Study of software reconfiguration after adapted service start

YOJI YAMATO*

*Network Service Systems Laboratories, NTT Corporation, Japan

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

We have proposed environment-adaptive software that automatically converts normal code in accordance with the environment and enables high-performance operation. In this paper, we study the overall processing performed for reconfiguration during operation, and newly propose resource amount reconfiguration, measure the processing time for reconfiguration during operation, and discuss reconfiguration timings.

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

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

1 INTRODUCTION

Moore's law is expected to slow down in the coming years. In response to this, the use of devices such as FPGAs (Field Programmable Gate Arrays) and GPUs (Graphics Processing Units) is increasing in addition to CPUs (Central Processing Units). For example, Microsoft uses FPGAs to improve search efficiency [1], and Amazon provides FPGA and GPU instances [2] with cloud technology (for example, [3]-[6]). Also, the use of IoT (Internet of Things) devices (for example, [7]-[19]) is increasing.

However, to properly utilize devices other than CPUs, programs need to be created that take device characteristics into consideration. Knowledge of OpenMP (Open Multi-Processing) [20], OpenCL (Open Computing Language) [21], CUDA (Compute Unified Device Architecture) [22], or embedded technology is required, so the skill barrier is high. [23]-[31] are other reconfiguration works. Therefore, we proposed the concept of environment-adaptive software that automatically converts and arranges the code once written so that it can use FPGAs and GPUs in the deployment environment, and it runs the application with high performance. At the same time, we have proposed and evaluated a method for automatically offloading application loop statements to FPGAs and GPUs, and a method for optimizing the amount and placement of

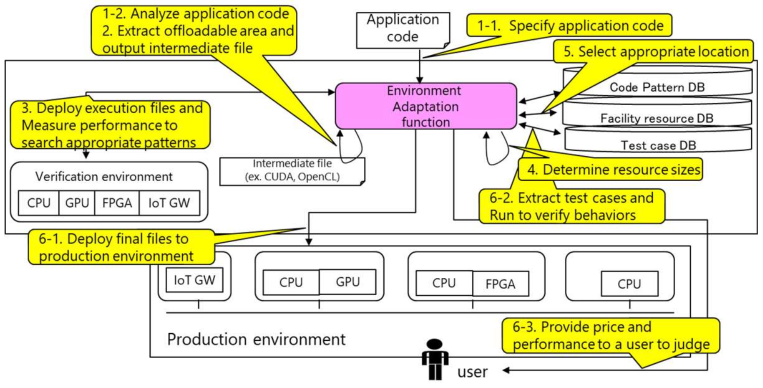

1-2. Analyze application code
2. Extract offloadable area and output intermediate file
Application code
1-1. Specify application code
5. Select appropriate location
Environment Adaptation function
Code Pattern DB
Facility resource DB
Test case DB
3. Deploy execution files and Measure performance to search 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
Production environment
6-3. Provide price and performance to a user to judge
user

Figure 1: Processing flow of environment-adaptive software.

application resources [32]-[36] In addition, after the start of operation, we propose and evaluate FPGA, GPU, and placement reconfiguration methods during operation that change the configuration based on actual usage.

In this paper, we examine the overall processing performed for reconfiguration during operation, which has been examined individually so far. As part of this, we newly consider reconfiguring the amount of resources that had not been considered. In addition, we measure the processing time of reconfiguration during operation and examine the execution timing.

2 RECONFIGURATION DURING OPERATION

2.1 Previous proposal

We previously proposed the processing flow shown in Fig. 1 for realizing software environment adaptation. The environment-adaptive software operates in cooperation with the verification environment, production environment, test case DB (database), code pattern DB, and facility resource DB, centering on the environment adaptation function.

Here, Steps 1-6 perform code conversion, resource amount setting, allocation location setting, and operation check, which are necessary before the start of operation. In Step 7, we reconfigure software when it is better to change the configuration.

