Olympiads in Informatics, 2014, Vol. 8, 169–178
© 2014 *Vilnius University, IOI*

REPORTS

omegaUp: Cloud-Based Contest Management System and Training Platform in the Mexican Olympiad in Informatics

Luis Héctor CHÁVEZ, Alan GONZÁLEZ, Joemmanuel PONCE
omegaUp
Hacienda de Coaxamalucan 138, Col. Hda. de Echegaray
Naucalpan, Estado de México, México CP 53300
e-mail: {lhchavez,alanboy,joe}@omegaup.com

Abstract. omegaUp is an open source cloud-based online contest and training platform for the Mexican Olympiad in Informatics. It is designed to be a robust, highly scalable, low cost and secure solution. To achieve our security goals, the platform leverages *minijail*, a modern and actively maintained sandbox from the ChromiumOS project with a good security track record as well as other features for grading task solutions reliably. Students can solve past and ongoing contests as well as training problems with live feedback. omegaUp also provides easy to use administrative features enabling users to upload tasks for automatic grading without staff assistance and create their own contests using any previously uploaded task. Since omegaUp's launch 3 years ago, it has hosted more than 400 contests for both IOI training and ACM-like competitions, graded more than 100,000 submissions, and helped the Mexican Team achieve their best performance in the IOI finals to date.

Keywords: automatic grading, contest management system, omegaup, *minijail*, cloud, security.

1. Introduction

In 2010 there were several online judges and training platforms in Mexico, each one targeting a narrow community. There were training gates for one particular language, like

Karelotitlan (Karelotitlán, 2011) (only hosting Karel problems for training), and some states also had their own training platform or contest management system. National contests usually suffered from scalability problems: when bursts of load happened, live feedback was often affected and the contestants' experience was unnecessarily stressful because of the grading system. States typically did manual or semi-automated offline grading. Due to the fragmented effort, none of the platforms invested enough time in designing and implementing a solution with the scalability, security and easy-to-use manageability features in mind that national competitions demand. It was often the case that only the developer of the platform had the ability to upload tasks, set up contests and make modifications, also making the process not scalable from the human resources perspective. Last-minute corrections to the test cases, and sometimes manual verification of the student's submissions made grading a time-consuming and error-prone process, which involved several extra hours of organizers' time. The result was that training for IOI and keeping track of the student's progress was a very ad-hoc process.

omegaUp was created to become a platform that could satisfy the needs of the whole country to have a centralized, properly maintained and reliable training gate and contest management system. It was designed to have easy to use administrative features, so contest organizers across the country can create, manage, and monitor their own contests and even upload their own tasks for automatic grading. This also means having a reduced dependency on the staff that runs the platform.

omegaUp is now the official platform to host contests for both the national and several state olympiads. Instead of every region maintaining their own platform and their own similar but slightly incompatible grading methods, having a single site for the whole country provides a more homogeneous experience for contestants across the country. This is also helpful for tracking progress since the pool of previous problems is now shared and available in a single portal.

The contest environment is designed to be a robust, highly scalable, low cost and secure solution. Security is achieved by designing a role-based permission model for the whole site and isolating all untrusted components as much as possible: nodes running contestants' code are hosted in virtual machines in the cloud and leverage *minijail* (Chromium, 2010), a modern and actively maintained sandbox from the ChromiumOS project with a good security track record. It is also possible to connect to the omegaUp server in a firewall-friendly way that locks down the permissions even more to make stronger isolation guarantees.

Running contestant's code in the cloud also helps the platform achieve our scalability goal: to guarantee a good experience for all the contestants during a live event, regardless of the size of the contest. The platform leverages from Microsoft Windows Azure (Windows, 2010) to host the compiling and grading processes, making possible to scale from one grading machine to dozens of them within minutes with very low cost. The site also makes heavy use of caching systems to store the result of expensive calculations temporarily, effectively improving the number of requests per second the system can handle.

In this paper, we describe how omegaUp works and how it is used to run the Mexican Olympiad in Informatics as well as other national programming contests.

2. Informatics Competitions in México

Mexico holds its national olympiad in informatics (Olimpiada Mexicana de Informatica, OMI) on a yearly basis. Each of the 32 Mexican States train and select their best 4 contestants who participate in the national contest. Our National Olympiad has 100 participants on average. Most of the states use a similar strategy and have their own established local informatics committees and run their own State Olympiad in Informatics as well as training tracks.

