# A Case for Remote GPUs over 10GbE Network for VR Applications

Shin Morishima, Masahiro Okazaki and Hiroki Matsutani

Keio University, 3-14-1 Hiyoshi, Kohoku-ku, Yokohama, Japan
{morisima,okazaki,matutani}@arc.ics.keio.ac.jp

## ABSTRACT

Due to advances on graphic processing technology and sensing technology in recent years, VR technology that enables users to experience environments made by computer similar to real environments has become popular. In VR technology, computation cost of graphic processing is high and thus it requires a high-end GPU because high-quality pictures that look like real environments are processed while reflecting sensor information. Therefore, users have to prepare a computer with a high-end GPU, which requires a high cost. In this paper, we propose to connect GPU cluster and user-side computers via 10GbE (10Gbit Ethernet) network so that graphic processing for HMDs is done by the GPU cluster via a network. In this way, users can use HMD with inexpensive computers because they do not have to prepare computers with high-end GPUs.

In this paper, we propose an index to evaluate performance of VR processing tasks in remote GPU environment and evaluate the performance in the remote GPU environment based on this index. We also propose a condition that does not degrade user experience in the remote GPU environment and allocation methods of multiple tasks to GPUs under this condition. The evaluation results of the proposed allocation methods show that if there is a high-load task, the method that preferentially allocates tasks to GPUs with low bandwidth achieves high performance; otherwise the method that preferentially allocates tasks to GPUs with high utilization achieves a high performance.

## 1. INTRODUCTION

VR (Virtual Reality) is a computer technology that enables users to experience environments made by computer similar to real environments. It has been widely used for video games and flight simulator.

A typical VR device is a HMD (Head-Mount Display) that a user wears like a hat. The components of HMD are a display and sensors. The sensors scan motion of line of sight and position of a user. The display projects a picture that is reflected by sensing data. Recently, VR with HMD has become popular, because graphic processing using GPU and sensing technologies have been improved. HMDs for general consumers (e.g., Playstation VR by Sony [3] and Oculus Rift by Oculus [2]) have been released, which demonstrates the popularization of VR with HMD.

In VR with HMD, computation cost of graphic processing is high and thus it requires a high-end GPU because high-quality pictures that look like real environments are processed while reflecting sensor information. Therefore, to use HMDs, users have to prepare a computer with a high-end GPU, which requires a high cost.

In this paper, we propose to connect GPU cluster and users' computers via 10GbE (10Gbit Ethernet) network so that graphic processing for HMDs is done by the GPU cluster via the network. In this way, a user can use HMD with inexpensive computer because the user does not have to prepare a computer with a high-end GPU. Additionally, our proposal can improve GPUs utilization because the number of processes allocated to GPU cluster can be more than the number of GPUs in GPU cluster.

However, in the case of the VR technology with such remote GPUs, which are connected via 10GbE network, the bandwidth between CPU-GPU is smaller and communication delay is larger than GPUs connected with PCIe directly. In this paper, we propose a measurement index to evaluate a processing performance of VR technology in remote GPU environment. We then evaluate relationships of the performance vs. the bandwidth and communication delay between CPU-GPU. Additionally, we consider the effective way of allocating VR processing tasks to GPU cluster based on performance of VR application with remote GPUs.

The rest of this paper is organized as follows. Section 2 surveys related work. Section 3 introduces VR systems using remote GPUs and proposes a performance index of VR assuming remote GPUs. Section 4 proposes the way of allocating VR processing tasks to GPU cluster. Section 5 evaluates performance of VR using remote GPUs and the VR task allocation methods. Section 6 concludes this paper.

## 2. RELATED WORK

We employ NEC Exp-Ether [10] as a PCIe over 10GbE technology to use GPUs via Ethernet network. The approach is to connect user-side computers and GPU devices via a PCIe over 10GbE assuming that these GPU devices form a GPU cluster (or GPU pool). By using Exp-Ether, PCIe packets for GPUs can be transported in 10GbE by encapsulating the packets into an Ethernet frame. In our case, pooled GPUs can be connected to user-side computers via 10GbE when users use VR application. This approach is well-suited to GPU as a Service (GaaS) clouds [6]. It is also suited to recent trends on rack-scale architecture [1][5] and software-defined infrastructure. Actually it has been adopted for compute-intensive applications that require many GPU devices other than VR, such as distributed databases [7] and distributed data processing [8]. For example, database queries are processed by remote GPU devices connected via 10GbE in [7] and Apache Spark operations are processed by remote GPU devices in [8]. Please note that we use Exp-Ether 10G that achieves up to 20Gbps bandwidth in this work, while Exp-Ether 40G that achieves up to 80Gbps is already available, which would significantly increase the CPU-GPU bandwidth.

