An Implementation of User-PC Computing System Using Docker Container

H. Htet, N. Funabiki, A. Kamoyedji, M. Kuribayashi, F. Akhter, and W.-C. Kao

***Abstract*—Recently, the computing ability of a personal computer (PC) has rapidly evolved with increases of the CPU clock rate, the number of cores, and the memory size. Then, the collection of idling resources of user-PCs will provide an efficient computing platform with very low costs. To realize this concept, we have studied the user-PC computing (UPC) system based on the master-worker model. By adopting the Docker container, our system allows various jobs or applications to run on PCs with different platforms and environments. In this paper, we present the details of the implementations of the UPC system, including the Docker image generation method for a given job. Here, to reduce the required time and file size, the existing environments of the target PC are checked first, and only the necessary ones are included in the image. For evaluations, we prepare four PCs with different specifications for the UPC system and nine jobs with various features as the applications. Then, we compare the CPU time and the file size to generate the Docker image for each job using two different PCs, when the proposed method is adopted or not. Besides, we measure the CPU time on the master and the worker to compute each job and the transmitted data size to the worker.**

***Index Terms*—User-PC computing, distributed system, resource usage, Docker, job management, Docker image generation.**

I. Introduction

Nowadays, the *Grid Computing (GC) system* has become popular as a parallel computing platform for large computing projects. By connecting a lot of computers via the Internet, the GC can provide a massive computing ability for engineers and scientists who need large computing resources. As a variation, the *desktop grid (DG) system* adopts *Personal Computers (PCs)* as computing machines because current PCs have achieved sufficient abilities for DG systems [1]. PCs have increased the computing performance with the faster CPU clock cycle, more CPU cores, the larger memory size, and the high storage capacity/access speed. Besides, PCs can be available with very low costs.

Previously, we studied the *User-PC Computing (UPC) system* as one style of the DG. The UPC can provide efficient computational platform for members in a group by using idling computing resources of their PCs [2]. Unlike the *Volunteer Computing (VC) system* [3], the UPC system can achieve the higher dependency by using the trusted PCs for the computing resources in the same organization or group. Using the *Docker container*, UPC allows various jobs or applications to run on PCs with various platforms and environments [4].

Container based technologies including Docker are much efficient at various computing environments. The Docker container technology is adopted in the UPC system to avoid platform constraints and dependencies in executing jobs on user-PCs. Docker can quickly assemble applications from components to afford different environments.

Docker is a software tool that has been designed to make it easier to create, deploy, and run an application program on various platforms using a container [5]. The *container* allows the application developer to package up the required software to run the application program, such as libraries, middleware, parameters, and other dependencies, into one package called the container image, to be shipped out. The *Docker container image* is a lightweight, standalone, and executable package of every software needed to run the application. It includes the codes, the runtime environment, the system tools, the system libraries, and the settings.

The UPC system has been designed with the master-worker model as in Fig. 1. The usage flow of the UPC system is as follows: 1) a UPC user submits a computing project to the *UPC master* through the Web server, 2) the master divides the project into a set of jobs, 3) the master generates the *Docker image* for each job, 4) the master finds the schedule of assigning the jobs to *UPC workers*, 5) the master transmits the *Docker images* of the jobs to the scheduled workers, 6) the UPC worker computes the assigned job using the *Docker* container technology and transmits the result to the master when it is finished, 7) the master receives the job result from the worker, and 8) the master returns the project result to the user when it receives the results for all the jobs.

In this paper, we present the implementation of the UPC system including the *Docker container*. To reduce the time to generate the Docker image for a submitted job from a user and reduce the size, the existing environments of the worker PC that will be assigned the job, are checked from the log information of the previous jobs on the PC. Then, only the

Manuscript received October 20, 2020; revised December 10, 2020. This work was supported by the Funabiki Laboratory (Distributed Systems Design Lab), Department of Electrical and Communication Engineering, Okayama University, Japan.

H. Htet, N. Funabiki, A. Kamoyedji, M. Kuribayashi, and F. Akhter are with the Electrical and Communication Engineering Department, Okayama University, Japan (e-mail: heinhtet@s.okayama-u. ac.jp, funabiki@okayama-u. ac.jp, cyberntique@gmail.com, kminoru@okayama-u. ac.jp, p90n6c8t@s.okayama-u. ac.jp).

W.-C. Kao is with the Department of Electrical Engineering, National Taiwan Normal University, Taipei, Taiwan (e-mail: jungkao@ntnu.edu.tw).

doi: 10.18178/ijfcc.2020.9.4.568

newly necessary environments are included in the Docker image.

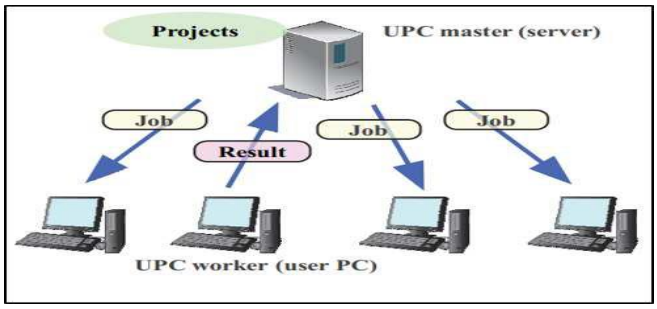

Fig. 1. Master-worker model in UPC system.

