An Elastic Platform for Large-scale Assessment of Software Assignments for MOOCs (EPLASAM)

Michael Walker, Douglas C. Schmidt, and Jules White

Vanderbilt University, Nashville Tennessee, USA
{michael.a.walker.1, douglas.c.schmidt, jules.white}@vanderbilt.edu

1 Introduction

1.1 Emerging trends and challenges

A *Massive Open Online Course* (MOOC) is a web-based class environment aimed at open large-scale global participation via the Web (EdX, 2015). In contrast to traditional forms of face-to-face (F2F) education, MOOCs enable flexible learning styles, where learners can pick and choose which classes they take, as well as when and where they do their work. Conventional methods and tools used in F2F education, however, are not ideal for use at scale in MOOCs. F2F methods and tools are particularly poorly suited to software-intensive MOOCs where the majority of the assessments focus on design and programming assignments. The following challenges must therefore be addressed by learning management platforms used for these types of MOOCs:

A. **Supporting the scale of MOOCs**. The number of learners in MOOCs is typically orders of magnitude greater than even the largest F2F courses (i.e., tens of thousands vs. hundreds). The tools and techniques for F2F courses are therefore not sufficient to handle the number of learners and/or assignments to assess. In particular, learning management platforms for software design and programming assignments require new tools/techniques to work effectively at the scale of MOOCs.

B. **Alleviating limitations of distance**. The distance between teaching staff and learners also increases significantly in MOOCs as compared to F2F classes, which introduces challenges that traditional F2F courses need not address. For example, it is infeasible for the teaching staff to meet with learners in software-intensive MOOCs to provide them with meaningful feedback on their assignments. Therefore, learning management platforms for these types of MOOCs must alleviate the challenges that distance can cause with respect to personalizing the learning experience.

C. **Providing security for the learning management platform and its users**. Security concerns arise when software submitted by learners is automatically compiled and tested. In particular, server-side compilation/execution and peer-review of code by other learners must be handled carefully to avoid security exploits, due either to malicious or accidental harm to the platform due to software bugs in learner solutions. The learning management platform for software-intensive MOOCs therefore requires a secure environment in which assignment submissions can be tested in isolation without harming the evaluation system or other learner submissions.

D. Minimizing the heterogeneity of system and network environments. Software-intensive MOOCs that focus on concurrent and distributed applications require multiple instances of

software running in their own networked environments. The architecture of these environments (e.g., the target architecture, such as mobile devices, embedded, or cloud services, etc.) may be heterogeneous and thus not identical to the assessment platform (e.g., x86 vs. ARM processors). The learning management platform should therefore support customizable target hardware architectures and dynamic virtual network topologies.

E. Reducing operational overhead. Various operational issues must be addressed by learning management platforms for software-intensive MOOCs, including information technology (IT) infrastructure tasks, such as logging and auditing of system execution and stability, load balancing, and data loss prevention strategies. Other operational issues relate to (1) transferring learner submissions from the front-end MOOC hosting server(s) to the learning manage platform and returning result(s) back to the front-end servers and (2) handling robust and secure peer-evaluation of learner submissions.

1.2 Solution approach → An *Elastic Platform for Large-scale Assessment of Software Assignments for MOOCs* (EPLASAM)

To address the challenges listed above, we are developing EPLASAM, which is both a learning management platform and method aimed at supporting the needs of learners and teaching staff in software-intensive MOOCs. EPLASAM is based on our experiences teaching the *Mobile Cloud Computing with Android* (MoCCA) MOOC course sequence, which is the first trans-institutional Specialization offered on the Coursera platform (Coursera, 2015). Over 400,000 learners have taken the MOOCs in the MoCCA Specialization over the past several years.

EPLASAM provides several contributions to R&D on learning management platform support for software-intensive MOOCs:

- It employs a model-based method of creating assignments, which addresses the challenges of (A) *Supporting the scale of MOOC courses* and (B) *Alleviating limitations of distance* listed above via various capabilities, such as (1) personalized assignment creation for different learner cohorts to minimize the impact of plagiarism and enable controlled experiments on different teaching methods, (2) testing-at-scale to evaluate learner submissions for correctness in an automated manner, and (3) meaningful evaluation and feedback to personalize learner experiences.
- It provides a scalable, secure, and customizable learning management platform for compiling, testing, executing, and assessing software-intensive assignments, which addresses the challenges of (C) *Providing security for the learning management platform and its users*, (D) *Minimizing the heterogeneity of system and network architectures*, and (E) *Reducing operational overhead* listed list above via various capabilities, such as (1) securely testing submitted code to ensure a robust process of compiling, executing, and assessing learner assignments, (2) standardizing the development and runtime environments to eliminate potential discrepancies in learner configurations, and (3) supporting assignments that require customizable hardware and network configurations.

Together, these contributions enable EPLASAM to enhance the delivery of meaningful feedback to learners, while simultaneously decreasing workload of the teaching staff in software-intensive MOOCs.

The remainder of this paper is organized as follows: Section 2 provides more detailed coverage of key challenges and design concerns confronting learners and teaching staff in software-intensive MOOCs; Section 3 describes the solutions applied in EPLASAM to resolve these challenges and design concerns; Section 4 compares our work on EPLASAM with related work on Learning-at-Scale; and Section 5 presents concluding remarks and outlines future work.

2 Key Challenges Facing Learners and Teaching Staff in Software-Intensive MOOCs

This section expands on the discussions in Section 1 related to the challenges of scale, distance, security, heterogeneity, and operational overhead facing learners and teaching staff in software-intensive MOOCs. Figure 1 shows a taxonomy of the design concerns we identified in each challenge area.

