Electronic Preprint for Journal of Information Processing Vol.26

Regular Paper

Development of a Web-based Front-end Environment to Aid Programming Lectures on Unix-like Systems

Syunji Yazaki[1,a)] Hideaki Tsuchiya[1,b)] Hiroaki Ishihata[2,c)]

Received: August 16, 2017, Accepted: February 1, 2018

Abstract: In this paper, we describe the details of the design and implementation of our Front-end Environment for Hands-on Activities (FEHA), which is a web-based programming environment. FEHA provides a programming environment on the web and utilizes existing Unix-like systems that equip a specialized programming environment as the build and runtime platform. FEHA controls the existing systems by using Secure SHell (SSH) and Rsync without any modification of the existing systems. We discuss a case study of FEHA in which it was applied to actual programming lectures at a university. In the lectures, 70% of the students completed registrations to use FEHA in about 3 min. In addition, they could understand how to use the FEHA and started submitting codes within several minutes after the registration. The case study shows that FEHA is able to provide a specialized programming environment for more than 100 students with a small amount of effort from the instructor and system administrator.

Keywords: programming education, web-based programming

1. Introduction

Education for special-purpose computing technologies is needed in recent computing industry, and the number of people becoming technicians is increasing [1]. Parallel processing, General-Purpose computing on Graphics Processing Units (GPGPU), and hardware design are examples of these specialized technologies.

Instructors providing programming lectures need to catch up with the state-of-the-art technologies that meet both industrial requirements and academic values. Web-based learning systems have been utilized to improve learning efficiency [2], [3], [4], [5]. Learners access learning content prepared by instructors by using their own web browser. Some systems provide not only a programming environment but also a comprehensive learning environment [6], [7]. Such systems are called Learning Management Systems (LMSs). Nowadays, LMSs are used to provide worldwide educational services called Massive Open Online Courses (MOOCs) [2], [8], [9]. MOOCs play an important role in providing a learning opportunity for everyone.

There are some difficulties in constructing a programming environment for teaching specialized technologies on LMSs. Ordinary software and hardware products are usually provided as out-of-box packages. However, most software and hardware for specialized technologies are not provided as such packages. The system administrator needs to build, install, and set them up with concise options referring to a desired programming environment. It requires significant knowledge and technology-related experience. In some cases, the instructor and system administrator might need additional development on the LMSs to associate such specialized software and hardware with the LMSs.

With this background, we proposed a pluggable web-based programming environment named Front-end Environment for Hands-on Activities (FEHA) [10]. FEHA works as a front-end of existing Unix-like computing systems that equip a specialized programming environment. FEHA does not make any modification to the existing systems. Instead, it utilizes two common tools, Secure SHell (SSH) and Rsync, to control the existing systems. These tools are installed on most Unix-like systems by default. FEHA also provides a minimum coding environment that includes an editor and buttons to specify some abstracted build and runtime options. FEHA helps instructors to provide lectures using the state-of-the-art technologies without any complex settings.

In this paper, we will provide a more detailed design concept and an extended implementation of FEHA than the previous paper [10]. In addition, we provide a case study in which FEHA was applied to actual lectures at a university.

The rest of this paper is organized as follows. We will introduce the related works and typical programming environment for computer science in Sections 2 and 3. On this basis, we will explain the purpose of system development in Section 4. Then, we will discuss about the system design and technological challenges for the implementation of FEHA in Sections 5 and 6. Next, we will discuss a case study of applying FEHA to actual lectures at a university in Section 7. Finally, we state the conclusions in Section 8.

[1] The University of Electro-Communications, Chofu, Tokyo 182–8585, Japan
[2] Tokyo University of Technology, Hachioji, Tokyo 192–0982, Japan
[a)] yazaki.syunji@uec.ac.jp
[b)] hideaki@cc.uec.ac.jp
[c)] ishihata@stf.teu.ac.jp

2. Related Works

2.1 Web-based SSH Client

There have been several web-based SSH clients such as WebSSH2 [11] and KeyBox [12]. These web applications provide an interactive Command Line Interface (CLI) on web browsers by using WebSocket technology. Users can access Unix-like systems that have an SSH server and then use programming environments prepared on the system through the web-based CLI without any installation of designated SSH client software.

2.2 Tutoring System

The use of tutoring systems is a better way for novice learners to learn a particular programming language [2]. Tutoring systems provide step-by-step tutorials that prevent learners from making arbitrary codes. By following the tutorial, learners can easily acquire fundamental programming skills. However, the learner needs to move on to other programming platforms to learn more complex and practical techniques.

2.3 Assistant Application

Glassman et al. proposed OverCode, which collects solutions submitted by students and then provides similar solutions to instructors. The instructors can obtain sophisticated solutions from the collection and use them for better teaching [13].

Another system records the learning activities [14]. This system records the events during a programming assignment, and these records are used to find learners who require a longer time to solve problems.

Wang et al. developed AutoLEP [15], which provides automatic syntactic and structural checking to help novice learners. Wei et al. proposed a technique to classify programs for hint generation [16]. In this technique, the programs are represented as a linkage graph to summarize the data flow in the programs. The graph helps teachers to understand the fundamental approach in the program created by the student. Paolo et al. developed a hint system for programming lectures [17]. The hint system displays a series of hints according to the requests from a student.

2.4 Web-based Programming Environment

Some systems that support programming lectures have been developed. Blackboard [7] and Moodle [6] are typical LMSs for MOOCs. Moodle is an open-source project that can be customized and extended by communities all over the world.

Further, plug-ins for Moodle are provided by many Moodle users. Virtual Programming Lab (VPL) [3] is a plug-in that supports programming exercises. VPL has a feature called "Jail," which provides a sandbox environment to run programs created by the learners safely. VPL also provides a web-based integrated development environment.

WebGPU [4] has been used in a MOOC for GPGPU programming lectures. WebGPU is designed to provide Graphics Processing Unit (GPU) resources to learners, and it supports general GPGPU-related technologies such as the Compute Unified Device Architecture (CUDA), Open Computing Language (OpenCL), and Open ACCelerator (OpenACC). It does not support other technologies since this system focuses on GPGPU.

