Evaluation of Re-Deployment for Environment-Adaptive Software

YOJI YAMATO*

*Network Service Systems Laboratories, NTT Corporation, Japan

yoji.yamato.wa@hco.ntt.co.jp

To use heterogeneous hardware, programmers needed sufficient technical skills such as OpenMP (Open Multi-Processing), CUDA (Compute Unified Device Architecture), and OpenCL (Open Computing Language). Therefore, I have proposed environment-adaptive software that enables high-performance operation by automatically converting and configuring the code once written, and have been working on automatic conversion and proper placement. However, until now, where to initially place the converted application has been considered, but the overall optimal placement has not been considered in consideration of the placement status of other users. In this paper, as a new element of environment-adaptive software, I study the relocation during operation, which improves the overall user satisfaction by considering the placement of other users, using a linear programming method. It was confirmed that it can be properly rearranged through simulation experiments.

CCS CONCEPTS • Software and its engineering • Software creation and management • Software development technique

Additional Keywords and Phrases: Environment-Adaptive Software, Automatic Offloading, Optimum Placement, Reconfiguration during Operation

1 INTRODUCTION

In recent years, Moore's Law is expected to slow down. Under such circumstances, not only CPUs (Central Processing Units) but also devices such as FPGAs (Field Programmable Gate Arrays) and GPUs (Graphics Processing Units) are being increasingly used. For example, Microsoft uses FPGAs to improve Bing search efficiency [1], and Amazon Web Services provides FPGAs and GPUs instances [2] with cloud technology (for example, [3]-[9]). In addition, the use of IoT devices in systems (for example, [10]-[18]) is increasing.

However, in order to properly utilize devices other than CPUs, it is necessary to create programs that are conscious of device characteristics, such as OpenMP (Open Multi-Processing) [19], OpenCL (Open Computing Language) [20] and CUDA (Compute Unified Device Architecture) [21] and embedded technology are required, so the skill barrier is high. Java [22] says that the code once written can be run anywhere, but the performance was out of consideration. Therefore, I have proposed environment-adaptive software which enables automatically conversion and deployment of the code once written so that the GPU, FPGA and many-core CPU that exist in the deployment destination environment can be appropriately used to operate the application with high performance. I also proposed and evaluated a method for automatically offloading application loop statements and functional blocks to FPGAs and GPUs as elements of environment-adaptive software [23]-[28]. Furthermore, I proposed a method for optimizing the placement destination of the converted application by satisfying the user's response time and price request.

In this paper, as a new element of environment-adaptive software, I propose and evaluate the reconfiguring method during operation in which the application that is converted, deployed and operated is relocated based on the placement status of other users to improve overall users' satisfaction. The contributions of this paper are as follows.

- I propose a method to calculate the relocation that improves the satisfaction of the entire user for every fixed number of user application placements using the linear programming method.
- For 700 applications, I perform a simulation to calculate the relocation and show that the relocation improves the satisfaction of the entire user.

2 EXISTING TECHNOLOGIES

2.1 Technologies on the market

CUDA is widespread as an environment for performing GPGPU (General Purpose GPU) (for example, [29]) that also uses the parallel computing power of GPU for general purposes. CUDA is an environment of NVIDIA for GPGPU, but OpenCL and its development environment (for example, [30]) has come out as specifications for handling heterogeneous devices such as FPGA, many-core CPU, GPU in the same way. However, although CUDA and OpenCL codes are forms in which the C language is extended, the difficulty of the program such as the description of memory processing is high.

For this reason, it is difficult for programmers with poor skills to use FPGAs and GPUs to speed up applications. In addition, even when using automatic parallelization technology (for example, [31]), trial operations of appropriate parallel processing part search was necessary to achieve high performances. As an effort to automate trial of parallel processing part search using instruction based specifications (for example, OpenACC (Open ACCelerators) [32][33]), I propose automatic GPU offloading using evolutionary computation method.

Regarding placement, there is research on optimizing the deployment position of VN (Virtual Network) for a group of servers on the network as an optimal use of network resources [34]. In [34], the optimum placement of VN is determined in consideration of communication traffic. However, for single-resource virtual networks, the purpose is to reduce carrier equipment costs and overall response time, and it did not consider conditions such as the processing time of different applications and individual user prices and response time requests.

