52	2482
8 UniLu Gaia 2014 [6]	48275	7.02	13161
9 PIK IPLEX 2009 [6]	643518	1.17	34389
10 RICC 2010 [6]	381397	1.53	11953

Baseline Algorithms

Selected baselines

Algorithms for the online interval scheduling

First Fit (FF)

Allocates the job to the first machine with available capacity

Best Fit (BF)

Allocates the job to the machine that leads to the least leftover capacity after allocation

BF-MAT

Allocates the job to the machine with the most actively processing jobs that has available capacity

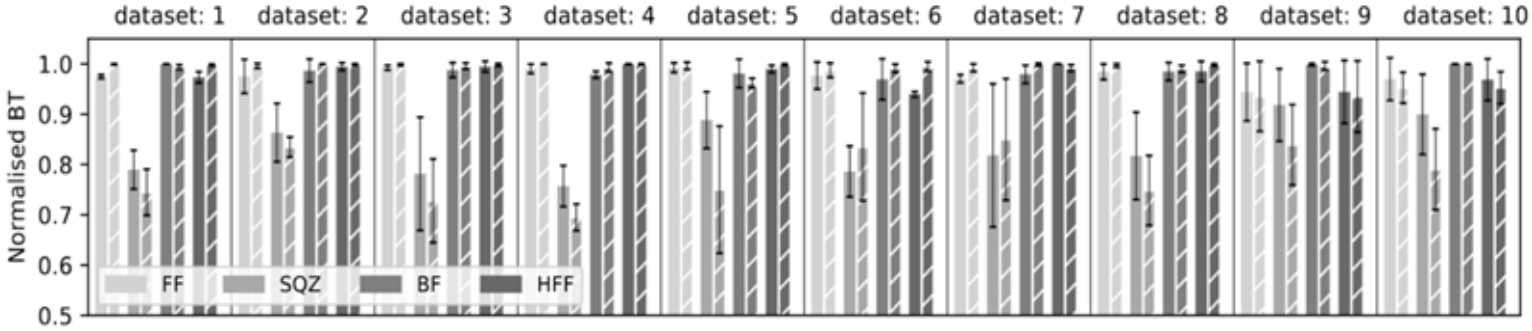
Hybrid First Fit (HFF)

Categorises jobs as small or large based on duration and packs them on separate machine categories

Baseline Moldability

To enable a fairer comparison in the moldable setting, each algorithm is modified to schedule jobs at their core-speedup knee-point using the calculation provided by the Downey Model.

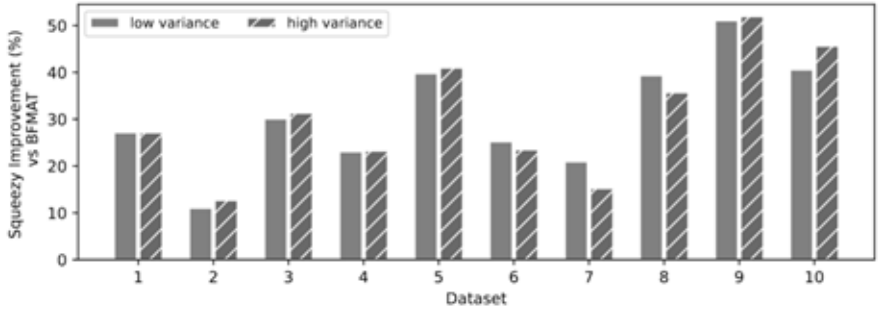
Average Normalised Busy Time for all capacities.
Solid bars denote results for the low variance.
Hatched bars denote results for the high variance parallelism model.