Another approach to use GPUs via Ethernet network is to use software service such as rCUDA [4] based on client-server model. In this approach, GPUs are directly connected to server machines via PCIe while clients do not have GPUs. Instead, a client requests GPU processing to the server using TCP/IP communication via Ethernet. Then the server processes the request using GPUs and returns the computation result to the client. If we use this approach, we need to modify the VR applications to support CUDA and rCUDA. It is not trivial because we need to make major changes on

This work was presented in part at the international symposium on Highly-Efficient Accelerators and Reconfigurable Technologies (HEART2017) Bochum, DE, June 7-9, 2017.

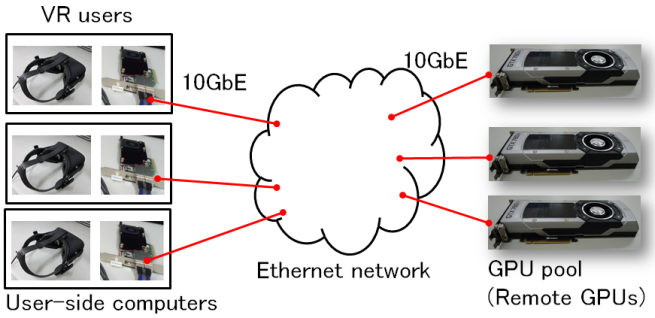


**Figure 1: VR system using remote GPUs**

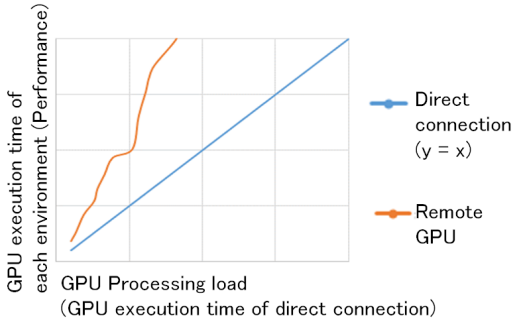


**Figure 2: Example of time-time graph as GPU performance index on VR in remote GPU environment**

the applications when we use this approach for existing VR applications. On the other hand, in our approach that uses PCIe over 10GbE technology, the remote GPUs are recognized by applications as if they are connected with PCIe directly. So, we can use remote GPU environment for VR applications without any changes on VR applications.

In the remote GPU environment, the latency between CPU-GPU is longer than that of existing VR environment. The impact of latency for VR user experience is analyzed in [11]. In the remote GPU environment, the bandwidth between CPU-GPU is restricted too. The influence of both the bandwidth and latency for application performance is evaluated in [9] by using rCUDA. In this paper, we propose task allocation methods for remote GPUs while taking into account the influence of latency and bandwidth.

Recently, NVLink attracts attention as a communication infrastructure between multiple GPUs [2]. NVLink is orthogonal to our proposal because it is mainly intended to accelerate the communication between GPUs located closely.

# 3. VR SYSTEM USING REMOTE GPUS

## 3.1 System Overview

Figure 1 illustrates the proposed VR system using remote GPUs. It consists of three components: user-side computers, GPU pool, and Ethernet network. In our remote GPU environment, NEC Exp-Ether interface [10] is installed to PCIe slot of the user-side computer instead of a high-end GPU device. The Exp-Ether interface card is connected to 10GbE network via one or two 10GbE SFP+ cables. As the GPU pool, we assume many GPUs are connected 10GbE network via NEC Exp-Ether interface with one or two 10GbE SFP+ cables. These pooled GPUs can be used by user-side computers via 10GbE network.

Assuming each computer requires a GPU connected via PCIe directly, the number of required GPUs is equal to the number of user-side machines. On the other hand, in the remote GPU environment, multiple GPU processing tasks can be allocated to one GPU, which means that we can reduce the number of GPUs required for the entire system.

Negative aspects of the remote GPUs come from lower CPU-GPU bandwidth and higher communication delay due to 10GbE network. It may degrade the frame rate (i.e., the number of pictures displayed during one second) of VR applications. Degradation of frame rate will degrade the user experience. In the remote GPU environment, we need to allocate GPU processing tasks to remote GPUs so as not to degrade the frame rate.