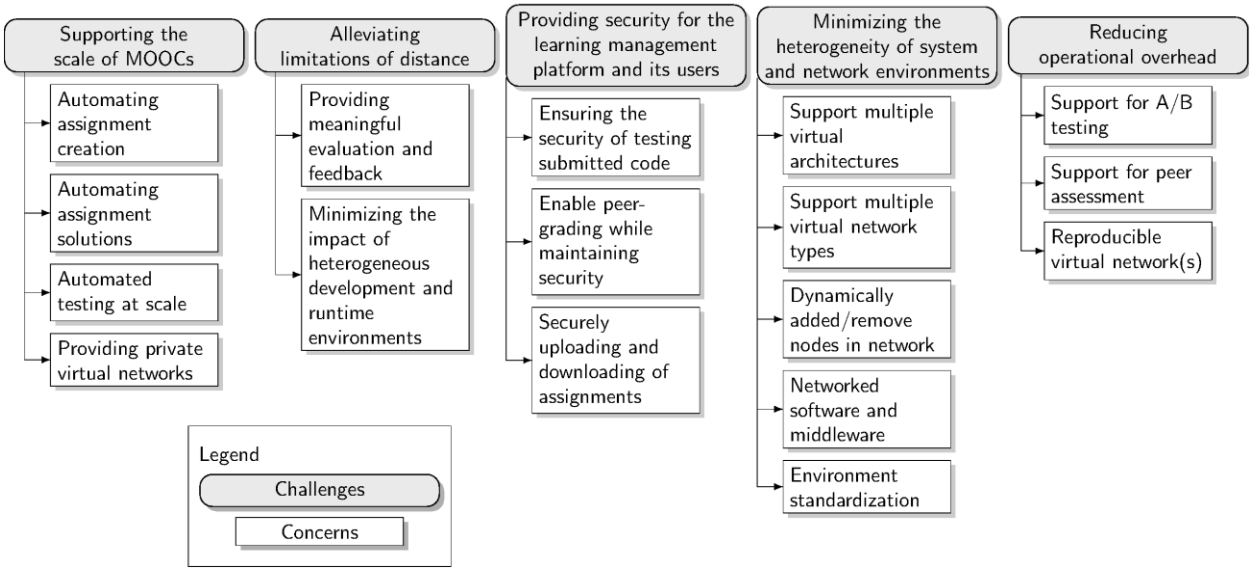

***Figure 1*. Taxonomy of Challenges and Design Concerns for Software-Intensive MOOCs.**

2.1 Supporting the Scale of MOOCs

The number of learners in MOOCs is orders of magnitude greater than even the largest F2F courses. Conventional tools and techniques devised for F2F courses are therefore often not adequate to handle differences in scale. For example, traditional F2F education has a much lower ratio of teaching staff to learners, so it is often feasible (and desirable) for the staff to individually assess each assignment of—and even individually meet with—every learner throughout the course. In contrast, it is infeasible for the teaching staff to individually assess each assignment in a MOOC with tens or hundreds of thousands of learners distributed around the world.

A particularly vexing challenge faced by software-intensive MOOCs involves scalably assessing assignment submissions from learners. Some types of courses lend themselves to questions (such as short answer, multiple choice, or fill-in-the-blank questions) that can be assessed automatically and scalably with high precision. These types of questions are often relevant in certain engineering and mathematics courses, where learners are tested on the proper application of formulas or algorithms via a range of randomized input values for questions asked of each learner. Likewise, short answer problems can be evaluated via regular expression checking for keywords

and multiple choice, matching, and fill-in-the-blank questions whose pass/fail states can be checked automatically.

Conversely, the assessment of software-intensive solutions—especially solutions to *design-oriented* software assignments—is hard to automate via conventional learning platforms available for MOOCs. In particular, since software-intensive assignments do not conform to the assessment categories outlined earlier they face challenges that other types of assignments do not when applied in MOOCs. We therefore identified three goals associated with creating assignments for software assignments: (1) creating assignments in a scalable manner, (2) facilitating meaningful feedback to the learners, and (3) deterring cheating. As shown in Figure 2, it is hard to achieve all of these goals simultaneously.

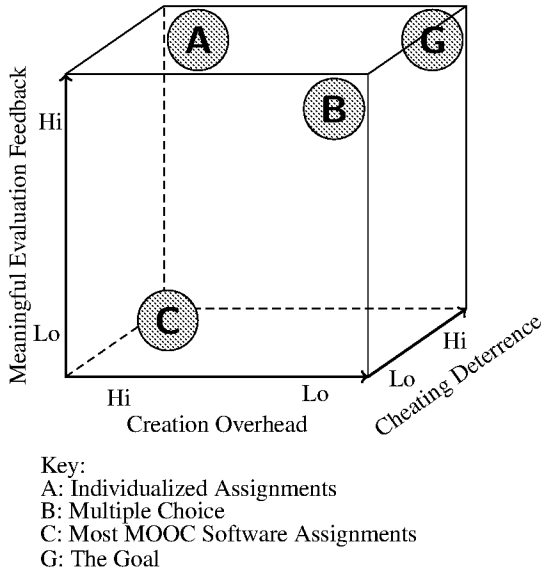

***Figure 2*. Assignment Creation Goals.**

The following are design concerns that must be resolved to address the challenge of scale in software-intensive MOOCs:

- ***Automating assignment creation***. Software-intensive assignments in F2F classes often have similar correct implementations, so it is hard to create assignments that both help deter cheating *and* are reusable for subsequent courses. This challenge becomes even more pronounced in a software-intensive MOOC, where the subject matter has a limited number of correct solutions, such as deadlock-free concurrency or robust distribution. An efficient means of creating assignments that deters cheating among learners is therefore needed for these types of MOOCs. For example, generating different variants of an assignment for each cohort of learners in a MOOC offering helps deter "cut-and-paste" plagiarism from assignments given in previous MOOC offerings. Automating the assignment creation process is essential to free the teaching staff from having to continuously and manually update learner assignments, which is not scalable without devoting substantial human resources.
- ***Automating the generation of assignment solutions***. Creating unique assignment variations for learner cohorts helps deter plagiarism, but without the corresponding assignment *solutions* it is infeasible to automatically assess assignments at scale. Certain considerations

therefore must be taken into account when generating assignments, such as storing the associated solutions to each learner cohort's unique assignments, as well as ensuring that all created assignments have valid solutions that can be assessed automatically.