2.2 Previous proposals

To adapt software to an environment, I previously proposed environment-adaptive software [28], the processing flow of which is shown in Figure 1. The environment-adaptive software is achieved with an environment-adaptation function, test-case database (DB), code-pattern DB, facility-resource DB, verification environment, and production environment.

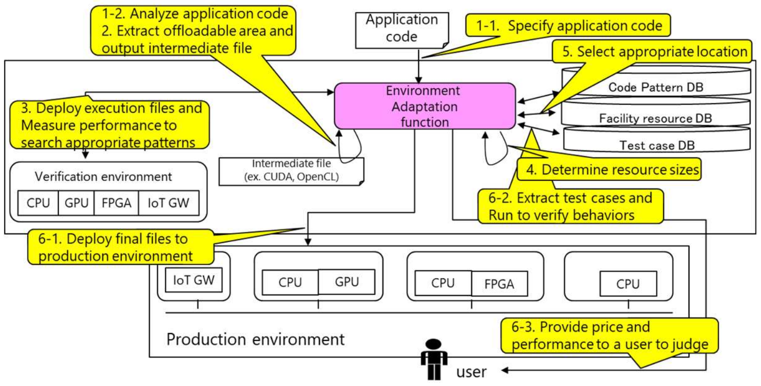
1-2. Analyze application code
2. Extract offloadable area and output intermediate file
Application code
1-1. Specify application code
5. Select appropriate location
Environment Adaptation function
Code Pattern DB
Facility resource DB
Test case DB
3. Deploy execution files and Measure performance to search appropriate patterns
Intermediate file (ex. CUDA, OpenCL)
4. Determine resource sizes
Verification environment
CPU
GPU
FPGA
IoT GW
6-2. Extract test cases and Run to verify behaviors
6-1. Deploy final files to production environment
IoT GW
CPU
GPU
CPU
FPGA
CPU
6-3. Provide price and performance to a user to judge
Production environment
user

Figure 1: Processing flow of environment-adaptive software.

Step 1: Code analysis
Step 2: Offloadable-part extraction
Step 3: Search for suitable offload parts
Step 4: Resource-amount adjustment
Step 5: Placement-location adjustment
Step 6: Execution-file placement and operation verification
Step 7: In-operation reconfiguration

In Steps 1-7, the processing flow conducts code conversion, resource-amount adjustment, placement-location adjustment, and in-operation reconfiguration for environment adaptation.

I will summarize this section. Because most offloading to heterogeneous devices is currently done manually, I proposed the concept of environment-adaptive software and automatic offloading to heterogeneous devices. However, regarding the placement of the application after automatic conversion, only the placement of the individual optimum according to the requirements of the individual user is examined, and the placement of the overall optimum is not examined in consideration of the placement situation of other users. Therefore, in this paper, I study to improve the satisfaction of multiple users to be reconfigured by reconfigured the placement after the start of operation.

3 RECONFIGURATION OF APPLICATION PLACEMENT

To embody the concept of environment-adaptive software, I have proposed automatic GPU and FPGA offloading of program loop statements and functional blocks and automatic adjustment of the offloading devices resource amounts. Based on these elemental technology studies, Section 3.1 outlines automatic GPU offloading technology for loop statements as an example, and in 3.2, the existing initial placement method and the necessity of reconfiguration will be described. In 3.3, I formulate a linear programming method and propose a method for appropriately reconfiguring placements of applications.

3.1 Automatic GPU offloading of loop statements

There are many cases where a program running on a normal CPU is speeded up by offloading it to a device such as a GPU or FPGA, but there are few cases where it is automatically conducted. For automatic GPU

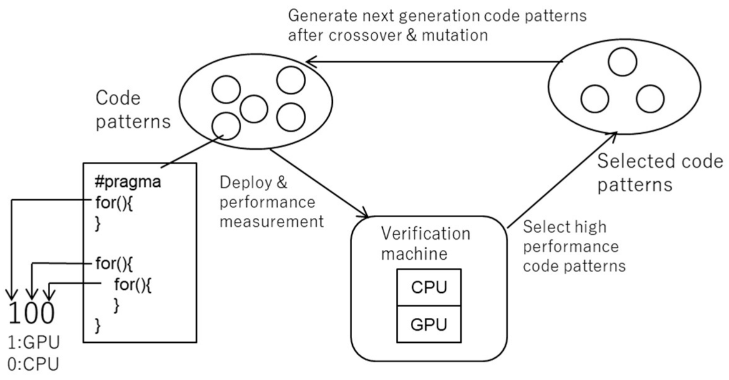
Generate next generation code patterns after crossover & mutation
Code patterns
Selected code patterns
#pragma
for(){
}
for(){
for(){
}
}
Deploy & performance measurement
Verification machine
CPU
GPU
Select high performance code patterns
100
1:GPU
0:CPU

Figure 2: Automatic GPU offload of loop statements.

offloading of loop statements, I have proposed several methods [23][28].