## 3.2 GPU Performance Index on VR in Remote GPU Environment

In the remote GPU environment, GPU performance on VR is limited due to lower CPU-GPU bandwidth and higher communication delay. In the remote GPU environment, since a lot of VR tasks are allocated to remote GPUs, we need to know the aggregated load that each remote GPU can process without degrading the user experience. As the aggregated load depends on the remote GPU performance, we analyze the relationship between GPU processing load and GPU performance. In the following, we first define the GPU processing load and GPU performance in VR processing and then propose a GPU performance index that represents the relationship between GPU processing load and GPU performance.

Typically, the GPU processing load affects a GPU core utilization ratio or an execution time of a GPU processing. Also GPU performance affects them too. This means that the GPU processing load increases in proportion to GPU core utilization or execution time of GPU processing; thus the GPU performance increases inversely in proportion to GPU core utilization or execution time of GPU processing.

In the VR processing, the GPU core utilization hardly changes and the execution time of GPU processing per frame changes when the GPU processing load changes. Similarly, the GPU utilization is constant while the GPU execution time changes when CPU-GPU bandwidth or communication delay changes. Based on the above observations, we propose to use the execution time of GPU processing per frame as an index for GPU performance and GPU processing load. In particular, the GPU execution time when the GPU is directly connected to a machine via PCIe is used as reference time, which is an index of GPU processing load. By comparing a GPU execution time with the reference time, we can estimate the magnitude of influence due to the overheads of remote GPU environment. Please note that the reference time is not the total processing time of VR application but the GPU processing time. Because the total processing time includes GPU processing time and CPU processing time, the total processing time is longer than the reference time.

Additionally, we propose a "time-time graph" in which x-axis is the reference time and y-axis is the GPU execution time in each environment to analyze a relationship between the GPU processing load and GPU performance on VR.

Figure 2 shows an example of a time-time graph as GPU performance index on VR in remote GPU environment. X-axis shows the GPU processing load, which means the GPU execution time with GPU directly connected via PCIe (i.e., reference time). Y-axis shows the measured GPU performance, which means the GPU execution time on a target environment (e.g., remote GPU environment). That is, this graph shows a relationship between GPU processing load (i.e., reference time) and actual GPU performance on a remote GPU environment. The performance of a directly-connected GPU is the line with $y = x$ because both of the axes show the same GPU execution time. The performance of a remote GPU is shown as a curve representing a relationship with GPU processing load; that is, the divergence between this curve and the line with $y = x$ shows the impact on performance caused by remote GPU environment. If y-axis is changed from execution time of GPU processing to execution time of entire processing, then the time-time graph represents a relationship between the execution time of entire processing per frame and

GPU processing load.

# 4. VR PROCESSING TASK ALLOCATION FOR REMOTE GPUS

In this paper, we assume that many user-side computers are connected to remote GPUs in the GPU pool via a PCIe over 10GbE network technology as mentioned in Section 3.1. In this case, the bandwidth between the computer and the remote GPU is constrained by the minimum bandwidth in the network between them. Also the communication delay is determined by the number of hops between them. Because we assume multiple GPU pools located in different places, the bandwidth and communication delay may change depending on location of the GPU used. In this case, because the GPU performance of VR processing changes depending on the GPU used, the throughput of an entire system depends on the allocation of users' GPU processing tasks to remote GPUs.

In this section, we consider the allocation methods to achieve the following two objectives.

1. Eliminating negative user experience due to remote GPUs
2. Maximizing throughput of an entire system while satisfying the first condition

## 4.1 Conditions for Minimizing Negative User Experiences

The GPU performance degradation caused by the remote GPU environment, such as reduced frame rate and frame drops, affects user experience. The frame rate determines a time limit for entire processing of one frame, and the limit is $\frac{1}{f}$ when $f$ is the frame rate. When the total processing time exceeds the time limit, the frame rate is decreased or some frames are dropped.

The following equation shows the condition for keeping a frame rate where the total processing time of a frame is $T$ and the frame rate is $f$.

$$T < \frac{1}{f} \tag{1}$$

In the remote GPU environment, we can allocate multiple VR processing tasks to one remote GPU. When one GPU processes multiple tasks, the GPU sequentially processes these tasks in a certain order. If all the tasks are completed before the next frame comes, the frame rate is kept. That is, assuming a GPU processing time per frame of the $i$-th task assigned to a GPU is $t_i$ and the number of tasks assigned to the GPU is $n$, the condition to keep the frame rate is expressed by the following equation.

$$\sum_{i=1}^{n} t_i < \frac{1}{f} \tag{2}$$

When all the GPUs satisfy Equation (2), the objective to eliminate negative user experience can be satisfied.

