Scaling Software Experiments to the Thousands

Christian Neuhaus[1], Frank Feinbube[1], Andreas Polze[1] and Arkady Retik[2]
[1]*Hasso Plattner Institute, University of Potsdam, Prof-Dr.Helmert-Str. 2-3, Potsdam, Germany*
[2]*Microsoft Corporation, One Microsoft Way, Redmond, U.S.A.*

Keywords: Massive Open Online Courses, Online Experimentation Platform, Software Engineering, Operating Systems, InstantLab, Windows.

Abstract: InstantLab is our online experimentation platform that is used for hosting exercises and experiments for operating systems and software engineering courses at HPI. In this paper, we discuss challenges and solutions for scaling InstantLab to provide experiment infrastructure for thousands of users in MOOC scenarios. We present InstantLabs XCloud architecture - a combination of a privat cloud resources at HPI combined with public cloud infrastructures via "cloudbursting". This way, we can provide specialized experiments using VM co-location and heterogeneous compute devices (such as GPGPU) that are not possible on public cloud infrastructures. Additionally, we discuss challenges and solutions dealing with embedding of special hardware, providing experiment feedback and managing access control. We propose trust-based access control as a way to handle resource management in MOOC settings.

1 INTRODUCTION

Knowledge acquisition has never had it better: While our ancestors had to buy very expensive books, we can look up virtually any information on the internet. Universities follow that trend by making their teaching materials available online: Lectures can be streamed and downloaded from platforms like *iTunes U*. Websites like *Coursera* and *Udacity* provide complete courses with reading materials, video lectures and quizzes - sometimes also run directly by universities (e.g. *Stanford Online* or *openHPI*). However, educational resources should not be limited to passively consumed material: 10 years of experience in teaching operating systems courses to undergraduate students at HPI have shown us that actual hands-on experience in computer science is irreplaceable.

InstantLab at HPI. InstantLab is a web platform that is used in our undergraduate curriculum to provide operating systems experiments for student exercises at minimum setup and administration overhead. InstantLab uses virtualization technology to address the problem of ever-changing system configurations: experiments in InstantLab are provided in pre-packaged containers. These containers can be deployed to a cloud infrastructure and contain virtual machine images and setup instructions to provide the exact execution environment required by the experiment. InstantLab's core component is a web application, through which users can instantiate and conduct experiments. The running instances of these experiments can be accessed and controlled through a terminal connection, which is set up from within the user's web browser (see figures 1, 2).

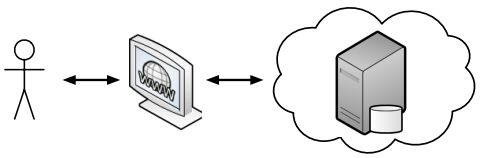

Figure 1: Browser-based access to experiments.

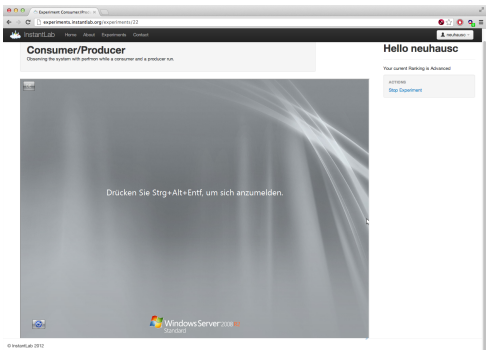

Figure 2: InstantLab: Live Experiments in the Browser.

Neuhaus C., Feinbube F., Polze A. and Retik A..
Scaling Software Experiments to the Thousands.
DOI: 10.5220/0004965505940601
In *Proceedings of the 6th International Conference on Computer Supported Education* (CSEDU-2014), pages 594-601
ISBN: 978-989-758-020-8

Over several terms of teaching operating systems courses, InstantLab has proven to be a helpful tool: it allows us to offer various software experiments with very little setup work, ranging from elaborate kernel-debugging exercises over demonstrations of historic operating systems (e.g. Minix, VMS) to more general software engineering exercises.

2 SCALING TO THE THOUSANDS

While InstantLab is used in teaching in our undergraduate classes with great success, we aim to bring this level of hands-on experience and practical exercises to massive open online-courses (MOOCs). This is where it is needed the most: Students at regular universities usually have access to university-provided hardware resources. Students of online courses however come from diverse backgrounds and are scattered all over the world – the only equipment that can be assumed to be available to the participants is a web browser. In this section, we review state of the art of MOOCs and software experiments and identify the challenges that lie ahead.