2.2 Overall processing of reconfiguration during operation

Reconfiguration during operation is performed in Step 7, here, the purpose of reconfiguration during operation is clarified. Before starting to use the environment-adaptive software, optimization is performed in accordance with the test pattern assumed by the user, and code conversion, setting of resource amount, and setting of placements are performed on the basis of on the assumed test case. However, after the actual operation starts, the request trend may possibly not be the one that was assumed in advance, and that the usage trend will be unexpected. For example, the operation started with FPGA logic that accelerates SQL processing, but after half a year, NoSQL queries became mainstream. To maintain high-speed performance by reconfiguring to a more appropriate configuration even in such cases, reconfiguration during operation is automatically performed by analyzing actual usage data. In the reconfiguration process during operation, the environment adaptation process, which is optimized before the start of operation, is appropriately performed on the actual usage, not on the assumed data in advance. The environment adaptation processing to be performed is FPGA logic reconfiguration, GPU logic reconfiguration, resource amount reconfiguration, and placement reconfiguration. All of these may be performed collectively or each step may be performed independently. These may be performed in accordance with the type of business form.

2.3 Each reconfiguration process during operation

2.3.1 *FPGA or GPU logic reconfiguration*

Regarding FPGA logic reconfiguration and GPU logic reconfiguration, we have proposed and evaluated the details in previous papers. Although the logic is reconfigured in the following six stages, the general concept of reconfiguration is the same for FPGAs and GPUs.

1. The system analyzes production request data history for a certain period of time, identifies multiple applications with the highest processing time load, and obtains the most frequent data size data when using those applications.
2. The system extracts offload patterns that speed up test cases for the most frequent data in multiple high-load applications through verification environment measurement trials.
3. The system measures the processing time of the current offload pattern and multiple extracted new offload patterns and obtains the performance improvement effect on the basis of the frequency of production use.
4. The system judges a reconfiguration proposal on the basis of whether the performance improvement effect of the new offload pattern is greater than the threshold of that of the current offload pattern.
5. The system proposes the execution of FPGA/GPU reconfiguration to the user and obtains an OK/Not OK response.
6. The system performs the static reconfiguration by starting another OpenCL/OpenACC (Open ACCelerators) [37] in a production environment.

2.3.2 *Resource amount reconfiguration*

Regarding the resource amount setting before the start of operation, we have proposed and evaluated the details in previous papers, but we will newly consider the reconfiguration of the resource amount in this paper.

Before starting operation, when determining the appropriate resource ratio between CPU and offload device, refer to [38] or so on to avoid any device processing becoming a bottleneck. Specifically, on the basis of the processing time of the assumed test case, the resource ratio is determined so that the processing time of the CPU and the offload device are of the same order. Next, we deploy the application to the production environment. When deploying the application to the production environment, we determine the amount of resources by maintaining the resource ratio as much as possible so as to meet the cost requirements specified by the user.

After the start of operation, the most frequent data when using the application is acquired instead of the pre-assumed test cases, and the appropriate resource ratio is calculated using the test cases with the most frequent data. Since a large price increase after the start of operation is difficult for users to tolerate, when determining the amount of resources, reconfiguration is proposed only when the price is the same as or lower than the initial price.

2.3.3 *Placement reconfiguration*

We have proposed and evaluated the details of the placement reconfiguration in previous papers. Regarding the placement of offload applications, before the start of operation (refer to [39] or so on), we can place them appropriately in accordance with the placement status of other applications. However, after the start of operation, the placement of other applications is increasing, so the overall placement needs to be optimized considering their placements.

The placement reconfiguration is calculated and determined by a linear programming solver such as GLPK [40] in accordance with formulas (a)-(c), and formula (a) for the value S relating to users satisfaction is newly added before the operation start. Regarding user satisfaction, if the pre-reconfiguration response time R_k^{before} becomes X times after reconfiguration R_k^{after}, then X is the relevant value, and the pre-reconfiguration price P_k^{before} becomes Y times after reconfiguration P_k^{after}, then P is the relevant value. The objective function of re-placement calculation is a value related to the satisfaction of all users to be reconfigured, and the placement that minimizes the sum of X+Y for all applications is calculated to find the overall optimum placement.