A computer cluster system designed for teaching parallel and distributed computing has been developed [5]. This system provides a web interface to manage source code and compile it. The built code can be submitted and run on the cluster.

3. Programming Environment on Unix-like Systems for Science and Technology

3.1 Computational Resource and Management

Software and hardware resources based on specialized technologies are limited. Therefore, the instructor must consider a way to share the limited resources among the learners for programming lectures.

Some particular types of resources such as GPU cores, Central Processing Unit (CPU) cores, memory bandwidth, network bandwidth, or reconfigurable logic gates are provided by dedicated devices. A GPU board, a Many Integrated Cores (MIC) board, High-Bandwidth Memory (HBM), high-performance interconnects, and a Field Programmable Gate Array (FPGA) board are typical examples.

These kinds of resources can be accessed from laboratories, computer centers, or supercomputing systems via resource management systems. A batch job system is a resource management system that is commonly used. SLURM, Torque, and OpenPBS are typical implementations of a batch job system.

A batch job system mainly provides two features to share computing resources among multiple concurrent users: job scheduling and queuing. The users submit jobs to a queue, which is associated with machines that provide particular computational resources. The jobs also include requests for the amount of computational resources. Once the jobs are received, they are kept in the queues. Then the system schedules the allocation of jobs on the basis of the requested resources and the remaining computational resources on the corresponding machines. If the requested amount of computational resources can be assigned to a job, the job will be allocated to run on the corresponding machines under the restriction of the allocated computational resources.

To use a batch job system, the user needs to describe a job as a script. The batch job script is a general Shell script with the annotations of a request for the queue and the amount of computational resources. Thus, the user can invoke any program in the batch job script. The annotation does not affect the behavior of the program; therefore, the batch job script can be run as a simple Shell script if the system allows it.

Batch job systems have been the de-facto standard for the management of computational resources. Therefore, the instructors need to support the use of a batch job system for lectures. Writing batch job scripts will not be a concern for professional software developers because they already have the skills to write them. However, the use of a batch job system can become a barrier for programming lectures because for the learners, it is difficult to write job scripts with a reasonable request for resources.

3.2 Development Tools and Library Management

Software development and the runtime environment depend on many kinds of development tools and libraries. In many cases, the

version of the software matters as well. The dependencies sometimes have complex structures. For instance, the GNU Compiler Collection (GCC), which provides commonly used programming environments packaged on CentOS Linux release 7.3, recursively depends on 55 software packages. Further, each dependent software package has its own dependencies.

Most of the development tools and libraries using specialized technologies are aggressively developed and updated. For practical software development, developers need to select the proper combination of development tools and libraries for the targeted platforms on which the developed software runs.

Software called "Environment modules" [18] or its alternative implementations [19] have been used to manage the combination of tools and libraries. This software has the capability to switch between different combinations of software without logoff/login or reboot operations. This feature can be utilized to prepare the environment for programming lectures with various types of specialized technologies.

4. Purpose of System Development

Typically, for delivering a programming lecture, the instructor follows an actual software development process. **Figure 1** illustrates a use case of the software development process. The system administrator prepares and maintains a development platform consisting of the required hardware and software. The engineer sets up personal environments for code, build, and execute programs. The engineer and system administrator can sometimes be the same person.

In a programming lecture, the learner practices programming by following a procedure similar to that of actual software development, as shown in **Fig. 2**. However, in contrast with actual development, the learner should work on setting up the personal environment with the instructor.

We observed two factors that possibly degrade the efficiency of teaching and learning. The first is that learners need to take care of their personal settings. The instructor also needs to prepare an account for each learner. If there are several hundreds of learners in a lecture, the instructor requires a long time for preparing accounts for all learners. It is also important that the instructor and learner keep the same personal settings to avoid unexpected troubles during programming lectures. However, adjusting the personal environment for a lecture requires a large amount of time since the personal environment varies depending on each user's activities.

The second is that learners also need to build source code and run the program with concise options for compilation and execution. It is better to experience all of the procedures of software development during the lecture as much as possible. However, building and running programs based on complex technologies requires complex designations through the options, which are tightly associated with the hardware and software configurations. For instance, some of the following options are required to build and run a parallel program using the Message Passing Interface (MPI), which produces interprocess communications: a particular compiler, a level of code optimization, a network architecture, an algorithm of communication, and the affinity of the CPUs. In addition, most development environments on Unix-like systems are assumed to be used through a CLI because they are remotely located to be shared and accessed via SSH. Acquiring skills to develop software with a CLI is important for engineering education. However, using a CLI for a programming lecture sometimes becomes a barrier for novice learners. Because learners need to understand how to access and use development environments by a CLI in advance of lectures. Thus, more simple interfaces to build and run programs is needed especially for novice programming learners.

We designed FEHA to resolve these problems. **Figure 3** (a) shows a use case of programming practice with FEHA, which we aim to achieve. FEHA basically takes care of two things: specifying concise options for building and executing programs and unifying the personal environment and providing the unified environment to all learners. With this support, the learner can focus on coding and checking the result in contrast to the use case shown in Fig. 2. In addition, the instructor can control the personal environments of all learners through FEHA. In compensation, additional use cases regarding the setup and maintenance of FEHA are needed.

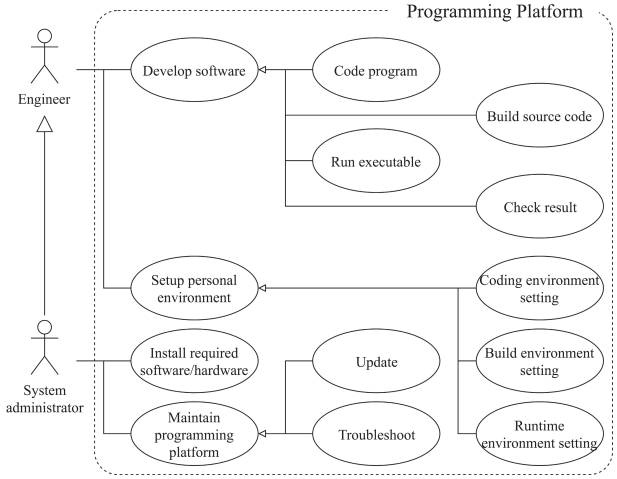


Fig. 1 Use case of software development by an engineer. The engineer takes care of the personal environment themselves.