Mexico started its participation in the International Olympiad in Informatics in 1992. Since then, the process of determining our IOI delegation has evolved (Cepeda and García, 2011). We currently select the best 32 competitors from our national contest and prepare them targeting the IOI to be held the year after our National Olympiad. Our 4 IOI participants are selected as the result of a training program that lasts about 10 months, consisting of 4 training camps and several rounds of online contests and practices. A group of 20 collaborators with previous international contests experience donate their time to prepare, create and translate more than 80 tasks used in the selection process.

We have identified several factors that make our informatics development challenging. Our territorial extension with respect to our economic development often bounds our training camp organization. Informatics is not considered a first-class assignment in our basic education system. Students usually arrive at a late age to competitive programming with respect to other countries. Furthermore, the understanding of English language is limited on most of our contestants in early stages, reducing the sources of self-training information they can read and practice against.

Our mission with omegaUp is to overcome the aforementioned challenges with a platform that sets our country in a position to achieve better results in the International Olympiad in Informatics and, in general, contribute to the development of computer science in our country.

3. Architecture and Design

The main design goals that omegaUp is set to solve are:

1. It must be an always-on Contest Management System where contents are provided and driven by the community itself, without site administrators' interaction.
2. It must provide a secure environment to run untrusted contestant's code, preventing cheating.
3. It must provide a scalable and low-cost contest environment where nodes could be added to the system using cloud computing services in case of an increase in load.

omegaUp uses a multi-tier service-oriented architecture, with physical separation between components, as seen in Fig. 1. The basic workflow of interaction between omegaUp services is as follows. When a contestant submits source code to be graded, the Frontend web interface relays that message to the Grader service, who orchestrates the workflow to get the code and input data (if needed) to an available Runner node that

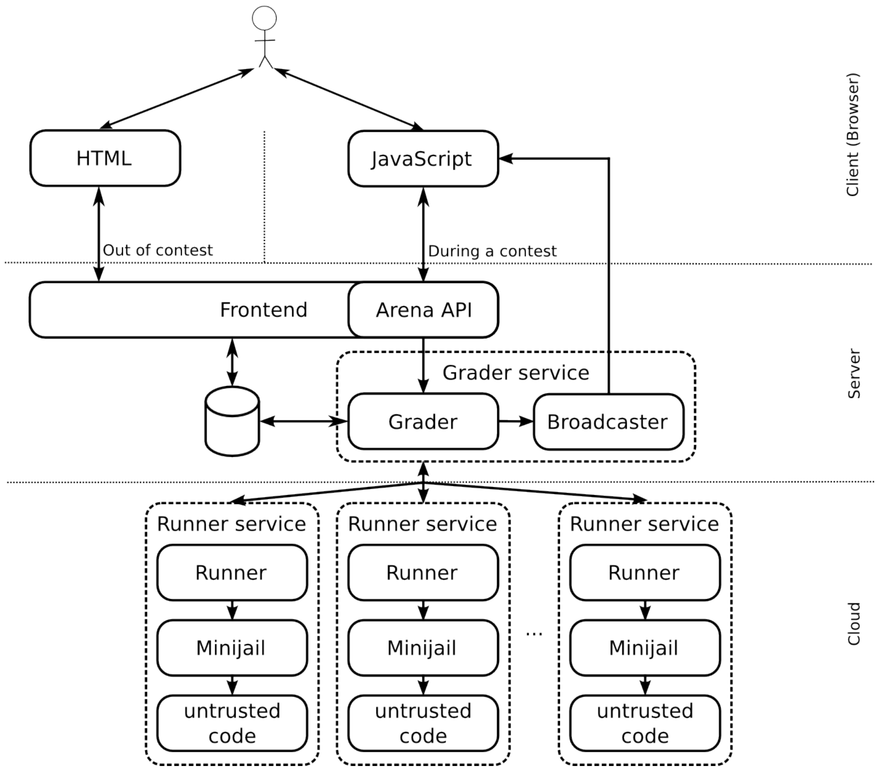

Fig. 1. omegaUp architecture.

compiles and executes the contestant's code in a sandbox. Afterwards, the Grader receives the outputs produced by the contestant's program from the Runner, compares it to the expected output and it emits a final verdict which is then sent back asynchronously to the contestant via a Broadcaster service. The next sections further explain each of the components.

All communication between components is encrypted using SSL to preserve integrity and confidentiality of the messages, and all server-side and cloud-side components use a certificate chain rooted in a self-signed Certificate Authority that provides mutual authentication to prevent contestants invoking any service directly. Isolating the expected case data is also important to provide security in depth, so even if contestants find vulnerabilities in the sandbox inside a Runner node, no expected outputs are ever stored in the machines that run untrusted code, making cheating extremely difficult.

