Analysis of Container Based vs. Jailed Sandbox Autograding Systems

Matthew Peveler   Evan Maicus   Timothy Cyrus   Buster Holzbauer   Barbara Cutler

Abstract

Traditionally, automated testing and grading of student programming assignments has been done in either a jailed sandbox environment or within a virtual machine (VM). For a VM, each submission is given its own instantiation of a guest operating system (OS) running atop the host OS, with no ability for a given submission to affect anything outside the VM. However, using a VM is expensive in terms of system resource usages, especially for RAM and memory, making it less than ideal for solutions without unlimited resources. Jailed sandboxes on the other hand allow student submissions to run directly on the server. Sufficient security measures must be implemented to ensure that students cannot access each others submissions or the server at large, and must prevent runaway programs, over-utilization of system resources. Jailed sandboxes have a larger attack vector than VMs.

Within the past several years, container systems have been gaining popularity and usage within the computer science industry, primarily through solutions such as Docker. These containers give similar security protections as a VM, but with better performance due to being able to utilize of resources installed within the host OS and other containers. However, containers do not have the full isolation of a VM, and thus implementing Docker for autograding ends up facing its own set of security concerns, as well as with the increased system resource usage. In this poster, we will analyze how well containers work for automated testing and grading of student homework, measuring system resources and throughput of submissions of containers against the traditional jailed environment.

Jailed Sandbox Environment

- The program executes on the bare-metal server.
- In Submitty, each parallel grading process has its own "untrusted" user.
- An "untrusted" user has minimal rights, both by the programs it can execute and the folders it is allowed to access.
- The application cannot affect or harm the server at large, nor freely access any resource.
- Each untrusted user has read/write permissions on a tmp folder, but cannot access the folders of any other untrusted user. (Student cannot access the submissions of other students, nor can they manipulate the grading process.)
- A user with elevated permissions copies student submissions into untrusted folders as needed during autograding and archives the results after grading.

While potentially 'easy' to setup, jailed sandbox environments may face problems in dealing with differing dependencies for all of the various courses Submitty supports at a university, as these classes may require different or conflicting versions of packages and software.