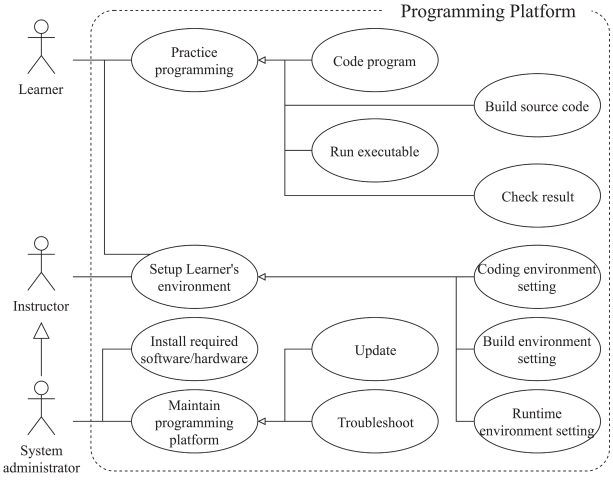
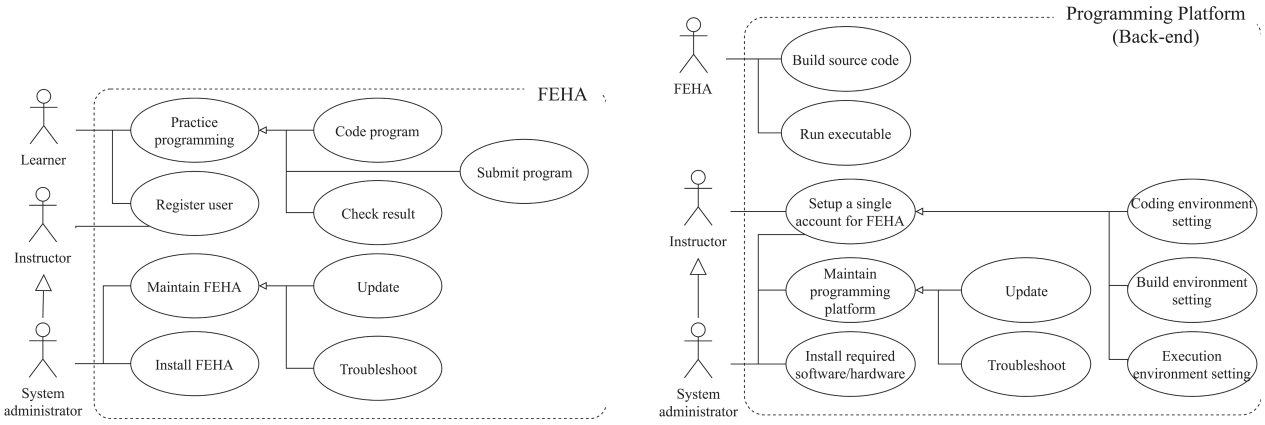


Fig. 2 Use case of programming practice. The learner follows the same flow of software development. The instructor takes care of the learner's personal environment.



(a) FEHA (b) Back-end

Fig. 3 Use case of programming practice with FEHA. (a) FEHA provides features to code and submit programs without complicated build and runtime options. This feature allows the learner to focus on coding activity to experience various technologies easily. The learner also can register themselves to FEHA. This self-registration feature helps the instructor to prepare accounts for many learners. The instructor and administrator need to set up and maintain FEHA. (b) The system administrator creates a single account for FEHA on the back-end. FEHA builds and runs the programs submitted by all FEHA users with this account.

On the back-end, as shown in Fig. 3 (b), the instructor and system administrator need to prepare a single account that builds and runs the programs submitted by all FEHA users.

5. System Design

5.1 Design Principle

On the basis of the discussion in Section 4, we implemented FEHA with the following policies:

- Utilize an existing programming environment on Unix-like systems
- Do not require any additional software installation on the existing systems
- Utilize the mechanism of computational resource management and environment switching on existing systems
- Use the minimum set of tools that are commonly installed on most Unix-like systems to control them from FEHA
- Use a local user account on each existing system as a reference of the personal environments distributed to all learners

These policies provide originality to FEHA compared to other web-based programming environments. That is, FEHA can work as a pluggable environment for programming lectures that can work on most existing Unix-like systems. In addition, FEHA does not require additional administration for existing Unix-like systems, except for the creation of one local user. These characteristics allow instructors to arrange a new lecture to learn a state-of-the-art technology quickly. Further, the instructor just needs to put FEHA onto a Unix-like system that has a programming environment related to the new technology to prepare the lecture.

In addition, we implemented following features to support programming lectures:

- A self-registration feature
- A feature providing code samples and templates that present outlines of learning programming code to the learners
- A feature to specify complex options to build and run by a summarized user interface

The self-registration feature allows users to register by themselves. We assume that FEHA will be used in a lecture that includes several hundreds of students. In this situation, making FEHA accounts for all users by the instructor or system administrator requires a large amount of time. Code samples and templates are provided to support lectures. The difference between the sample codes and the code templates is that the samples are codes that are ready to run without modification, whereas the templates are pieces of code that must be modified. A learner can run the samples for experiencing the use of FEHA and program execution without coding as a quick start. Then, the instructor can utilize the templates to teach coding techniques. A summarized user interface equips some forms and buttons to allow learners to specify complex options for building and running by a simplified interface.

5.2 Configuration

Figure 4 shows an overview of the FEHA configuration. A host that contains an instance of FEHA is represented as the FEHA host, while existing Unix-like systems that will be controlled by FEHA are described as the back-end. Existing Unix-like systems can be simple machines or computer clusters that have one or more login nodes. We assumed that the FEHA host is separated from the back-end; however, the back-end can be a FEHA host if the system administrator allows it.

The solid and dashed arrows represent data and control flows, respectively. As we described in the design policy, FEHA controls the back-end by using a minimum set of tools. We choose the “ssh” and “rsync” commands as the tools. “ssh” provides the SSH session to invoke any allowed Unix command on the FEHA host and back-end. “rsync” incrementally synchronizes files between the FEHA host and back-end. The file synchronization runs over the SSH session created by “ssh.”