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

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

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

3 PROCESSING TIME MEASUREMENT

So far, only individual reconfigurations have been evaluated, but in this paper, we implement all the reconfiguration processing performed by the environment-adaptive software, measure the processing time of each processing, and discuss the execution timing.

3.1 Measurement conditions

In the FPGA reconfiguration, the signal processing tdFIR [41] is offloaded to the FPGA by the existing method [33] before the start of operation, and the image processing MRI-Q [42] running on the CPU was included with operation. We apply a request load for a certain period of time and confirm a reconfiguration proposal to the new offload pattern that has a high performance improvement effect.

FPGA offload target loop statements: tdFIR 6, MRI-Q 16

Arithmetic intensity narrowing: Narrowing to top 4 loop statements for arithmetic intensity analysis

Resource efficiency narrowing: Narrowing to top 3 loop statements in resource efficiency analysis

Number of measured offload patterns: 4 (The first time measured the top three loop statement offloads, and the second time measured the combination pattern of the two loop statement offloads that performed well in the first time.)

Request load: tdFIR 200 req/h, MRI-Q 10 req/h load with three data sizes. tdFIR requests 162 KB, 2.06 MB, and 33.0 MB sample data with a ratio of 75:120:5. MRI-Q requests 32*32*32, 64*64*64, double size of 64*64*64 sample data (64*64*64 copied and added) with a ratio of 3:5:2.

Load analysis time: 1 hour

Number of load top applications: 2

Performance improvement effect threshold: 2

In the GPU reconfiguration, the Fourier transform NAS.FT [43] was offloaded to the GPU by an existing method [36] before the start of operation, and the incompressible fluid analysis Himeno benchmark [44] running on the CPU was included with the operation. We apply a request load for a certain period of time and confirm a reconfiguration proposal to the new offload pattern that has a high performance improvement effect.

GPU offload target loop statements: NAS.FT 81, Himeno Benchmark 13

Number of individuals M: NAS.FT 30, Himeno Benchmark 10

Number of generations T: NAS.FT 30, Himeno Benchmark 10

Goodness of fit: (processing time)^{-1/2}

The shorter the processing time, the higher the goodness of fit. By using the (-1/2) power, it is possible to prevent the search range from narrowing due to too high matching of a specific individual whose processing time is very short.

Selection: Roulette Selection. However, elite preservation, which preserves the highest goodness of fit gene in the next generation without crossover or mutation, is also performed.

Crossover rate Pc: 0.9

Mutation rate Pm: 0.05

Request load: NAS.FT 20 req/h, Himeno Benchmark 30 req/h load with 3 data sizes. In NAS.FT, the sample data sizes of classes W, A, and B are requested in a ratio of 3:5:2. In the Himeno benchmark, the sample data sizes of M, L, and XL are requested in a ratio of 2:5:3.

Load analysis time: 1 hour

Number of load top applications: 2

Performance improvement effect threshold: 2

In resource amount reconfiguration, NAS.FT and Himeno Benchmark are used to determine the amount of resources on the basis of the initially assumed data size, and after starting operation, it is confirmed that the resource amount is reconfigured when the actual data size differs significantly from the expected size.

The initial assumed data size is class W for NAS.FT and size XL for Himeno Benchmark. NVIDIA vGPU [45] is used for dividing and reallocating GPU resources. Resources can be allocated in units of 1 Core for CPU and 4 GB RAM for GPU, and the cost ratio of CPU 1 Core: GPU 4 GB RAM is 1:2.5.

In placement reconfiguration, we simulate overall placement optimization that improves the average satisfaction of multiple users when NAS.FT is offloaded to GPU and MRI-Q is offloaded to FPGA. When 300 applications are placed in order and then 100 new applications are placed, the optimal placement is calculated for 100, 200, and 400 applications for re-placement calculation. The simulation conditions are as follows.

The deployment topology consists of 4 layers, with 5 locations of cloud layer, 20 locations of carrier edge layers, 60 locations of user edge layers, and 300 input nodes.