For evaluations of the implemented UPC system, we measure the CPU time to compute nine jobs of various features when they are computed on the system with four PCs that have different CPU architecture including the number of logical and physical cores. These jobs represent a network simulation, a convolutional neural network, an image compression, and are implemented by C++, C, Assembly, and Python.

In addition, to verify the effectiveness of the proposed Docker image generation, we measure the CPU time to generate the Docker image and the size for the same jobs on two different PCs, and evaluate the reductions of the total CPU time and the size of the Docker images over nine jobs by applying our proposal.

The rest of this paper is organized as follows: Section II discusses relevant works in literature. Sections III, IV, and V present the implementation of the UPC Web server, the master, and the worker, respectively. Section VI evaluates the implementations in the system. Finally, Section VII concludes this paper with future works.

II. Related Works in Literature

In this section, we briefly present related works in this paper.

In [6], Benjamin *et al*. presented comparisons of the behaviors of four virtualization tools in grid computing environments. The authors measured the CPU, memory, disk, and network usage by executing the micro benchmark programs in each VM tools and evaluated the linearity, overhead, and performance isolation. This work helps the user to select the suitable tool according to the application's nature.

In [7], Xavier *et al*. presented performance evaluations between the containerization and the virtualization for HPC applications. The authors found that the containerization is the lightweight alternative to the virtualization for HPC applications.

In [8], Park *et al*. presented a container-based cluster management platform to provide dynamic distributed computing environments desired by users. The authors compare the performance between the Docker-based execution and the native one by using two benchmark tools implemented with C, Java, Python, and R, and showed that Docker offers the near-native performance.

In [9], Jaikar et al. focused on executions of scientific jobs that require intensive resources in cloud computing environments. They proved that the Docker container outperforms the OpenStack virtual machine to execute the CPU and memory intensive jobs. Besides, the Docker container consumes the less power while executing scientific jobs.

III. Implementation of UPC Web Server

In this section, we present the software platform and fundamental functions of the UPC Web server.

A. Software Platform

The UPC Web server is implemented using *Node.js* that is an open source server environment and can run on various platforms including Linux, Windows, UNIX, and Mac OS. Thus, the functions of job accepting, result reception and downloading are described by *JavaScript*. *Node.js* offers the interpreter and the running environment of *JavaScript* source codes on the server [11].

In the UPC system as revealed in Figure 2, *Node.js* runs on the Linux OS. Node.*js* has a built-in module called HTTP to create the HTTP-based server that listens to server ports and gives responses to the UPC master. The user interface is implemented using HTML5 and CSS files. The file system is also controlled by *JavaScript* codes.

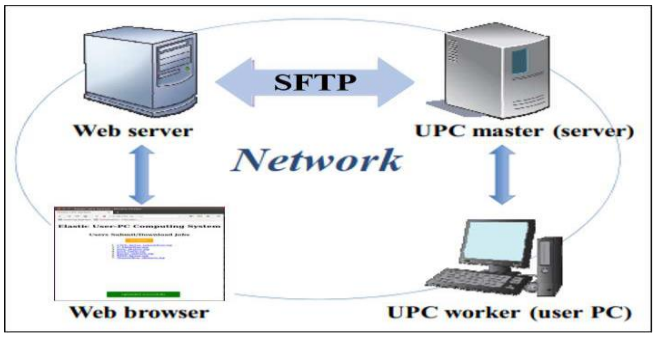

Fig. 2. User interface for UPC system.

B. Threads for Basic Functions

The three basic functions of the UPC Web server are implemented with different threads as follows.

1) The *job acceptance thread* may accept the jobs that are submitted through the Web interface. A user can submit a job that consists of the source codes, the required platform and libraries lists.
2) The *job transmission thread* transfers the submitted jobs to the UPC master using *SSH File Transfer Protocol (SFTP)* [12].
3) The *result reception thread* receives the results of the job from the UPC master and store them so that the user can download them from the Web interface.

C. Data Synchronization

For both interactive and automated file transfers between the Web server and the UPC master, *SSH File Transfer Protocol (SFTP)* is used as the secure file transfer protocol supporting the full security, so that the file synchronization is secured and relied. *SSH File System (SSHFS)* [13] is used at the master to mount the directories and the files located in the Web server.

IV. Implementation of UPC Master

In this section, we present the software platform and fundamental functions of the UPC master.

A. Software Platform

The source codes of the UPC master are implemented using *Python* for the server offering the multi-threaded programming [18].

The *MySQL* server [19] is adopted for the database to manage the information of the PCs for the UPC worker. Multiple UPC workers can connect to the UPC master at the same time, where one thread is allocated to each worker. The *Python* multi-threaded module can provide the powerful, high-level supports for the threads.