FEHA consists of a web User Interface (UI), controller, and database. The web UI provides a programming environment to the learner and also shows the summarized execution results of the submitted programs. The controller controls all processes

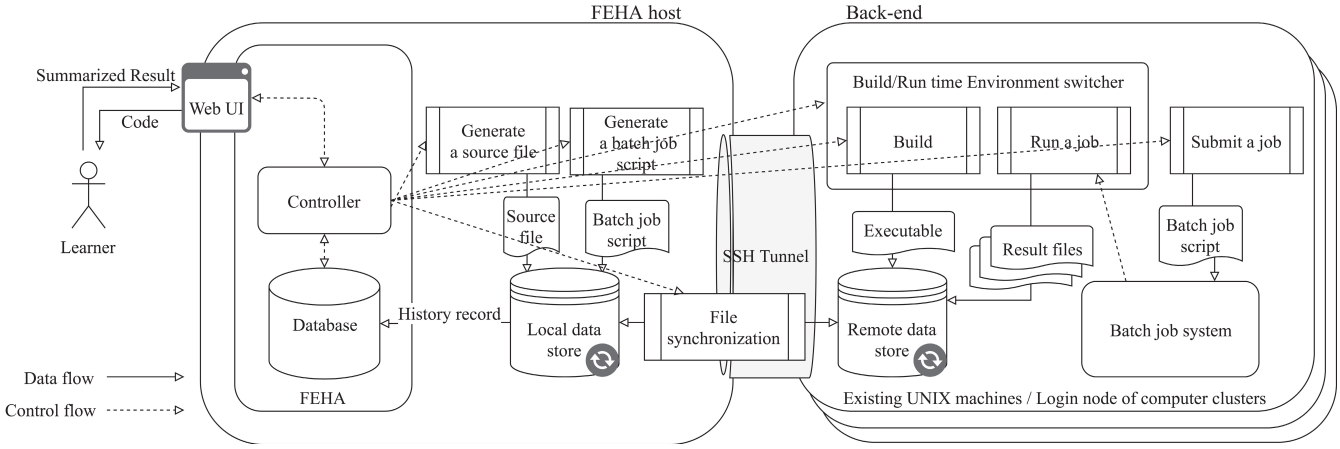


Fig. 4 Construction of FEHA. The learner interacts with FEHA via the web UI. FEHA utilizes the environment switcher and batch job system prepared on the back-end. FEHA proceeds with the program execution request in the left-to-right direction.

done by the FEHA. The database stores all information regarding the creation, submission, and execution of programs.

On the back-end side, FEHA assumes that the back-end has a build and runtime environment switcher and a batch job system, as described in Section 3. FEHA can also build and run programs without these features if the back-end does not have them. In that case, FEHA just uses the default environment to build and run the programs without computational resource management.

The FEHA host and back-end have local and remote data stores, respectively. The data stores are actual directories on each file system, which contain the source file, batch job script, executable, and result files. The stored files are also recorded as history in the database.

5.3 Implementation Details

We implemented FEHA by using Node.js with Express and MongoDB. Node.js and Express are a JavaScript runtime environment and web framework, respectively. Express provides an Application Programming Interface (API) to construct a web service. This is known to be an efficient architecture [20], [21]. We also used MongoDB, which implements the document-oriented data store as the database.

Express running on Node.js provides a web UI for the learners as a web server. The learners code programs and submit them to the web server as a HyperText Transfer Protocol (HTTP) request. The submitted code and options for building and running are sent with this request as HTTP parameters.

We defined designated HTTP endpoints by using Express to receive some types of requests. The requests are mainly categorized into "Submit C/C++ program," "Submit Verilog program," "Submit MIPS Assembly program," and "Get result." When an endpoint received a request, the corresponding procedure implemented on Node.js proceeds. When Node.js receives a "Submit C/C++ program," "Submit Verilog program," or "Submit MIPS Assembly program" request, Node.js extracts the submitted code and options for building and running from HTTP parameters. Then Node.js proceeds with the program submission procedure described in Section 5.4 according to the extracted information. For a "Get result" request, Node.js retrieves submission records including the execution results. Then, Node.js summarizes the records and places the summary on the web UI. Node.js ensures access control to accept these requests according to the user's permissions.

In the procedures, Node.js invokes external "ssh" and "rsync" commands. External commands generally cause a longer latency because the process waits for the completion of the commands. However, given the nature of Node.js, these external commands are issued asynchronously. This nonblocking mechanism of Node.js allows FEHA to process many requests effectively.

5.4 Program Execution Procedures

The main task of FEHA is to execute programs submitted by learners. FEHA roughly proceeds with a program execution request in the left-to-right direction in Fig. 4. The detailed process is as follows:

(1) Program submission
  (a) Create the local data store if it does not exist.
  (b) Generate a source file and batch job script on the basis of the user input. The source file contains submitted code. The job script is automatically generated according to the options specified by the user.
  (c) Store the source and script in the database.
  (d) Create the remote data store if it does not exist.
  (e) Synchronize the data stores to transfer the files to the back-end.
  (f) Build the source file at the back-end to generate the executable.
  (g) Synchronize the data stores to transfer the executable to the FEHA host.
  (h) Store the executable in the database. If the build fails, FEHA records and retires to proceed with the request.
  (i) Submit the job and wait for job allocation and execution, which are done by the batch job system. The batch job system puts the result files into the remote data store.
(2) Result acquisition
  (a) Synchronize the data stores to pull the result files into the local data store and then store them in the database.

