Oct 10th, 2024 ICSME, Flagstaff, AZ, USA

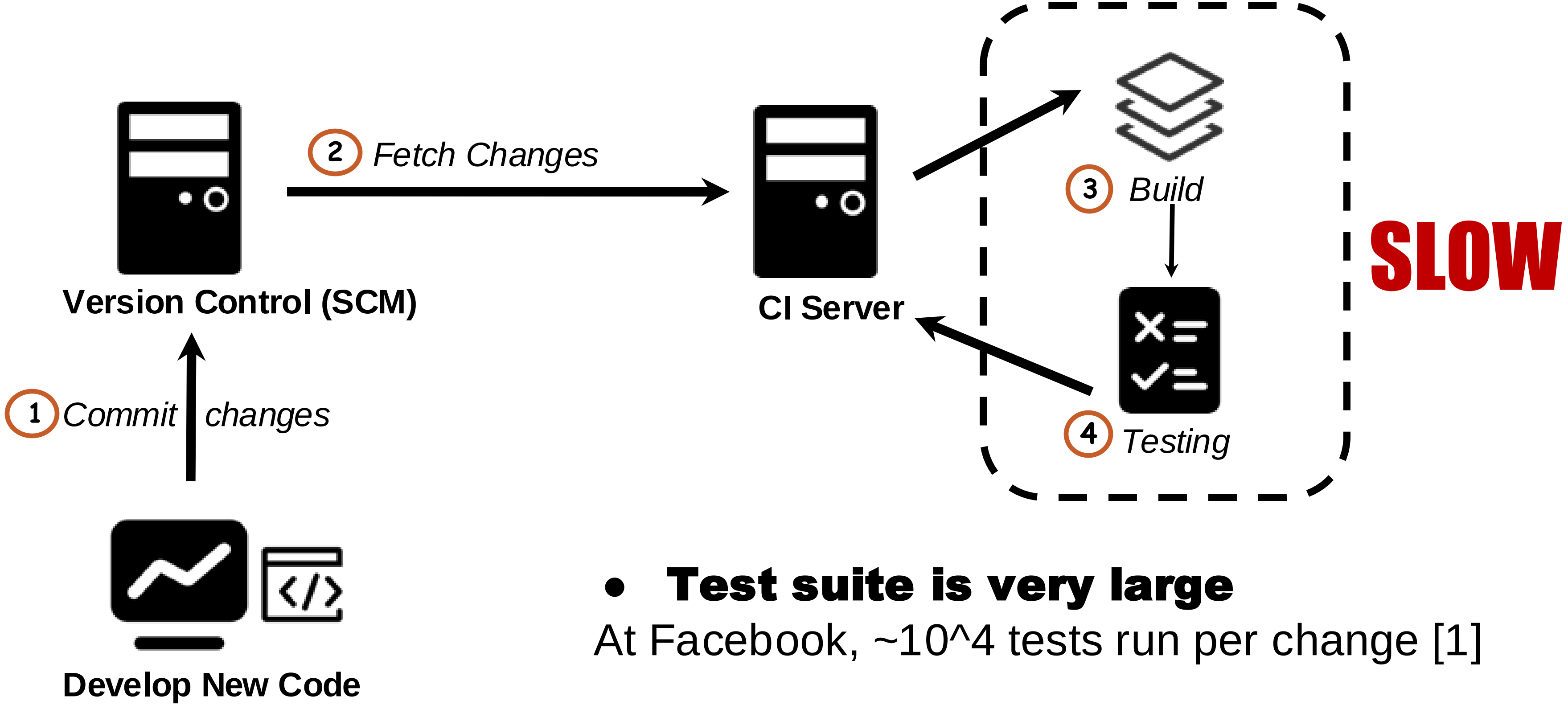
Test Scheduling Across Heterogeneous Machines While Balancing Running Time, Price, and Flakiness