Besides, the *Docker* container technology [20] is used to provide the flexibility and portability for running various jobs on different worker PC platforms. *Docker* is an open source platform for developing, shipping, and running an application on an arbitrary PC environment. It builds the *Docker* image to offer the software environment for running each job, including the source codes, so that the job can run on a worker without considering the installed software.

B. Threads for Basic Functions

The four basic functions of the UPC master are implemented with different threads as follows.

1) The *job management thread* receives the request for a new job from the Web server by detecting the newly updated files using SFTP. Then, it prepares a new job by unzipping, inserting and modifying the *Docker file template*, then builds and saves the complete job running environment to the job queue.
2) The *worker management thread* receives the joining request from a UPC worker. When the master receives the request, it creates a new thread for the new worker, collects the information on the worker, and stores them at the master' database.
3) The *job transmission thread* sends a job in the job queue to the assigned worker, which is continued until the job queue is empty.
4) The *result uploading thread* sends the result from the worker to the UPC Web server using SFTP.

C. Docker Image Generation

The UPC master accepts the jobs from the Web server and prepares the *Docker file* for each job to build the *Docker image* that bundle the environment and the application, and execute it as a Docker container, as shown in Figure 3. The Docker image can be generated from the *Docker file* that contains the list of the instructions. In our Docker image generation method, the *Docker file* is automatically created by analyzing the list of requirements for the job from the user and the extensions of source codes. To reduce the generation time and the size of the Docker image file, it checks the previously built Docker image for the worker PC, and only the necessary files are included in the image. Figure 4 shows the details of the process.

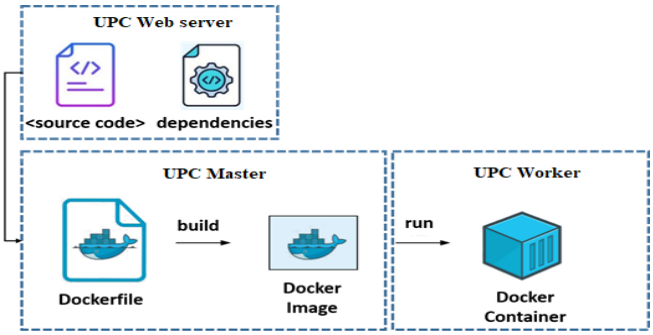

Fig. 3. Docker image generation process overview.

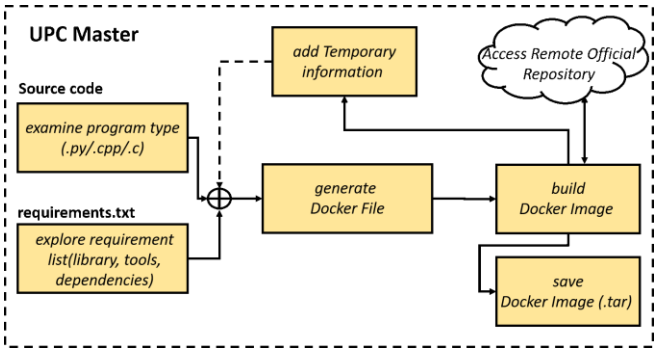

Fig. 4. Docker image generation process details.

In details, the UPC master performs the following steps to generate the Docker Image for each submitted job.

1) It unzips the job, examines the program type, and explores the requirement list.
2) It compares and checks the information obtain at step 1. with the log data under the temporary information directory that stores the previously built Docker image information.
3) It refers the previous built Docker image if the running environment, libraries, and dependencies are almost similar with the current job's requirements.
4) Otherwise, it refers the base image of the previously built Docker image when only the running environment is same.
5) Otherwise, it generates a new Docker image for the current job by following the instructions of the generated *Docker file*.
6) It accesses to *Public Remote Repository* to download and install the necessary images, libraries, and platforms, and chooses the small and light package to reduce the image size to a minimum.
7) It saves them as a *Docker* image when successfully finished, and adds it in the correspondence job list.

D. Worker Management

When a PC joins the UPC system as a worker, the UPC master collects the static performance information of the PC, such as the memory size, the CPU frequency, the number of cores, and the hard disk size. The master also periodically collects the dynamic performance information of the PC, such as the percentage of the current resource usage and the available resource status. The UPC master records all the information in the database. Thus, if the worker cannot keep running the job because the resource usage exceeds the upper limit, the UPC master can detect it. In this case, the UPC master can send stop alert of running UPC job to the worker and resume alert when resources are available to use.

V. Implementation of UPC Worker

In this section, we present the software platform and fundamental functions of the UPC worker.

A. Software Platform

The source codes of the UPC worker are also implemented using *Python* for the clients offering the multi-threaded programming. The *Docker* container technology is used to run the *Docker* image for each job on the worker assigned by the UPC master.

B. Threads for Basic Functions

The five basic functions of the UPC worker are implemented with different threads as follows.