Containers

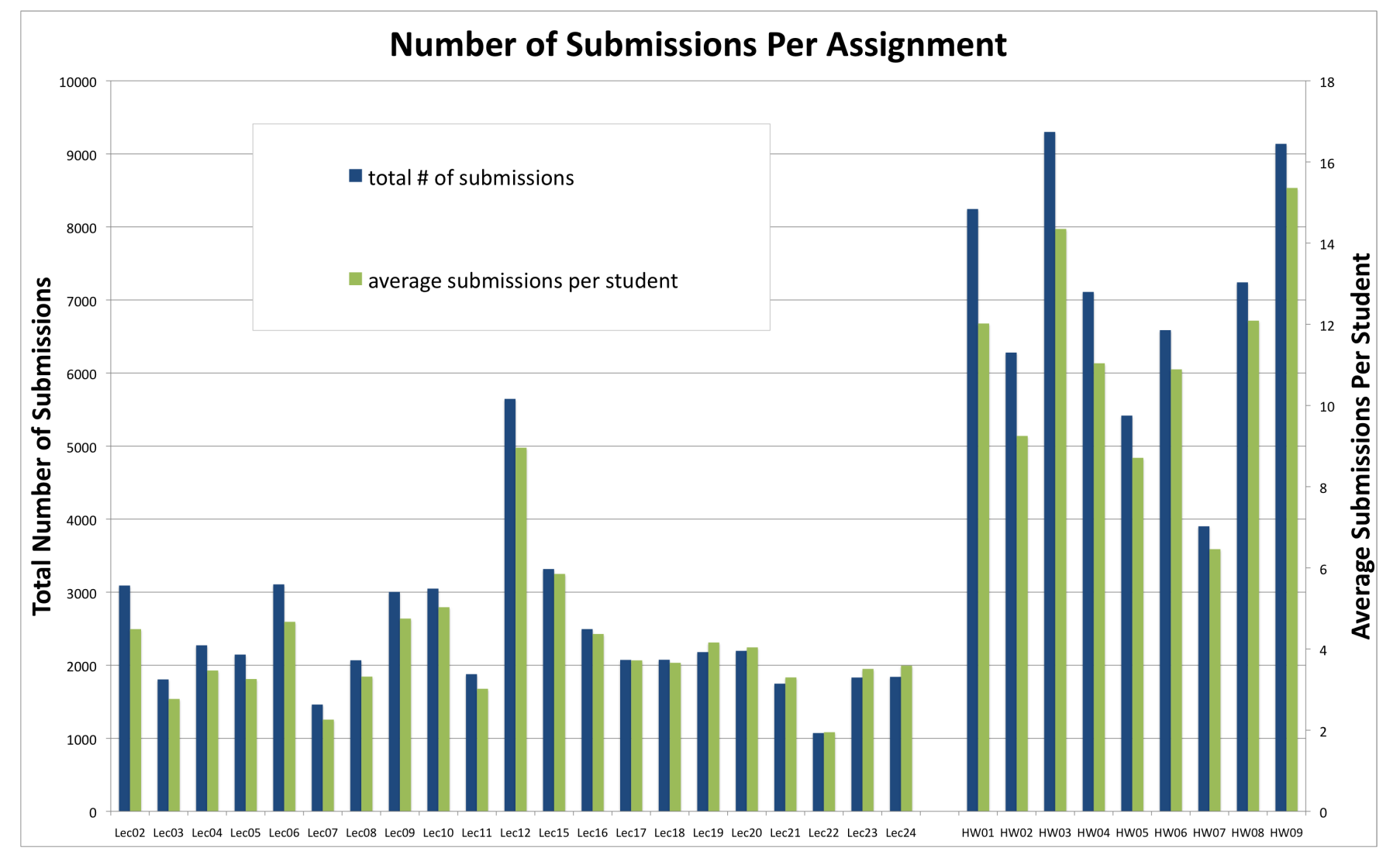
- Underlying technology of cgroups and namespaces since 2008.
- Popularized in 2013 by Docker.
- Virtualization at the operating-system level, avoiding overhead of a full virtual machine.
- Share folders between the host and container via mounting 'volumes'.
- Containers can have different installed software, available programming languages, packages, etc.
- Container configuration can be customized per course or per assignment.
- Special users and file permissions should still be carefully established when using containers, which run everything as root by default.
- Not insignificant costs to spin up/destroy a container as well as some additional load on CPU and RAM.