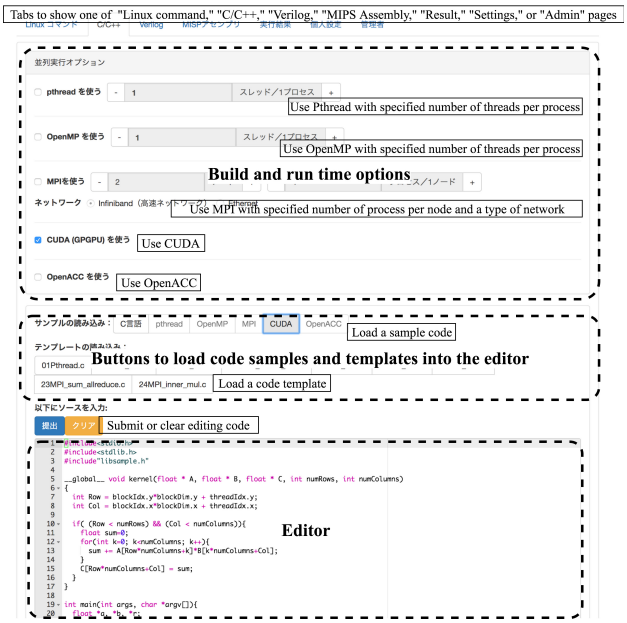


Fig. 5 Coding environment of FEHA. Learners input a code from the screen and submit it with the build and runtime options.

(b) Display an abstracted result of program execution.

According to the learner's request, program submission and result acquisition can occur individually.

For building and running programs, FEHA dynamically switches the development and runtime environment by using Environment Modules or Lmod described in Section 3.2. This makes FEHA adaptable to various programming technologies with small modifications of its implementation.

FEHA runs a submitted program by using one of two batch job systems: Torque or SLURM. The batch job system ensures the execution order of the submitted program and the fairness of resource assignment on behalf of FEHA. On the other hand, interactive or real-time applications cannot be run through FEHA.

5.5 Coding Environment

FEHA provides a coding environment that consists of an editor, a sample or template, and options buttons for building and running programs. **Figure 5** shows an example of the coding environment. For simplicity, FEHA allows the learners to input a code in a single file from this screen and submit it. If some additional files to be compiled with submitted codes are needed, the instructor can add them in the common settings of FEHA for the lecture.

Figure 6 shows an example of screen displaying the history of submissions and their details. The history of execution is shown on the left side as a list. The right side displays the details of each result. FEHA shows messages from the standard output and error on building and execution. FEHA also shows additional data when a program generates it. If the additional data includes text-based vector graphics, FEHA tries to show it as a Scalable Vector Graphics (SVG) image. Some results can be visualized by using this feature.

5.6 Supported Programming Activities and Technologies

FEHA supports the following activities.

Fig. 6 An example screen displaying the results. The history is shown on the left side. The details of each result are displayed on the right side.

C/C++ programming Fundamental C/C++ programming with GCC-compatible compilers.

Thread programming with C/C++ Thread-based parallel C/C++ programming with POSIX Thread (Pthread) and Open Multi-Processing (OpenMP).

MPI-based distributed parallel programming with C/C++ Distributed C/C++ parallel programming with MPI libraries on Ethernet and Infiniband.

GPGPU parallel programming with C/C++ CUDA and OpenACC programming with the CUDA Toolkit and PGI Compiler.

C programming for learning MIPS assembly C programming to learn MIPS assembly codes with the MIPS SDE toolchain.

Verilog HDL programming for RTL simulation Verilog Hardware Description Language (Verilog HDL) programming for Register Transfer Level (RTL) circuit simulation with Icarus Verilog. Programmers can specify input signals for simulation and check the result via a visualized signal chart graph.

Verilog HDL programming for logic synthesis Verilog HDL programming for logic synthesis with Yosys. Programmers can check a visualized net list graphs generated by Yosys.

If some lectures require new technology, the following development actions are mainly required.

(1) Design and add a new submission form that has simplified build and runtime options in a separate tab on FEHA's web UI.

(2) Add an HTTP endpoint to accept requests regarding the new technology.

(3) Implement a build and run procedure and associate it with the new endpoint.

6. Technological Challenges on Implementation

6.1 Implementing a Usable System for Novice Programming Learners

Since we designed FEHA for novice programming learners, the system should be usable for such learners. Once we performed a preliminary implementation of FEHA, we conducted a usability test by using Web Usability Scale (WUS) questionnaires developed by Nakagawa et al. [22].

We have shown the evaluation results in our previous paper [10]. Thus, we briefly summarize the results here. The WUS consists of 21 questions in seven factors: "Favorability," "Usefulness," "Reliability," "Layout," "Operability," "Visibility," and "Responsibility." Subjects (students) rate each question on a scale of 1 (poor) to 5 (good). We applied the WUS for two thread programming lectures to teach Pthread and OpenMP technologies, where 108 and 116 students respectively attended each lecture. We received 66 and 51 valid answers from the students.

Looking at the results, all factors graded as 3.1~3.5 on average. From the results, we confirmed that FEHA is designed as a useful system for most of the students.

6.2 Runtime Security

The availability of learning systems is one important concern during lectures. As we explained in Section 5.1, FEHA uses a local user account at the back-end to run the programs submitted by all learners working on FEHA. That is, the local account plays the role of a single proxy user shared by all learners. This means that a failure encountered by one learner may affect all other learners.

The learners who run programs through FEHA cannot exceed the restrictions set to the local user at the back-end. In most Unix-like systems, the system administrators do not grant any privilege to general users that are not administrators, and they usually restrict the computational resources and file access for general users. The runtime security of FEHA completely relies on the restrictions imposed for the proxy user at the back-end.

On the other hand, the back-end restriction does not expect that a single general user is shared by multiple programmers. Therefore, FEHA must take care of this and give appropriate restrictions to all learners. Examples of risks are as follows:

(1) A program goes into an infinite loop and blocks the programs submitted by other learners.
(2) A program exhausts computational resources.
(3) A learner might remove the files generated by other learners.

These risks may occur accidentally or intentionally.

We utilized the restriction features present in the batch job system and Unix shell. Since FEHA generates a batch job script for every submission, FEHA can add any option to restrict the resources for a job. In addition, as explained in Section 3.1, the batch job script is actually a simple shell script. Thus, FEHA can also add resource restrictions provided by the "limit" or "ulimit" commands that are built into the shells.

Other systems [3], [4] employ a sandbox, container, or virtual machine approach. However, these approaches require particular software, and it cannot be guaranteed that the back-end has such software. Therefore, we did not employ these approaches according to the design policy at this time.