1) The ***connection initiation thread*** finds the address and the port of the UPC master from the socket. Then, the worker is connected to the UPC master by sending the necessary information.
2) The ***job reception thread*** receives the Docker image for the job with the .tar file and temporarily allocates it in the disk space of the worker.
3) The ***job execution thread*** starts to load and run the received *Docker* image as a container.
4) The ***job restoring thread*** saves the current running states of the job in the hard disk and sends the state to the master when the worker PC runs out all the available resources.
5) The ***result transmission thread*** transfers the result of the job when successfully completing it. Then, it automatically removes the *Docker* image and the container from the disk space of the worker.

C. Resource Usage Measurement

The resource usage of the UPC worker is measured using *psutil* (process and system utilities), a Python cross-platform library. *psutil* can monitor, profile, and limit the process resources, and manage the running processes [10].

Figs. 5 and 6 show the CPU and memory usage rates of one student's PC in our group for one weekday. *psutil* recorded the resource usage rates at every one minute. Both usage rates significantly increased at the daytime from 8:30am to 6:30pm. It is found that the CPU usage rate is not high for the whole day where it stays between 2% and 5%. Thus, the CPU has sufficient capability of running the job in the UPC system.

On the other hand, the memory usage rate is relatively high for the whole day where it stays between 56% and 62%. Therefore, it is necessary to consider the proper memory use for running the job in the UPC system.

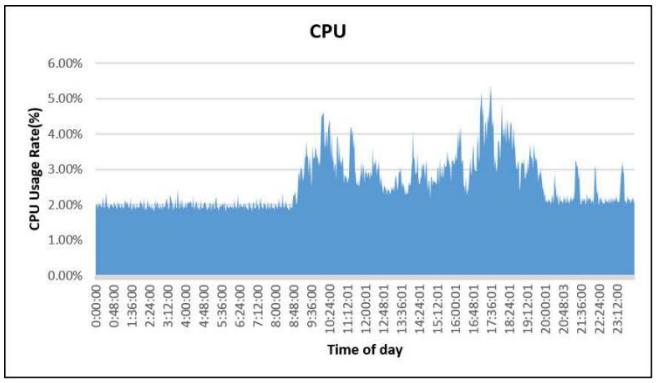

Fig. 5. CPU usage rate of student's PC.

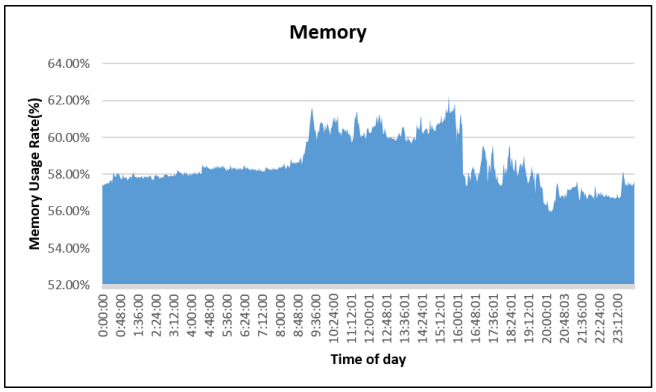

Fig. 6. Memory usage rate of student's PC.

D. Job Control Function

In the UPC system, the running job on a worker must not disturb the use of the PC by the owner. As shown in the previous subsection, the memory usage rate is generally high.

Therefore, the job control function is implemented to stop the running job and free the memory for the job, when the memory usage rate becomes higher than the given threshold. In this study, 90% is selected for the threshold from our experiment results that will be shown in Section VI. Then, we discuss the implementation of worker PC memory control on Linux or Windows operating system.

First, we discuss the implementation for Linux. **kill** command is used to stop the job. Then, ***'kill -STOP #Process-Name'*** command is used to free the memory. If the job can run there again, ***'kill -CONT #ProcessName'*** command is used to resume the job.

Next, we discuss the implementation for Windows. **taskkill** command is used to stop the job. Then, ***'Stop-Process -Name #ProcessName'*** command is used to free the memory. If the job can run there again, ***'Cont-Process -Name #ProcessName'*** command is used to resume the job.

VI. Experiments

In this section, we conduct experiments of the improved UPC system.

A. Experiment Setup

In our experiments, we adopt one Master PC and four worker-PCs in Table I. These worker PCs can be regarded into two groups as PC-1, 2 and PC-3, 4 depending upon the number of cores. Then, we adopt two C++, three C and four Python programs for jobs in Table III. The PCs are connected with the master through the 100Mbps wired Ethernet, and have the SSD disks.

The two C++ programs (Palabos [14], Flow [15]) were physic simulation programs that consumed memory usage a lot. The two C programs (Network Simulator, Optimization Algorithm) were developed in our group for wireless network studies [16]. The three Python programs (DCGAN, RNN, and CNN) were picked up from the *GitHub* repository for neural networks [17]. They require high computing resources. The remaining one C program (FFmpeg) [21] and one Python program (Converter) [22] are related with processing of multimedia content. Nowadays, multimedia content processing is becoming popular and working with large content takes more processing time.