Methodology

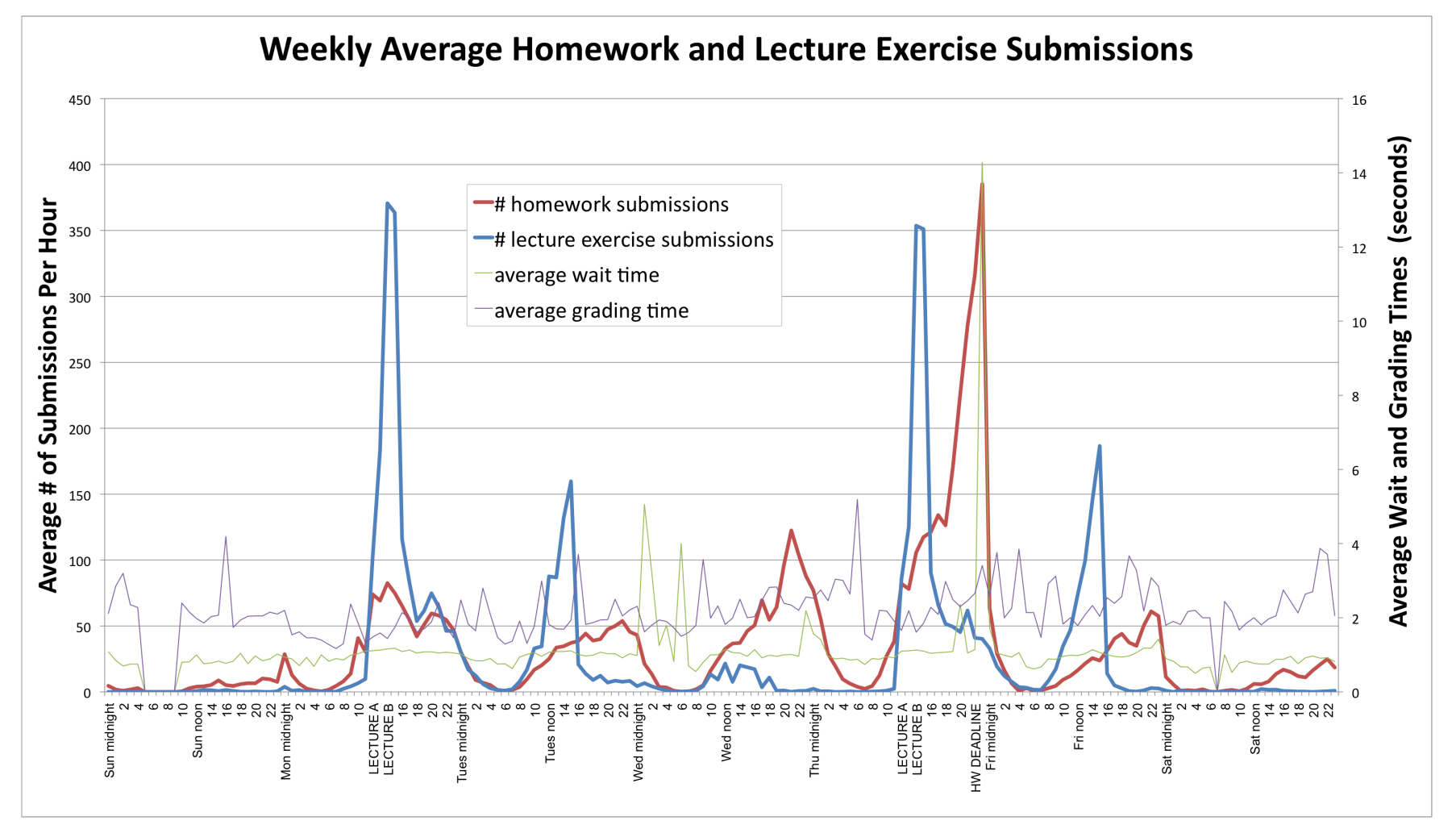
- Identified time slices from previous semesters' data with:
  - Maximum number of submissions per hour.
  - Longest average grade time (weeks with computationally intensive homework and/or many buggy infinite loop submissions that were terminated for time).
  - Elevated wait times (the machine was running at capacity with many concurrent jobs).
- We 'replayed' these hours twice, once with our jailed sandbox and a second time with containers (using Docker)
- We logged:
  - How long a submission waited in the grading queue,
  - How long it took to run the grading scripts over a submission,
  - CPU usage percentage,
  - RAM usage percentage,
  - How long it took to spin up (Docker only)
  - How long it dook to destroy (Docker only)
- The machine used was a Dell Poweredge R520 with an Intel Xeon ES-2470 (8 Cores, 16 Threads) CPU and 32 GB of RAM.