Given the elasticity of the computing resources, it is possible to run 5-hour contests with an average verdict latency of 2–30 seconds (depending on the number of cases and time limit) with a total cost of 3 US dollars. Based on the same elasticity property, we can still achieve a good and responsive contestant experience even when there are changes to the task's test cases in the middle of the contest that require a sudden burst of hundreds of submissions for re-grading. Furthermore, the current price trend of cloud services with major providers indicates that running a contest will cost us even less in the future.

One trade-off about running contestants' code in cloud computing services is an expected lower consistency of performance between machines: there are slight but detectable variations between instances of a cloud compute provider's service, even at the same price level (Ou *et al.*, 2012). To avoid this, other online judge services such as Sphere Online Judge have dedicated hardware clusters to evaluate contestants' code (Sphere, 2008). Nonetheless, it does not affect us too much, given that most tasks are designed to clearly distinguish between implementations of the expected complexity and a worse one. Running a 5% sample of the corpus of submissions on different clusters of the same provider at the same price level[1] resulted in a score change for at most 0.73% of the submissions. Furthermore, when comparing Amazon's m3.medium against Windows Azure's A1 instances, we found that only 2.06% of those submissions would have changed score if evaluated in the other platform.

3.1. *Frontend*

All user-facing interaction is done through the Web Interface layer, written in HTML 5 and Javascript. Users of omegaUp do not have to download any external plugin or dependency to use the service.

Since security is a priority for omegaUp and to use all HTML 5 features, support for Internet Explorer previous to version 8 was explicitly dropped. This also reduces development costs. Modern browser security features such as Content-Security-Policy and Strict-Transport-Security are used by omegaUp and further guarantee the integrity of the platform.

All business logic for contest creation and management is processed by the Frontend web service. The Frontend service is written in PHP, running on top of Facebook's HHVM and served by nginx. To maintain proper separation of concerns, all interactions between the Web Interface and the Frontend service are strictly done via JSON messages using a RESTful API (omegaUp, 2011). When a contestant submits a source code to be graded, the Web Interface calls a Frontend API to register the grading request in omegaUp. The Frontend then informs the Grader service about the new submission via Grade Request Message delivered through a REST API call.

The Frontend provides the administrative interface to manage contests and tasks, generates statistics and rankings, and hosts the main contest interface which has the standard CMS features such as clarifications, viewing past submissions, and a real-time scoreboard using WebSockets.

This layer is also the one responsible for all role-based authentication, sign-on through Google, Facebook, and native logins, as well as contest policy enforcement. There is an internationalization infrastructure built in and we support both English and Spanish, although the vast majority of task descriptions are only available in Spanish.

[1] m3.medium in the case of Amazon and A1 for Windows Azure

A lockdown mode is available for contests that require tighter security controls and network-level isolation. Visiting the site using an alternative URL[2] enables this mode and ensures all resources are served from the same endpoint, simplifying the firewall rules that are needed in the physical contest site to achieve the desired level of isolation. Furthermore, most of the site's features are disabled in this mode, especially those that can modify contests or problems, as well as all known scenarios in which contestants can communicate with each other through site features like viewing source code of past submissions. This mode only requires that passwords are secret and are not shared between contestants.

3.2. *Grader*

The Grader service is a fully asynchronous Scala service that communicates through JSON messages over HTTPS with client authentication. After a Grade Request Message is received, it is routed to one of several runner queues that have an associated pool of Runner nodes. This allows us to provide coarse Quality-of-Service guarantees: tasks that are designed to always finish within 30 seconds in the worst case will run in the default queue and provide a very fast response time to contestants; tasks that can take more than 30 seconds will be sent to a slower queue to avoid bogging down the whole system. Additionally, if the time to response for a contest needs to be isolated from the effects of other submissions in the system, a private runner pool can be associated with a queue that will exclusively process submissions to said contest. If a task's test cases are modified while running a contest, re-grading of submissions can be done in yet another queue to further minimize impact. This allows us to very quickly react to unexpected live contest issues and mitigate disruption to contestants. Submission states are backed by a MySQL database and Grader can rebuild the submission queues upon restart.

Each queue has an associated pool of Runner nodes that handle the request in a producers-consumers fashion. Messages to the runner nodes are also sent using JSON over HTTPS. Once the runner node finishes running the task, its output is compared in the Grader service against the expected output using one of several built-in tokenizers, or can be sent back to a Runner node if a custom grader is required. This scenario usually happens when tasks do not have a unique solution, so more untrusted code needs to be executed to come up with a verdict.

Graded submissions' results are then sent to the Broadcaster component, which updates the contest scoreboard and notifies contestants of the verdict of their submissions. Broadcaster uses WebSockets to send near-real-time updates directly to contestants' browsers, avoiding constant periodic polling and providing a low-latency solution.