In other words, the following two conditions should be satisfied in the entire system.

1. All the users' tasks satisfy Equation (1).
2. All GPUs satisfy Equation (2).

$T$ and $t_i$ can be estimated based on the time-time graphs which are drawn by measuring the execution times of GPU processing and entire processing by varying the GPU processing load.

## 4.2 Task Allocation Methods for Remote GPUs

There are two factors that affect the GPU processing performance in the remote GPU environment: 1) bandwidth and 2) the number of hops between user-side computer and remote GPU. The number of hops affects the communication delay, but as described in Section 5, the negative impact of communication delay on application performance is small compared to that of bandwidth. Thus in this paper our task allocation methods mainly consider the difference in the bandwidth as the primary performance index [1].

In this paper, we propose the following three allocation methods and compare them in terms of performance.

- Method 1: GPUs with higher bandwidth are preferentially used.
- Method 2: GPUs with lower bandwidth are preferentially used.
- Method 3: GPUs with higher utilization rate are preferentially used.

These methods determine which GPU is preferentially used when multiple GPUs are available, under the conditions that do not degrade user experience based on Equations (1) and (2).

When a GPU has a high bandwidth and thus the GPU processing time is short, more GPU processing tasks can be allocated to the GPU. Method 1 that priorities GPUs with higher bandwidth aims at enhancing the overall performance by increasing the utilization of high performance GPUs.

Method 2 that prioritizes GPUs with lower bandwidth assigns GPU processing tasks on GPUs with lower performance. In Method 1, the GPU with the largest bandwidth is prioritized based on Equation (2), but Equation (1) is not considered. Based on the condition of Equation (1), there is a possibility that GPUs with low bandwidth may not be used at all so as not to degrade the user experience. In other words, GPU processing tasks that require high bandwidth have to be allocated to GPUs with high bandwidth, but if the other tasks have been already assigned to such high-performance GPUs with large bandwidth, the condition of Equation (2) cannot be satisfied. In such cases, to allocate new tasks, the task allocation has to wait until one of previously-allocated tasks is completed, resulting in performance degradation of the entire system. Method 2 has a potential to improve the performance by avoiding such problem. By assigning GPU processing tasks preferentially to GPUs with lower bandwidths, utilization of GPUs with higher bandwidths is reduced. As a result, based on the condition of Equation (1), Method 2 can decrease the possibility that the task allocation waits until an empty time slot becomes available on high-bandwidth GPUs.

In Method 3, the utilization is derived as a ratio of the allocated GPU processing tasks over the maximum processing capacity of the GPU. The maximum processing capacity is represented by the right side of Equation (2), and the amount of allocated tasks is represented by the left side of Equation (2). Thus, the utilization $R$ is expressed by the following equation.

$$R = \frac{\sum_{i=1}^{n} t_i}{\frac{1}{f}} \tag{3}$$

Method 3 aims at increasing the utilization of each GPU by preferentially allocating GPU processing tasks to GPUs with higher utilization. Since the utilization is prioritized in Method 3, GPUs with lower bandwidth are more likely to be given a higher priority because the utilization occupied by each task becomes high. Therefore, similar to Method 2, Method 3 can avoid the possibility to wait for a free time slot on high-bandwidth but heavily-loaded GPUs due to the condition of Equation (1).

# 5. EVALUATIONS

[1]Depending on a given network environment, communication delay (or both) should be selected as the primary performance index.