Hengchen Yuan*, Jiefang Lin*, Wing Lam†, August Shi*

* The University of Texas at Austin
† George Mason University

CCF-2145774
CCF-2217696
CCF-2338287

Background: Regression Testing

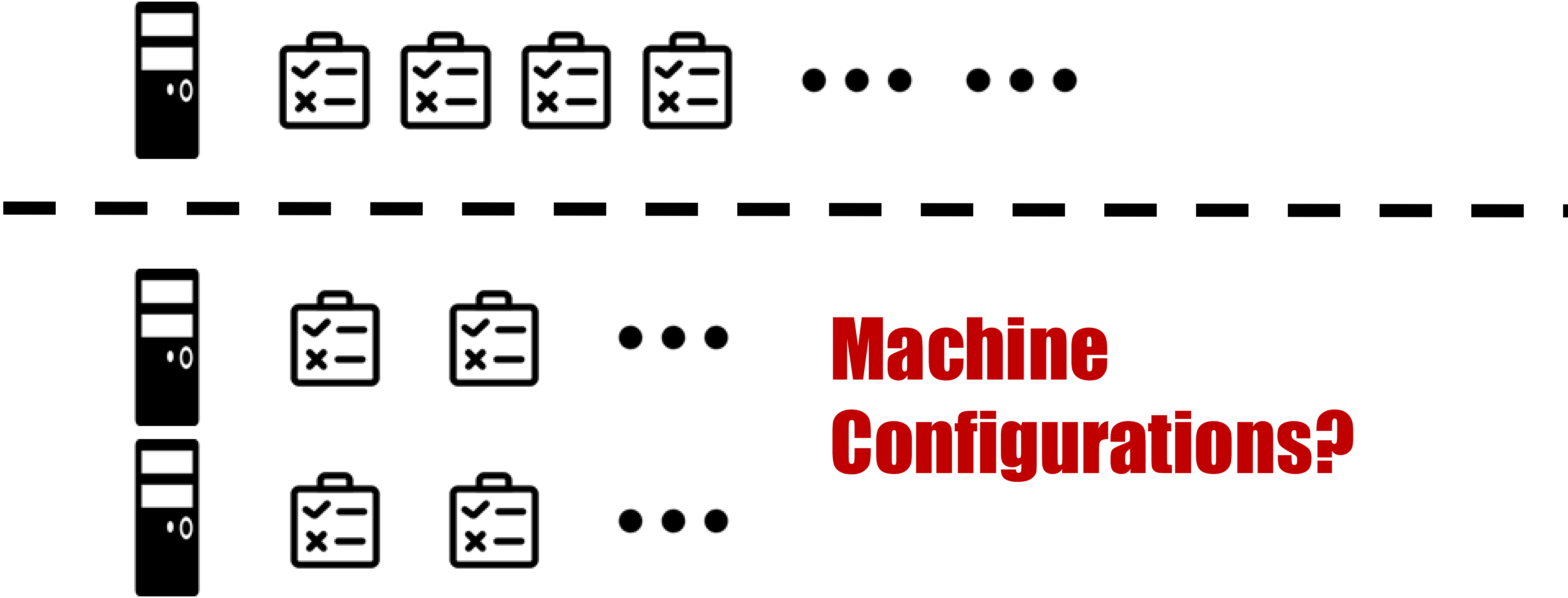


- **Test suite is very large**

At Facebook, ~10^4 tests run per change [1]

[1] Machalica et al., "Predictive Test Selection", ICSE SEIP 2019.

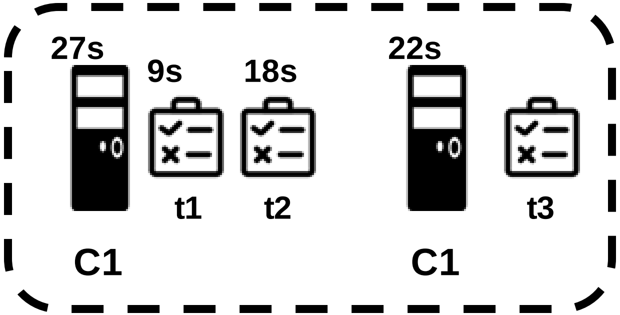
Test Scheduling



Test Scheduling Example

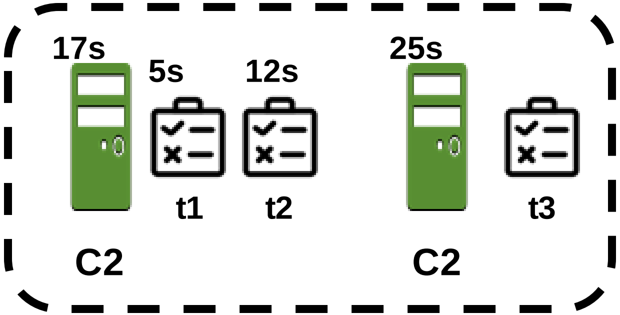
Allocate these 3 tests on 2 machines while minimize the running time:

Test	Config 1 (C1) run time
t1	9
t2	18
t3	22



27s

C2 runs faster!

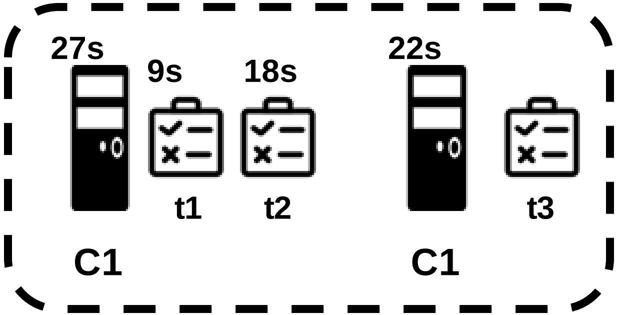


25s

Test Scheduling Example

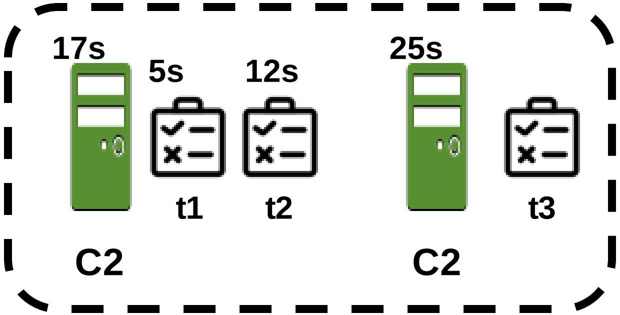
What about Price? **C1: 0.004 cent (¢)/s, C2: 0.006 cent (¢)/s**

Test	Config 1 (C1) run time	Config 2 (C2) run time
t1	9	5
t2	18	12
t3	22	25



(27s + 22s) * 0.004¢ = 0.196¢

C1 is cheaper!



(17s + 25s) * 0.006¢ = 0.252¢

Test Scheduling Example

What about Flaky tests?

Flaky test: the test can pass or fail on the same code version

Test	Config 1 (C1) run time	Config 2 (C2) run time
t1	9	5
t2 (flaky test)	18 **(fail rate: 0.2)**	12 **(fail rate: 0.5)**
t3	22	25

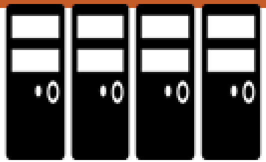
Some flaky tests are configuration-affeected! [1]

What if consider the fail rate?
How to take it in to consideration?

[1] Silva, D., Gruber, M., Gokhale, S., Arteca, E., Turcotte, A., d'Amorim, M., ... & Bell, J. (2024). The effects of computational resources on flaky tests. *IEEE Transactions on Software Engineering*

Test Scheduling Example

What about Flaky tests?



Homogeneous

Test	Config 1 (C1) run time	Config 2 (C2) run time
t1	9	5
t2 (flaky test)	18 * **1.25 = 22.5**	12 * **1.94 = 23.28**
t3	22	25

Model the flaky-failure rate into running time

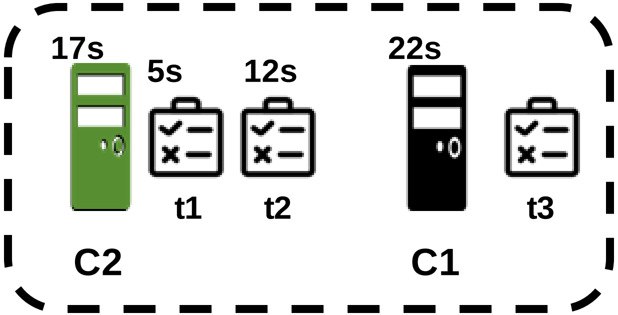
Expected running time if rerun flaky test *until it passes once* (up to 10):

For the test with the failrate 0.5, we expect it to be run $1 + \sum_{i=1}^{10} 0.5^i = 1.94$ times

Test Scheduling Example

What if we use different configurations?