Table 1 summarizes the restrictions done by FEHA. FEHA restricts computational resources by referring permissions set to the local account on the back-end and for the use of the batch job system. FEHA also restricts access to file contents by using an access control feature provided by the database.

Concerning file content protection, FEHA guarantees reproducibility for all submitted programs. Files submitted by one FEHA user are potentially broken by another FEHA user's program because FEHA shares a single account on the back-end for all FEHA users. This problem comes from the nature of the file access permission mechanism of UNIX-like systems. To take care of this problem, FEHA stores a submitted source file and generated batch job script in the database before synchronization, as described in the submission process described in Section 5.4. Since the database restricts the modification of the file contents stored in the database, as indicated in Table 1, each FEHA user can reload their past submitted codes from the database and resubmit them on FEHA's UI to reproduce the same result.

Table 1 List of restrictions for job execution.

Restricted by	Restrictions
Back-end account	Number of processes, number of threads, memory size, file size, CPU time, and so on.
Batch job system	All restrictions listed the above and job execution time.
Database in FEHA	Modifying contents of program source, batch job script, execution result, and so on.

7. Case Study

7.1 Lecture Outline

As a case study, we applied FEHA to a computer architecture class at a university. This class was opened for third-year undergraduate students.

The first and second lectures were on 14th and 21st July 2017, respectively. About 160 students attended both lectures. In the lectures, we taught a digital hardware design approach using Verilog HDL. We also followed the RTL simulation and circuit logic synthesis as a typical flow of digital hardware development.

Before applying FEHA, we were not be able to arrange any HDL programming activities during the lecture because of difficulties related to the preparation of the HDL programming environment. Using an HDL programming environment generally requires complicated personal settings for RTL simulation, logic synthesis, and visualization of the circuit netlist or input/output signals.

We arranged several programming assignments during two lectures by using Verilog HDL, which is a common programming HDL used for hardware product development. We provided these assignments through FEHA because the development environment of such digital hardware requires specialized tools and complex environment settings. We did not announce any preparation for the programming to the students.

7.2 Configuration of the Back-end Cluster

For this case study, we used a cluster that we usually use for other research projects as the back-end. The cluster consists of one login node, six GPU nodes, four multicore CPU nodes, and one large Symmetric MultiProcessing (SMP) node that has 24 cores in total. All nodes are connected with a single TCP/IP network using 1-Gb Ethernet. In addition, the login node, multicore CPU nodes, and SMP nodes are connected via Infiniband QDR, which is a high-performance network for parallel computing. The cluster has Torque 4.2.5 and Environment Modules 3.2.10 as the batch job system and environment switching software. We created one local user on the cluster for FEHA. We did not perform

Table 2 Summary of submission records.

Lecture	Total submissions	User submissions (Ave. / Min. / Max)	Active users (Attendance)	Number of Assignments
1st (14/7/2017)	1,629	10.9 / 1 / 25	150 (161)	6
2nd (21/7/2017)	867	5.9 / 1 / 19	148 (163)	3

any modification to use FEHA on the cluster.

The cluster has a toolchain for hardware development using Verilog HDL. We used the toolchain, which includes iverilog 0.9.20120609 and yosis 0.5 as the RTL simulator and logic synthesizer.

7.3 Setup of FEHA

The minimum functional requirements for the back-ends working with FEHA are that the back-ends have to provide "ssh" and "rsync" commands as well as a batch job system. If the back-ends have the Environment-modules capability, FEHA allows the instructors to add settings for each programming environment easily as an option. Most public computing systems can meet these requirements. Thus, FEHA is pluggable for most public computing systems.

To setup FEHA, the instructor and system administrator need to follow the following steps:

(1) Prepare a user account on the back-end (back-end user) and apply concise settings for the programming environment for the back-end user.
(2) Prepare a user account on the FEHA host (FEHA host user).
(3) Register an SSH public-key of the FEHA host user in the list of the back-end user's authorized keys on the back-end. FEHA does not support password authentication for security reasons. It is better to enable the SSH session sharing option provided by OpenSSH for the FEHA host user. This can dynamically reduce the latency of communications over an SSH session.
(4) Install FEHA itself on the FEHA host with the FEHA host user.
(5) Write the general configuration file on the FEHA host. The file contains settings for the web service, load balancing for HTTP, a database, and logs.
(6) Write the cluster profile. This profile contains information about the programming environment supported by the back-end. The information includes the types of the batch job system and environment switching software as well as the configuration of the compiler and libraries.
(7) Write the lecture profile. This profile contains the names of the lectures and the programming environment required for the lectures. The instructor can add their own libraries and code templates for the lectures.

In this case study, we installed FEHA on a server (Intel Xeon E3-1241 v3, 3.50 GHz, 8 cores; 16 GB DDR3, 1,600 MHz), and invoked eight instances of FEHA under the management of process manager PM2 2.4.6 for load balancing.

7.4 User Registration

At the beginning of the first lecture, we introduced FEHA to the students and then prompted them to register themselves on FEHA. The registration procedure includes just entering an email address and password on the registration page provided by FEHA. **Figure 7** shows the number of completed registrations on a time series from 15:00 to 16:00 during the lecture. The total number of completed registrations during this time span was 149. We note that the overall number of registrations was more than 149 because FEHA had been opened for students in the class before the lecture, and some students registered before the lecture.

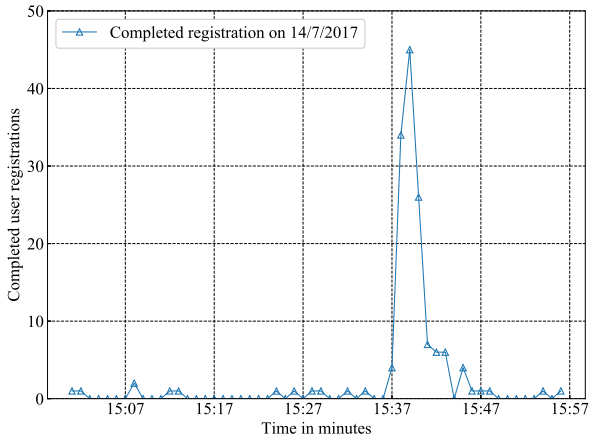


Fig. 7 Completed registrations on a time series. Within 3 min, 105 of 149 (≃70%) registrations were completed.