- ***Automating testing at scale***. Each assignment submitted by learners should be assessed individually, ideally in a private workspace that does not affect other learner submissions. Moreover, assessing the quality of software designs in a MOOC environment is hard since the appropriate choices of component design and implementations often involve subjective criteria, such as "understandability" and "extensibility," which are hard to automate. Software-intensive solutions can also be hard to analyze automatically if learners have improperly implemented portion(s) of a known design or have chosen to apply a poor design.
- ***Providing private virtual networks***. Assignments containing network communication create additional challenges for a scalable learning management platform attempting to host and automatically assess MOOC assignments. Firstly, the platform must support not only a private workspace for each component, but also a virtual network between workspaces. Handling this networking infrastructure at scale requires complex resource management to satisfy quality-of-service (QoS) requirements and prevent degradation of other assignment processing due to resource limitations, such as network link speeds.

Sections 3.1, 3.2, 3.4, and 3.5 describe how EPLASAM addresses the design concerns associated with the scale challenge via the use of assignment randomization and assessment, Linux containers, and a container distribution manager.

2.2 Alleviating Limitations of Distance

The increased distance between teaching staff and learners also increases significantly when transitioning from F2F courses to MOOCs, which introduces challenges that traditional F2F courses often need not address. For example, it is infeasible for the teaching staff to meet personally with all learners in a MOOC. The following are design concerns that must be resolved to address the challenge of distance in software-intensive MOOCs:

- ***Providing meaningful evaluation and feedback*.** The need to create meaningful feedback for learners has always existed, but with F2F courses it has been possible to individualize the feedback, often creating it on-demand when questioned by a learner. This level of personalized feedback is not possible in MOOCs due to the inability to meet and assist every learner. Moreover, it is hard to provide learners with feedback on their work in an informative manner that is neither so overly-constrained that learners lack sufficient information to gain insight from further analysis or so extensive that learners can derive the correct answer(s) without having to conduct deeper analysis on their own, on the other hand. While these issues occur in traditional F2F courses, they are exacerbated in MOOCs due to the increased scale and distance between learners and teaching staff, which motivates the need for meaningful automated evaluation and feedback.

- ***Minimizing the impact of heterogeneous development and runtime environments.*** Software development and design courses, even F2F ones, face challenges with software development tool standardization and compatibility. Traditionally these challenges have been surmountable due to the relative small number of learners in a course. Moreover, each learner receives personalized assistance from teaching staff, who can help install the necessary software and/or help learners debug obscure configuration problems. Software development in MOOCs, however, is complicated due to the heterogeneous learning configuration space and subtle interactions between different variants of operating systems, middleware, software development kits (SDKs), programming language compilers, software library versions, etc.

Sections 3.3 and 3.4 describe how EPLASAM addresses the design concerns associated with the distance challenge via the use of automating and standardizing tooling used for building, testing, and assessing programming assignments.

2.3 Providing Security for the Learning Management Platform and Its Users

Security concerns arise when testing untrusted software logic submitted by learners. For example, submissions should be considered untrusted at all points to prevent malicious or faulty logic from harming the evaluation system or interfering with other learner's. The following are design concerns that must be resolved to address the challenge of security in software-intensive MOOCs:

- **Ensuring security while testing untrusted learner-submitted code**. For programming assignments that require both compilation and execution of untrusted (and possibly malicious) code, steps must be taken to prevent learner-submitted code from interfering with the automated testing system as a whole and/or with other learner submissions.
- **Enable peer-grading while maintaining security.** To support peer-grading as a learning tool, mechanisms are needed to prevent peer assessors (both human and machine) from exposing their own personal system to potential security threats.
- **Securely uploading and downloading of assignments**. Secure authentication is needed to protect the integrity and privacy of learners by encrypting transmission of all assignments to and from the learner, the learning management platform, and any other MOOC servers (such as forums, grade book, etc.). Moreover, the learning management platform should be integrated with the compilation build environment and associated analysis tools to select only the desired files for assessment.

Sections 3.1, 3.2, and 3.3 describes how EPLASAM addresses the design concerns associated with the security challenge via the use of Linux containers, a distribution manager to securely automate container processing, and secure submission of assignments via the compilation tooling.

2.4 Minimizing the Heterogeneity of System and Network Environments

Software-intensive MOOCs that focuses on concurrent and distributed applications require multiple instances of software running in their own networked private environments. The private environments and system architectures (e.g., ARM, x86, AVR/Arduino) may be heterogeneous

and thus not identical to the assessment platform (e.g., x86_64). In this case, the learning management platform must support efficient and scalable execution of emulators.

Even if the architecture is the same for both development and assessment platforms, the need for networked separate environments still exists. Complicating the matter, user-definable network topologies will be required. Custom network designs not only require customizable topologies, but must be repeatable with customizable background network traffic and QoS settings. In addition, the communication infrastructure should support dynamic configurations that can simulate either changing network or physical conditions. These features are needed to develop and test network-based code that can adapt to changing network conditions.

The following are design concerns that must be resolved to address the challenge of heterogeneity in software-intensive MOOCs:

- ***Support multiple virtual architectures***. To enable adaptability in the learning management platform, multiple architectures should be supported. A wide variety of architectures are in use today and with the advent of the Internet of Things (IoT), the need to support an even wider range of heterogeneous architectures will increase.
- ***Support multiple virtual network types***. The ability to configure and adapt to the communication infrastructure is also required. While the processing of the learners' assignments is typically handled in a cloud environment, it may be necessary to simulate network connections of different types, such as 100mbps Ethernet, 10gbps Ethernet, 802.11N wireless, and Near Field Communication (NFC).
- ***Dynamically add/remove nodes in network***. The learning management platform will need to create and remove nodes dynamically to simulate real world events, such as devices moving away from each other, going offline/online, and new resources being added to the network.
- **Networked software and middleware**. Development and analysis of networked software—especially concurrent and distributed middleware—requires scalable environments for experimental evaluation. These environments should be able to isolate a group of private virtual networks together and provide realistic communication between each virtual network based on targeted channel capabilities, such as wireline or wireless channels. A Learning management platform therefore requires the ability to create repeatable virtual networks between unique workspaces that have learner submitted code executing. These virtual networks must also ensure end-to-end QoS requirements (e.g., for latency and bandwidth) and have the ability to handle configurable network traffic and topologies.
- **Environment standardization**. If the development environment (which consists of compilers, operating systems, and associated programming and testing tools) is not defined in a standard manner, various compilation and compatibility issues can arise, which are hard for the teaching staff to assist individuals with at scale. Even minor differences between development environments can yield significant differences in the generated results that increase the amount of effort required by the (often limited) MOOC teaching staff. The learning management platform should therefore standardize these potential incompatibilities to lessen the burden on the teaching staff while the MOOC is live.