TABLE I: PC Specifications in Experiments

PC		PC-1	PC-2	PC-3	PC-4	Master
processor type		core-i3	core-i5	core-i7	core-i9	core-i5
number of cores		4	4	8	16	4
clock frequency (GHz)		1.70	2.60	3.40	3.60	3.20
memory (GB)	**available**	2	2	4	8	8
	total	4	4	8	16	8
disk (GB)	**available**	64	64	64	64	225
	total	500	500	500	500	225

B. Worker Usability by CPU Rate

First, we conduct the experiment of verifying the usability of the worker, when CPU usage rate of the PC is very high while running a UPC job. Here, we run the Python program for *Convolutional Neural Network (CNN)* five times on PC-3, and check the operability of the PC.

Fig. 7 shows the change of the CPU usage rate and the CPU time of the job execution. At the first run, the job occupies 332% of the total CPU resource, which indicates that three cores are fully occupied. At the second-fifth runs, when the PC owner has daily computational processes of Word, PowerPoint, Web access, the operating system automatically controls them at the higher priority than the jobs for the UPC system, and reduces the assigned CPU resource to the job at 296%, 224%, 181%, and 315%, respectively. Thus, no disturbance occurs in handling the owner processes.

C. Worker Usability by Memory Rate

Next, we conduct the experiment of verifying the usability of the worker PC, when the memory usage rate of the PC is very high. Figure 8 shows the change of the memory usage rate and the CPU time of the job program. The PC does not work properly at the fourth run. When it exceeds 90%, the PC is hung up and needs to be rebooted, where all the running processes are lost. Therefore, the memory usage rate for the UPC job must be carefully controlled to avoid the problem.

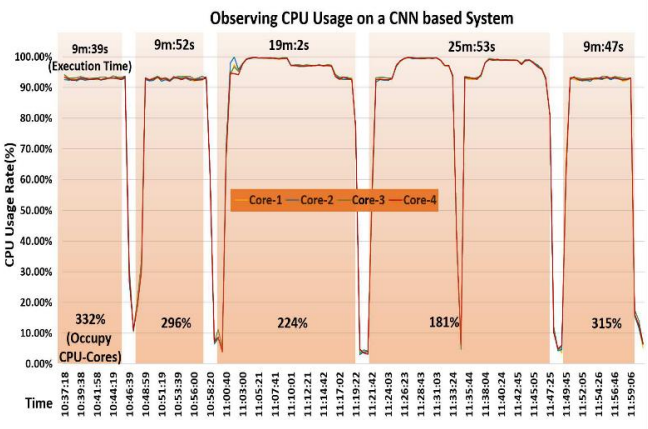

Fig. 7. CPU usage rate by CNN program.

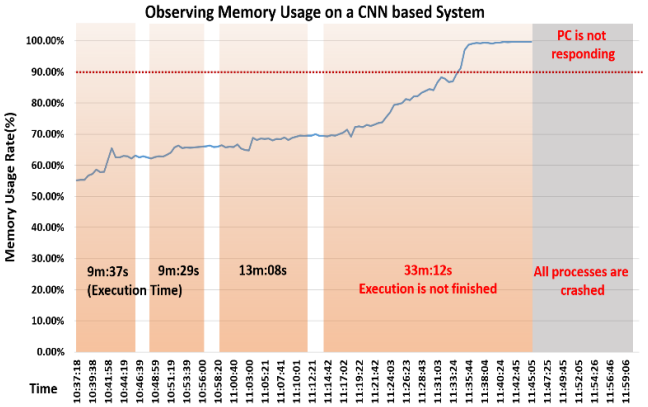

Fig. 8. Memory usage rate without control.

D. Job Control Result

Then, we apply the implemented job control function. Fig. 9 shows the change of the memory usage rate when the same CNN program runs on the PC five times. Every time the rate exceeds the given threshold (90%), the job is automatically stopped and about 36% of the memory is released to keep running daily processes by the PC owner.

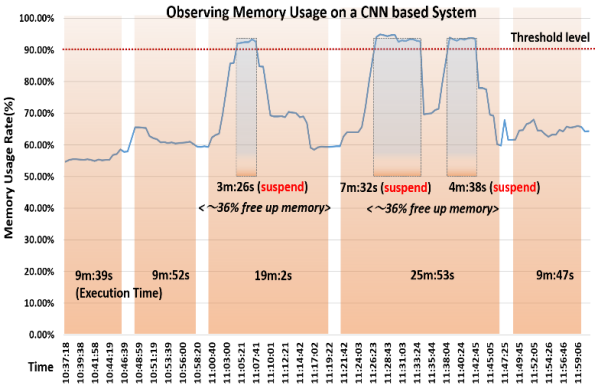

Fig. 9. Memory usage rate with control.

E. Docker Image Generation

Besides, to verify the effectiveness of the proposed Docker image generation method, we measure the CPU time to generate the Docker image and the file size for the same jobs

on two different PCs, and evaluate the reductions of the total CPU time and the size of the Docker images by our proposal. Usually, the Docker image is generated at the UPC master in the UPC system. Table II compares the CPU time and the file size on the master and PC-1 that is the slowest PC, before and after applying the proposal. As the effectiveness, the total image generation time is reduced to 7min. 24sec. at the master and 17min. 04sec. at PC-1 and the total image size for the nine jobs is reduced to around 3.126 GB.