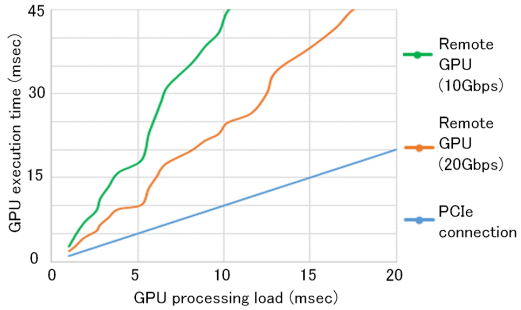


**Figure 3: Relationship between CPU-GPU bandwidth and GPU execution time per frame of VR application**

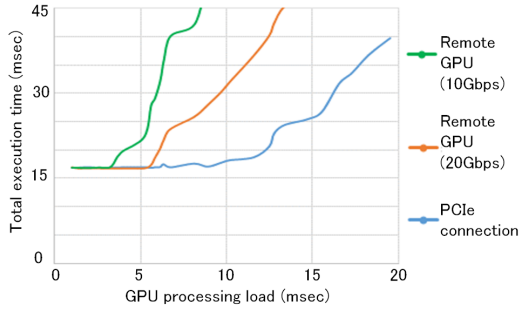


**Figure 4: Relationship with CPU-GPU bandwidth and execution time of entire VR processing per frame**

## 5.1 Evaluation Environment

In the experiments, we use a computer in which the processor is Intel Xeon E3-1225v2 and memory capacity is 16GB. The GPU used is NVIDIA GeForce GTX980Ti. Table 1 lists specification of the GPU. The GPU is installed to PCIe 3.0 x16 slot of the computer in the directly-connected GPU case, while it is connected to the computer with one or two 10GbE SFP+ cables using NEC ExpEther 10G in the remote GPU cases. The bandwidth of PCIe 3.0 x16 is up to 256Gbps, while that of ExpEther 10G is 20Gbps using two 10GbE cables at the maximum. We use 10GbE switches to evaluate the impact of communication delay with multiple hops. As a benchmark program, VR application was created using Unity 5.4.0 as a development environment and Oculus Rift was used as a VR device. The benchmark VR application is continuously generating many identical objects that move in constant motion as a video. We prepared multiple videos by changing the number of objects in the video in order to generate different GPU loads.

**Table 1: GPU spec used in the experiments**

| | GeForce GTX 980 Ti |
|---|---|
| Number of cores | 2,816 |
| Core clock | 1,038MHz |
| Memory clock | 7,010MHz |
| Memory datapath width | 256bit |
| Memory capacity | 6GB |

## 5.2 Relationship between CPU-GPU Bandwidth and VR Processing Performance

The factors that the remote GPU affects the VR processing performance are the bandwidth and communication delay between CPU and GPU, and this section focuses on the bandwidth. In the remote GPU environment assumed in this paper, user-side computers and remote GPUs are connected via 10GbE network. As mentioned above, the CPU-GPU 10GbE network bandwidth is lower than that of the direct connection via PCIe 3.0 x16. As the bandwidth becomes lower, the overhead of CPU-GPU communication becomes larger, and the VR processing performance is degraded. Its negative impact is evaluated by using the time-time graph described in Section 3.2.

Figure 3 shows the relationship between the CPU-GPU bandwidth and the GPU execution time per frame of the VR application. The direct PCIe connection has the highest bandwidth. Since its GPU execution time is used as the index of GPU processing load, the relationship between its GPU execution time and GPU processing load becomes a straight line of $y = x$. The remaining two curves show the GPU execution times in two remote GPU environments with different bandwidths. As shown in the graph, the remote GPU environments increase the GPU execution time in any GPU processing load. By comparing the remote GPU environments with 20Gbps and 10Gbps bandwidths, the lower CPU-GPU bandwidth environment increases the GPU execution time. In the VR application, since the transfer of the GPU processing contents from CPU to GPU and the transfer of the GPU processing result from GPU to CPU are performed for each frame, these transfer overheads are incurred for each frame[2].

The increase in execution time of the GPU processing affects the performance of an entire VR application. Figure 4 shows the relationship between the CPU-GPU bandwidth and the execution time per frame of an entire VR application. When an entire processing task is finished earlier than the required frame rate, the GPU simply waits for the next frame. The frame rate is set to 60fps (frame per second) in this evaluation environment; thus, the frame length is about 16msec. When the total execution time is less than this, the GPU waits for the next frame so that the total execution time becomes 16msec. When the total execution time increases more than this, on the other hand, the frame rate is decreased.

From Figure 4, it can be seen that the total execution time of the entire VR application increases as the GPU processing time in the remote GPU environment (Figure 3) increases.

## 5.3 Relationship between CPU-GPU Communication Delay and VR Processing Performance

The second factor that affects the VR processing performance in the remote GPU environment is the CPU-GPU communication delay. In the remote GPU environment, user-side computers and GPU pool are connected via 10GbE network and there may be multiple switches (or hops) between them. The communication delay increases as the number of CPU-GPU hops increases. Thus, the relationship between the communication delay and the VR processing performance is evaluated by varying the number of 10GbE switch hops for the CPU-GPU communication.