Submission Data

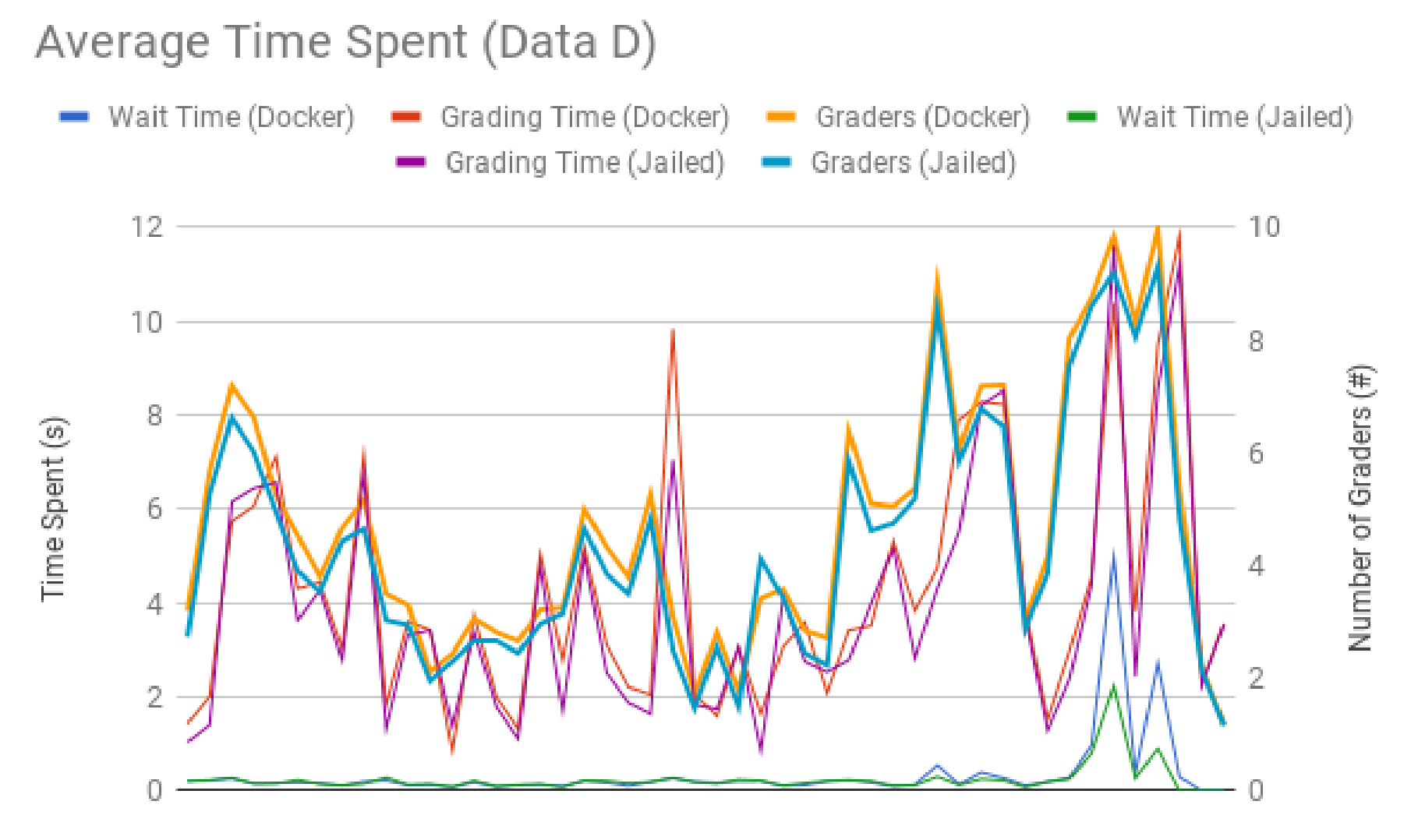
- Submitty at RPI: ~12-14 courses per term, ~2000 different students per year.
- In our CS1 course (> 700 students in Fall 2017), Submitty is used to autograde both lecture exercises and longer weekly homeworks.
- Students are allowed and encouraged to resubmit their assignments to correct errors.