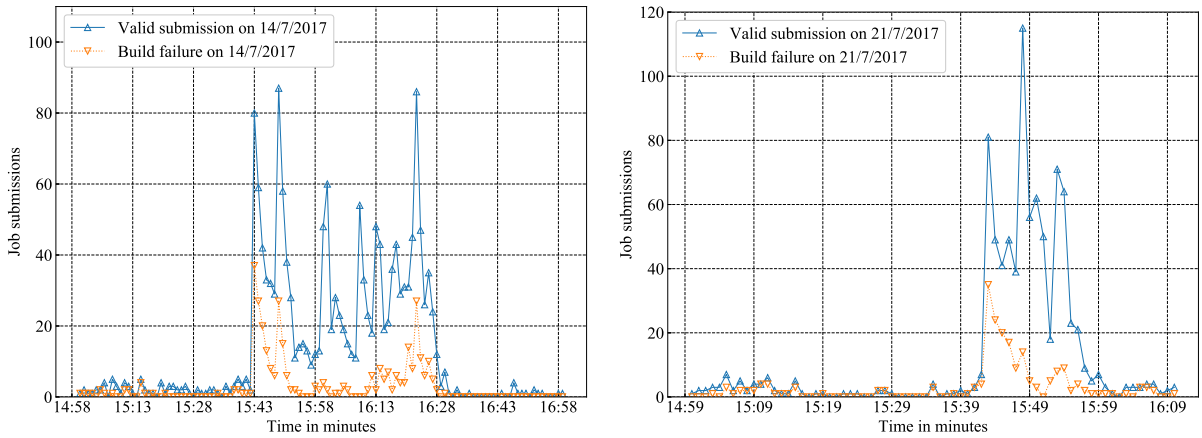
The graph shows that 70% (=105/149) of the students completed self-registration within 3 min (from 15:38 to 15:40). Moreover, the graph also shows that a total of 94% (=140/149) of the students completed self-registration by 15:43. We observed that programming activities began at 15:43 from the log. From this result, we supposed that most of the students who tried self-registration did not encounter a significant difficulty. Thus, we confirmed that FEHA aided to the preparation of a personal environment, as we aimed.

7.5 Program Submission

We have summarized the submission records of two lectures in **Table 2**. From the results, we found that the average number of submissions per user (student) exceeded the number of assignments corresponding to the lecture. This indicated that more than 90% students submitted one or more codes for each assignment. This indicates that most of the students were able to experience HDL programming without a fatal problem.

We also plotted the submission records on a time series, as shown in **Fig. 8**. The graphs show the numbers of submissions and build errors encountered by the students. From Fig. 8 (a), it can be seen that the submissions started around 15:43 during the first lecture. On the other hand, most of the registrations were completed around 15:40, as shown in Fig. 7. This indicates that the students could understand how to use FEHA and start to submit codes within several minutes after registration. This result answers the purpose of FEHA development.

Next, we focused on the difference between the numbers of



(a) 1st lecture (14/7/2017)

(b) 2nd lecture (21/7/2017)

Fig. 8 Numbers of submissions and build failures during submission on a time series. Students experienced a coding flow including build failure and program completion during 45 and 30 min in 1st and 2nd lectures, respectively.

submissions and build failures in Fig. 8. We constructed two types of assignments in both lectures: "Running a sample" and "Filling in a blank." "Running a sample" involves running completed code samples to learn how to use FEHA and to experience the technologies simply. "Filling in a blank" involves coding the missing parts of code templates to learn more about programming techniques.

Around 15:55-16:20 in Fig. 8 (a) and 15:50-16:00 in Fig. 8 (b), there are time spans in which we can observe relatively small numbers of build failures. The students were working on sample assignments during this time. In contrast, the number of build failures occupies about 1/3 of the submissions in other time spans. The students were trying to fill in the missing parts of the templates during this time.

In addition, most submissions approximately occurred within 45 and 30 min during the first and second lectures, respectively. These time spans occupy about 1/2 to 1/3 of the time slot of the lectures because the time slot of the lecture is 90 min. This indicates that most of the students were able to experience a coding flow including build failure and completion of the program during this shorter time span by utilizing the sample and template feature.

From this case study, we confirmed that FEHA helps learners experience a specialized technology in a reasonable time span. This result meets the purpose of FEHA.

8. Conclusion

In this paper, we described the details of the design and implementation of our FEHA, which is a web-based programming environment. FEHA is designed to utilize existing Unix-like systems that are equipped with a specialized programming environment for programming lectures. FEHA controls existing systems by using SSH and Rsync so that FEHA does not make any modification to the existing systems. FEHA also utilizes a batch job system installed on existing systems to fairly share computational resources between programming learners. This means that FEHA can be set up on most public computing systems and utilize their computational resources. Thus, FEHA increases the utilization of computational resources of existing computing systems and saves time during the preparation of the programming environment for programming lectures. FEHA also allows novice programming learners to experience various technologies by a summarized web-based UI.

On the other hand, FEHA only supports particular programming activities implemented in FEHA. This restricts learner's programming activities. To add new technologies or programming environments to extend the programming activities supported by FEHA, additional development is required. In addition, we assume that there is a condition that needs to be considered while using FEHA. The use of FEHA might cause a violation with the use of the back-end. For example, the use of some software installed at the back-end might be restricted for use by an individual user or a dedicated organization. The system administrator needs to pay attention to this kind of violation.

We showed a case study of FEHA when we applied it to programming lectures for digital circuit design. About 160 students attended the lectures. In the case study, we confirmed that 70% of the students completed registration within about 3 min. In addition, they were able to understand how to use the FEHA and started submission codes within several minutes after registration. The results showed that the students could experience a coding flow including build failure and program completion within 1/2 to 1/3 of the time slot of the lectures. The case study showed that FEHA helps more than hundreds of learners experience a specialized programming environment with a small amount of effort from the instructor.