Sections 3.1, 3.2, and 3.3 describe how EPLASAM addresses the design concerns associated with the heterogeneity challenge via Linux containers that can run isolated emulators to provide support for additional architectures, a distribution manager to handle all communication-related issues (such as virtual private networks), and Linux containers and compilation tooling providing environment standardization.

2.5 Reducing Operational Overhead

Numerous operational challenges arise when developing a learning management platform for software-intensive MOOCs, including traditional information technology (IT) tasks, such as logging and auditing of system execution and stability, load balancing, and data loss prevention strategies. There are other desired features that conventional learning management platforms either do not support or do not support in the domain of software-intensive MOOCs, including private virtual networks and secure peer review. The following are design concerns that must be resolved to address the challenge of logistics in software-intensive MOOCs:

- ***Support for A/B testing***. Validation of teaching methods, practices, and hypothesis can be supported via a platform that generates assignments for MOOC learners. Two-sample hypothesis testing (often known as A/B Testing) allows statistical analysis of hypothesis with two variants. A similar approach, called split testing, can be applied with more than two variants. To reduce teaching staff overhead during the MOOC, the learning management platform should be able to configure certain criteria for partitioning learners, learner assignments, peer-reviewers, and other possible criteria into either two or more groupings automatically.
- ***Support for peer assessment***. Peer assessment helps reduce the overhead of teaching staff grading at-scale, as well as improving learner abilities to, read, judge, and improve each other's' code. The learning management platform must therefore automatically facilitate various aspects of peer assessment, such as *peer review* (e.g., grading others' code by applying a rubric) and *peer evaluation* (e.g., having peers compile, execute, and evaluate each other's assignments).
- ***Reproducible virtual network configuration(s)***. Certain types of software design and programming assignments (e.g., peer-to-peer file sharing, network discovery protocols) require various types of network topologies, bandwidth, QoS settings, and background traffic. If any of these conditions change the results may vary. The learning management platform must therefore reproducibly create the same virtual network so that individual learner submissions can be tested under the consistent conditions.

Sections 3.1, 3.2, and 3.3 describes how EPLASAM addresses the design concerns associated with the logistics challenge via the use of Linux containers, a distribution manager, and compilation tooling to provide an automated system with low operational overhead.

3. The Structure and Functionality of EPLASAM

EPLASAM consists of several interchangeable hierarchies and parts, each providing a specific portion of its overall capabilities. Linux containers provide individualized, secure, and potentially customized runtime environments for code compilation, testing, and evaluation. EPLASAM handles

the distribution of Linux containers to different hosts. It also uses interchangeable command-line build-tools to run each step of the compilation, testing, and evaluation. Finally, it provides generators that automatically produce assignments that are customized for learner cohorts. The remainder of this section describes each of these capabilities and outlines how they address the challenges presented in Section 2.

3.1 Linux Containers

The technology for general purpose OS-Level Virtualization for Linux has existed as open-source software since 2005 with OpenVZ. OS-Level virtualization is a technique where virtual private instances of the same OS as the server are provided to virtualized application(s) running on the host server. This approach offers many advantages over other virtualization technologies, such as hypervisors and full VM emulations. In particular, it enables Virtual Private Server (VPS) instances, each of which runs its own copy of an OS and enables users to install almost any software that runs on that OS.

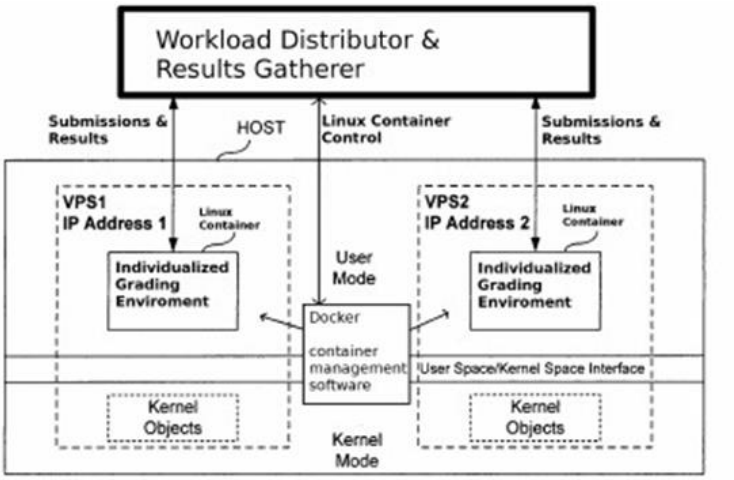

***Figure 3*. Diagram of Docker Container Deployment and Management.**

A VPS differs from a hypervisor or fully emulated virtual machine technologies in several ways that make them attractive for use in EPLASAM. Figure 3 shows how EPLASAM employs OS-level virtualization to provide a VPS to each learner application submission. We manage each VPS via a central workload distribution manager and Docker, which automates the deployment of applications by providing an additional layer of abstraction and automation of OS-level virtualization on Linux (Docker, 2014).

Two key ways in which a VPS differs from the other virtualization options are the startup-time and resource overhead of each instance. For example, the time for a VPS instance to start is significantly shorter than for an entire OS to boot, as is the case for hypervisor and full virtual machine emulation technologies. In particular, the base OS in a VPS has already been instantiated, so only the portions required for the VPS to instantiate must be started. Moreover, the resource overhead of VPS is significantly lower than a full virtual machine emulation instance.

VPS does have several drawbacks, however. For example, the host machine's kernel must support OS-Level virtualization at the OS kernel level. Likewise, the OS type on the host machine must be the same OS type on the virtualized instance.

Until recently, distribution of OS-level virtualization, also called “software containers,” were not easily automated and therefore required a high degree of skill and knowledge to configure properly. The broad adoption of Docker, however, enables automated deployment of application(s) in these containers in networked configurations, thereby simplifying ease of use and reducing deployment time.