2.1 State of the Art

Recently, massive open online courses have received plenty of attention as a new way of providing education and teaching. This is reflected by a several web platforms supporting MOOCs (e.g. Coursera, Udacity, edX, Khan Academy, Stanford Online, openHPI, iversity). On the software side, new software frameworks (e.g. *lernanta*[1], *Apereo OAE*[2]) and Software-as-a-Service platforms (e.g. Instructure Canvas) offer the technical foundation for MOOCs.

In learning theory, the need and usefulness for learning from practical experience has been recognized (Grünewald et al., 2013; Kolb et al., 1984). Therefore, most MOOCs incorporate student assignments complementing the teaching material, where results are handed in and are used for feedback. Most of these assignments are quizzes, which can be easily evaluated by comparing the student's input to a list of correct answers. More complex assignments (e.g. calculations, programming exercises) are usually conducted by students handing in results (e.g. calculation results, program code). A solution for automated generation of feedback for programming exercises was proposed by (Singh et al., 2013). However, conducting programming assignments and software experiments on students' hardware creates several challenges: The heterogeneity of students' hardware and software makes it difficult to ensure consistent, reproducible experiment conditions. This makes software problems hard to troubleshoot (Christian Willems, 2013). Additionally, licensing conditions and pricing of software products can limit the availability and prevent use for open online courses. Therefore, practical assignments in MOOCs are currently mostly limited to non-interactive tasks that do not offer students students hands-on experience with practical software experiments.

2.2 Challenges and Chances

In this paper we present an architecture for flexible and interactive software experiments for massive open online courses. We build on our experiences with InstantLab at HPI using pre-packaged software experiments that are executed in virtual machines and present an architecture that leverages public cloud infrastructure resources to cope with the high user load. However, offering live software experiments at MOOC-typical scale is fundamentally different from a classroom scenario: On one hand, the massive scale poses new challenges that have to be met. On the other hand, the large number of participants offers new opportunities of to improve education and the platform itself. In this paper, we address the following issues:

GPU and Accelerator Hardware. Embedding special hardware resources into cloud-hosted experiments is a difficult task: We show how new virtualization technology can assign physical hardware to different users and employ a compilation-simulation-execution pipeline to use scarce physical hardware resources only for tested and correct user programs (see section 3.3).

Automated Feedback & Grading. Learning from experience with practical experiments is only possible when feedback on a students performance is provided. To gather the required information, we propose methods for experiment monitoring by using virtual machine introspection to monitor students performance during assignments (see sections 4,5).

Automated Resource Assignment. Access to expensive experiment resources should only be granted to advanced and earnest participants of a course. Traditional access control mechanisms fall short as they require manual privilege assignment. Instead, we propose a trust-based access

[1]https://github.com/p2pu/lernanta/

[2]http://www.oaeproject.org/

control scheme to automatically govern access to experiment resources (see section 6).

3 XCloud ARCHITECTURE

This section provides an overview of our XCloud architecture that allows us to set up, manage and maintain complex experiment testbeds of heterogeneous hardware on cloud infrastructure (see figure 3). Conceptionally, it consists of three distinctive parts: The *Middleware*, the *Execution Services* and the *Data Storage*.

The *Middleware* hosts a web application that allows to start experiments and access virtual desktop screens. Users can connect to it using a web browser or arbitrary network protocols (e.g. SSH). The *Middleware* manages user identities and mediates their access to the actual experiments hosted by the *Execution Services*. It also controls the deployment of experiments on appropriate hardware resources.

The *Execution Services* manage hardware resources for execution of experiments and provide an infrastructure to monitor the user activity by a variety of means ranging from simple log mechanisms to more elaborate techniques such as virtual machine introspection.

The *Data Storage* service is our reliable store for user data, experiment results and the activity log that holds all the information about the users activities. These logs are the basis for our trust-based access control as described in section 6. Furthermore, the *Data Storage* holds virtual machine disk images encapsulating our experiments.