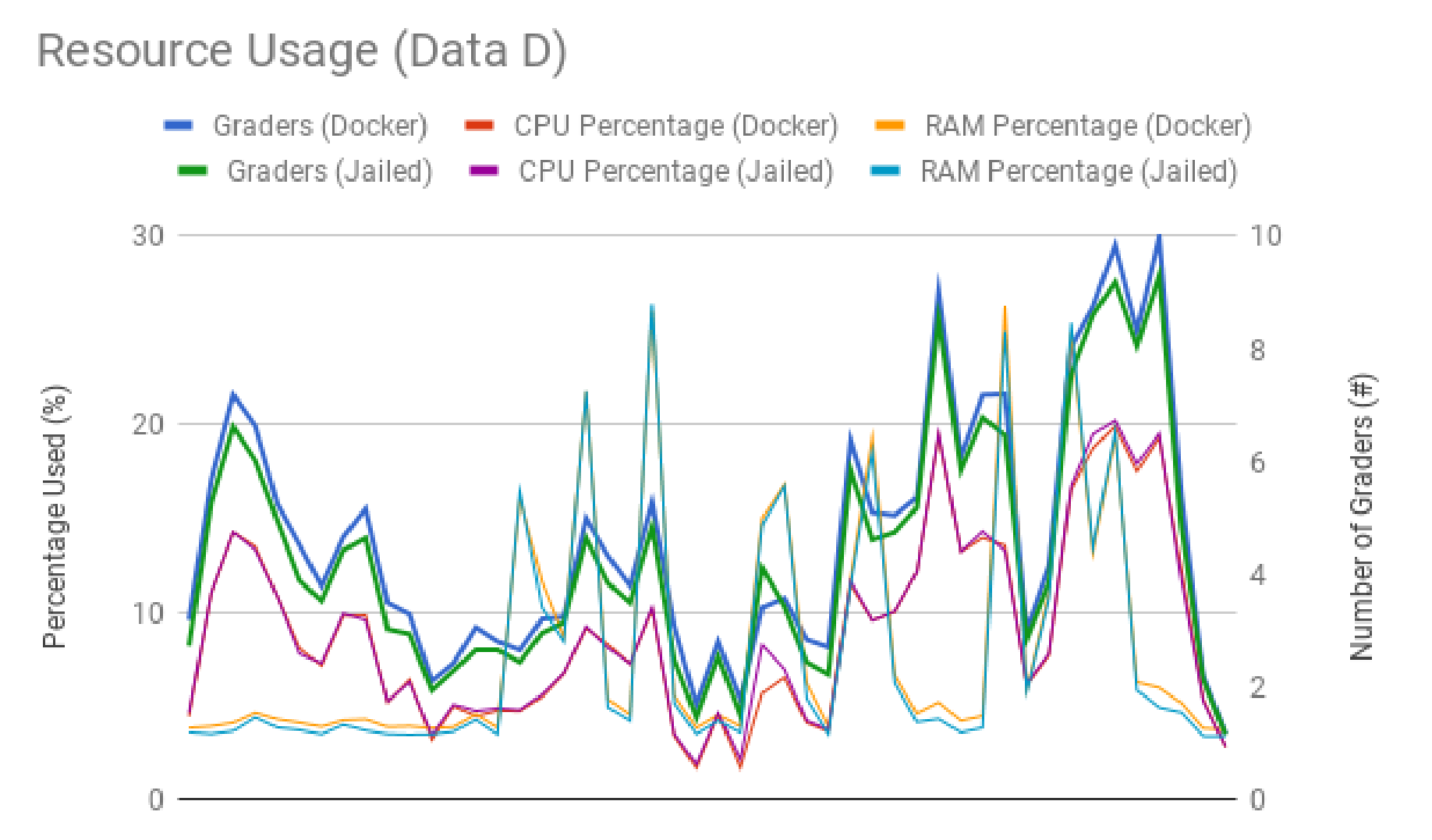


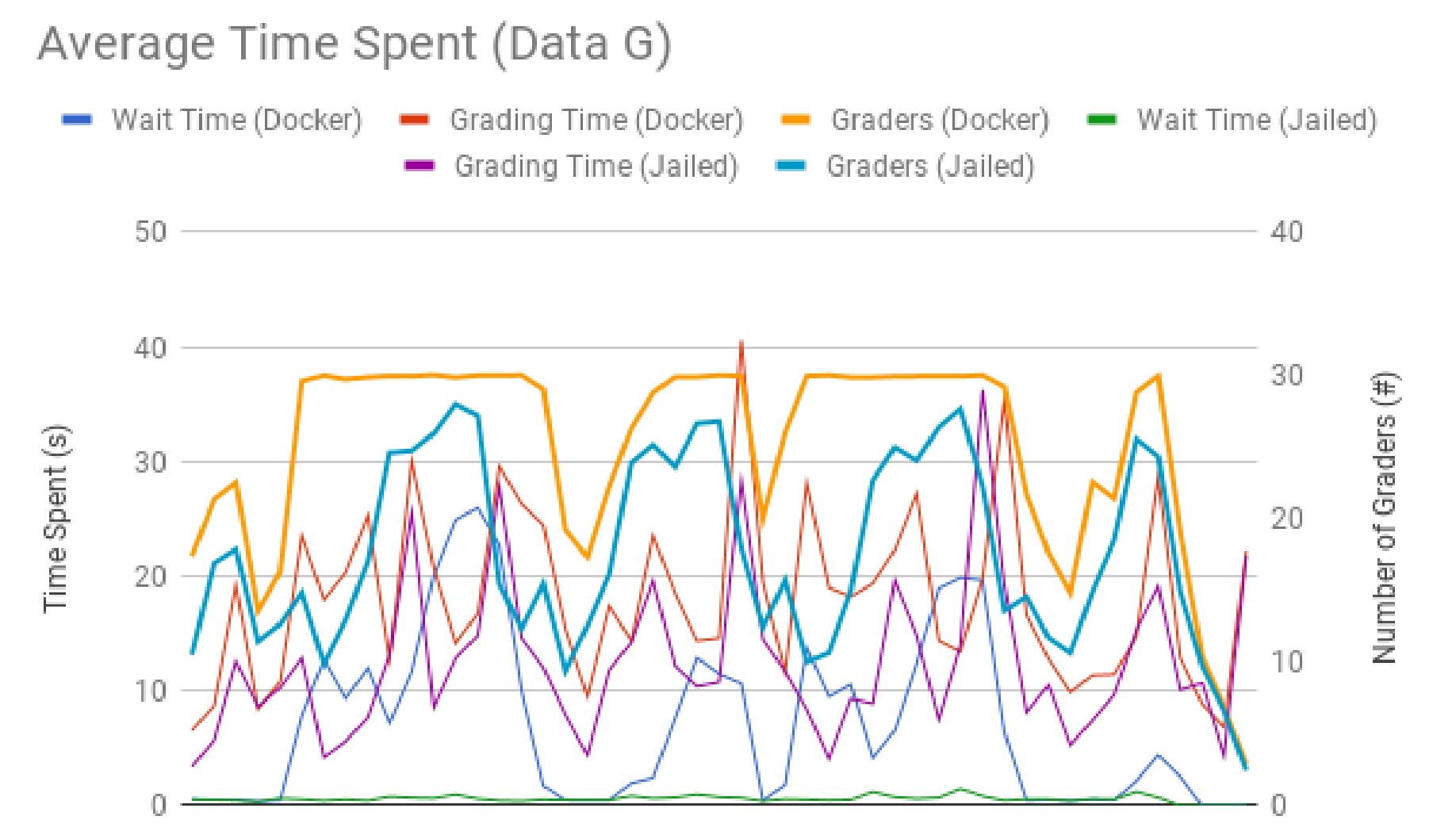
- The server experiences heavy load during these lecture times and the night of the homework deadline.