In the cloud, there are 8 CPUs, 4 GPUs with 16 GB RAM, and 2 FPGAs servers. In the carrier edge, there are 4 CPUs, 2 GPUs with 8 GB RAM, and 1 FPGA servers. In the user edge, there are 2 CPUs and 1 GPU with 4 GB RAM servers. The monthly charges for all resources used by one server were 50,000, 100,000, and 120,000 yen in the cloud, respectively. Carrier edge and user edge are relatively expensive, and we set them at 1.25 and 1.5 times that of the cloud. The link bandwidth is 100 Mbps between cloud and carrier edge, and 10 Mbps between carrier edge and user edge. The link cost was 8,000 yen/month for a 100 Mbps link and 3,000 yen/month for a 10 Mbps link.

As for the resources used by the applications, NAS.FT uses GPU 1GB RAM, bandwidth 2Mbps, transfer data amount 0.2 MB, and processing time 5.8 seconds. MRI-Q uses 10% of FPGA server resources, bandwidth 1 Mbps, transfer data amount 0.15 MB, and processing time 2.0 seconds.

Applications placement requests are generated randomly from 300 input nodes to upper nodes. As for the number of placement requests, the ratio of NAS.FT: MRI-Q = 3:1 makes 300 requests for the initial placement of the application. As user requirements, price, response time, or both are selected when requesting placement. In the case of NAS.FT, an upper limit price of 7,500 yen (a), 8,500 yen (b), or 10,000 yen (c) per month is selected, and an upper limit response time of 6 seconds (A), 7 seconds (B), or 10 seconds (C) is selected. In the case of MRI-Q, the upper limit of price is 12,500 yen (x) or 20,000 yen (y) per month, an upper limit response time of 4 seconds (X) or 8 seconds (Y) is selected. For NAS.FT, a, b, c, A, B, C, aC, bB, bC, cA, cB, cC are set to 1/12 each, and for MRI-Q, x, y, X, Y, xY, yX, yY are set to 1/7 each.

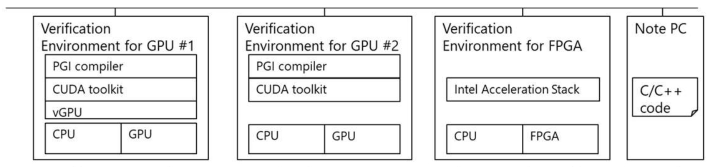
3.2 Verification environments

Figure 2 shows the measurement environment. For FPGA reconfiguration, Intel FPGA PAC D5,005 is used and controlled by Intel Acceleration Stack 2.0 [46]. For GPU reconfiguration, NVIDIA GeForce RTX 2,080 Ti is used and controlled by PGI Compiler 19.10 [47]. In the resource amount reconfiguration, the GPU resources of NVIDIA Tesla T4 are divided and used by NVIDIA vGPU 12.2 [45]. For placement reconfiguration, simulation is performed with GLPK 5.0 [40].

3.3 Results

The improvement results and processing time by reconfiguration of FPGA reconfiguration, GPU reconfiguration, resource amount reconfiguration, and placement reconfiguration are shown.

Figure 3(a) shows the improvement in FPGA offload processing time before and after the reconfiguration proposal and the total processing time for a certain period related to it. Before the reconfiguration, tdFIR was offloaded, and the total request processing time of 79.7 seconds was 1 hour of load. As a result of the analysis, tdFIR and MRI-Q are the two applications with the highest load. By searching for offload patterns by using the most frequent data after the start of operation and multiplying the number of production uses, the degree of improvement in processing time reduction is 41.1 seconds/hour for tdFIR before reconfiguration and 252 seconds/hour for MRI-Q after reconfiguration. From Fig. 3(a), a reconfiguration is proposed that checks the improvement threshold of 2.0 and changes the offload application from tdFIR to MRI-Q. Since the compilation time of the application is dominant, the processing time depends on it. For both applications, 1 compilation takes about 6 hours, so it takes about 1 day to measure 4 patterns.