First, as a basic problem, the compiler can find the limitation that this loop statement cannot be processed in parallel on the GPU, but it is difficult to find out whether this loop statement is suitable for parallel processing on the GPU. Loops with a large number of loops are generally said to be more suitable, but it is difficult to predict how much performance will be achieved by offloading to the GPU without actually measuring it. Therefore, it is often the case that the instruction to offload this loop to the GPU is manually given and the performance measurement is tried. On the basis of that, [23] proposes automatically finding an appropriate loop statement from codes parsed by parsing library such as Clang [35] that is offloaded to the GPU with a genetic algorithm (GA) [36], which is an evolutionary computation method. From a general-purpose program for normal CPUs that does not assume GPU processing, the proposed method first checks the parallelizable loop statements. Then for the parallelizable loop statements, it sets 1 for GPU execution and 0 for CPU execution. The value is set and geneticized, and the performance verification trial is repeated in the verification environment to search for an appropriate area. By narrowing down to parallel processing loop statements and holding and recombining parallel processing patterns that can be accelerated in the form of gene parts, patterns that can be efficiently accelerated are explored from the huge number of parallel processing patterns (see, Figure 2).

The work of Yamato (2019) proposes transferring variables efficiently [28]. Regarding the variables used in the nested loop statement, when the loop statement is offloaded to the GPU, the variables that do not have any problem even if CPU-GPU transfer is performed at the upper level are summarized at the upper level. This is because if CPU-GPU transfer is performed at the lower level of the nest, the transfer is performed at each lower loop, which is inefficient. I also proposed a method to further reduce CPU-GPU transfers. Specifically, for not only nesting but also variables defined in multiple files, GPU processing and CPU processing are not nested, and for variables where CPU processing and GPU processing are separated, the proposed method specifies to transfer them in a batch.

Regarding GPU offload of loop statement, automatic offload is possible by optimization using the evolutionary computation method and reduction of CPU-GPU transfer.

Using a similar method, I have proposed automatic conversion of loop statements for FPGA and automatic offloading of function blocks. I also proposed a method of automatically adjusting the resource amounts of the

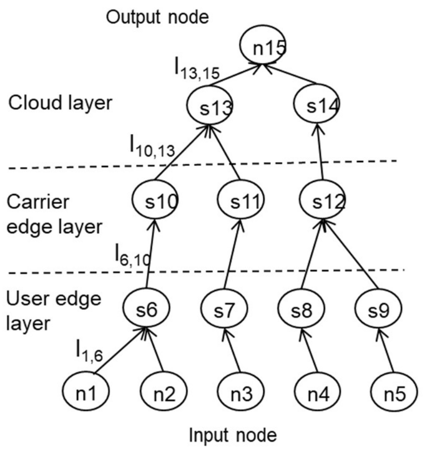

Figure 3: Network topology example.

offload destination device in order to operate with good cost performance after automatic code conversion.

3.2 Existing method and necessity of reconfiguration

As the computational node link, we assume a three-layer topology of user edge, carrier edge, and cloud (for example, Figure 3). Computational nodes can be divided into three types; CPU, GPU, and FPGA. Nodes equipped with GPU and FPGA are provided as GPU and FPGA instances by virtualization technology (for example, [37]), including CPU resources.

In the past, as in the study of [34], for example, the location where the server accommodating the virtual network is placed was systematically designed in view of long-term trends such as increased traffic. On the other hand, environment-adaptive software has two characteristics. The first is that the placement application is not statically determined, but is automatically converted for GPU and FPGA, and the offload pattern suitable for the usage is extracted through actual measurement through GA and so on, so the application code and performance can be dynamically changed. Second, it is not only necessary to reduce carrier equipment costs and overall response time, but it is also necessary to meet individual user requirements for response time and price, and the application placement policy can be changed dynamically.

