Development of an Open Source Programming Contest System

Jamel Charouel
University of Virginia
Jtc2dd@virginia.edu

Nathan Williams
University of Virginia
Nlw2sx@virginia.edu

Derek McMahon
University of Virginia
Dmm7aj@virginia.edu

Vivian Liu
University of Virginia
Wl2nw@virginia.edu

Austin Petrie
University of Virginia
Adp4fm@virginia.edu

Jason Deng
University of Virginia
Jgd3hb@virginia.edu

ABSTRACT

Student programmers across the globe compete in programming contests to showcase their problem-solving skills in a fast-paced environment. The logistics of running a programming contest can be difficult with many participants and many coding problems. There is free software available that can be clunky to use, and there is well put together software that must be bought or subscribed to in order to use. With the Programming and Instructional Contest Organizer (PiCO), an open source programming contest platform, contest organizers can run their own contests with software that is free and not clunky to use. Open sourcing the platform allows the public to add extensions to PiCO that might be specific to them, or add enhancements that might benefit anyone using the software and push them to PiCO's repository. This will also allow the software to be updated in the future. The combination of open source and quality design removes the need to compromise between price and reliability.

1 INTRODUCTION

The customer our team worked with was the University of Virginia Computer Science Department. Specifically, our team worked with Professor Mark Floryan who helps organize coding contests for students at UVA. These students practice to compete in regional coding contests under the International Collegiate Programming Contest (ICPC) in the Fall, where the top 100 (or so) teams qualify to compete at the world finals in the Spring.

Currently, the most widely used free system designed to run programming contests (and the one that our client currently uses) is the Programming Contest Control system (PC^2). Although this system is still updated for semi-regular use, it is sometimes difficult to use and has had problems in the past that have caused coding contests to be stopped and reset entirely.

There are many components to PC^2 in terms of software that must be run in tandem with one another in order for both judges and contestants to have a straightforward contest be run. This requires many windows be open on one person's computer at the same time, resulting in confusion and frustration. Problems with running contests can result in moving back an entire contest schedule due to technical issues.

The Service Learning Practicum class that our project is centered around is a year long course that places students into groups early on and has them work in sprints to complete requirements set by the customer, who is in this case chosen by the course administrator. Meetings are held every other week between the customer and the students to ensure the feature implementations align with the expectations of the customer. Another meeting is held with a graduate student in order to be sure that the team is on track to finish. Certain overall project deliverables are also outlined every two weeks that must be complete by the end of every iteration, with final requirements due at the end of each semester in the fall and spring.

Two Science, Technology, and Society (STS) projects that our group members conducted are related to the PiCO system. One researches the design decisions in programming contest management software and how they impact the computer science community. It goes into detail on the growth of online coding competitions and the direct effect it has on the technology industry. The other explores the role of open source education software and its potential to catalyze changes in global political paradigms via stimulation of education access in presently marginalized regions.

2 BACKGROUND

The system is built on the Django web application framework with the assistance of JavaScript. Docker, an

open-source project that automates the deployment of application inside software containers[1], is used to sandbox code execution. It is deployed on UVA's server at http://libra.cs.virginia.edu/pico.

3 RELATED WORK

There are currently two systems, PC^2 and Kattis, in the market that are used by the programming contest community. Both systems are custom-designed for programming contest purposes.

PC^2, short for Programming Contest Control System, is an open source application developed at California State University, Sacramento (CSUS) in support of International Collegiate Programming Contest activities of the Association for Computing Machinery (ACM)[2]. It is used in the ICPC regional contest and provides a medium through which teams may submit solutions to computing problems. Through the same tool, judges can review submissions and provide feedback. The problems are computationally involved and submissions are in the form of a program that outputs a solution as prescribed by the corresponding problem.

The system is a stand-alone application that needs to be downloaded and set up properly before it can be used. It only supports running one input file when judging the submission, but some programming contests provide test cases in multiple input files. Moreover, users found some features are not easy to use. For example, output of participant's submission and the example output is not well-aligned. The user has to manually scroll down both sides to find lines where differences appear. Also, although the system supports automatically judging the submissions, judges need to manually create multiple new users as auto-judges and assign each to a problem.