A key improvement to Linux container technology provided by Docker is the creation and use of a central server hosting images (which is called a Registry) and the build scripts required to create those images called DockerFiles. We apply Docker in a partially closed-source environment, i.e., where answers to assignments are kept secret, which leverages its ability to host a private Registry.

The central workload distribution manager mentioned above stores the deployment strategies EPLASAM uses to create the rules for container deployment and virtual network configuration. Existing tools, such as Shipyard (Shipyard, 2014), handle the low-level migration, monitoring, and management of Docker servers and Docker images. Shipyard can easily be controlled via a public API, thereby enabling the use of EPLASAM’s custom Domain-Specifc Modeling Language (DSL) tools provided by the Distribution Manager to manage physical servers, container deployment and execution, handling of assignment files, and setting up of virtual networks.

The abilities that Docker containers give the EPLASAM platform help address concerns from sections 2.1, 2.3, 2.4, and 2.5. Specifically, it gives the ability to provide private networks of virtual machines at scale from section 2.1. The VPS design provides separation of each student's virtual environment from section 2.3. Containers help to standardize student submission environments, supports multiple virtual networks and architectures via emulators from section 2.4. They also provide support for reproducible virtual networks by providing easily reproducible environments from section 2.5.

3.2 Distribution Manager

The design of EPLASAM requires a means to automate the distribution of containers to the required hosts, as well as the distribution of submitted assignments to these servers for processing. Figure 3 shows how Docker is used along with a Workload Distributor to distribute the processing of assignments. Currently EPLASAM can use multiple server host machines and multiple Docker VPS per host, though the distribution of assignment workloads must be determined manually. Automating this distribution is an area we are looking at for future work.

We observed two general approaches in Docker management: (1) either have the deployment management system itself manage container deployment strategies or (2) allow users to manually deploy containers where they choose. In our initial experiments we opted for manually deploying containers and have used Shipyard since it offers the most feature from amongst the myriad of software solutions comprising the Docker software ecosystem. In particular, Shipyard has an HTTP/REST-based API—unlike many alternatives, such as Kubernetes (Kubernetes, 2015)—that allows finer control of which server(s) specific container(s) are executed. This capability helps EPLASAM configure the virtual private networks used to connect containers.

Shipyard works via an installed controller on each host machine that is configured with Docker. It requires configuring a single central Shipyard instance to manage all the hosts that Shipyard is

responsible for. It then connects to a private repository of DockerFiles describing each container image, after which point management of all VPSs and containers can begin.

The management of the physical servers, container deployment and execution, handling of assignment files, and setting up of virtual networks are handled by EPLASAM's custom Domain Specific Language (DSL) and tool-chain. This custom DSL interacts with different levels of the platform, multiple custom applications, and other DSLs. For clarity, we call the top-level DSL that interacts with the other layers/DSLs within the platform the "Platform-DSL" and call the DSLs for assignments the "Assignment-DSLs." The Platform-DSL is designed to allow hierarchical composition of multiple assignment-DSL model instances, thereby allowing EPLASAM to handle simultaneous assessment of multiple different assignments and adjust to handle complex interactions between different assignments. For example, one adjustment involves distributing network-bound and a CPU-bound processes across the available hardware to minimize interference. The Assignment-DSL likewise needs to support multiple container definitions, expected container runtime requirements and profiles, assignment variations, and virtual network requirements. EPLASAM is designed via a multi-layered approach, where each component has well-defined interaction with the overall platform, to facilitate adoption since it is easier to add new/modify features or components.

Existing distribution managers, such as Shipyard, provide the EPLASAM platform the ability to address concerns from sections 2.1, 2.3, 2.4, and 2.5. A distribution manager provides the ability to automate assignment grading at scale from section 2.1. It also facilitates the secure transfer of assignment submissions between different servers in the process of grading from section 2.3. A Distribution Manager addresses the concerns of dynamically adding/removing network nodes and supporting multiple virtual network types from section 2.4. In addition, it also addresses the concern of reproducible virtual networks from section 2.5.

3.3 Compilation Tooling

EPLASAM does not require any specific project build tool or integrated development environment (IDE). Since Linux containers are only executable via the command-line, the use of an automated build tool is an integral part of the overall framework we have developed. The grading of programmatic questions is decomposed into two phases: compilation and testing. Both phases may have sub-phases that will depend upon the problem and code being evaluated, though EPLASAM just focuses on these two phases for now (the specifics of sub-phases will be addressed by both the choice of build-tools chosen and the desired evaluation techniques chosen by the staff of a course).

Although EPLASAM is build-tool agnostic, support for each tool/tool-chain must be integrated into the platform. This requirement enables the proper handling of both the specific starting commands and reading of results from the compilation and testing framework execution. We adopted Gradle as the default build tool for EPLASAM. Other build automation tools, such as Maven, Ant, and Make, were considered, but Gradle was selected since it is the build system utilized in the material our MOOCs are covering. The specific build system chosen will depend upon the programming language being tested, its conventions, and the personal choice of the MOOC staff.

Currently, EPLASAM contains manual script implementations to launch and evaluate compilation results, unit testing, and integration testing results. Future work will expand these abilities to operate via a plugin system that enables the use of other tool/tool-chains, such as Gradle, Make, and Ant. Future work will also provide an automated and secure method for assignment transfer via the chosen tool/tool-chains to/from the learner and EPLASAM.

Automated build tools help to address the concerns from sections 2.2, 2.3, 2.4, and 2.5. Automating build tools helps to provide meaningful evaluation and feedback, even at a distance, and reducing the impact heterogeneous development and runtime environments. Which address concerns from section 2.2. Automated build tools also help to address the concern of providing security for testing submitted code from section 2.3. They also help to address the concern of environment standardization from section 2.4. The use of automated build tools also helps to reduce the overall operational overhead concern from section 2.5.

3.4 Programming Assignment Assessment

Assessing programming assignments yields additional challenges not faced by commonly asked question types, such as multiple choice or true/false questions. In particular, how can the teaching staff craft a validation method for an assignment that is both effective and provides learners with meaningful feedback. If a suite of unit tests for an assignment returns a number of passed tests, but has no feedback as to what or even why a test was not passed, then it will fail the 'Alleviating Limitations of distance' challenge shown in Figure 2.