Based on these two characteristics, I previously proposed a method which uses a linear planning method to calculate the response time and price of the converted application when the user requested the application placement in order to satisfy the user's response time request and price request. The proposed method minimizes either the response time or the price as the objective function for users' requests. Using the proposed method, I have confirmed the effectiveness by simulating the placement optimization of 1000 applications with GLPK (Gnu Linear Programming Kit) 5.0, assuming multiple types of applications.

However, in the proposed method, the application is arranged according to the response time and price request of each individual user, so that it is basically a first-come-first-served basis. For example, if there are only requirements that prioritize cheapness, the cloud will be filled, and if there are only requirements that prioritize speed, the edges will be filled, and after filling, it will be necessary to deploy to other servers. Therefore, in order to ease the first-come-first-served placement, it is necessary to reconfigure the placement

$$S = \sum_{k \in App} \left(\frac{R_k^{after}}{R_k^{before}} + \frac{P_k^{after}}{P_k^{before}} \right) \quad (1)$$

$$\sum_{i \in Device} (A_{i,k}^d \cdot B_{i,k}^p) + \sum_{j \in Link} (A_{j,k}^l \cdot \frac{C_k}{B_k^l}) = R_k^{after} \le R_k^{upper} \quad (2)$$

$$\sum_{i \in Device} a_i \left(\frac{A_{i,k}^d \cdot B_k^d}{C_i^d} \right) + \sum_{j \in Link} b_j \left(\frac{A_{j,k}^l \cdot B_k^l}{C_j^l} \right) = P_k^{after} \le P_k^{upper} \quad (3)$$

$$\sum_{k \in App} (A_{i,k}^d \cdot B_k^d) \le C_i^d \quad (4)$$

$$\sum_{k \in App} (A_{j,k}^l \cdot B_k^l) \le C_j^l \quad (5)$$

a_i: Device usage cost
b_j: Link usage cost
C_i^d: Device calculation resource limit of #i
C_j^l: Link bandwidth limit of #j
C_k: Data size of #k application
$A_{i,k}^d$: Whether to use of #k application on #i device
$A_{j,k}^l$: Whether to use of #k application on #j link
B_k^d: Calculation resource of #k application
B_k^l: Bandwidth usage of #k application
$B_{i,k}^p$: Processing time of #k application on #i device

not only before the start of operation but also after the start of operation.

3.3 Reconfiguration method

In this subsection, I propose a reconfiguration method for reconfigure the application to an appropriate placement location taking into account the placement status of other users, including the formulation of the linear programming method.

In the reconfiguration, for placement of a certain number of applications (for example, 100 applications and so on), the placement and reconfiguration of a certain number of applications (for example, 100 applications and so on) is calculated on a trial in a form that satisfies the initial requirements of multiple users. Through this process, the proposed method improves the total user satisfaction determined by changes in response time and price. Reconfiguration is actually performed only when the effect of reconfiguration is high, such as when the total satisfaction level exceeds a certain threshold as a result of trial calculation of reconfiguration. Since the actual reconfiguration requires a change in the application execution server, the proposed method uses a way such as live migration to suppress the user influence.

The linear programming equations for reconfiguration are shown in (1)-(5), and the parameters are shown in (1) is the objective function to evaluate total user satisfaction evaluation, (2)(3) are the constraint conditions of response time and price which are specified by the user request for each application deployment, and (4)(5) are the constraint conditions of server resource upper limit.

First, the new placement is arranged according to the equations (2)-(5). At the time of placement, the user can set requirements for response time or price or both. Response time Request R_k^{upper} specifies within seconds a response is required, and price request P_k^{upper} specifies how much or less a month. If only one of (2)(3) is specified, the minimization of the opposite is the objective function, and if both are specified, the objective function is specified by the user to minimize response time or price. (4)(5) are constraint conditions that set the upper limit of the calculation resource and communication band, and it is calculated including the application placed by other users, and prevents the resource upper limit excess due to the new application placement. New placement is performed by sequentially performing the calculations (2)-(5) in response to the user's placement request.