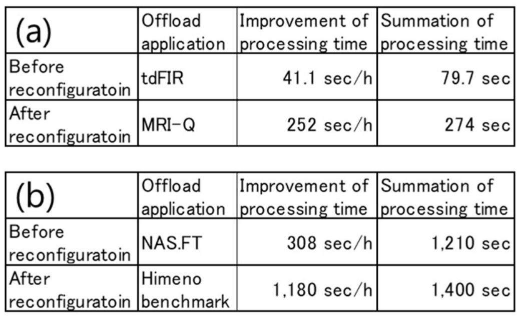
Name	Hardware	CPU	RAM	GPU/FPGA	OS	NVIDIA vGPU	CUDA toolkit	PGI compiler	Intel Acc
Verification Environment for GPU #1	Supermicro SYS-1029GP-TR	Intel Xeon Silver 4210R (10 core, 2.4GHz)	128 GB	NVIDIA Tesla T4 *2 (CUDA core: 2560, Memory:GDDR6 16GB)	RedHat Enterprise Linux 7.9	12.2	10.1	19.10	
Verification Environment for GPU #2	LEVEL-F039-LCRT2W-XYVI	AMD Ryzen Threadripper 2990WX	64 GB	NVIDIA GeForce 2080 (CUDA core: 4352, Memory:GDDR6 11GB)	Ubuntu 16.04.6 LTS		10.1	19.10	
Verification Environment for FPGA	Dell PowerEdge R740	Intel(R) Xeon Bronze 3206R *2	128 GB	Intel PAC D5005 (Intel Stratix 10 GX FPGA)	CentOS 7.9				2.0
Client	HP ProBook 470 G3	Intel Core i5-6200U	8GB		Windows 10 Pro				

Figure 2: Verification environments.

Figure 3(b) shows the degree of improvement in GPU offload processing time before and after the reconfiguration proposal and the total processing time for a certain period related to it. Before the reconfiguration, NAS.FT was offloaded, and the total request processing time of 1,210 seconds was one hour of load. As a result of the analysis, Himeno Benchmark and NAS.FT are the two applications with the highest load. By searching for offload patterns using the most frequent data after the start of operation and multiplying the number of production uses, the degree of improvement in processing time reduction was 308 seconds/hour for NAS.FT before reconfiguration, and 1,180 seconds/hour for the Himeno benchmark after reconfiguration. From Fig. 3(b), a reconfiguration is proposed that checks the improvement threshold of 2.0 and changes the offload application from NAS.FT to Himeno Benchmark. The processing time depends on the size of the application, such as the number of for statements, but in the case of NAS.FT, the offload pattern search after reconfiguration takes about 6 hours.

Figure 4(a) shows the set resource amount, cost, and cost-effectiveness before and after reconfiguration. After optimizing NAS.FT with data size W and Himeno Benchmark with data size XL and starting operation, the most frequent data after the start of operation are B and M, respectively. In NAS.FT, the data size has increased from W to B, and the amount of calculation has increased, so it is calculated that it is better to increase the GPU resource relative to the CPU. It was thought that increasing the GPU to 16 GB RAM would be more cost-effective, but reconfiguration is not proposed as it would increase the price significantly. On the other hand, in the Himeno benchmark, the data size decreased from XL to M, and the amount of calculation decreased, so it is calculated that GPU resources can be reduced relative to CPU. A proposal was made to reduce the GPU to 8 GB RAM, increasing the cost-effectiveness by a factor of 1.5. Proposals for reconfiguring resource amount are made within seconds.