3.3. *Runner*

The Runner service is written in Scala, and runs using cloud computing services. We have used both Amazon Web Services and Microsoft Windows Azure as virtual machine

[2] `https://arena.omegaup.com/`

providers with good results. Running a task is straightforward: the runner receives a JSON message containing the source file, the execution limits and other task parameters. The sandbox is then invoked for both the compilation and evaluation of each of the tasks' cases, and sends the results back to the grader using a combination of JSON and a custom bzip2-ed stream with the case results over HTTPS.

Runners are designed to hold the least possible amount of state and be light on configuration. Runners do not need to be pre-registered on the grader: they register themselves dynamically upon start on the default queue in case there is a need to spin off new runners. Task inputs are lazily deployed from the Grader into a Runner until a submission needs it and are cached for subsequent requests using the SHA-1 hash of the data as key. Contest administrators can request the re-grading of any submission in a way that debugging information and errors are displayed in the administrative interface, which is especially helpful to diagnose issues with interactive tasks or custom graders.

3.4. *Sandbox: minijail*

When the omegaUp project started, Moe sandbox (Mareš, 2009) was chosen as the sandbox implementation, since it was also used in the IOI. Some modifications were made to support multithreaded languages (such as Java) with very lightweight race condition exploit mitigation, multi-process support to also sandbox code compilation, and syscall interception to be able to fake dangerous calls without crashing (e.g. Java tries to open sockets during initialization).

As omegaUp evolved, it soon became obvious that the approach using a ptrace-style sandbox was not sufficient for a variety of reasons, including being vulnerable to TOCTOU races (Isolation, 2014), introducing a large overhead per system call (Merry, 2010), and that maintaining it was costly since several updates to the kernel broke the sandbox in non-trivial ways.

For the second version of the sandbox, *minijail* from the ChromiumOS project was chosen (Chromium, 2010). It has a more modern architecture, and uses two recent Linux kernel security mechanisms: seccomp-bpf moves the syscall filtering to the kernel using compiled bytecode (Drewry, 2012), making it both very efficient and immune to race conditions while still providing syscall interception to return an error on certain syscalls instead of terminating the process; Kernel namespaces provide process-level isolation to the rest of the file system and the network.

minijail allowed the use of unmodified interpreters by providing a whitelist of allowed system calls with negligible overhead. This allowed us to expand the list of supported languages to C/C++/C++11, Pascal, Java, Python, Haskell, Ruby, and a Pascal-based Karel compiler with minimal effort. *minijail* is also actively maintained and used in a commercial application, so any exploits found are likely to be fixed quickly.

It is important to note that omegaUp considers both contestant's code execution and the compilation process as untrusted, so they are run within the sandbox. There have been Denial of Service (DoS) attacks on compilers such as the Java double parse bug (MITRE, 2010) and it is very easy to abuse the C++ error messages to generate gigabytes of output with very small inputs (TGCEEC, 2014).

4. Contest Management

Any user of omegaUp can create and manage their own contests as well as tasks. Contest administrators are not limited to IOI or ACM-style contests. Instead, traits such as the penalty policy, a score decay factor, and scoreboard display policy are freely configurable. Tasks and contests can be configured to be either public or private, with security options to avoid information leaks.

There is support for editing task descriptions online using a slightly modified Markdown syntax, and infrastructure changes are underway to enable a peer-review system to raise the quality bar of problem statements and minimize last-minute changes. The use of Markdown instead of free-style HTML was chosen to maintain a more consistent look and feel for tasks descriptions.

Having a centralized national task repository with standardized rules and expectations has helped not only train for the IOI, but has spawned or helped improve several other regional and national level contests, since they do not have to worry about the infrastructure to make a big successful contest. We even have visitors and contests from other South American countries, such as Colombia and Bolivia.

5. Open Source and Openness

All of omegaUp's source code is freely available from GitHub with a BSD license[3]. We also run on top of a fully open source stack: nginx, HHVM, Debian/Ubuntu GNU/Linux with MySQL databases. Contributing to omegaUp is also easy since we provide a downloadable Vagrant instance that is set up to match our production configuration.

We are also committed to building an open platform: we encourage people that upload tasks to omegaUp to make them public for everybody to use and practice. This is a vast improvement from the previous status quo of contests in Mexico: once a contest was finished, task data was rarely provided, and even when it was, the data was usually lost after some time, or was provided in a format that was not useful outside of the particular contest environment used.