Next, the reconfiguration is considered. The reconfiguration is calculated according to (1)-(5), but in particular, the calculation of the value S which is related to total user satisfaction is newly added. Regarding to satisfaction of individual users, 1 point for response time and 1 point for price in the placement before reconfiguration. If the response time before reconfiguration R_k^{before} is multiplied by X to R_k^{after} after reconfiguration, X is the value related to response time satisfaction, and the price before reconfiguration P_k^{before} is multiplied by Y to P_k^{after} after reconfiguration, Y is the value related to price satisfaction . The objective function of the trial calculation of reconfiguration is the total user satisfaction of a certain number of applications to be reconfigured, and the placements that minimize the summation of (X + Y) for multiple applications is calculated. The specific content of the objective function is described in (1). Also, if the user has specified only one constraint condition at the time of new placement, only one of (2) or (3) is specified in that application.

Based on the linear programming equations (1)-(5), the effect of reconfiguration is calculated by deriving a solution using a linear programming solver such as GLPK or CPLEX (IBM Decision Optimization). The number of applications to be reconfigured is a fixed value and may not be all applications. The solver calculation time increases as the number of applications that calculate reconfiguration increases. Therefore, the setting of a certain number of application can be variable, and optimization of 100 applications for every 100 placements, optimization of 1000 applications or so on are determined by adjusting the size according to the solver calculation time.

4 EVALUATION

4.1 Evaluation conditions

4.1.1 *Applications to be simulated*

The placed applications are the Fourier transform and image processing, which are expected to be used by many users.

The Fast Fourier Transform (FFT) is used in various situations of monitoring in IoT such as analysis of vibration frequency. NAS.FT (NASA Fourier Transform) [38] is one of the open source applications for FFT processing. It calculates the 2,048 * 2,048 size of the built-in sample test.

MRI-Q (Magnetic Resonance Imaging - Q) [39] computes a matrix Q, representing the scanner configuration for calibration, used in 3D MRI reconstruction algorithms in non-Cartesian space. During application performance measurement, MRI-Q executes 3D MRI image processing to measure processing time using 64*64*64 size sample data.

Using the author's GPU and FPGA automatic offload technology [23][28], it was found that NAS.FT can be accelerated by GPU and MRI-Q can be accelerated by FPGA, and the performance is 5 times and 7 times that of CPU, respectively.

4.1.2 *Evaluation method*

The topology for arranging applications is composed of 3 layers as shown in Figure 3, with 5 sites in cloud layers, 20 sites in carrier edge layers, 60 sites in user edge layers, and 300 input nodes.

In terms of servers, there are 8 CPU servers, 4 GPU servers with 16GB RAM (Random Access Memory) and 2 FPGA servers in the cloud, 4 CPU servers, 2 GPU servers with 8GB RAM, and 1 FPGA server in the carrier edge, and 2 CPU servers and 1 GPU server with 4GB RAM in the user edge. When all resources of one server on the cloud will be used (when using 16GB RAM for GPU server), the monthly fee is 500, 1,000, and 1,200 USD. Due to the aggregation effect, the monthly fee is 1.25 and 1.5 times that of the cloud, assuming that the carrier edge and user edge will be expensive.

For links, a bandwidth of 100 Mbps is secured between the cloud and the carrier edge, and a bandwidth of 10 Mbps is secured between the carrier edge and the user edge. For the link cost, I set prices that 100Mbps link fee is 80 USD per month, 10Mbps link fee is 30 USD per month.

As the resource used by the application, the value when actually offloaded to GPU or FPGA is used for the processing time. NAS.FT uses GPU 1GB RAM, usage band 2Mbps, transfer data amount 0.2MB, and processing time 5.8 seconds. MRI-Q uses 10 % of the FPGA server (the number of Flip Flop and Look UP Table used is the FPGA resource usage), the usage band is 1 Mbps, the transfer data amount is 0.15 MB, and the processing time is 2.0 seconds.

First, 600 applications are initially placed. Random placement requests are generated from 300 input nodes. As the number of placement requests, the application placement request is made 600 times at a ratio of NAS.FT:MRI-Q = 3:1.

As a user request, a price condition or a response time condition or both is selected for each application when requesting placement. In the case of NAS.FT, the monthly upper limit of 75 (a), 85 (b) or 100 (c) USD (US Dollar) is selected for the price, and the 6 (A), 7 (B) or 10 (C) second upper limit is selected for the response time. In the case of MRI-Q, the monthly upper limit of 125 (x) or 200 (y) USD is selected for the price, and the 4 (X) or 8 (Y) second upper limit is selected for the response time. As a user request, in NAS.FT, a, b, c, A, B, C, aC, bB, bC, cA, cB, and cC are selected with a probability of 1/12 each, In MRI-Q, x, y, X, Y, xY, yX, yY are selected with a probability of 1/7 each. When the user upper limit request is one index, the minimization of another index is the objective function, and when there are two indexes, one is randomly selected and the minimization is the objective function.