TABLE II: EVALUATION OF DOCKER IMAGE GENERATION

jobs	Image generation time (H:M:S)				Image size (GB)	
	master		PC-1		master	
	before	after	before	after	before	after
Simulator	0:01:03	0:00:19	0:02:36	0:00:38	0.395	0.217
Algorithm	0:00:42	0:00:12	0:01:10	0:00:12	1.5	0.892
DCGAN	0:01:35	0:01:09	0:03:33	0:02:28	1.9	1.83
RNN	0:01:27	0:00:28	0:03:18	0:00:42	1.8	1.78
CNN	0:02:24	0:01:36	0:05:16	0:04:25	3.3	3.26
FFmpeg	0:02:37	0:02:10	0:06:37	0:06:13	4.4	3.16
Convert	0:03:49	0:01:47	0:11:07	0:04:34	3.5	2.89
Palabos	0:06:19	0:05:04	0:12:07	0:09:48	6.7	6.34
Flow	0:01:31	0:01:18	0:03:36	0:03:16	0.441	0.441
total	0:21:27	0:14:03	0:49:20	0:32:16	23.93	20.81
Reduced	0:07:24		0:17:04		3.126	

F. Measurements of Resource Usage by Jobs

Now, we measure the resource usage of the four PCs by the nine jobs. Table III shows the CPU time, the memory usage rate, and the required disk space of each job.

TABLE III: FEATURES OF NINE JOB PROGRAMS

Jobs	**CPU time (H:M:S)**				**Memory usage rate (%)**				**Disk space (GB)**			
	PC-1	**PC-2**	**PC-3**	**PC-4**	**PC-1**	**PC-2**	**PC-3**	**PC-4**	**PC-1**	**PC-2**	**PC-3**	**PC-4**
Simulator	02:16:12	01:08:10	00:55:47	00:41:06	0.68	0.67	0.17	0.08	0.393	0.393	0.393	0.393
Algorithm	00:44:04	00:28:59	00:22:48	00:16:14	0.69	0.67	0.18	0.07	1.37	1.37	1.37	1.37
DCGAN	01:41:29	01:13:43	00:26:59	00:17:00	38.69	37.76	12.18	4.2	1.87	1.87	1.87	1.87
RNN	00:21:19	00:15:37	00:10:39	00:09:13	30.36	29.77	7.86	2.08	1.84	1.84	1.84	1.84
CNN	00:32:23	00:28:41	00:13:26	00:11:43	35.03	36.87	11.74	4.53	4.04	4.04	4.04	4.04
FFmpeg	00:52:57	00:38:09	00:19:43	00:14:19	21.69	18.04	5.12	1.04	4.43	4.43	4.43	4.43
Converter	00:24:25	00:18:50	00:12:57	00:12:09	18.89	18.03	5.23	2.19	3.46	3.46	3.46	3.46
Palabos	00:27:28	00:23:15	00:20:01	00:19:06	42.96	47.98	14.63	4.46	6.68	6.68	6.68	6.68
Flow	00:27:09	00:16:34	00:12:57	00:10:21	47.97	48.05	15.14	5.08	0.43	0.43	0.43	0.43
total	07:47:26	05:11:58	03:15:17	02:31:11								

TABLE IV: CPU TIME AND TRANSMISSION DATA SIZE FOR NINE JOB PROGRAMS

Jobs	**CPU time on master (H:M:S)**			**CPU time on worker (H:M:S)**			**Total CPU time**	**Transmission data size**		**Assigned worker**
	Build	**Save**	**Transfer**	**Load**	**Run**	**Transmit**		***Docker* (GB)**	**Result (KB)**	
Simulator	00:00:19	00:00:56	00:00:11	00:00:07	00:53:25	00:00:40	00:55:38	0.217	17	PC-3
Algorithm	00:00:12	00:01:04	00:01:05	00:01:51	00:23:16	00:00:40	00:28:08	0.892	8	PC-2
DCGAN	00:01:09	00:01:29	00:01:37	00:00:34	00:11:09	00:00:55	00:16:53	1.83	20	PC-4
RNN	00:00:28	00:01:32	00:01:36	00:00:22	00:04:36	00:00:40	00:09:14	1.78	8	PC-4
CNN	00:01:36	00:02:09	00:02:34	00:00:59	00:03:42	00:00:40	00:11:40	3.26	4	PC-4
FFmpeg	00:02:10	00:01:54	00:02:16	00:01:23	00:04:58	00:01:50	00:14:31	3.16	1.7e+6	PC-4
Converter	00:01:47	00:03:50	00:01:39	00:04:51	00:07:54	00:00:30	00:20:31	2.89	179300	PC-2
Palabos	00:05:04	00:01:22	00:00:11	00:02:52	00:20:20	00:01:25	00:31:14	6.34	250	PC-1
Flow	00:01:18	00:00:21	00:00:20	00:00:49	00:24:43	00:00:00	00:27:31	0.438	0	PC-1
total							03:35:20			

The first two jobs are C programs that do not use *multi-thread*. Thus, the CPU time is not much different between the worker PCs except PC-1 that has lack of maximum turbo frequency feature. They do not consume much memory. The next three neural networks jobs are Python programs that use *multi-thread*. Thus, the CPU time is much different between the worker PC-1,2 and PC-3,4. They consume much memory.