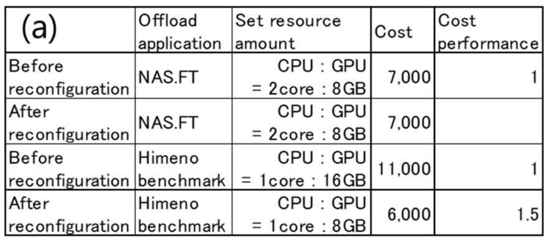
Figure 4(b) shows average value of R_k^{after}/R_k^{before}+P_k^{after}/P_k^{before} of the re-placed applications on the vertical axis in the reconfiguration simulation. Although there are some variations, about 10% or less of the number of applications to be calculated for re-placement are actually reconfigured. From Fig. 4(b), the average of R_k^{after}/R_k^{before}+P_k^{after}/P_k^{before} is about 1.96 for the re-placed applications. This value is not greatly improved from 2. For example, when NAS.FT is re-placed from the carrier edge to the cloud, the response time drops from 6.6 to 7.4 seconds, but the price drops 7,000 yen from 8,400 yen, and the value will drop from 2 to 1.954. As the number of applications to be calculated increased, the number of conditional expressions for linear programming increased, but even 400 applications were completed within 1 minute.



(a)	Offload application	Improvement of processing time	Summation of processing time
Before reconfiguratoin	tdFIR	41.1 sec/h	79.7 sec
After reconfiguratoin	MRI-Q	252 sec/h	274 sec

(b)	Offload application	Improvement of processing time	Summation of processing time
Before reconfiguratoin	NAS.FT	308 sec/h	1,210 sec
After reconfiguratoin	Himeno benchmark	1,180 sec/h	1,400 sec

Figure 3: a) FPGA logic reconfiguration results. (b) GPU logic reconfiguration results.



(a)	Offload application	Set resource amount	Cost	Cost performance
Before reconfiguration	NAS.FT	CPU : GPU = 2core : 8GB	7,000	1
After reconfiguration	NAS.FT	CPU : GPU = 2core : 8GB	7,000	
Before reconfiguration	Himeno benchmark	CPU : GPU = 1core : 16GB	11,000	1
After reconfiguration	Himeno benchmark	CPU : GPU = 1core : 8GB	6,000	1.5

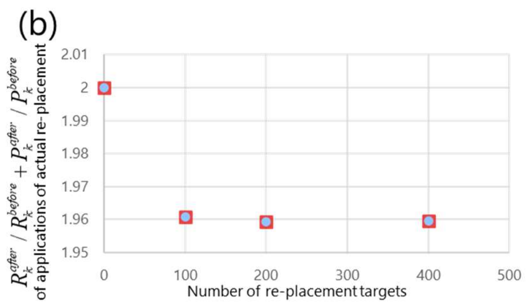


Figure 4: (a) Resource amount reconfiguration results. (b) Applications re-placement results.

3.4 Discussion

The reconfiguration of FPGA, GPU, and placement is evaluated in other papers, but the newly proposed resource amount reconfiguration is also highly user-friendly. It analyzes data size trends and proposes cost-effective reconfiguration when prices are similar or lower.

The processing time for each reconfiguration during operation depends on the application, but in this measurement, FPGA reconfiguration takes 1 day, GPU reconfiguration takes 6 hours, resource amount reconfiguration takes several seconds, and placement reconfiguration takes 1 minute for 400 applications. In the case of FPGA, it takes about 6 hours to compile an application, so even if the number of verification environment measurements is 4, it takes about 1 day for 1 application. In the case of GPU, genetic algorithm [48] is used to search for reconfiguration patterns, so it takes about six hours for one application because it performs many measurements with multiple generations and multiple individuals. Reconfiguration of the resource amount involves selecting the most frequent data and calculating the resource ratio and amount in accordance with the processing time at that time, which takes several seconds. Placement depends on the solver's linear programming calculation time. It takes less than 10 seconds for 100 applications, but it takes 1 minute for 400 applications, and it takes longer as the number of applications increases.

Frequent changes are not desirable because changes in resource amount and placement affect prices. Therefore, the reconfiguration trial is assumed to be about once every one to several months. However, at the start of operation, all adaptive processing is performed before operation is started, but reconfiguration during operation may be performed independently for each process. For example, FPGA, GPU logic, and resource amount reconfiguration may be done once every three months, and placement reconfiguration may be done at fixed increments, such as every 100 applications, depending on the type of business form.