In the future, FEHA will support virtualization technologies equipped on the back-end since it has become a key feature for any type of computing. From the point of view of file protection, FEHA should have some sandbox-like mechanism to prevent malicious activities as an option. Filesystem in USErspace (FUSE), a virtual machine, or system-call-overriding technologies could be a candidate to resolve this problem. Further, we will also apply FEHA to other lectures using state-of-the-art technologies.

Acknowledgments Part of this work was supported by JSPS KAKENHI Grant Numbers JP26330143 and JP16K00490. The

authors would like to thank the reviewers for their valuable comments and suggestions to improve the quality of this paper.

References

[1] Denning, P.J. and Gordon, E.E.: A Technician Shortage, *Comm. ACM*, Vol.58, No.3, pp.28–30 (2015).
[2] Codecademy: Codecademy (online), available from ⟨https://www.codecademy.com/⟩ (accessed 2017-08-01).
[3] Rodríguez-del Pino, J.C.: VPL, Virtual Programming lab for Moodle (online), available from ⟨http://vpl.dis.ulpgc.es/⟩ (accessed 2017-08-01).
[4] Dakkak, A., Pearson, C. and Hwu, W.M.: WebGPU: A Scalable Online Development Platform for GPU Programming Courses, *2016 IEEE International Parallel and Distributed Processing Symposium Workshops* (*IPDPSW*), pp.942–949 (2016).
[5] Lin, H.: Teaching Parallel and Distributed Computing Using a Cluster Computing Portal, *2013 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum*, pp.1312–1317, IEEE (2013).
[6] MoodleHQ: Moodle, Moodle Pty Ltd. (online), available from ⟨https://moodle.org/⟩ (accessed 2017-08-01).
[7] Blackboard: Blackboard (online), available from ⟨http://www.blackboard.com/⟩ (accessed 2017-08-01).
[8] KhanAcademy: Khan Academy (online), available from ⟨https://www.khanacademy.org/⟩ (accessed 2017-08-01).
[9] Courcera, Inc.: Coursera (online), available from ⟨https://www.coursera.org/⟩ (accessed 2017-08-01).
[10] Yazaki, S., Kikuchi, T., Tsuchiya, H. and Ishihata, H.: FEHA: An Adaptive Web-Based Front-End Environment to Support Hands-On Training in Parallel Programming, *Proc. Conference Future of Education*, pp.166–170 (2016).
[11] Bill, C.: WebSSH2 (online), available from ⟨https://github.com/billchurch/WebSSH2⟩ (accessed 2017-12-11).
[12] Kavanagh, S.: KeyBox: Web-Based Bastion Host and SSH Key Management (online), available from ⟨https://github.com/skavanagh/KeyBox⟩ (accessed 2017-12-11).
[13] Glassman, E.L., Scott, J., Singh, R., Guo, P. and Miller, R.: OverCode: Visualizing Variation in Student Solutions to Programming Problems at Scale, *Proc. Adjunct Publication of the 27th Annual ACM Symposium on User Interface Software and Technology, UIST'14 Adjunct*, pp.129–130, ACM (2014).
[14] Hashiura, H., Mori, K., Tanaka, T., Hazeyama, A. and Komiya, S.: An Environment for Collecting Fine-Grained Development Records to Help with Programming Exercise, *3rd International Conference on Advanced Applied Informatics*, pp.739–744 (2014).
[15] Wang, T., Su, X., Ma, P., Wang, Y. and Wang, K.: Ability-training-oriented Automated Assessment in Introductory Programming Course, *Computers & Education*, Vol.56, No.1, pp.220–226 (2011).
[16] Jin, W., Barnes, T., Stamper, J., Eagle, M., Johnson, M. and Lehmann, L.: Program Representation for Automatic Hint Generation for a Data-Driven Novice Programming Tutor, *Lecture Notes in Computer Science*, Vol.7315, Springer (2012).
[17] Antonucci, P., Estler, C., Nikolić, D., Piccioni, M. and Meyer, B.: An Incremental Hint System For Automated Programming Assignments, *Proc. 2015 ACM Conference on Innovation and Technology in Computer Science Education*, *ITiCSE '15*, pp.320–325, ACM (2015).
[18] Furlani, J.L. and Osel, P.W.: Abstract Yourself With Modules, *Proc. 10th USENIX Conference on System Administration, LISA '96*, pp.193–204, USENIX Association (1996).
[19] Geimer, M., Hoste, K. and McLay, R.: Modern Scientific Software Management Using EasyBuild and Lmod, *2014 First International Workshop on HPC User Support Tools*, pp.41–51, IEEE (2014).
[20] TechEmpower: Web Framework Benchmarks, Round 14 (online), available from ⟨https://www.techempower.com/benchmarks/⟩ (accessed 2017-05-10).
[21] Liu, D. and Deters, R.: The Reverse C10K Problem for Server-Side Mashups, *ICSOC 2008 International Workshops, Service-Oriented Computing*, pp.166–177 (2009).
[22] Nakagawa, K., Suda, T., Zempo, H. and Matsumoto, K.: The Development of Questionnaire for Evaluating Web Usability, *Proc. Human Interface Symposium 2001*, pp.421–424 (2001). (in Japanese).

Syunji Yazaki received his Ph.D. from The University of Electro-Communications (UEC) in 2007. From 2007 to 2016, he worked as an assistant professor at Tokyo University of Technology and UEC. From 2012 to 2013, he was a visiting scholar at The Ohio State University. From 2016 to 2018, he was a research associate at Hitotsubashi University. He moved to UEC as a project associate professor in 2018. His research interests include high-performance computing and Internet operation technology. He is a member of IPSJ, IEICE, IEEE, and ACM.

Hideaki Tsuchiya received his M.S. and Ph.D. degrees in Information Theory from The University of Electro-Communications in 1993 and 1997, respectively. He is an associate professor in Information Technology Center, the University of Electro-Communications. He is a member of IPSJ, IEEE, and IEICE.

Hiroaki Ishihata was born in 1957. He worked at Fujitsu Laboratories from 1980 to 2007. He received his Ph.D. from Waseda University in 1992. He became a professor at Tokyo University of Technology in 2007. His current research interests include high-performance computing. He is a Member of IPSJ, IEICE and IEEE-CS.