Test	**Config 1 (C1) run time**	**Config 2 (C2) run time**
t1	9	5
t2 (flaky test)	18 **(fail rate: 0.2)**	12 **(fail rate: 0.5)**
t3	22	25

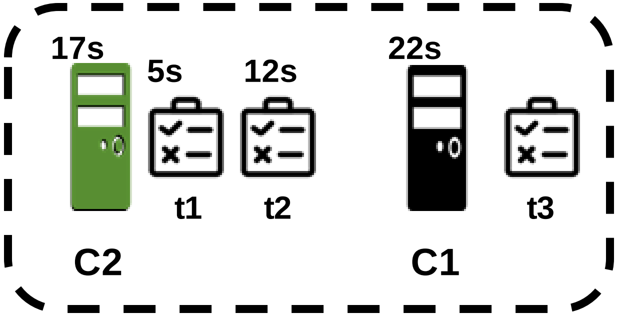


22s compared to 25s before

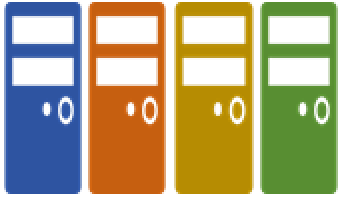
Test Scheduling Example

What about Price? **C1: 0.004 cent (¢)/s, C2: 0.006 cent (¢)/s**

Test	Config 1 (C1) run time	Config 2 (C2) run time
t1	9	5
t2 (flaky test)	18 **(fail rate: 0.2)**	12 **(fail rate: 0.5)**
t3	22	25



Heterogeneous scheduling is both cheaper and faster!



Heterogeneous

17s * 0.006¢ + 22s * 0.004¢ = <u>0.19¢</u> compared to 0.196¢ before

Heterogeneous Test Scheduling

Input

Project’s tests’ info (running attributes on each candidate configuration)

Number of machines needed

Output

A combination of machines with different configurations on which to run tests

An allocation scheme is a mapping of which tests to run on which machine

Challenge

Large search space of possible combinations of machines configurations

C candidate machine configurations, ***M*** machines for solution: Close to ***C^M*** possible combinations!

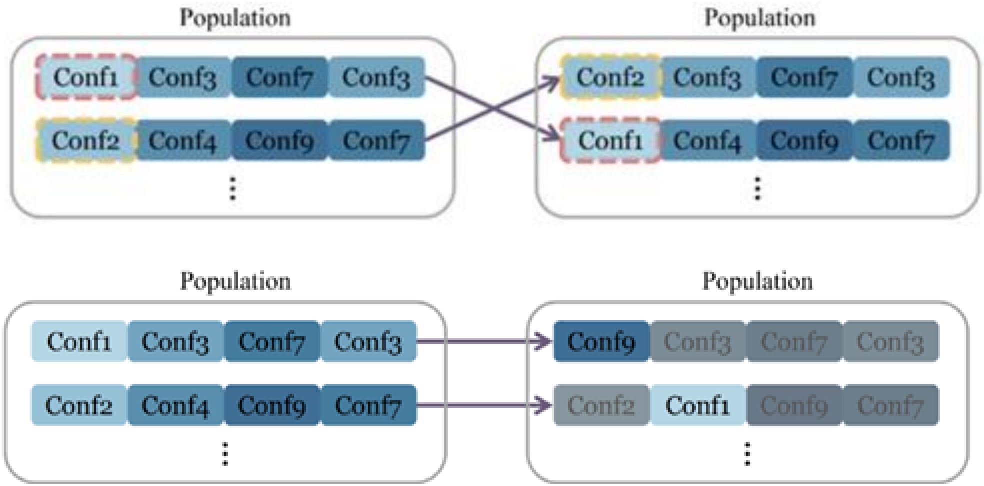
Hard to produce an optimal allocation on a given combination