To address this issue, EPLASAM employs advanced built tools, such as Gradle, to automate the entire process of compiling, unit testing, and integration testing. Gradle currently supports C, C++, and Java and is on the road map to become the official build script for Android. We therefore chose it as our standard automation script. Since EPLASAM is designed in a modular manner, however, if Gradle does not support the programming language the teaching staff intends to use for a course or assignment it is relatively straightforward to integrate the desired command-line build system in its place. This modularity adds flexibility to EPLASAM to support languages and features beyond the scope of a single build system. It also provides the ability to use additional testing frameworks, such as Junit or SureLogic, that best fits a specific assignment type.

The use of custom code-analysis tools in addition to testing-frameworks is supported with both Gradle and Maven, but we also plan to support individualized command-line executions to support for custom evaluation and grading techniques. This customizability will further enhance EPLASAM's flexibility by allowing the application of more advanced analysis tools and techniques.

In our experience teaching MOOCs, we found it helpful to withhold some tests from learners, thereby providing a multi-tiered grading process. The teaching staff is responsible for selecting which tests, if any, to keep from learners. This capability enables the teaching staff to create their own grading criteria and methodologies or to create a series of questions/assignments that reveal more of the tests to learners over time to help refine and improve their understanding of the material.

The design of EPLASAM helps to address the concerns from section 2.1 and 2.2. Assignment creation techniques help to address the concerns of assignment and solution automation from section 2.1. Automated assignment and solution creation, helping address the concern of providing meaningful evaluation feedback from section 2.2.

3.5 Assignment Randomization

Assignment creation techniques aimed at preventing cheating and facilitating the potential reuse of assignments in a MOOC environment face challenges that traditional educational situations do not. In particular, the geographic distance between the teaching staff and learners makes it harder to detect certain types of cheating. Some problem types (such as short answer, multiple choice, and fill-in-the-blank) have exactly one correct answer or answer combination. These problems can be pre-created and a random subset can be given to each learner to help address the needs of cheating prevention and facilitate question reuse. Essays or short answer-based questions can use statistical analysis methods to detect (and thereby prevent) cheating. The 'correct' answers to these types of problems need not be determined exactly. Instead, keywords/order input can be entered into a regular expression parser to check validity/uniqueness.

Software-based assignments are often have structurally similar solutions, depending on the assignment specification. Thus, if a single correct solution leaks to the learner community, the question cannot be reliably reused in future offerings due to the likelihood that learners will find this solution online and apply it without having to do the work themselves. Therefore, to address the issues that software-based questions have with respect to similarity of assignment solutions between learner cohorts—and to help prevent reuse of solutions between offerings—EPLASAM applies a model-based approach to individualized assignment generation.

Our initial work for this approach has yielded a generator that takes a model as input and outputs the full source code and all files required to compile a data-centric Create, Read, Update, & Delete (CRUD) based Android application. This generator is given the desired model file as input and uses custom template groups to define the output files. It is possible to turn individual templates from within a group on or off, giving the teaching staff the ability to choose what code they wish to generate. This feature allows the creation of variants of assignments, which can be used for either subsequent assignments or for creating different versions for A/B Testing.

The input model for the custom code generator is an XML file describing the data-centric view of the information to store in the desired application, along with some meta-data. Figure 4 shows the application specific section of a sample model. This model describes the application, its underlying database table, and the fields to store in the table. All components of the model have a 'name' value used for variable name and string construction for use in different locations within the application. These locations range from the underlying database to the user interface. Field values have a 'type' that describes the type of data to store. The default value of a field is configurable via the 'default_val' property. The base data types supported currently are those of the Java primitive types, with the addition of String and byte[], which are referred to as STRING and BLOB, respectively. Future work will expand these options to include more complex data types, such as images, audio, and video.

```
<application name="example app">
  <content-provider
          name="provider example">
    <table name="table 1">
      <field type="TEXT"
            name="field 1"
            default_val="" />
      <field type="LONG"
            name="field 2"
            default_val="-1" />
      <field type="CHAR"
            name="field 3" />
      <field type="BLOB"
            name="field 4" />
      <field type="SHORT"
            name="field 5" />
      <field type="INTEGER"
            name="field 6" />
      <field type="FLOAT"
            name="field 7" />
      <field type="DOUBLE"
            name="field 8" />
      <field type="BOOL"
            name="field 9" />
    </table>
  </content-provider>
</application>
```

***Figure 4*. Sample Application Definition Model.**

EPLASAM's focus on Android for the initial code to generate with our custom DSL code-generator tool stems from the component-based architecture of the Android platform, which allows us to remove portions of code from the list of templates to generate. This feature, in turn, helps narrow the code that is given learners at any one time to test them on the material being covered. It also allows the use of the same code to teach multiple concepts, thereby allowing the teaching staff to create exercises that reinforce each other.

Assignment randomization helps to address the concerns of automated assignment & assessment creation from section 2.1. In particular it helps to allow the re-use of created assignments in subsequent offerings of the same course.

4 Related Work

This section compares EPLASAM with related work that focuses on three areas: (1) the use of Linux containers for virtualization, (2) generation of assignments, and (3) automated generation of grading and meaningful feedback for programming assignments. These areas are relevant since developing a platform for automated grading of software-intensive assignments in a MOOC environment requires effective resource management at the cloud infrastructure-level, as well as the ability to scale both assignment creation and evaluation with the addition of (potentially many) new learners.

4.1 Related Work on Linux Containers

The Docker (*Docker Inc. official site*) open-source project we utilize in EPLASAM automates the deployment of applications via software containers utilizing operating system (OS)-level virtualization. Docker is not an OS-level virtualization solution; rather it uses interchangeable execution environments such as Linux Containers (LXC) and its own *libcontainer* library to provide container access and control.