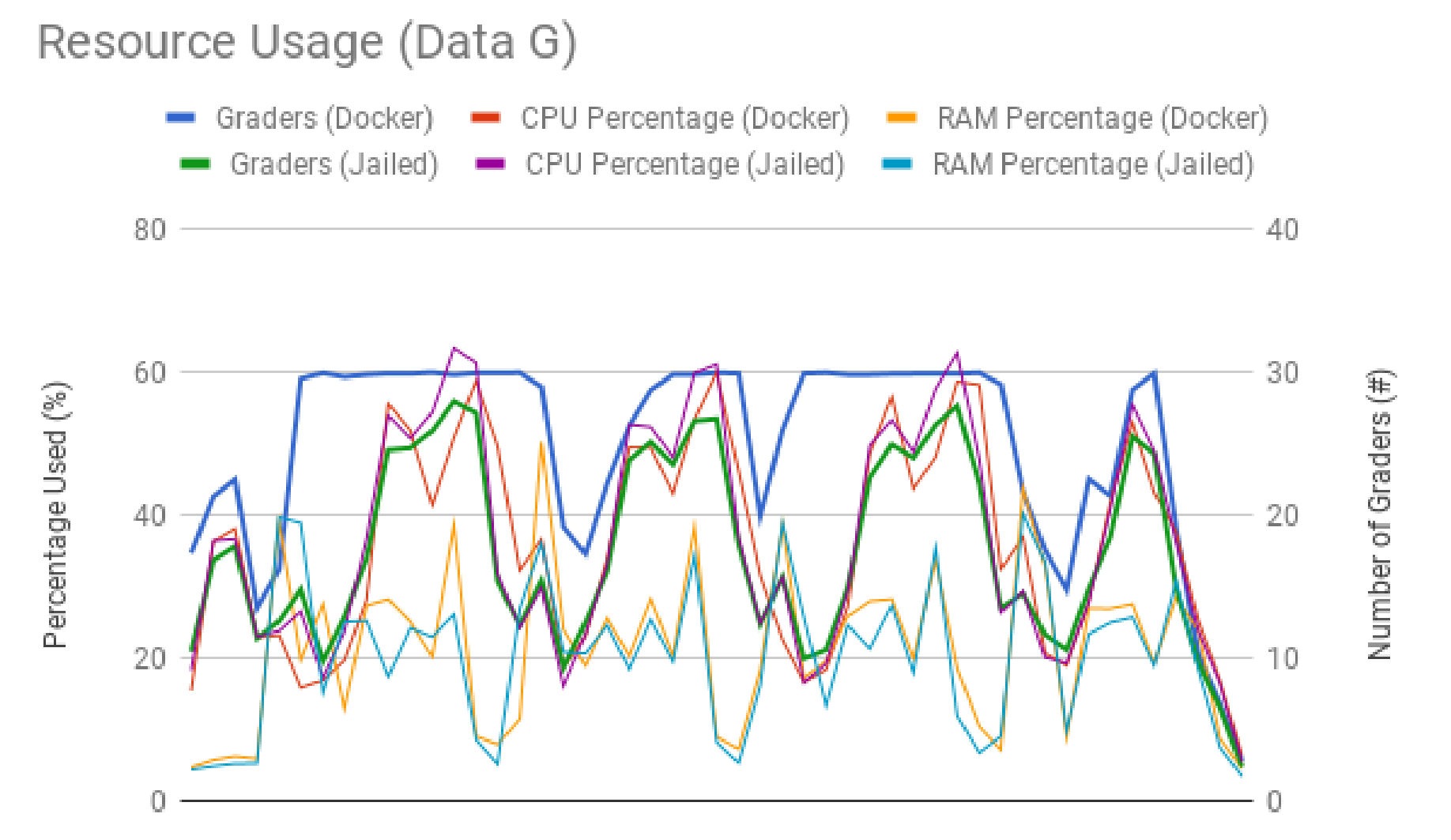


Results









Discussion

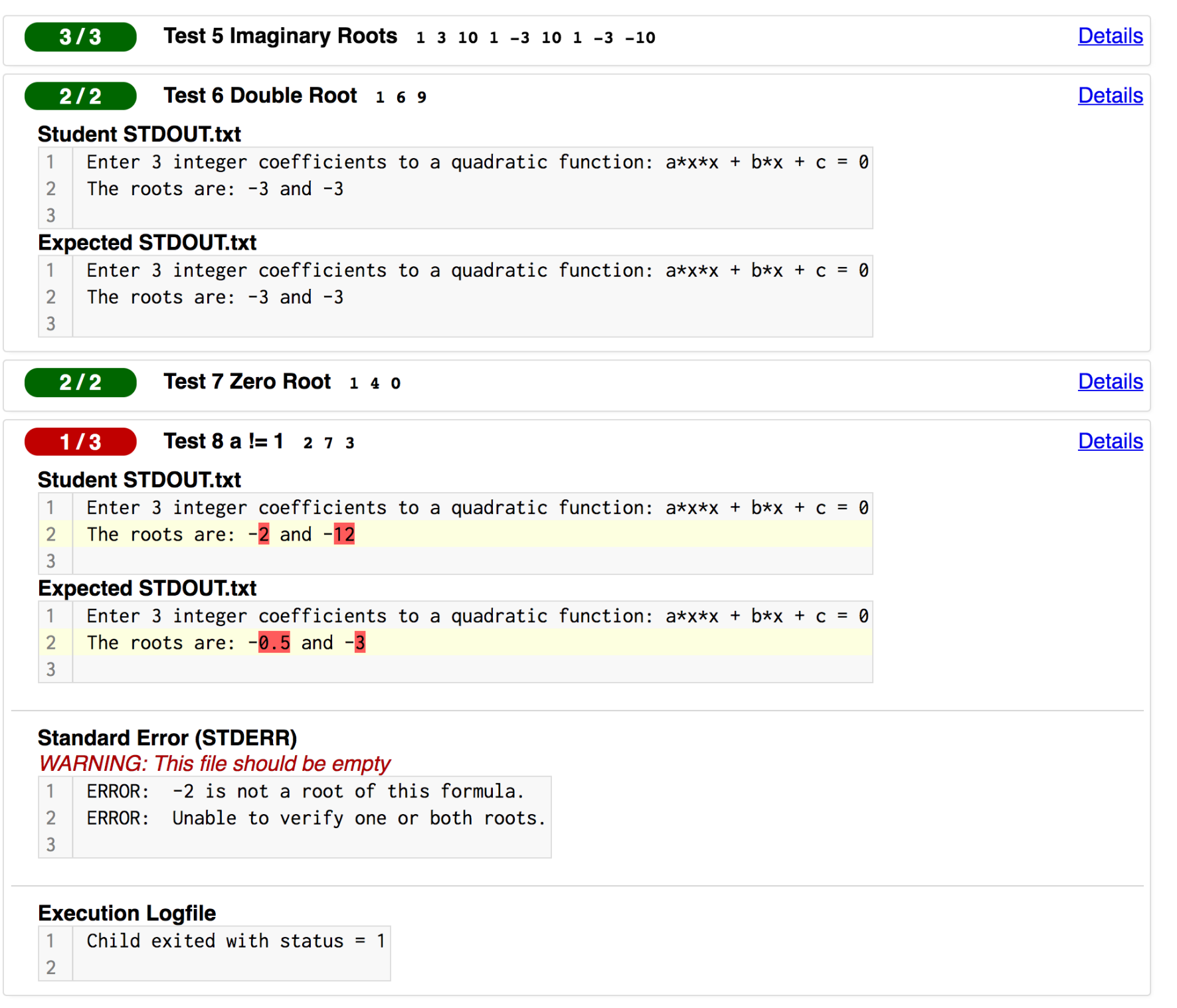
- CPU was roughly the same for both, while RAM was slightly higher for containers
- Grading time took an additional 3 - 25 seconds for docker compared to jailed sandboxes due to spinning up and destroy containers
- On a maxed machine (all grading processes engaged), grading throughput will be less for Docker
- Creating more containers than available virtual CPU cores causes system instability