For each test, there are ***M*** ways to allocate it. Close to ***M^T*** possible allocation scheme on a given combination for ***T*** tests

Approach: GASearch

Genetic Algorithm → Efficiently search machine configurations list

- Randomly create a population of solutions (e.g., 50)
  - solution → a sequence of configurations
- **Calculate fitness of each solution in population**
- Create new solutions based on fitness (e.g., Create offspring by recombining some parts of parent's representation + mutating offspring)

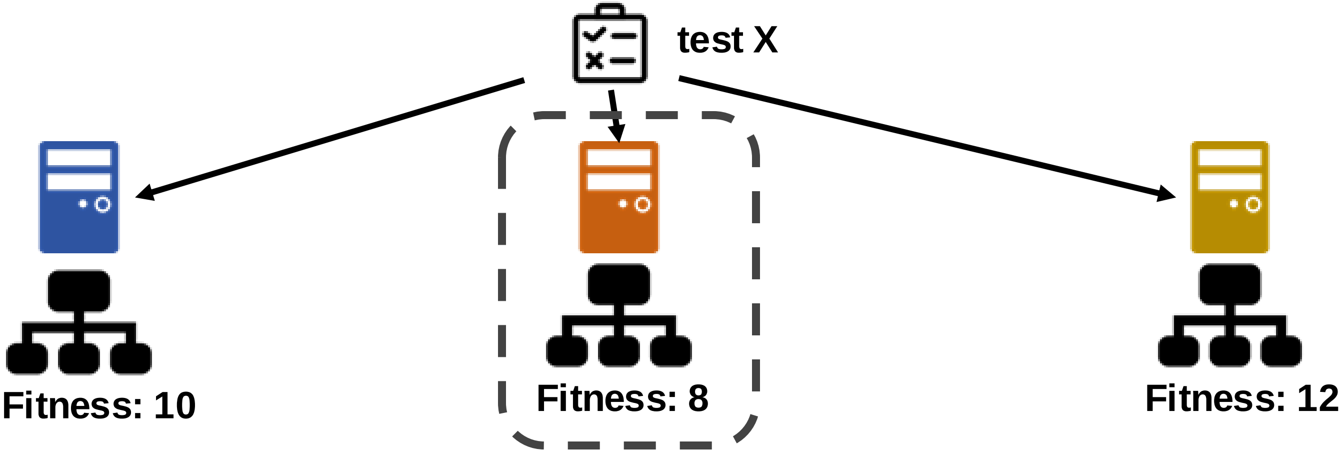


- Only create offspring using the fittest members of the population
- Evolve several generations until find optimal solution

Approach: GASearch

Fitness-based greedy test allocation

- Fitness is calculated based on the allocation scheme (which machine a test should run on)
- Use a greedy algorithm to iteratively allocate each test on given machines



Fitness = α*Time + (1-α)*Price

Evaluation Setup

12 different candidate machine configurations

24 modules across 22 projects as baseline

We run each test approximately 300 times on each configuration to collect running time.

Projects are from Github, the number of tests goes from 14 to 6267

ConfigID	# CPU	Mem (GB)	Price (USD/Hour)
C1	0.1	1	0.002548
C2	0.1	2	0.003881
C3	0.25	2	0.005703
C4	0.5	2	0.008739
C5	0.5	4	0.011406
C6	1	4	0.017478
C7	1	8	0.022812
C8	2	4	0.029622
C9	2	8	0.034956
C10	2	16	0.045624
C11	4	8	0.059244
C12	4	16	0.069912

* Machine configurations from [1]. “# CPU” with non-integer value means a core is shared across multiple tasks [2].
Hourly costs are specified on AWS FRAGATE [3]

[1] Silva, D., Gruber, M., Gokhale, S., Arteca, E., Turcotte, A., d'Amorim, M., ... & Bell, J. (2024). The effects of computational resources on flaky tests. *IEEE Transactions on Software Engineering*.
[2] Runtime options with Memory, CPUs, and GPUs,” https://docs.docker.com/config/containers/resource constraints, 2024.
[3] “AWS Fargate,” https://aws.amazon.com/fargate, 2024.