3.1 XCloud by Example

Within our InstantLab curriculum, we have two special teaching objectives that require the extension of the standard IaaS concepts and that are consequently incorporated into our XIaaS model: *network-related experiments* and *accelerator experiments*. For network experiments, we extend the IaaS concept with an explicit notion of a (virtualized) network interconnect within the cloud. For accelerator experiments, we distinguish between different types of processing units (like x64 or Itanium) and dedicated accelerator boards (like Nvidia Tesla or Intel Xeon Phi).

As a proof-of-concept, we currently run our experiments on our XIaaS implementation that is based on HPs Converged Cloud and hosted within the FutureSOC-Lab at the Hasso Plattner Institute in Germany.

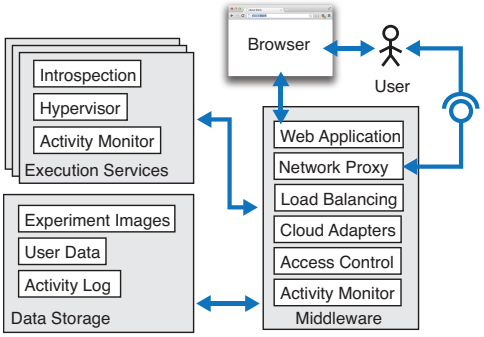

Figure 3: XCloud architecture.

As they are conventional XIaaS services, our components could be accessed separately by clients. When working with the experiment scenarios, though, our students experience the XIaaS infrastructure as a big multicomputer with compute nodes, accelerator nodes, and an interconnection network.

As an example, consider experiments that allow students to monitor the boot phase of an operating system or debug its kernel. In these experiments it is crucial that the student can connect remotely to the systems that are studied from another virtual machine providing its with appropriate monitoring tools. With XCloud, we treat the two virtual machines as *XIaaS components* and create the required connection (e.g. serial cable) by linking their *hardware interfaces*. *XIaaS components* are discussed in the next section.

To support these kinds of scenarios while enforcing our security policies, the communication endpoints that the clients receive are grouped and governed be our trust-based access control scheme (see section 6). For the convenience of our students, these services are not restricted to the web service protocol only. Students can also use standard remote display connections (RDP) to connect to their virtual machines or directly connect to the XCloud interconnect using protocols such as the Message Passing Interface (MPI).

Another class of experiments that heavily rely on XCloud's interconnect features, are parallelization tasks where students need to create and run parallel applications that utilize heterogeneous compute resources using a master-worker-style execution pattern over multiple virtual machines.

3.2 XIaaS Components

With XCloud, we introduce *XIaaS components* (see Figure 4) which extend the IaaS abstraction to support our teaching objectives. A *XIaaS component* has the following characteristics:

Network Interface. XIaaS components can not only

be accessed using standard network services provided by their operating systems, but can also be interconnected to form a desired network topology.

Hardware Interface. The hardware interfaces provided by XIaaS components allow users to access arbitrary ports of the virtual hardware, like screen connection, Ethernet, USB, debugger connections, etc. using tunneling protocols. Furthermore, they can be redirected to hardware interfaces of other XIaaS components, allowing for a variety of new usage opportunities.

Physical Devices. To support modern accelerator cards or other specialized hardware setups, InstantLab provides generic deployment mechanisms that use hardware descriptions to configure the hardware according to the needs of the usage scenario.

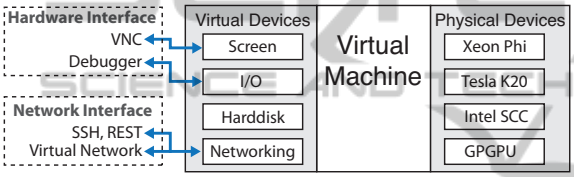

Figure 4: XIaaS Component.

3.3 Accelerator Hardware Virtualization

The current trends in hardware developments lead to an increasing amount of variation, both in hardware characteristics and hardware configurations. In the future, general purpose processors will be supported by a variety of special accelerators, which will require diligent distribution of computation tasks. The first type of accelerators that gained a great popularity are GPU Compute Devices. Current versions support a wide variety of applications and programming styles and the literature is full of success stories of algorithms that gain drastic performance benefits from GPU acceleration. Intel, joining this trend, introduced a new type of accelerators with its Intel's Xeon Phi accelerator boards which are special in the sense of being fully x86-compatible. Because of the benefits of accelerators for performance-hungry workloads, several cloud-hosters such as Amazon, Nimbix and peer1hosting recently added accelerator-based cloud products to their portfolio. These solutions are mostly IaaS products providing exclusive direct access to physical GPUs, thus providing high performance, and fidelity, while disallowing multiplexing, and interposition (Dowty and Sugerman, 2009).