Among two multimedia processing jobs, C program (FFmpeg) use *multi-thread* and so, the CPU time is much different, however, Python program (Converter) do not use *multi-thread* and the CPU time is not much different between the worker PC-1,2 and PC3,4. The last two physic simulation jobs are C++ programs that use only two and four threads during execution. Therefore, the CPU time is not much different be-tween PC-1,2 and PC-3,4. However, these simulation programs consume much memory.

G. Measurements of CPU Time and Disk for UPC System

Then, we measure the CPU time and the disk space required to execute each job in the UPC system. After a job is submitted to the UPC master through the Web server, the job is processed in the following six steps:1) building the *Docker* image file at the master (Build), 2) saving the image file in the disk at the master (Save), 3) sending the image file from the master to one worker (Transfer), 4) receiving the image file and loading it into the memory at the worker (Load), 5) running the job program at the worker (Run), and 6) sending back the result from the worker to the master (Transmit). Thus, the CPU time for each of the six steps is measured.

Table IV shows the CPU time required for each step of the nine jobs on the master and on one worker, and the data size for transmissions between the master and the worker. Here, we assign a job to a worker in descending order of the CPU time for the jobs in Table III. The image building time is reduced by referencing the similar previously built image around one minute. However, it can be saved a lot time for the regions where the Internet communication speed is poor to download the necessary packages from the remote official repositories and for low performance PC to install all the downloaded packages. The running time on a worker dominates the required time for each job. Thus, the proper worker assignment for each job is critical in improving the performance of the UPC system. It will be in our future works.

VII. Conclusion

This paper presented the implementation of the *UPC system* using the *Docker container* to run various jobs on various worker PCs. The CPU time was measured when nine jobs with various features were computed on four PCs with different CPU architecture. The effectiveness of the *Docker image generation method* was verified by comparing the total CPU time and the file size before and after applying the proposed method on two PCs for each job. In future works, we will implement the *job migration function* of dynamically changing the assigned worker of the currently running job to another one, when the performance of the current worker is low and that of the new worker is high, and study the *job scheduling method* to efficiently assign the jobs to the workers including the job migration.

Conflict of Interest

The authors declare no conflict of interest.

Author Contributions

H. Htet designed and implemented the system; N. Funabiki supervised the research and revised the paper; A. Kamoyedji and F. Akhter took part in conducting measurements; M. Kuribayashi co-supervised the research; W.-C. Kao analyzed the paper and checked the grammatical errors; all authors had approved the final version.

Acknowledgment

I would like to express my deep gratitude to Professor Nobuo Funabiki for his valuable and constructive suggestions during the planning and development of this research work. I would also like to thank to Professor Wen-Chung Kao and Associate Professor Minoru Kuribayashi for their patient guidance, encouragement and useful critiques of this research. My grateful thanks are also extended to Mr. Ariel Kamoyedji and Ms. Fatema Akhter for their support in conducting measurements and useful advices. Finally, I wish to thank my parents, Prof. Nobuo Funabiki, Japanese Government and MEXT scholarship Japan for supporting and encouraging throughout my study.