4 CONCLUSIONS

In this paper, the overall processing performed for the reconfiguration during operation of the important elements of environment-adaptive software is examined. As part of this, we have added a new method for reconfiguring the resources sizes. We measured each processing time of reconfiguration during operation.

For FPGA and GPU logic, we find appropriate patterns by periodically performing optimization trials in a verification environment using the most frequent data after the start of operation. For the amount of resources, we find the appropriate resource ratio and amount from the CPU and offload device processing time for the most frequent data. For the placement, we find an appropriate placement by calculating the global optimization with the response time and the price condition by using a linear programming method. We implemented all the reconfiguration processes and measured the processing time. It took 1 day for FPGA reconfiguration, 6 hours for GPU reconfiguration, several seconds for resource amount reconfiguration, and 1 minute for placement reconfiguration with 400 applications.

In the future, we will discuss the commercial implementation timing of each reconfiguration process in accordance with the confirmed processing time of reconfiguration during operation.

REFERENCES

[1] A. Putnam, et al., "A reconfigurable fabric for accelerating large-scale datacenter services," ISCA'14, pp.13-24, June 2014.

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

[3] O. Sefraoui, et al., "OpenStack: toward an open-source solution for cloud computing," Int. J. Comput. Appl., Vol.55, No.3, 2012.

[4] Y. Yamato, "Automatic Verification for Plural Virtual Machines Patches," ICUFN 2015, pp.837-838, 2015.

[5] Y. Yamato, et al., "Fast and Reliable Restoration Method of Virtual Resources on OpenStack," IEEE Transactions on Cloud Computing, DOI: 10.1109/TCC.2015.2481392, Sep. 2015.

[6] Y. Yamato, et al., "Fast Restoration Method of Virtual Resources on OpenStack," IEEE CCNC 2015, pp.607-608, Jan. 2015.

[7] M. Hermann, et al., "Design Principles for Industrie 4.0 Scenarios," Rechnische Universitat Dortmund. 2015.

[8] H. Noguchi, et al., "Autonomous Device Identification Architecture for Internet of Things," IEEE WF-IoT 2018, pp.407-411, Feb. 2018.

[9] Y. Yamato, et al., "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, 2016.

[10] Y. Yokohata, et al., "Service Composition Architecture for Programmability and Flexibility in Ubiquitous Communication Networks," IEEE SAINT 2006, pp.142-145, 2006.

[11] Y. Yamato, et al., "Development of Service Control Server for Web-Telecom Coordination Service," IEEE ICWS 2008, 2008.

[12] Y. Yokohata, et al., "Context-Aware Content-Provision Service for Shopping Malls Based on Ubiquitous Service-Oriented Network Framework and Authentication and Access Control Agent Framework," IEEE CCNC 2006, pp.1330-1331, 2006.

[13] M. Takemoto, et al., "Service Elements and Service Templates for Adaptive Service Composition in a Ubiquitous Computing Environment," The 9th Asia-Pacific Conference on Communications (APCC 2003), Vol.1, pp. 335-338, 2003.

[14] Y. Yamato, et al., "Study and Evaluation of Context-Aware Service Composition and Change-over Using BPEL Engine and Semantic Web Techniques," IEEE CCNC 2008, pp.863-867, 2008.

[15] Y. Yamato, et al., "Method of Service Template Generation on a Service Coordination Framework," UCS 2004, Nov. 2004.

[16] Y. Yamato, et al., "Proposal of Lambda Architecture Adoption for Real Time Predictive Maintenance," The Fourth International Symposium on Computing and Networking Workshops (CANDARW 2016), pp.713-715, Nov. 2016.

[17] H. Noguchi, et al., "Device Identification Based on Communication Analysis for the Internet of Things," IEEE Access, DOI: 10.1109/ACCESS.2019.2910848, Apr. 2019.

[18] H. Noguchi, et al., "Distributed Search Architecture for Object Tracking in the Internet of Things," IEEE Access, DOI: 10.1109/ACCESS.2018.2875734, Oct. 2018.