Direct access allows applications to use all the capabilities of the GPUs and exploit their full performance. This is beneficial for us, since representative performance and high fidelity is especially important in our scenario, since we want our students to learn about performance characteristics and performance optimization techniques of accelerators ((Kirk and mei Hwu, 2010),(Feinbube et al., 2010)). The overhead introduced by more sophisticated virtualization mechanisms could have a negative effect on execution speed and the range of supported features of the virtualized device.

The lack of mediation between virtual machines and physical hardware makes it impossible to support features that are usually inherent to virtualization technologies: VM migration, VM hibernation, fault-tolerant execution, etc. The required interposition could be realized by implementing means to stop a GPU at a given point in time, read its state, and to transfer it to another GPU that will then seamlessly continue with the computation. For our scenario this is not as important, though, because student experiments do not require a highly reliable execution: they can simply be repeated if they fail.

Another important feature that is usually expected from virtualization is the idea of multiple VMs sharing physical hardware. For us, the amount of multiplexing that is required depends primarily on the number of accelerators that we can provide and the number students that we want to allow to work with our MOOC exercise system concurrently.

In an ideal world, we could provide every student with an direct, exclusive access to one accelerator, allowing them the best programming experience. The drawback would be that we would then need to either restrict the number of students or dynamically increasing the number of accelerators by relying on other cloud offerings resulting in additional costs.

The other extreme would be a situation where we only have a very small number of accelerators for our exercises. This might be the case because the accelerator type is novel and not provided by any cloud hoster. To allow for realistic execution performance and high fidelity in such a scenario, the only option would be to use a job queue for the students' task submissions and execute one task after another on each accelerator. Since the user experience is rather limited in these situations, we described some techniques to improve both, the user experience and the performance of MOOCs where many students are working with single unique physical experiments in (Tröger et al., 2008). One possibility is to separate the compilation of the students experiment code from the actual execution on the experiment. The compilation can be

performed on generic VMs without dedicated accelerator hardware. This enables quick feedback during development without actually using the accelerator hardware. For running the compiled code, virtualization techniques should allow multiple users to work simultaneously with software experiments on the same physical hardware without interfering with each other. To support this scenario GPUs need multiplexing capabilities. While GPU vendors are currently working with vendors of virtualization solutions like VMware (Dowty and Sugerman, 2009) and Xen (Dalton et al., 2009) on architectures for performant GPU virtualization, some researchers found ways to allow for this even today: Ravi et al. implemented a GPU virtualization framework for cloud environments (Ravi et al., 2011) by leveraging the ability of modern GPUs to separate execution contexts and the ability to run concurrent kernels exclusively on dedicated GPU streaming processors. (NVIDIA Corporation, 2012)

4 USER ACTIVITY MONITORING

The key difference between MOOCs and traditional schools or universities is the proportion between the number of students and teaching staff. Thousands of users regularly sign up for open online courses which are offered and organized by only a small team. This proportion means that personal support work for individual participants to assign experiment resources and provide tutoring and feedback has to be kept at a minimum. To address this problem, we propose automatic feedback and grading mechanisms (see section 5) and automatic resource assignment (see section 6) based on observed behavior of students on the platform. Both mechanisms require information about how users interact with the platform. In this section, we propose activity monitoring on the web platform and observation of active software experiments.

The web platform itself provides valuable information of student activity during a course. The **web platform** can record student activity such as logins to the platform, access of video lectures and starting, finishing or abortion of experiments. This information is stored in the *activity log* (see figure 3) and evaluated to assess the students' continuity in following the contents of the course curriculum.