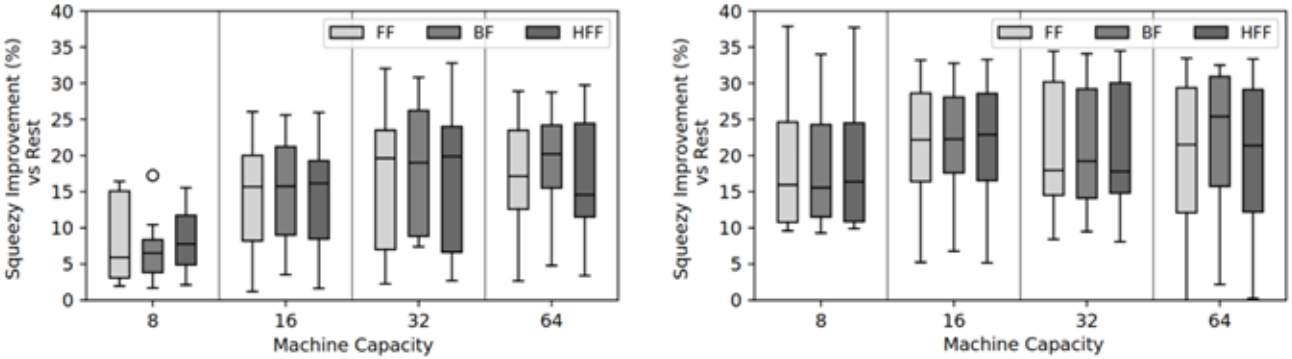
Observations

- The bars show the average BT for all capacities
- The error bars visualising the variance.
- The results show that Squeezy achieves the lowest busy time among all baselines across all datasets

Improvement VS BF-MAT shown separately



Percentage improvement of Squeezy vs the rest of the algorithms



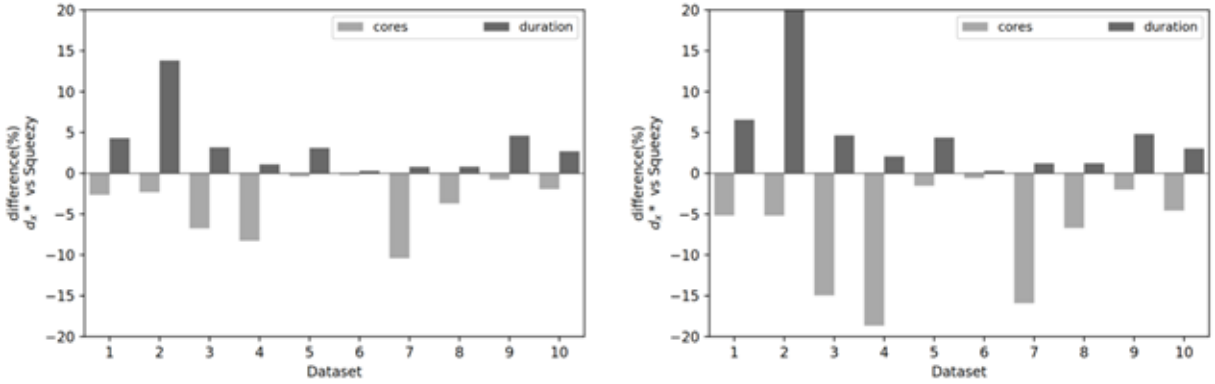
(a) low variance model

(b) high variance model

Observations

- Showcases the improvement of Squeezy over the baselines for each machine capacity
- Under high variance, Squeezy's improvement over the baselines is greater than low variance case
- Since Squeezy works by reducing allocated cores, it naturally aligns with the high variance setting where additional cores yield diminishing speedup returns.

Effect of Squeezy on allocated cores and durations of jobs versus core-speedup knee-point



(a) low variance model (b) high variance model

Observations

- The reductions in allocated cores results in minor increases in execution duration
- The reduction in allocated cores frequently exceeds the increase in execution duration
  - a consequence of the flattening speedup curve near the knee-point
- Squeezy exploits this property by allocating slightly below the knee-point
  - This allows assignments to existing machines instead of provisioning new ones

Conclusion and Future Work

Conclusion

- **Squeezy dynamically reduces core allocations** to enable more efficient assignments, while using a mutual exclusivity threshold to limit resizing and incompatible job assignments.
- **Squeezy consistently outperforms baseline algorithms** across different job parallelism profiles using real-world traces.
- **Small core adjustments significantly reduce busy time**, with only a small increase in job duration, highlighting the value of **moldability**.

Future Work

- Designing adaptive threshold strategies by using reinforcement learning techniques
- Considering environments with heterogeneous machine capacities
- Considering additional optimisation dimensions and constraints
  - Job slowdown
  - SLA violations

THANK YOU FOR YOUR ATTENTION!

Source code available at: https://github.com/GiorgDiama/Moldable-Interval-Scheduling-Simulator

ANY QUESTIONS?

George Diamantopoulos: giorg.diama@gmail.com
Panagiotis Oikonomou: paikonom@uth.gr