References

[1] Desktop grid. [Online]. Available: http://www.desktopgrid.hu
[2] N. Funabiki, K. S. Lwin, Y. Aoyagi, M. Kuribayashi, and W. C. Kao, "A user-PC computing system as ultralow-cost computation platform for small groups," *Application and Theory of Computer Technology*, vol. 2, no. 3, pp. 10-24, 2017.
[3] L. F. G. Sarmenta, "Sabotage-tolerance mechanisms for volunteer computing system," *Future Gen. Com. Sys.*, vol. 18, no. 4, pp. 561-572, 2002.
[4] H. Htet, N. Funabiki, A. Kamoyedji, and M. Kuribayashi, "Design and implementation of improved user-PC computing system," *Technical Committee on Network Systems (NS)*, pp.37-42, 2020.
[5] Docker. [Online]. Available: https://docs.docker.com/get-docker/
[6] Q. Benjamin, N. Vincent, and C. Franck, "Selecting a virtualization system for grid/P2P largescale emulation," Workshop on Experimental Grid testbeds for the assessment of large-scale distributed applications and tools, France, 2006.
[7] X. Miguel, N. Marcelo, R. Fabio, F. Tiago, L. Timoteo, and R. Cesar, "Performance evaluation of container-based virtualization for high performance computing environments," in *Proc. 16th Euro micro Conference on Parallel, Distributed and Network-Based Processing*, pp. 233-240, 2013.
[8] P. J. Won and H. Jaegyoon, "Container-based cluster management system for user-driven distributed computing," *KIISE Transactions on Computing Practices*, pp. 587-595, 2015.
[9] J. Amol, B. Sangwook, H. Heejune, K. Byungyun, A. Syed, and N. Seo-Young, "OpenStack and docker comparison for scientific workflow w.r.t. execution and energy," 2016.
[10] Psutil. [Online]. Available: https://pypi.org/project/psutil/
[11] Node.js. [Online]. Available: https://nodejs.org/en/
[12] SFTP. [Online]. Available: https://www.ssh.com/ssh/sftp/
[13] SSHFS. [Online]. Available: https://wiki.archlinux.org/index.php/SSHFS
[14] J. Latt, "Palabos, parallel lattice Boltzmann solver," FlowKit, Lausanne, Switzerland, 2009.
[15] Open Porous Media (OPM) Initiative, "Flow: Fully Implicit Black-Oil Simulator," 2018.
[16] M. M. Islam, N. Funabiki, M. Kuribayashi, S. K. Debnath, K. I. Munene, K. S. Lwin, R. W. Sudibyo, and M. S. A. Mamun, "Dynamic access-point configuration approach for elastic wireless local-area network system and its implementation using Raspberry Pi," *Int. J. Netw. Comput.*, vol. 8, no. 2, pp. 254-281, July 2018.
[17] Neural network. [Online]. Available: https://github.com/aymericdamien/TensorFlow-Examples/tree/master/examples/3_NeuralNetworks
[18] Python. [Online]. Available: https://docs.python.org/3/library/socketserver.html
[19] MySQL. [Online]. Available: https://dev.mysql.com/doc/refman/8.0/en/
[20] Docker container. [Online]. Available: https://www.docker.com/resources/what-container
[21] FFmpeg. [Online]. Available: https://github.com/FFmpeg/FFmpeg
[22] Converter. [Online]. Available: https://github.com/andyp123/mp4_to_mp3

H. Htet received the B.E. and M.E. degrees in information science and technology from the University of Technology (Yatanarpon Cyber City), Myanmar, in 2015 and 2018, respectively. He is currently a Ph.D. student in Graduate School of Natural Science and Technology at Okayama University, Japan. His research interests include distributed computing system, big data analysis, computer networks, heterogeneous computing

devices, and controller.

N. Funabiki received the B.S. and Ph.D. degrees in mathematical engineering and information physics from the University of Tokyo, Japan, in 1984 and 1993, respectively. He received the M.S. degree in electrical engineering from Case Western Reserve University, USA, in 1991. From 1984 to 1994, he was with Sumitomo Metal Industries, Ltd., Japan. In 1994, he joined the Department of Information and Computer Sciences at Osaka University, Japan, as an assistant professor, and became an associate professor in 1995. He stayed at University of Illinois, Urbana Champaign, in 1998, and at University of California, Santa Barbara, in 2000-2001, as a visiting researcher. In 2001, he moved to the Department of Communication Network Engineering (currently, Department of Electrical and Communication Engineering) at Okayama University as a professor. His research interests include computer networks, optimization algorithms, educational technology, and Web technology. He is a member of IEEE, IEICE, and IPSJ.

A. Kamoyedji is currently a Ph.D. student in Graduate School of Natural Science and Technology at Okayama University, Japan. His research interests include are optimization, scheduling algorithm Design and UPC system. He holds a master degree in system and information engineering from Ashikaga Institute of Technology, Japan.

M. Kuribayashi received the B.E., M.E., and D.E. degrees from Kobe University, Kobe, Japan, in 1999, 2001, and 2004. From 2002 to 2007, he was a research associate in the Department of Electrical and Electronic Engineering, Kobe University. In 2007, he was appointed as an assistant professor at the Division of Electrical and Electronic Engineering, Kobe University. Since 2015, he has been an associate professor in the Graduate School of Natural Science and Technology, Okayama University. His research interests include digital watermarking, information security, cryptography, and coding theory. He received the Young Professionals Award from IEEE Kansai Section in 2014. He is a senior member of IEEE.

D. Akhter received her B.S. in computer science and engineering from Jatiya Kabi Kazi Nazrul Islam University, Bangladesh in 2016. In recognition of her academic achievement, she was awarded Gold Medal from the President of Bangladesh in 2017. She is currently doing her M.S. in Electronic and Information Systems Engineering, Okayama University, Japan under MEXT scholarship of Japanese Government. Her research interest includes wireless communication and network security. She is a student member of IEEE.

W.-C. Kao received the M.S. and Ph.D. degrees in electrical engineering from National Taiwan University, Taiwan, in 1992 and 1996, respectively. From 1996 to 2000, he was a Department Manager at SoC Technology Center, ERSO, ITRI, Taiwan. From 2000 to 2004, he was an Assistant Vice President at NuCam Corporation in Foxlink Group, Taiwan. Since 2004, he has been with National Taiwan Normal University, Taipei, Taiwan, where he is currently a Professor at Department of Electrical Engineering and the Dean of School of Continuing Education. His current research interests include system-on-a-chip (SoC), flexible electrophoretic display, machine vision system, digital camera system, and color imaging science. He is a senior member of IEEE.