In addition, we extend the activity monitoring to **active software experiments**, which are running inside of virtual machines in our approach. This enables us to gather more detailed information required for feedback and grading which is not only based on the outcome of assignments but can consider how the given task was solved. A monitoring solution for active experiments should be susceptible to user manipulation (tamper-resistant), not alter the execution of the experiment (transparent) and have only a small performance impact (efficient). These criteria are not met by in-VM monitoring modules. Instead, we propose *virtual machine introspection*-based monitoring (Pfoh et al., 2009; Garfinkel et al., 2003). By implementing monitoring on the hypervisor level, it is protected against manipulation by the user and does not require changing the software images of experiments to install monitoring modules. Introspection on the hypervisor level is facing the challenge of bridging the *semantic gap* (Chen and Noble, 2001): Context on data structures is not available from the operating system but has to be inferred from context. However, since we know the exact software versions in our experiment images we can provide the necessary context information to interpret observations. While VM introspection can cover virtually all activity in experiments, certain events are of particular interest: The use of privileged operations (system calls, hardware access, I/O) can be efficiently monitored as they trap into the hypervisor – this gives a good overview of activity on the system. Additionally, the memory can be monitored to detect loaded software modules or check the integrity of code sections. Further information can be gathered by monitoring the activity on the network link.

Software support for VM introspection is available for different platforms and hypervisors. The open source library *LibVMI*[3] is compatible with *KVM* and *Xen* hypervisors while *VProbes*[4] is offered for VMWare solutions.

5 AUTOMATED FEEDBACK AND GRADING

To fully benefit from learning at scale, both the feedback and the grading procedures must be supported by automated tools. The quality of these tools must increase with both present students and those from former courses. Since most people do not have outstanding autodidactic abilities, for MOOC-based courses getting a large number of participants involved is crucial. A good learning environment can only be provided if the discussion forums are populated with active people. Their collaborations in exer-

[3] https://code.google.com/p/vmitools/

[4] http://www.vmware.com/products/beta/ws/vprobes_reference.pdf

Table 1: Means and metrics for student achievements as a basis for feedback and grading.

Scope	Means	Metrics
Students (System Interaction Tasks)		
1.1 Knowledge	Quizzes; clozes	correct / incorrect answers
1.2 Actions	VM Introspection, scripts	correct / incorrect VM states; violations
Student Programs (Programming Tasks)		
2.1 Source code	keyword scans; pattern detection; AST analysis;	correct / incorrect keywords and libraries
2.2 Actions	Integration tests; run-time monitoring; sensor data;	correct / incorrect results and states See 1.2 (student actions) as well.

cises and discussions of the learning material not only help the students to get a better understanding of the course contents, but also document the problems that students found in lectures and exercises and thus are good source for ideas on how to improve and extend the course for the future.

One of the most sensitive topics, though, is the grading of learning results. For the purpose of our lectures, students are graded by their knowledge and actions, as well as, the source code and behavior of their respective programs. Table 1 gives an overview of the means that can be used to acquire information about their qualification and the according metrics. We provide four levels of information that can be combined to evaluate and grade our students.

The knowledge that the student acquired in the course can be automatically tested and graded using quizzes and clozes. They get instant feedback about their understanding of the course material and it is very straightforward to use this information to generate grades automatically.

Since our courses heavily rely on hands-on experiences and students directly manipulating system states, documenting those states is crucial. We use the VM Introspection mechanisms to check whether or not a student achieved a desired system state (see section "Experiment monitoring"). Furthermore special scripts provided by the course teaching team can be introduced and run inside the VM for additional state checks. Many user interactions are already measured for our trust-based access control technology (see section 6). The level of trustworthiness and the number of deviations from expected behavior can also be taken into account when it comes to direct feedback and grading of a students performance.

Programming exercises offer additional information that can be used for student assessments: the source code and the run-time behavior of the students' programs. Source code can not only be checked for correct compilation, but analyzed in more sophisticated ways. Often it is required, that students use predefined keywords or library functions to achieve the assigned task. Other keywords or library calls may be forbidden. Simple white- and black-lists and text scans can be used for these purposes. Based on such a checker, patterns could be investigated, e.g. if after an "open()" call there is also a "close()" call. Doing this in a more elaborate way, though, requires to work on the abstract syntax tree (AST) representation of the program code. Besides simple pattern detection ASTs allow to compare solutions with one another. Student programs can be clustered and compared with a variety of standard solutions. The actions that a running student program executes can be checked in the same fashion as the students actions himself. Furthermore integration tests can be applied to see if the programs behavior and results match the expectations. Language runtimes like the Java Virtual Machine also allow to use sampling, profiling, reflection and other runtime monitoring mechanisms to evaluate the behavior of student programs in detail. If physical experiments are involved sensor data provided by these setups can also be used. All the information that can be acquired at those four levels can be used to provided immediate feedback and automated grading.