The other standard system used in ACM ICPC competitions is Kattis. It is currently the most popular commercial option available. Kattis is a general purpose online coding challenge website. It advertises as a platform for companies to interview potential candidates, universities to assess their students, and individuals/teams to practice and compete in coding challenges. Kattis is similar in many ways to other popular coding challenge web applications such as Topcoder and hackerrank.com. It has a wide array of capabilities that provide users with the tools to create coding problems and assign them automatic graders to judge solutions submitted by participants. It has also become a community for programmers to discuss problems and connect with employers looking to fill software positions.

Beyond these primary aspects Kattis markets to, it is also used as the contest system for the ICPC World Finals, along with other regional and smaller-scale competitions. For the most part, it excels at everything in which PC^2 falls short. It requires much less effort to set up and can handle more participants and problems per contest. It is a very powerful and versatile product, but the key limiting factor for its use in more competitions is its steep costs. The lowest starting price is $900 per month, for up to five users. This price makes it difficult for most individuals, teams, and competitions that aren't significantly funded from being able to gain access to such a system. Therefore, Kattis was mainly able to serve as a useful guide of features for us to implement while designing the PiCO system.

4 SYSTEM DESIGN

Organizing the system at a high level was a challenging aspect in the earlier stages of design. Naturally, as PiCO evolved, the structure of the system needed to adapt. Nevertheless, the system architecture eventually reached an equilibrium that favored simplicity and reusability, while avoiding excessive fragmentation.

While applications within the Django project were created to satisfy new system requirements, models were assigned to the most appropriate application. In many ways, the models actually determine what apps should exist within the project. A template hierarchy gradually formed as well, taking advantage of the template inheritance capabilities built into the Django framework.

Alongside the development of the system architecture, the team faced a number of pivotal decision points in terms of design: the flow of the contest functionality, execution of code, the diff table within the judging interface, and the user statistics.

4.1 System Architecture

The system consists of five applications: users, teams, contests, alerts, and stats. The user app is developed base on Django's user model, with the support of a user profile that allows the user to change the website theme. The team app supports user invites, joining, and leaving a team. Teams can be public or private, requiring a request to join if private. The contests app handles all functionalities related to a programming contest, including contest template, contest invitation, problem, submission, participant, notification of submission, and scoreboard. The alert app handles generating and displaying alerts for team invitation and join request. The stats app creates statistics reports for teams' past contests.

Templates are separated into different folders according to which app it is serving. A base template is extended by all templates. It contains the style of the website. Contest also has a base template extending from the general base and is extended by all contest templates. It contains common parts shared by the contest pages such as navbar.

4.2 Creating and Running a Contest

When creating a contest, the user has the option to load a previously saved contest template which consists of

information such as contest length, languages supported, time penalty, judges, and participants. Since programming contests usually have similar settings, this feature is designed so that the creator of the contest can apply the default template and save time configuring redundant information.

The Participant model is designed as a link between teams and contests. Once a contest is created, an invitation will be sent to the teams who were added as participants. A team needs to accept the invitation before a Participant object containing the information of the contest, team, and score is created, otherwise the invitation is deleted. This model makes it easier to keep track of the team behavior within a contest.

One challenge in running a contest is handling the authority. Only a subset of the participants can view different sections of the contest, and the authority changes along with the contest stage. A contest has three stages: created but inactive, active, and past. A creator and a judge of the contest can view the pages in any stage. A user can only view the contest when it is active or past if he/she is in the team participated in the contest. Judges may view the submissions from all participants in the contest, while users may only view submissions from their own team in a contest.

The contest scoreboard is used to display the overall team's progress on each problem as well as who is in the lead in terms of problem completion and time penalties/attempts at a problem. The scoreboard utilizes the Problems model alongside AJAX requests in order to update whenever a submission is submitted or judged, showing a team's pending submission or the judge's decision on their submission.

The problem in the contest page will be colored once a user submits an attempt. It uses yellow, green, and red to represent the submission as unhandled, correct, or incorrect. A notification window will pop up alerting the user when the submission is handled and returned from judge.

4.3 Contest Environment

Until fairly late into development, the format of a contest page was identical to any other page within the system once a user was authenticated. Navigation was standard across all of these templates, but in an effort to provide a distinct user experience while inside a contest, the base template was modified with that goal in mind.