Next, after placing 600 applications, the proposed method reconfigures a certain number of applications for

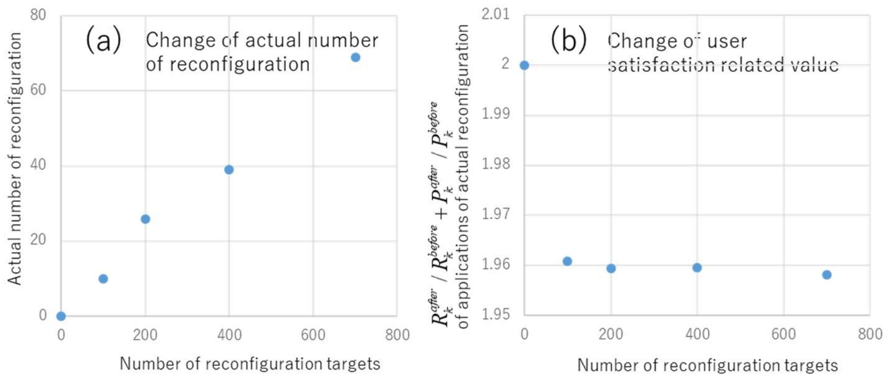

Figure 4: (a) Changes in the number of actual reconfiguration applications, (b) Changes in user satisfaction-related indicators for actual reconfiguration applications..

100 applications placements. In this experiment, the number of placements in the placement cycle is fixed at 100, but the number of reconfigured applications is variable to 100, 200, 400, 700 and total user satisfaction is calculated.

As a comparison of the proposed methods, user satisfaction is also obtained for the method of reconfiguration by randomizing the placement order of the applications to be placed in order to ease the first-come-first-served basis.

4.2 Results

The experiment was performed by simulation using solver GLPK5.0. Figure 4 (a) is a graph with the number of target applications to be reconfigured on the horizontal axis and the number of applications actually reconfigured on the vertical axis, and Figure 4 (b) is a graph with the number of target applications to be reconfigured on the horizontal axis and average value of actually reconfigured applications of R_k^{after}/R_k^{before}+P_k^{after}/P_k ^{before} on the vertical axis.

Regarding the calculation time, in the case of new placement, a total of 700 applications are calculated and placed in order, so it takes less than 2 minutes. On the other hand, the reconfiguration calculation time is less than 10 seconds for 100 applications, but less than 2 minutes for 700 applications, although the conditional expression of the linear programming increases as the number of target applications increases.

Regarding Fig. 4 (a), although there are some variations, it can be seen that about 10% of the number of target applications is actually reconfigured. In the new placement, the optimal placement is made individually for each application, but in the reconfiguration, it can be seen that reconfigurable applications can be found to some extent by calculating the appropriate placement for multiple applications at once.

Regarding Fig. 4 (b), the average of R_k^{after}/R_k^{before}+P_k^{after}/P_k^ {before} is about 1.96 for the reconfigured applications, and it has been improved. This value is not a big improvement from 2, because for example, when NAS.FT is relocated from the carrier edge to the cloud, the response time will be 7.4 seconds from 6.6 seconds, but the price will be about 70 USD from about 84 USD, then the value will be 1.954 from 2. From Fig. 4 (b), it can be seen that this value is almost constant regardless of the number of target applications, and it is not necessary to target all applications for reconfiguration. Experimental results show that relocating about 10% of deployed applications improves overall user satisfaction. Since the improvement in satisfaction is almost constant regardless of the number of applications for which relocation is calculated, it is sufficient to calculate the relocation certain fixed number of operational user application deployments.

5 CONCLUSIONS

In this paper, as a new element of the environment-adaptive software, I proposed a reconfiguration method that improves the satisfaction of the target users by relocating the application in consideration of the application placement status of other users after the start of operation.

The proposed method uses a linear programming method and performs trial calculation to maximize total satisfaction of the users of the reconfigured application as the objective function. Specifically, after satisfying the user's response time and price requirements, the user satisfaction determined from the response time and price at the time of reconfiguration is calculated. In the calculation, the linear programming solver GLPK is used to find the solution for multiple applications to be reconfigured.