Previous work exists on the creation (Menage, 2007) and benchmark testing (Soltesz, Pötzl, Fiuczynski, Bavier, & Peterson, 2007) of generic Linux-based containers, shows that Linux containers are a more lightweight means of virtualization compared to traditional hypervisors. Similarly, related work uses containers as a means to provide isolation and a lightweight replacement to hypervisors in specific use cases. For example, High Performance Computing (HPC), reproducible network experiments, and peer-to-peer testing environments focus on different aspects of Linux containers that they leverage. Xavier et al. (Xavier et al., 2013) show that in the domain of HPC that containers provide an excellent lightweight hypervisor replacement, but at the time the solutions available were immature and unable to provide effective isolation. Handigol et al. (Handigol, Heller, Jeyakumar, Lantz, & McKeown, 2012) discuss the creation of reproducible network experiments via container-based emulation, which is intended to spur other researchers to publish runnable versions of their experiments. They leverage the ability to package, distribute, and run containers. Bardac et al. (Bardac, Deaconescu, & Florea, 2010) show how peer-to-peer testing environments can be developed using Linux containers. The focus on EPLASAM in this paper is on the scalability and limitations of networked applications in a container-based testing environment.

Both the papers from Bardac et al. and Handigol et al. use Linux containers to provide reproducible network-based application testing. However, the focus in each paper differs slightly. In particular, Bardac et al. are concerned with repeatability, whereas Handigol et al. are concerned with the scalability of network-based applications, particularly peer-to-peer applications. In contrast to these efforts, our work on EPLASAM uses all three of these aspects, i.e., lightweight, isolation, and ease of distribution. Moreover, EPLASAM leverages some of Docker's features to create custom containers and share them in a manner that's more straightforward than possible in prior work.

4.2 Related Work on Generation

This section compares EPLASAM with related work in the areas of generating assignments in a learning-at-scale environment and generating automated feedback to programming assignments.

4.2.1 Assignment Generation

With the advent of MOOCs, the interest in automatically generating assignments at scale has grown. Automating Exercise Generation (Sadigh, Seshia, & Gupta, 2012) describes using a template-based generator together with mutation and satisfiability solving to automatically generate assignment problems, valid solutions, and grading of submitted solutions. The problems they target are a subset of problems from an embedded systems class textbook, which they found had problems that did not fit well into their platform.

In addition to automatic assignment generation, related work has focused on creating Domain Specific Languages (DSL) for code generation of mobile and cloud-based applications. PhoneApps (Mannadiar & Vangheluwe, 2010) is a custom DSL built to help simplify the creation of stand-alone mobile applications using Statecharts with modular components at different levels of abstraction. Manjunatha et al. design an approach for creating MobiCloud (Manjunatha, Ranabahu, Sheth, & Thirunarayan, 2010), which is a DSL for creating Cloud Mobile Hybrid applications via an agnostic Model-View-Controller (MVC) code generation pattern, treating both the cloud and mobile portions of an application as single entity. MobiCloud supports multiple target mobile device and cloud hosting platforms but requires a custom code generator that accepts the custom ModiCloud-DSL model as input, for each target platform. The MVC pattern is also used by Buck et al. the Objektgraph IDE (Buck, Diethelm, & Sheneman, 2013), which uses a graphical UML editor to generate Java applications, and anticipates Android and Google Web Toolkit support.

EPLASAM is similar to MobiCloud and Objektgraph in that it uses the MVC pattern with templates to generate our targets. In addition, however, EPLASAM can build a standard DSL interface that simplifies swapping of future generator components. This interface provides flexibility to the overall platform and simplifies adoption allowing the use of custom code generation tools.

4.2.2 Meaningful Feedback Generation

In addition to automated generation of assignments and mobile applications, related work has focused on generating meaningful feedback to MOOC learners on their programming assignments. Singh et al. describe a technique for automated feedback generation of introductory programming assignments (Singh, Gulwani, & Solar-Lezama, 2013). Their technique requires a correct reference implementation and a set of rules in a custom error model language (EML) that define errors (and corresponding corrections) that learners are likely to produce. These EML rules are used to define a space of candidate programs that will be generated from the learner submission and then searched for the minimal number of corrections required to transform the submission to match the reference program.

Carbunescu et al. present their results and experiences from using a framework they designed for auto-grading of parallel code for the 2013 and 2014 XSEDE Parallel Computing Course (Carbunescu et al., 2014). Their platform used a C program to verify the output results and a Python script to manage file handling, job processing, and final grade calculations. Grades are based upon runtime performance for a variety of input sizes for each of three assignments in the course to evaluate both strong and weak potential scalability. They present the concessions and modifications they made when designing each assignment's implementation, such as how verification of non-deterministic simulations can be hard to validate.

This related work shows the extent to which trial and error can be applied to create tests with meaningful feedback. These results led us to design EPLASAM so that it is agnostic with respect to specific tests and testing methodologies it supports. The goal is to provide an environment for flexibility creating and evaluating new techniques and approaches.

5 Concluding Remarks

This paper presented the motivations, challenges, and solutions related to EPLASAM, which is a platform for software-intensive MOOC assignment creation and grading. We discussed the

EPLASAM design goals and the related challenges associated with creating meaningful software programming assignments that deter cheating. We show how the overall platform is split into two major portions: (1) the server portion, which acts as the infrastructure, and (2) the tools portion, which facilitates teaching staff creating new assignments that are automatically generated and assessed at scale. We then explain how EPLASAM accomplishes its design goals and explain why various technical choices were made. Finally, we compared EPLASAM with related work to show how our approach builds upon and extends previous work to leverage our experiences teaching software-intensive MOOCs.

Three main categories of future work stem from our work on EPLASAM:

- **Enhanced platform analysis and development**. We are currently integrating EPLASAM into our Mobile Cloud Computing with Android (MoCCA) Specialization. Measurements and analysis of its performance and scalability will be conducted to evaluate its effectiveness. We anticipate that additional software capabilities will be needed to create the DSL(s) and related tools required to manage the platform logistics of container management and deployment, particularly in a heterogeneous hosting scenario where the platform utilizes multiple cloud services, such as Amazon EC2, Microsoft Azure, DigitalOcean, RackSpace, etc.
- **Applying the platform in Learning-at-Scale environments and analyzing its impact on learners**. Using EPLASAM our MoCCA Specialization will provide an opportunity for research into the science of education on software-intensive design and programming topics. For example, the ability to assess A/B group learners at the scale of tens-to-hundreds of thousands of learners will enable us to refine our current teaching and testing techniques and explore new ones. With tighter coupling between the testing framework and other course material, it will be possible to test things such as which order of introducing topics leads to better overall results. It will also be possible to explore variations in both the number and difficulty of programming assignments and determining which combinations provide the best overall result. Future work will be possible on the analysis of learner learning results based on the quality and quantity of meaningful feedback they receive from the build system, e.g., using the assignment submissions to adapt and enhance auto-grading tools. Improvements in accuracy, quality feedback, runtime efficiency, and breadth of testing coverage are metrics by which differing approaches can be evaluated to identify effective methods.
- **Using the platform for research on advanced middleware and networked applications**. Future work in the area of middleware development and analysis will be enabled by EPLASAM. The platform, with its dynamic virtual networking and repeatable experiments, will assist in addressing the challenges Chaqfeh et al. concerning Internet-of-Things enabled middleware (Chaqfeh & Mohamed, 2012). The ability to simulate both variable networking conditions and resource constraints allows better development and analysis into specialized network-based middleware. Two important industries that are ever increasing are video games (Correa, 2013), which has been larger than Hollywood for more than a decade (Yi, 2004), and consumer mobile devices. Both these domains have specialized requirements that EPLASAM can assist with in terms of middleware development and analysis, such as strict timing constraints or minimum levels of robustness that EPLASAM can simulate. In addition, both video games and mobile devices are popular topics that can serve as a basis for future MOOCs that teach additional core computer science principles, methods, and tools.

References

Bardac, M., Deaconescu, R., & Florea, A. M. (2010). Scaling peer-to-peer testing using linux containers. In Roedunet international conference (roedunet), 2010 9th (pp. 287–292). Retrieved from http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5541555

Buck, D., Diethelm, I., & Sheneman, S. (2013). Objektgraph: why code when mvc applications can be generated with uml-based diagrams? In Proceedings of the 2013 companion publication for conference on systems, programming, & applications: software for humanity (pp. 25–26). Retrieved from http://dl.acm.org/citation.cfm?id=2514576

Carbunescu, R., Devarakonda, A., Demmel, J., Gordon, S., Alameda, J., & Mehringer, S. (2014). Architecting an autograder for parallel code. In Proceedings of the 2014 annual conference on extreme science and engineering discovery environment (pp. 68:1–68:8). New York, NY, USA: ACM. Retrieved from http://doi.acm.org/10.1145/2616498.2616571 doi: 10.1145/2616498.2616571

Chaqfeh, M. A., & Mohamed, N. (2012). Challenges in middleware solutions for the internet of things. In Collaboration technologies and systems (cts), 2012 international conference on (pp. 21–26). Retrieved from http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6261022

Correa, C. (2013, April). Why video games are more addictive and bigger than movies will ever be. http://www.forbes.com/sites/christophercorrea/2013/04/11/why-video-games-are-addictive-and-bigger-than-movies-will-ever-be/. (Accessed: 2014-10-24)

Coursera - Free Online Courses From Top Universities. (n.d.). Retrieved February 23, 2015, from https://www.coursera.org/specialization/mobilecloudcomputing2/36

Docker inc. official site. (n.d.). http://docker.com. (Accessed: 2014-10-24)

Handigol, N., Heller, B., Jeyakumar, V., Lantz, B., & McKeown, N. (2012). Reproducible network experiments using container-based emulation. In Proceedings of the 8th international conference on emerging networking experiments and technologies (pp. 253–264). Retrieved from http://dl.acm.org/citation.cfm?id=2413206

How It Works. (2013, November 13). Retrieved February 15, 2015, from https://www.edx.org/how-it-works

Kubernetes by Google. (n.d.). Retrieved January 25, 2015, from http://kubernetes.io/ . (Accessed: 2014-10-24)

Manjunatha, A., Ranabahu, A., Sheth, A., & Thirunarayan, K. (2010). Power of clouds in your pocket: An efficient approach for cloud mobile hybrid application development. In Cloud computing technology and science (cloud-com), 2010 ieee second international conference on (pp. 496–503). Retrieved from http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5708492

Mannadiar, R., & Vangheluwe, H. (2010). Modular synthe-sis of mobile device applications from domain-specific mod-els. In Proceedings of the 7th international workshop on model-based methodologies for pervasive and embedded software (pp.21–28). Retrieved from http://dl.acm.org/citation.cfm?id=1865879

Menage, P. B. (2007). Adding generic process containers to thelinux kernel. In Proceedings of the linux symposium (Vol. 2,pp. 45–57). Retrieved from https://www.kernel.org/doc/ols/2007/ols2007v2-pages-45-58.pdf

Sadigh, D., Seshia, S. A., & Gupta, M. (2012). Automating exer-cise generation: A step towards meeting the mooc challenge forembedded systems. In Proceedings of the workshop on embedded and cyber-physical systems education (p. 2). Retrieved from http://dl.acm.org/citation.cfm?id=2530546

Shipyard. (2014). http://shipyard-project.com/. (Accessed: 2014-10-24)

Singh, R., Gulwani, S., & Solar-Lezama, A. (2013). Automatedfeedback generation for introductory programming assignments.In Proceedings of the 34th acm sigplan conference on program-ming language design and implementation (pp. 15–26). NewYork, NY, USA: ACM. Retrieved from http://doi.acm.org/10.1145/2491956.2462195 doi:10.1145/2491956.2462195

Soltesz, S., Pötzl, H., Fiuczynski, M. E., Bavier, A., & Peterson, L.(2007). Container-based operating system virtualization: a scal-able, high-performance alternative to hypervisors. In Acm sigops operating systems review (Vol. 41, pp. 275–287). Retrieved from http://dl.acm.org/citation.cfm?id=1273025

Xavier, M. G., Neves, M. V., Rossi, F. D., Ferreto, T. C., Lange, T.,& De Rose, C. A. (2013). Performance evaluation of container-based virtualization for high performance computing environments. In Parallel, distributed and network-based processing(pdp), 2013 21st euromicro international conference on (pp.233–240). Retrieved from http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6498558 (Accessed: 2014-10-24)

Yi, M. (2004, December). They got game / stacks of new releases for hungry video game enthusiasts mean it's boom time for an industry now even bigger than hollywood. http://www.sfgate.com/news/article/THEY-GOT-GAME-Stacks-of-new-releases-for-hungry-2663371.php. (Accessed: 2014-10-24)