A minimal navigation bar was favored, which does not include direct links to the other core apps (e.g. users, teams). Instead, a user who wishes to leave the contest environment exits through the logo in the navbar, directing them back to the home page. The remaining real estate in the navigation bar was designated to a breadcrumb for simple navigation within the contest.

The motivation behind this design was to ingrain users in the contest experience and not distract them with other portions of the website. Since PiCO is first and foremost a programming contest platform, it seemed appropriate to distinguish the contest experience from the rest of the system.

4.4 Code Execution

The execution of submitted code is largely centered around two technologies: Python's subprocess library and Docker. The subprocess library was used to kick off the Docker container, mount a temporary directory containing the necessary files onto that container, and then execute code within it.

Since there is a chance that a team submits malicious code to the server, submissions should not be executed directly. Docker provides sandboxing by executing the code in an isolated container. A virtual machine (VM) would be more difficult to break out of than a container and thus be more secure, but that was not feasible since VMs can take minutes to start up which was an unacceptable time frame for running code submissions. Docker, on the other hand, starts in a matter of seconds and still provides a significant level of security.

Code is run on the fly (i.e. at the time the judge goes to review the submission) so that the output does not have to be stored indefinitely on the system. This saves memory space on the server, although often slightly increasing the load time of the judge page.

4.5 Diff Table

In the judging interface, providing a means of comparing the output of a program submission to the expected output was imperative. This was an important feature to aid the judging process. The solution quickly settled upon was a table with a side-by-side comparison of the real and expected outputs with highlighted differences. To produce such a table, the two text groups being compared are represented as string lists, where each string is a line, and those two lists are diffed.

As opposed to delivering both outputs to the client and diffing the outputs on the frontend, diffing was executed on the backend in order to take advantage of the Python library *difflib*, which could generate a rich delta that could be processed into a table.

While digesting the delta, differences were labeled as whitespace or empty lines where appropriate. By this method, judges can change what differences are counted as errors and highlighted in the diff report on the client-side, avoiding a server ping to regenerate the program output and recreate the diff table under different settings.

4.6 Statistics

Teams or individuals that participate in UVa's ICPC competitions may want to be able to access information

regarding their past performances. The stats application displays to users various statistics regarding their past performances, such as number of problems solved, previous teammates, and more.

Users can select a past contest to view, which will display a graph showing how many problems they correctly solved in that contest and how long it took them to solve each problem.

We chose the statistics displayed to be those that are not necessary for the platform to function but are interesting for users to keep track of. Some of the information, such as 'Past Teammates', may help users find teammates for future contests as well. By keeping track of this cumulative data, participants can also identify trends in their working patterns that may help them improve in future competitions.

5 PROCEDURE

PiCO will be used by the UVA ACM ICPC team as a replacement for the currently used contest management software, PC^2. With the system deployed on the university server, users can sign up and organize into teams. Public teams can be joined by anyone, while users must submit a join request or be invited to join a private team. The public attribute of a team may be toggled by any member of the team.

With teams established, any user may create a contest and invite teams to participate. Each contest has a set of problems as well as a set of users who have judging privileges for that contest. To save time when creating future contests with similar properties, contest templates can be created and used to pre-populate fields in the contest creation form. Following the creation of the contest, a judge can activate it, allowing participating teams to begin submitting solutions to the problems. Judges can judge those submissions and provide feedback. Participants can also view a scoreboard inside the contest platform.

Users are given the capability to edit their profile information, and under user settings a user can customize the appearance of the site for their account by choosing a predefined theme. Site admins also have the ability to monitor disk usage, which distinguishes usage by the PiCO source code from the database disk real estate and consumed disk space from outside the system.

6 RESULTS

PiCO is effective at simplifying user needs regarding contests and contest management. The entire system is maintained through a website on which users can register accounts for free. This makes our system very attractive and accessible to new users as well as new contest-based organizations to use as a platform for managing their contests. Users will no longer need to keep their system up to date with the latest versions of an application program, which is an inconvenience associated with PC^2. Hosts will not have to pay any expensive subscription fees, either, which is a downside of Kattis. Additionally, the PiCO platform is released as open-source software, allowing users with technical knowledge to customize the platform to their own contest-specific needs. This also promotes the expansion of our contest management platform into a wider range of use cases.