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

[20] T. Sterling, et al., "High performance computing : modern systems and practices," Cambridge, MA : Morgan Kaufmann, ISBN 9780124202153, 2018.

[21] J. E. Stone, et al., "OpenCL: A parallel programming standard for heterogeneous computing systems," Computing in science and engineering, Vol.12, No.3, pp.66-73, 2010.

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

[23] Iman Attarzadeh and Siew Hock Ow, "Software Development Effort Estimation Based on a New Fuzzy Logic Model," International Journal of Computer Theory and Engineering vol. 1, no. 4, pp. 473-476, 2009.

[24] Anggy Trisnadoli and Indah Lestari , "Analysis of Software Quality Requirements for Mobile Learning Application for High School," International Journal of Computer Theory and Engineering vol. 10, no. 3, pp. 73-76, 2018.

[25] Shweta Sharma and S. Srinivasan, "A Survey on Software Design Based and Project Based Metrics," International Journal of Computer Theory and Engineering vol. 14, no. 2, pp. 54-61, 2022.

[26] Tanveer Hassan, Chaudhary Wali Mohammad, and Mohd. Sadiq, "Using Social Network and Fuzzy Set Theory for Elicitation and Prioritization of Software Requirements ," International Journal of Computer Theory and Engineering vol. 14, no. 3, pp. 126-134, 2022.

[27] Nedhal A. Al-Saiyd, "The Impact of Reusing Open-Source Software Model in Software Maintenance," International Journal of Computer Theory and Engineering vol. 9, no. 1, pp. 6-10, 2017.

[28] S. Keshavarz and Reza Javidan, "Software Quality Control Based on Genetic Algorithm," International Journal of Computer Theory and Engineering vol. 3, no. 4, pp. 579-584, 2011.

[29] O. Moravcik, D. Petrik, T. Skripcak, and P. Schreiber, "Elements of the Modern Application Software Development," International Journal of Computer Theory and Engineering vol. 4, no. 6, pp. 891-896, 2012.

[30] Sushil Garg, K. S. Kahlon, and P. K. Bansal, "Testability Analysis of Aspect Oriented Software," International Journal of Computer Theory and Engineering vol. 2, no. 1, pp. 119-124, 2010.

[31] Jaswinder Kaur, Satwinder Singh, Karanjeet Singh Kahlon, and Pourush Bassi, "Neural Network-A Novel Technique for Software Effort Estimation," International Journal of Computer Theory and Engineering vol. 2, no. 1, pp. 17-19, 2010.

[32] 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.

[33] Y. Yamato, "Automatic Offloading Method of Loop Statements of Software to FPGA," International Journal of Parallel, Emergent and Distributed Systems, Taylor and Francis, DOI: 10.1080/17445760.2021.1916020, Apr. 2021.

[34] Y. Yamato, "Improvement Proposal of Automatic GPU Offloading Technology," ICIET 2020, pp.242-246, Mar. 2020.

[35] Y. Yamato, "Proposal of Automatic Offloading for Function Blocks of Applications," ICIAE 2020, pp.4-11, Mar. 2020.

[36] 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.

[37] S. Wienke, et al., "OpenACC-first experiences with real-world applications," Euro-Par 2012 Parallel Processing, pp.859-870, 2012.

[38] K. Shirahata, et al., "Hybrid Map Task Scheduling for GPU-Based Heterogeneous Clusters," IEEE CloudCom 2010, pp.733-740, 2010.

[39] C. C. Wang, et al., "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.

[40] GLPK website, https://sites.google.com/site/nssvdabb/glpk

[41] Time domain finite impulse response filter web site, http://www.omgwiki.org/hpec/files/hpec-challenge/tdfir.html

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

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

[44] Himeno benchmark web site, http://accc.riken.jp/en/supercom/

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

[46] Intel Acceleration Stack website, https://www.intel.com/content/www/us/en/software-kit/665814/intel-fpga-pac-d5005-acceleration-stacks-v2-0-1.html

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

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