6 AUTOMATIC RESOURCE ASSIGNMENT

Offering live VM-based software experiments to a large audience is a resource-intensive workload. Due to budget constraints of teaching institutions, the amount of these resources is limited. While cloud platforms can scale up to handle high-load situations, this also increases cost. In addition, specialized hardware (e.g. accelerator cards, GPGPUs) often cannot be used by different users simultaneously (i.e. no overcommitment) and are only present in limited numbers. Therefore, a resource assignment scheme is required that can automatically assign experiment resources to earnest participants of the course and detect and prevent misuse of the provided resources.

To achieve a self-managing solution, we propose a *trust based access control (TBAC)* scheme, which assigns resources to students based on their previous behaviour on the platform. TBAC is a dynamic access control scheme based on the notions of *reputation* and *trust* (Josang et al., 2007): Reputation is information that describes an actor's standing in a system. Based on such information, trust levels can express a subjective level of confidence that this actor will behave in a desired way in the future. These trust levels can be represented as values on numerical scales $x \in [0;1]$, where low values indicate low confidence and 1 indicates certainty. These levels can be calculated based on personal experience with actors and also include reputation information gathered from other actors (federated reputation system) or a centralized authority (centralized reputation system). Based on these concepts, Trust-based Access Control (TBAC) represents a dynamic extension to traditional models of access control (mandatory, discretionary, role-based) ((Boursas, 2009)(Chakraborty and Ray, 2006)(Dimmock et al., 2004)) to incorporate trust between actors into the access policy definitions. Policies can then specify access criteria based on a certain level of trust. Access is granted, if the requesting actor has a sufficient level of trust assigned and denied otherwise. This concept has been proposed for use in pervasive computing ((Almenárez et al., 2005)), P2P-networks (Tran et al., 2005) and is used to govern user permissions on the popular website *Stackoverflow*.

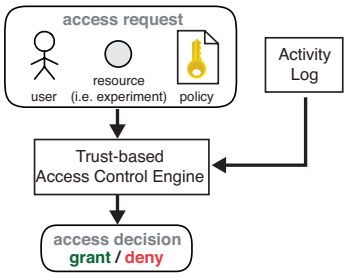

Figure 5: Trust-based Access Control System.

The advantage of using TBAC for resource assignment is self-management of the system: trust levels are automatically derived from a user's behavior on the platform, by using the information gathered through monitoring of interactions with the platform and experiments, which is stored in the **activity log** (see figure 5). Access policies for resource-intensive software experiments can then be specified using these trust-levels. This makes policies specific to experiments, but not to users – eliminating the need for manual creation of user-specific access policies.

The properties of the curriculum of online courses

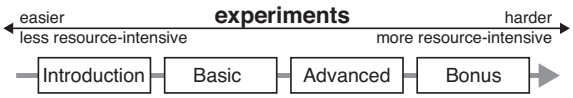

Figure 6: Course curriculum: Difficulty Stages.

can be used in designing such TBAC systems and policies. We assume that the curriculum starts out with introductory lessons, where students familiarize themselves with basic material, where software experiments are not yet required (see figure 6). In subsequent lessons, the content will become more demanding and require practical experiments, which should only be available to students who successfully completed the previous lessons. We make use of these properties by employing a step-wise approach in building trust levels. To get access to the software experiments in a new curriculum stage, students have to build according trust levels by completing assignments in the previous stage. As trust levels start out at 0 and do not fall below this level, the mechanism cannot be circumvented by starting with a fresh account.

7 SUMMARY

Massive Open Online Courses enjoy great popularity and are used by millions of users worldwide. However, the possibilities for practical assignments and hands-on exercise are mostly limited to simple quizzes. In this paper, we introduce *InstantLab* – a platform for running software experiments on cloud infrastructure and making them accessible through a browser-based remote desktop connection. To truly scale this approach to thousands of users, several challenges have to be met: We address the problem of embedding specialized accelerator hardware into virtualized experiments. As experiment resources are limited, they should be protected from abuse and assigned to users with an earnest learning intent. We employ an automatic resource assignment based on trust levels which are derived from users activities on the platform. As practical experiments requires detailed feedback on the students action to be effective: We draw information from runtime virtual machine introspection to provide detailed feedback about assignments during and after experiments.