Future Work

- Development of automated scheduling algorithms to pre-spin up containers
- Improvement of interface for instructors to build custom Dockerfiles
- Dockerfile builder to ensure correctness of packages
- Build interface similar to Docker for grading on full VMs/external servers

Submitty `http://submitty.org`

Submitty is an open source programming assignment submission system from the Rensselaer Center for Open Source Software (RCOS), launched by the Department of Computer Science at Rensselaer Polytechnic Institute.



```
3 / 3   Test 5 Imaginary Roots  1 3 10 1 -3 10 1 -3 -10                Details

2 / 2   Test 6 Double Root  1 6 9                                       Details
Student STDOUT.txt
Enter 3 integer coefficients to a quadratic function: a*x*x + b*x + c = 0
The roots are: -3 and -3

Expected STDOUT.txt
Enter 3 integer coefficients to a quadratic function: a*x*x + b*x + c = 0
The roots are: -3 and -3

2 / 2   Test 7 Zero Root  1 4 0                                         Details

1 / 3   Test 8 a != 1  2 7 3                                            Details
Student STDOUT.txt
Enter 3 integer coefficients to a quadratic function: a*x*x + b*x + c = 0
The roots are: -2 and -12

Expected STDOUT.txt
Enter 3 integer coefficients to a quadratic function: a*x*x + b*x + c = 0
The roots are: -0.5 and -3

Standard Error (STDERR)
WARNING: This file should be empty
ERROR:  -2 is not a root of this formula.
ERROR:  Unable to verify one or both roots.

Execution Logfile
Child exited with status = 1
```

Related Publications

- *Correlation of a Flexible Late Day Policy with Student Stress and Programming Assignment Plagiarism*
  Breese, Maicus, Peveler, and Cutler. SIGCSE 2018 Poster
- *Program Analysis Tools in Automated Grading*
  Dinella, Breese, Maicus, Cutler, Holzbauer, and Milanova.
  SIGCSE 2018 Poster

Acknowledgments

- Rensselaer Center for Open Source (RCOS)
- Red Hat Software
- Google Summer of Code 2018
- `https://github.com/Submitty/`