I conducted simulation experiments assuming multiple types of applications, confirmed the improvement in user satisfaction when reconfiguring with the proposed method, and showed the effectiveness. Experiments have shown that relocating about 10% of deployed applications improves user satisfaction-related values from 2 to 1.96. In the future, I will consider expanding the scope of reconfiguration, such as resource balance reconfiguration of offload devices, as well as placement.

REFERENCES

[1] A. Putnam, A. M. Caulfield, E. S. Chung, D. Chiou, K. Constantinides, J. Demme, H. Esmaeilzadeh, J. Fowers, G. P. Gopal, J. Gray, M. Haselman, S. Hauck, S. Heil, A. Hormati, J.-Y. Kim, S. Lanka, J. Larus, E. Peterson, S. Pope, A. Smith, J. Thong, P. Y. Xiao and D. Burger, "A reconfigurable fabric for accelerating large-scale datacenter services," Proceedings of the 41th Annual International Symposium on Computer Architecture (ISCA'14), pp.13-24, June 2014.

[2] Amazon Web Services EC2 web site, https://aws.amazon.com/ec2/instance-types/

[3] O. Sefraoui, M. Aissaoui and M. Eleuldj, "OpenStack: toward an open-source solution for cloud computing," International Journal of Computer Applications, Vol.55, No.3, 2012.

[4] Y. Yamato, "Automatic system test technology of virtual machine software patch on IaaS cloud," IEEJ Transactions on Electrical and Electronic Engineering, Vol.10, Issue.S1, pp.165-167, Oct. 2015.

[5] Y. Yamato, "Server Structure Proposal and Automatic Verification Technology on IaaS Cloud of Plural Type Servers," International Conference on Internet Studies (NETs2015), July 2015.

[6] Y. Yamato, Y. Nishizawa and S. Nagao, "Fast Restoration Method of Virtual Resources on OpenStack," IEEE Consumer Communications and Networking Conference (CCNC2015), Las Vegas, pp.607-608, Jan. 2015.

[7] Y. Yamato, "Automatic verification for plural virtual machines patches," The 7th International Conference on Ubiquitous and Future Networks (ICUFN 2015), pp.837-838, Sapporo, July 2015.

[8] Y. Yamato, "Proposal of Optimum Application Deployment Technology for Heterogeneous IaaS Cloud," 2016 6th International Workshop on Computer Science and Engineering (WCSE 2016), pp.34-37, June 2016.

[9] Y. Yamato, Y. Nishizawa, S. Nagao and K. Sato, "Fast and Reliable Restoration Method of Virtual Resources on OpenStack," IEEE Transactions on Cloud Computing, DOI: 10.1109/TCC.2015.2481392, Sep. 2015.

[10] M. Hermann, T. Pentek and B. Otto, "Design Principles for Industrie 4.0 Scenarios," Rechnische Universitat Dortmund. 2015.

[11] Y. Yamato, "Proposal of Vital Data Analysis Platform using Wearable Sensor," 5th IIAE International Conference on Industrial Application Engineering 2017 (ICIAE2017), pp.138-143, Mar. 2017.

[12] Y. Yamato and M. Takemoto, "Method of Service Template Generation on a Service Coordination Framework," 2nd International Symposium on Ubiquitous Computing Systems (UCS 2004), Nov. 2004.

[13] Y. Yamato, Y. Fukumoto and H. Kumazaki, "Proposal of Real Time Predictive Maintenance Platform with 3D Printer for Business Vehicles," International Journal of Information and Electronics Engineering, Vol.6, No.5, pp.289-293, Sep. 2016.

[14] Y. Yamato, Y. Fukumoto and H. Kumazaki, "Security Camera Movie and ERP Data Matching System to Prevent Theft," IEEE Consumer Communications and Networking Conference (CCNC 2017), pp.1021-1022, Jan. 2017.

[15] Y. Yamato, Y. Fukumoto and H. Kumazaki, "Proposal of Shoplifting Prevention Service Using Image Analysis and ERP Check," IEEJ

Transactions on Electrical and Electronic Engineering, Vol.12, Issue.S1, pp.141-145, June 2017.