Figure 5 shows relationship between the CPU-GPU hops and the GPU execution time per frame of the VR application. The communication bandwidth is set to 10Gbps in this figure. In this graph, 0-hop indicates that there is no switch between CPU and GPU, that is, the user-side computer and GPU are directly connected by a 10GbE SFP+ cable with ExpEther 10G. 1-hop indicates that the user-side computer and GPU are connected via 10GbE network through one 10GbE switch. When comparing the remote GPU environments with 1-hop and 0-hop, the GPU execution time of 1-hop is slightly longer than that of 0-hop, but the difference is not significant as shown in the graph. That is, negative impact of the communication delay due to intermediate switch on performance is quite small as compared to that of the bandwidth. In this case, when allocating tasks to remote GPUs, the bandwidth should be considered at first rather than the number of hops. Please note that the proposed task allocation methods

[2]Although we employ ExpEther 10G in this work, the bandwidth will be significantly improve with ExpEther 40G that achieves up to 80Gbps, which will be our future work.

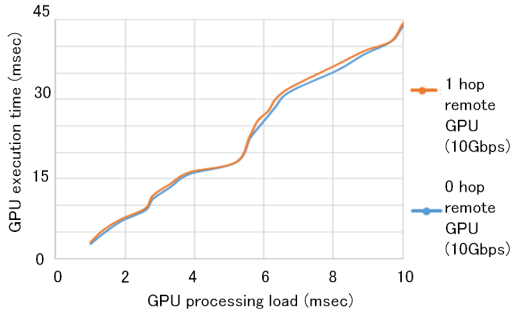


**Figure 5: Relationship between CPU-GPU hops and GPU execution time per frames of VR application**

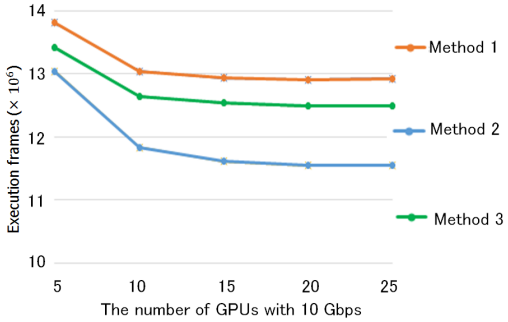


**Figure 6: Relationship between the number of GPUs with 10Gbps CPU-GPU bandwidth and the number of execution frames in high load case**

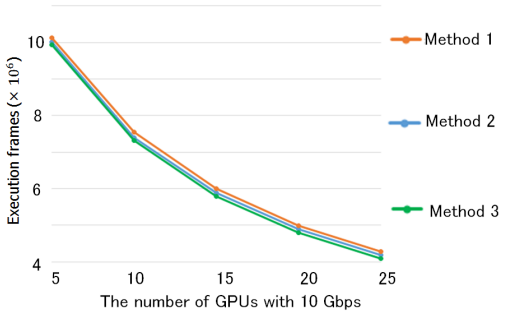


**Figure 7: Relationship between the number of GPUs with 10Gbps CPU-GPU bandwidth and the number of execution frames in low load case**

described in Section 4.2 consider the bandwidth.

## 5.4 Task Allocation Methods for Remote GPUs

We evaluate the three tasks allocation methods described in Section 4.2 based on simulations using the execution times for each bandwidth measured in Section 5.2. The task allocation is performed under the conditions of Equations (1) and (2) described in Section 4.1, so it is necessary to obtain the parameters used in these equations. Again, $T$ is the execution time of an entire VR processing shown in Figure 4. $t_i$ is the execution time of each GPU processing shown in Figure 3. The frame rate $f$ requested by the application is set to 30fps which is the frame rate used in television and the like.

We assume that each GPU processing lasts a certain number of frames with a certain workload. In this case, they have two variables: the GPU processing load $t$ and the number of frames $k$. The workload of each GPU processing is represented by $t \times k$. Assuming the number of tasks to be processed in a simulation is $N$, the total workload $S$ is expressed by the following equation.

$$S = \sum_{i=1}^{N} t_i k \tag{4}$$

$S$ is set to a constant value. Tasks are generated until the total workload reaches to $S$. These tasks are processed with GPUs. The performance of the entire system is evaluated based on the number of frames processed until all the tasks are completed. In this simulation, $S$ is set to 100 million. To analyze the effects of the number of GPUs and network configuration, we evaluate the performance by changing the number of GPUs and network bandwidth in the remote GPU environment. As the bandwidth, we assume 20Gbps and 10Gbps in the evaluation shown in Figure 3. More specifically, the number of GPUs with 20Gbps bandwidth is fixed to five and the number of GPUs with 10Gbps bandwidth is varied. As can be seen from Figure 4, the total processing time $T$ of the entire VR processing also increases as the GPU processing load $t$ of each task increases. Therefore, from Equation (1), the task cannot be allocated to GPU with low bandwidth if $t$ is large; thus the possible range of $t$ affects the performance of the system. In this evaluation, there are two possible ranges of $t$:

- $1 \leq t \leq 6$ that satisfies Equation (1) by GPUs with 10Gbps for all the tasks
- $1 \leq t \leq 8$ that satisfies Equation (1) by GPUs with 20Gbps for all the tasks

The above $t$'s ranges were obtained so that corresponding $T$ values satisfy Equation (1) based on $T$-to-$t$ relationship in Figure 4. We randomly generate GPU processing loads so as to satisfy the above $t$'s ranges. In this evaluation, the first range of $t$ is denoted as "low load case" and the second range is "high load case". The number of consecutive frames $k$, which is another variable of task, is set to a random number in the range of $100 \leq k \leq 1000$.

Figure 6 shows the performance of the three allocation methods when the number of GPUs with 10Gbps bandwidth is varied in the high load case. The vertical axis represents the number of frames necessary for completing a certain amount of tasks and the smaller number indicates that the throughput is higher. A task with a high load cannot be allocated to GPUs with 10Gbps from the restriction of Equation (1). Therefore, since tasks with high load become bottleneck, in all the methods the performance improvement by increasing the number of GPUs is saturated when the number of GPUs is 15 or more.

When comparing the three methods, Method 2 that preferentially allocates tasks to GPUs with small bandwidth achieves a good performance. This is because it can allocate more tasks, which are high load and bottleneck, to GPUs with 20Gbps bandwidth.

Figure 7 shows the performance of the three allocation methods when the number of GPUs with 10Gbps is varied in the low load case. Unlike at the high load case, since there is no task that does not satisfy Equation (1), performance of all the methods improves as the number of GPUs increases. Performance differences between these methods are smaller than those at the high load case, but Method 3 achieves the highest performance. In Method 3, tasks are preferentially allocated to GPUs with high utilization ratio. Since it can increase the utilization of all GPUs regardless of bandwidth, it achieves a higher performance than the other methods.

Based on these results, Method 2 should be adopted if there is a high-load task that may not satisfy Equation (1); otherwise, Method 3 should be adopted.

As task allocation to remote GPUs, multiple tasks can be allocated to one GPU as long as the conditions of Equations (1) and (2) are satisfied. On the contrary, in the existing VR environment that does not use remote GPUs, since process-

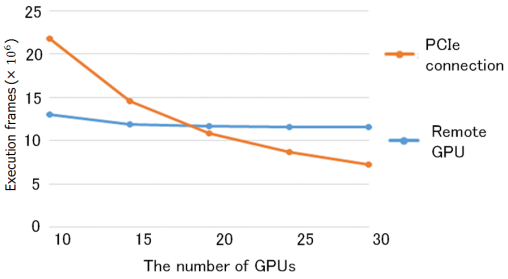


**Figure 8: Performance of remote GPU environment and directly-connected GPU environment in high load case**

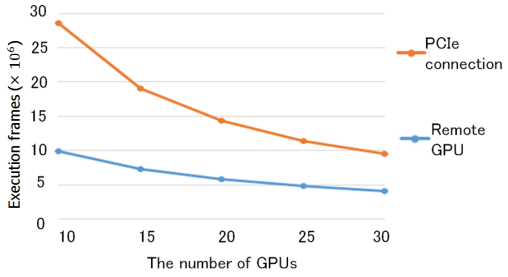


**Figure 9: Performance of remote GPU environment and directly-connected GPU environment in low load case**

ing tasks of each user-side computer are performed by a local GPU directly connected to the computer, only one task can be allocated to each GPU. Therefore, if more than one tasks can be allocated to one GPU on average, the remote GPU environment achieves a higher overall system throughput than the conventional environments. The high throughput means that tasks of the same number of users can be processed with fewer GPUs compared to the conventional environments.

Figures 8 and 9 show the performance of GPUs directly-connected by PCIe and those of remote GPU when the same number of GPUs are used at high load and low load respectively. As the remote GPU allocation method in the experiments, we choose the best method for each case. The ratio of 20Gbps/10Gbps bandwidth GPUs in the remote GPU environment is the same as in Figures 6 and 7. These figures show that the performance of the remote GPU environment is higher in two cases: 1) high load case with fewer GPUs and 2) low load case with any number of GPUs. In the case of low load case, since all the GPUs satisfy Equation (1) and one or more tasks are always allocated, the performance becomes higher than the directly-connected GPU case. On the other hand, in the high load case with more GPUs, because of the increase of the number of GPUs that cannot satisfy Equation (1), the performance is lower than the directly-connected GPU case. Assuming the maximum bandwidth between the user-side computers and GPU pool is given, it is important to configure the ratio of different bandwidth GPUs by considering the distribution of application workloads, such as the numbers of tasks with high and low loads.