Because PiCO is currently designed for programming contests, it aims to be as user-friendly as possible. Contest participants and managers are presented with relevant information such as their scores and time remaining without extraneous distractions on the landing page, allowing contestants to focus on the problems instead of navigating the system. Contestants can see their submissions in a compact easy-to-read window, allowing them to directly compare their submission's output with expected output. The code execution capabilities using Docker allows users to execute code with reduced risk of damage being done to the server. Because Docker loads much more quickly than a virtual machine would, it allows users to receive their judging feedback faster. Contest organizers are able to efficiently create new contests by uploading problem files and solutions. Participants can be added quickly through drop-down selections as well within the same form. During active contests, they can access the submissions of various participants for manual judging and evaluation. Alternatively, autograding is a setting that contest organizers can toggle in order to speed up the grading process. Using PC^2, a separate user must be created to be given auto-judge privileges, which also requires two machines being used, one to host the contest and another to run the autograder. Using PiCO, the autograding functionality is a setting within contest creation, requiring only one computer and no extra users.

These features make PiCO an excellent option to use for running programming contests. They focus on usability and allowing users to spend more time on the contest rather than learning to navigate the system.

7 FUTURE WORK

Given more time to enhance the system, one improvement would be to "Dockerize" the entire web app. As previously mentioned, a docker container was implemented in order to provide some level of sandboxing to code execution. However, rather than providing sandboxing, running the entire system in a Docker container would instead capitalize on the increased portability provided by containerization. By wrapping the application and its dependencies into a lightweight package, the app would run consistently on any platform that has Docker installed.

A significant addition that would provide the most value to the current client would be to add classroom functionality. Professor Floryan of the University of Virginia has suggested that the system could be modified for use in a

flipped classroom setting, where there are lecture videos that students watch outside of class and graded problems that are done on the site in class. This could be accomplished either by making the system more general purpose or providing separate modules for both classroom and programming contest settings.

Other useful additions would enhance the capabilities of contest participants and administrators. For instance, some existing online contest websites (e.g. Leetcode) provide avenues for challenging the solutions of another team if they believe that they plagiarized or cheated the system. To enable plagiarism checking, an algorithm would have to be implemented to analyze the submissions for code copying. A similar functionality related to invalid submissions would be giving the judges the capability to declare built in functions that are illegal in a given question. If the problem is to manually implement atoi(), then clearly the built-in atoi() would be prohibited and the system should ensure that submissions do not use it. Additionally, it could be useful to give judges the ability to use validators to check solutions. The client had made this optional since he doesn't believe it is used that often and it can be difficult to acquire the validator to a problem. However, it would be useful to have, as major competitions such as the ICPC have defined standards for validators.

Improvements also can be made facilitating the interaction and collaboration of users. For example, at the conclusion of the contests discussion boards could open that allow people to talk about different solutions to the problems and would include standard upvote/downvote functionality to bring the highest quality posts to the surface. This would help to foster a community centered on the growth of computer science knowledge. Building off of this would be features that allow for more personalization of individual profiles. Giving a user the ability to put a picture with their profile would enable others to recognize them in the discussion board, and histories of their posts would provide insight into their track record.

A more subjective addition would be the development of a section strictly for practice. Including past problems and related solutions would give users a way to improve their skills outside of a live contest. However, the added complexity in the code base and the site as seen by the end user may not be worth it when considering all of the existing external resources for practicing this skillset. The development of this feature would certainly be subject to future analysis regarding the balance between fitness for the original purpose of managing programming contests and providing a full user experience in a single location.

ACKNOWLEDGMENTS

Professor Mark Floryan of the University of Virginia was an active and helpful resource throughout the development process, providing useful feedback and well-defined requirements. He was a responsive customer and involved stakeholder of PiCO.

REFERENCES

[1] "Docker". Wikipedia, The Free Encyclopedia. April 10, 2017. https://en.wikipedia.org/wiki/Docker_(software)

[2] Measure, M. CSUS Programming Contest Control System. (PC^2). http://pc2.ecs.csus.edu/

[3] Sale, D. Dockerinzing A Python Django Web Application, Semaphore. September 7, 2016. https://semaphoreci.com/community/tutorials/dockerizing-a-python-django-web-application

[4] N.A. Docker Security. Docker Docs. 2017. https://docs.docker.com/engine/security/security/#docker-daemon-attack-surface