[16] Y. Yamato, Y. Fukumoto and H. Kumazaki, "Analyzing Machine Noise for Real Time Maintenance," 2016 8th International Conference on Graphic and Image Processing (ICGIP 2016), Oct. 2016.

[17] Y. Yamato, "Experiments of posture estimation on vehicles using wearable acceleration sensors," The 3rd IEEE International Conference on Big Data Security on Cloud (BigDataSecurity 2017), pp.14-17, May 2017.

[18] P. C. Evans and M. Annunziata, "Industrial Internet: Pushing the Boundaries of Minds and Machines," Technical report of General Electric (GE), Nov. 2012.

[19] T. Sterling, M. Anderson and M. Brodowicz, "High performance computing : modern systems and practices," Cambridge, MA : Morgan Kaufmann, ISBN 9780124202153, 2018.

[20] J. E. Stone, D. Gohara and G. Shi, "OpenCL: A parallel programming standard for heterogeneous computing systems," Computing in science & engineering, Vol.12, No.3, pp.66-73, 2010.

[21] J. Sanders and E. Kandrot, "CUDA by example : an introduction to general-purpose GPU programming," Addison-Wesley, 2011.

[22] J. Gosling, B. Joy and G. Steele, "The Java language specification, third edition," Addison-Wesley, 2005. ISBN 0-321-24678-0.

[23] Y. Yamato, T. Demizu, H. Noguchi and M. Kataoka, "Automatic GPU Offloading Technology for Open IoT Environment," IEEE Internet of Things Journal, DOI: 10.1109/JIOT.2018.2872545, Sep. 2018.

[24] Y. Yamato, "Study and Evaluation of Automatic GPU Offloading Method from Various Language Applications," International Journal of Parallel, Emergent and Distributed Systems, Taylor and Francis, DOI: 10.1080/17445760.2021.1971666, Sep. 2021.

[25] Y. Yamato, "Study and Evaluation of Improved Automatic GPU Offloading Method," International Journal of Parallel, Emergent and Distributed Systems, Taylor and Francis, DOI: 10.1080/17445760.2021.1941010, June 2021.

[26] Y. Yamato, "Improvement Proposal of Automatic GPU Offloading Technology," The 8th International Conference on Information and Education Technology (ICIET 2020), pp.242-246, Mar. 2020.

[27] Y. Yamato, "Proposal of Automatic Offloading for Function Blocks of Applications," The 8th IIAE International Conference on Industrial Application Engineering 2020 (ICIAE 2020), pp.4-11, Mar. 2020.

[28] Y. Yamato, "Study of parallel processing area extraction and data transfer number reduction for automatic GPU offloading of IoT applications," Journal of Intelligent Information Systems, Springer, DOI:10.1007/s10844-019-00575-8, 2019.

[29] J. Fung and M. Steve, "Computer vision signal processing on graphics processing units," 2004 IEEE International Conference on Acoustics, Speech, and Signal Processing, Vol. 5, pp.93-96, 2004.

[30] Xilinx SDK web site, https://japan.xilinx.com/html_docs/xilinx2017_4/sdaccel_doc/lyx1504034296578.html

[31] E. Su, X. Tian, M. Girkar, G. Haab, S. Shah and P. Petersen, "Compiler support of the workqueuing execution model for Intel SMP architectures," In Fourth European Workshop on OpenMP, Sep. 2002.

[32] S. Wienke, P. Springer, C. Terboven and D. an Mey, "OpenACC-first experiences with real-world applications," Euro-Par 2012 Parallel Processing, pp.859-870, 2012.

[33] M. Wolfe, "Implementing the PGI accelerator model," ACM the 3rd Workshop on General-Purpose Computation on Graphics Processing Units, pp.43-50, Mar. 2010.

[34] C. C. Wang, Y. D. Lin, J. J. Wu, P. C. Lin and R. H. Hwang, "Toward optimal resource allocation of virtualized network functions for hierarchical datacenters," IEEE Transactions on Network and Service Management, Vol.15, No.4, pp.1532-1544, 2018.

[35] Clang website, http://llvm.org/

[36] J. H. Holland, "Genetic algorithms," Scientific american, Vol.267, No.1, pp.66-73, 1992.

[37] NVIDIA vGPU software web site, https://docs.nvidia.com/grid/index.html

[38] NAS.FT website, https://www.nas.nasa.gov/publications/npb.html

[39] MRI-Q website, http://impact.crhc.illinois.edu/parboil/