Community

The community around omegaUp is not limited to the contest management system. We also maintain a Q&A site (similar in spirit to StackOverflow) where students can ask and answer each other's questions. We also run a blog where task creators can post explanations to the solutions of their tasks to help students that are stuck.

Also, since the contest system is not designed exclusively for IOI-style tasks, ACM-ICPC student chapters across the country and other organizations with national programming contests like the Mexican National Open Programming Contest CONACUP (CONACUP, 2013) have chosen omegaUp to host their contests and training sessions.

[3] `https://github.com/omegaup/omegaup/`

6. Results and Future Work

We are convinced that omegaUp is well positioned to solve most of the problems it was originally designed to attack. In particular, the state of online contests in Mexico has greatly improved, a large and growing number of states use the platform for their local olympiad qualifying contests and to help with their training tracks. New types of competitions have been created thanks to the openness of the platform. Over the next months omegaUp will become easier to use for novice students and non-students, and the platform will move to a more autonomous system, where the users provide and review the site content.

Since omegaUp was launched in 2011 it has hosted more than 400 contests for both IOI training and ACM-like competitions, graded more than 100,000 submissions and currently provides more than 1,000 practice problems.

Other goals like improving the results of the Mexican Team in the IOI and building an integral training gate targeted to younger students are more long-term, so it will be an ongoing effort for a few years to come. So far we can say that, since 2011, omegaUp has helped Mexico win 4 Bronze and 2 Silver medals. This already represents roughly 50% of what we achieved from 1993 to 2011: 7 Bronze and 1 Silver medals. We believe these are just the early signs of its full potential.

7. Acknowledgments

César Cepeda for supporting the project financially and providing invaluable guidance. Also thanks to all the open source contributors, in order of number of commits: Juliana Peña for doing the initial user experience implementation, Pablo Aguilar, Abraham Toriz, Hugo Dueñas, Alexis Cervantes, Freddy Román, Omar Ocegueda, Alberto Munguía, and Rubén Rodríguez.

References

Cepeda, A. and García, M. (2011). Mexican Olympiad in Informatics. *Olympiads in Informatics*, 5, 128–130.

ChromiumOS Design Docs – System Hardening (2010).
http://www.chromium.org/chromium-os/chromiumos-design-docs/system-hardening

CONACUP – Concurso Nacional Abierto de Programación. (2013). http://conacup.org/

Drewry, W. (2012). *Dynamic Seccomp Policies* (*Using BPF Filters*).
http://lwn.net/Articles/475019/

Isolation – The Confinement Principle. (2014).
https://crypto.stanford.edu/cs155/lectures/03-isolation.pdf

Karelotitlán, (2011). http://www.cmirg.com/karelotitlan/

Mareš, M. (2009). Moe – design of a modular grading system. *Olympiads in Informatics*, 3, 60–66.

Merry, B. (2010). Performance analysis of sandboxes for reactive tasks. *Olympiads in Informatics*, 8, 87–94.

MITRE CVE-2010-4476: The Double.parseDouble method in Java Runtime Environment (JRE). (2010).
https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-4476

omegaUp REST API documentation. (2011).
https://github.com/omegaup/omegaup/wiki/REST-API

Ou, Z., Zhuang, H., Nurminen, J. K., Ylä-Jääski, A., Hui, P. (2012). Exploiting hardware heterogeneity within the same instance type of Amazon EC2. 4–4.

https://www.usenix.org/system/files/conference/hotcloud12/hotcloud12-final40.pdf
SPOJ Clusters. (2008). http://www.spoj.com/clusters/
The Grand C++ Error Explosion Competition. (2014). http://tgceec.tumblr.com/
What is Windows Azure. (2010).
http://azure.microsoft.com/en-us/overview/what-is-azure/

L.H. Chávez is an ACM-ICPC world finalist (2010) and has a bachelor's degree in computer science (2011) from Tecnológico de Monterrey, Campus Querétaro. He has been involved in several efforts to improve the state of programming contests in Mexico since 2007, and is one of the co-founders of omegaUp. He is currently employed at Google in the Chrome team and is also studying towards a MSc in computer science from Stanford.

A. González has a bachelor's degree in computer systems engineering (2012) from Instituto Tecnológico de Celaya. He created Teddy Online Judge in 2008, which hosted several national-level contests. He is currently a Software Engineer in Microsoft, working in the Operating Systems group.

J. Ponce participated in the IOI 2005 and was Mexican IOI Deputy Leader in 2009 and 2010. Has a bachelor's degree in computer science (2011) from Tecnológico de Monterrey, Campus León. He contributes to the Mexican Olympiad in Informatics coordinating the national team selection process since 2009 and currently works for Microsoft as a Software Engineer in Windows Azure SQL DB.