REFERENCES

Almenárez, F., Marín, A., Campo, C., and García R, C. (2005). Trustac: Trust-based access control for pervasive devices. *Security in Pervasive Computing*, pages 225–238.

Boursas, L. (2009). *Trust-based access control in federated environments*. PhD thesis, PhD thesis, Technische Universität in München.

Chakraborty, S. and Ray, I. (2006). Trustbac: integrating trust relationships into the rbac model for access control in open systems. In *Proceedings of the eleventh ACM symposium on Access control models and technologies*, pages 49–58. ACM.

Chen, P. M. and Noble, B. D. (2001). When virtual is better than real [operating system relocation to virtual machines]. In *Hot Topics in Operating Systems, 2001. Proceedings of the Eighth Workshop on*, pages 133–138. IEEE.

Christian Willems, Johannes Jasper, C. M. (2013). Introducing hands-on experience to a massive open online course on openhpi. In *Proceedings of IEEE International Conference on Teaching, Assessment and Learning for Engineering (TALE 2013)*, Kuta, Bali, Indonesia. IEEE Press.

Dalton, C. I., Plaquin, D., Weidner, W., Kuhlmann, D., Balacheff, B., and Brown, R. (2009). Trusted virtual platforms: a key enabler for converged client devices. *SIGOPS Oper. Syst. Rev.*, 43(1):36–43.

Dimmock, N., Belokosztolszki, A., Eyers, D., Bacon, J., and Moody, K. (2004). Using trust and risk in role-based access control policies. In *Proceedings of the ninth ACM symposium on Access control models and technologies*, pages 156–162.

Dowty, M. and Sugerman, J. (2009). Gpu virtualization on vmware's hosted i/o architecture. *SIGOPS Oper. Syst. Rev.*, 43(3):73–82.

Feinbube, F., Rabe, B., Löwis, M., and Polze, A. (2010). NQueens on CUDA: Optimization Issues. In *2010 Ninth International Symposium on Parallel and Distributed Computing*, pages 63–70, Washington, DC, USA. IEEE Computer Society.

Garfinkel, T., Rosenblum, M., et al. (2003). A virtual machine introspection based architecture for intrusion detection. In *NDSS*.

Grünewald, F., Meinel, C., Totschnig, M., and Willems, C. (2013). Designing moocs for the support of multiple learning styles. In *Scaling up Learning for Sustained Impact*, pages 371–382. Springer.

Josang, A., Ismail, R., and Boyd, C. (2007). A survey of trust and reputation systems for online service provision. *Decision Support Systems*, 43(2):618–644.

Kirk, D. and mei Hwu, W. (2010). *Programming Massively Parallel Processors: A Hands-on Approach*. Morgan Kaufmann, 1 edition.

Kolb, D. A. et al. (1984). *Experiential learning: Experience as the source of learning and development*, volume 1. Prentice-Hall Englewood Cliffs, NJ.

NVIDIA Corporation (2012). NVIDIA's Next Generation CUDA Compute Architecture Kepler GK110.

Pfoh, J., Schneider, C., and Eckert, C. (2009). A formal model for virtual machine introspection. In *Proceedings of the 1st ACM workshop on Virtual machine security*, pages 1–10. ACM.

Ravi, V. T., Becchi, M., Agrawal, G., and Chakradhar, S. (2011). Supporting gpu sharing in cloud environments with a transparent runtime consolidation framework. In *Proceedings of the 20th international symposium on High performance distributed computing*, HPDC '11, pages 217–228, New York, NY, USA. ACM.

Singh, R., Gulwani, S., and Solar-Lezama, A. (2013). Automated feedback generation for introductory programming assignments. In *Proceedings of the 34th ACM SIGPLAN conference on Programming language design and implementation*, PLDI '13, pages 15–26, New York, NY, USA. ACM.

Tran, H., Hitchens, M., Varadharajan, V., and Watters, P. (2005). A trust based access control framework for p2p file-sharing systems. In *System Sciences, 2005. HICSS'05. Proceedings of the 38th Annual Hawaii International Conference on*, pages 302c–302c. IEEE.

Tröger, P., Rasche, A., Feinbube, F., and Wierschke, R. (2008). SOA Meets Robots - A Service-Based Software Infrastructure for Remote Laboratories. *International Journal of Online Engineering (iJOE)*, 4.