Evaluation Setup - Baselines

Github Baseline

Only use configuration of 2 CPUs and 8 GB RAM, same as GitHub Actions [4]

Smart Baseline

HOMOGENEOUS Baselines

Optimal homogeneous machines using fitness

Random Baseline

Randomly choose heterogeneous machines

[4] “About GitHub-hosted runners,” https://docs.github.com/en/actions/using-github-hosted-runners/using-github-hosted-runners/about-github-hosted-runners, 2024.

Evaluation - Research Question

RQ1. GASearch compared to baselines on single goal

RQ2. How does the weight factor affect GASearch

RQ3. What is the flaky-failure rate?

RQ4. Comparison against brute-force search?

RQ5. Effectiveness when using less test data?

Please check our paper for those details

RQ1. Heterogeneous vs Homogeneous Machines

The value shows the ratio of GAsearch result to the baseline result.

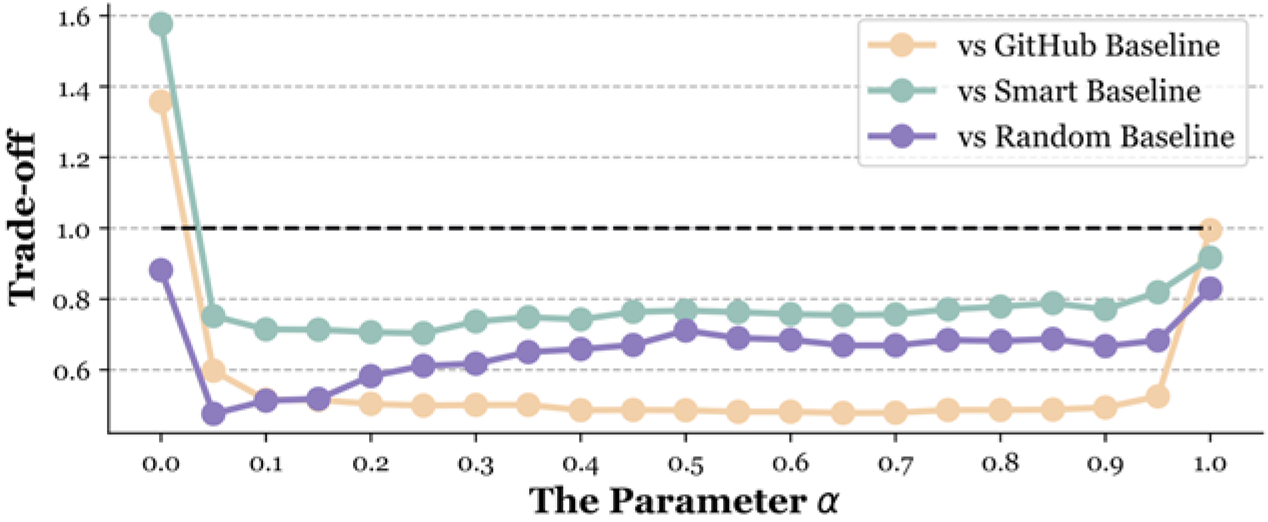
	Optimizing for Price			**Optimizing for Running Time**		
Baselines	Min.	Max.	Avg.	Min.	Max.	Avg.
Github	0.04	0.84	0.45	0.33	1.00	0.91
Smart	0.54	1.00	0.84	0.85	1.00	0.99
Random	0.61	1.00	0.88	0.41	1.00	0.83

Finding: Heterogeneous generally performs better at optimizing price. GASearch can find better solution for running time or price over the baselines.

RQ2. Effect of Fitness Function Weight

Fitness = α*Time + (1-α)*Price

$$Tradeoff(A_G, A_B) = \frac{Time_{para}(A_G)}{Time_{para}(A_B)} \times \frac{Price(A_G)}{Price(A_B)}$$



Finding: GASearch's improvement on tradeoff between the two factors is much better when balancing both within the fitness function.

Conclusion

We propose scheduling tests using heterogeneous machines

We implement GASearch, a genetic algorithm approach to schedule tests across heterogeneous machines

GASearch provides better running time and price, as well as better trade-offs against baselines

Website:

Contact: Hengchen Yuan *hcyuan@utexas.edu*