# 6. SUMMARY

VR technology requires a high graphic processing power and thus it requires a high-end CPU for each user; so the cost to enjoy VR applications is typically high. In this paper, we proposed to use the remote GPUs for VR applications so that the user-side computers and pooled GPUs are connected via 10GbE network. With this approach, users can utilize pooled GPUs via network and do not have to prepare computers with high-end GPUs; thus the cost to start VR applications can be reduced. In this paper, we proposed an index to evaluate the VR application performance in the remote GPU environment. We then evaluated relationships between the performance and the CPU-GPU bandwidth and communication delay. The evaluation result showed that the VR application performance is affected by the CPU-GPU bandwidth. We also proposed a condition that does not degrade user experience in remote GPU environment and allocation methods of multiple tasks to GPUs under this condition. The evaluation results of the proposed allocation methods showed that if there is a high-load task, the method that preferentially allocates tasks to GPUs with small bandwidth achieves a high performance; otherwise, the method that preferentially allocates tasks to GPUs with high utilization achieves a high performance. Please note that this is the first work to address the initial cost for VR applications by borrowing remote GPU environment and the prototype of the proposed VR system has already been working comfortably in the lab.

**Acknowledgements** This work was supported by JSPS KAKENHI Grant Number JP16J05641.

# 7. REFERENCES

[1] Intel Rack Scale Architecture: Faster Service Delivery and Lower TCO. `http://www.intel.com/content/www/us/en/architecture-and-technology/intel-rack-scale-architecture.html`.

[2] Oculus Rift. `https://www3.oculus.com/en-us/rift/`.

[3] PlayStaion VR. `http://www.jp.playstation.com/psvr`.

[4] J. Duato, A. Pena, F. Silla, R. Mayo, and E. Quintana-Orti. rCUDA: Reducing the Number of GPU-based Accelerators in High Performance Clusters. In *Proc. of the International Conference on High Performance Computing and Simulation (HPCS'10)*, pages 224–231, Jun 2010.

[5] S. Legtchenko, N. Chen, D. Cletheroe, A. Rowstron, H. Williams, and X. Zhao. XFabric: A Reconfigurable In-Rack Network for Rack-Scale Computers. In *Proc. of the USENIX Symposium on Networked Systems Design and Implementation (NSDI'16)*, pages 15–29, Mar 2016.

[6] M. Liu, T. Li, N. Jia, A. Currid, and V. Troy. Understanding The Virtualization "Tax" of Scale-Out Pass-Through GPUs in GaaS Clouds: An Empirical Study. In *Proc. of the International Symposium on High Performance Computer Architecture (HPCA'15)*, pages 259–270, Feb 2015.

[7] S. Morishima and H. Matsutani. Distributed In-GPU Data Cache for Document-Oriented Data Store via PCIe over 10Gbit Ethernet. In *Proc. of the International Workshop on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Platforms (HeteroPar'16)*, Aug 2016.

[8] Y. Ohno, S. Morishima, and H. Matsutani. Accelerating Spark RDD Operations with Local and Remote GPU Devices. In *Proc. of the International Conference on Parallel and Distributed Systems (ICPADS'16)*, pages 791–799, Dec 2016.

[9] C. Reao, R. Mayo, E. S. Quintana-Ort, F. Silla, J. Duato, and A. J. Pea. Influence of InfiniBand FDR on the performance of remote GPU virtualization. In *Proc. of International Conference on Cluster Computing (CLUSTER'13)*, pages 1–8, Sep 2013.

[10] J. Suzuki, Y. Hidaka, J. Higuchi, Y. Hayashi, M. Kan, and T. Yoshikawa. Disaggregation and Sharing of I/O Devices in Cloud Data Centers. *IEEE Transactions on Computers*, 66(10):3013–3026, Oct 2016.

[11] T. Waltemate, I. Senna, F. Hülsmann, M. Rohde, S. Kopp, M. Ernst, and M. Botsch. The Impact of Latency on Perceptual Judgments and Motor Performance in Closed-loop Interaction in Virtual Reality . In *Proc. of the International Conference on Virtual Reality Software and Technology (VRST'16)*, pages 27–35, Nov 2016.