I. INTRODUCTION

Recently, HPC users are interested in running HPC applications on Cloud computing since they are looking Cloud computing as an alternative to dedicated supercomputers. In HPC Cloud computing, users rent high-end computing infrastructure from service provider and pay money as they used. This scheme is now called HPC as a Service (HPCaaS). In addition, GPGPU is now one of the most efficient way to boost up scientific applications. Many HPC applications got better performance by using GPU programming models such as CUDA and OpenCL.

Despite the advantage of using GPGPU, most of Cloud computing OS do not support GPU as a primary device. There are some commercial Cloud service that provides GPU as an instance such as Amazon EC2, Penguin Computing, and Nimbix. But still these commercial Cloud services provide GPU instance with PCI pass-through technique that virtual machines (VMs) cannot access other host’s GPU. Using GPUs across network is important in HPC Cloud computing since most of the time users will use VMs without GPU, and not every compute node will have GPUs attached. If node with a GPU is occupied by VMs that don’t use a GPU, there should be way to provide the GPU, which remains idle, to other VMs.

II. A GPGPU HPC CLOUD PLATFORM BASED ON OPENSTACK

Our GPGPU HPC Cloud platform used OpenStack Cloud OS, KVM hypervisor, and rCUDA [1] API forwarding technique to build a software stack of the Cloud platform. With API forwarding, our Cloud platform can support multiple VMs to share single GPU which is important to maximize GPU resource utilization. Also it is possible for single VM to use multiple GPUs across different hosts.

We propose GPGPU HPC Cloud platform with OpenStack, KVM, and rCUDA. And to make efficient use of GPU resource in Cloud computing, we introduce two GPU resource scheduling concepts: centralized and distributed. Finally, we evaluate performance of our Cloud platform with three different testing environment.

Our GPGPU HPC Cloud platform gives many opportunity to GPU resource scheduling on Cloud computing. Previous GPU resource scheduling was subordinated to CPU resource scheduling. Due to limitation of PCI pass-through, GPU was dedicated to the VM in initial step and cannot change to use other GPUs after VM is in running state. But in our cloud environment, VM can use other host’s GPUs if dedicated GPU is not powerful enough to support user’s Service Level Agreement (SLA). This environment change can also give challenges to live migration of VMs with GPU resource. In this work, we present on-going research of GPU resource scheduling on GPGPU enabled Cloud computing. One is centralized GPU resource scheduling, and the other is distributed GPU resource scheduling.

A. Centralized GPU resource scheduling

Our centralized GPU resource scheduler is inside the nova compute in controller node. All the resource information, including GPU, is gathered in OpenStack database. Our scheduler is separated to 2 parts: initial placement and dynamic allocation. At initial placement step, scheduler decides which GPUs will be attached to user’s VM. In dynamic allocation part, scheduler monitors GPU resource usage and re-schedule VMs to GPUs if resource is not fully utilized. This step is related to live migration of VMs with GPU resource. To make efficient scheduler, we consider GPU core utilization, GPU memory usage, network topology, and path latency as inputs of scheduling.

B. Distributed GPU resource scheduling

The aim of distributed GPU resource scheduler is to remove the single point of failure as compared to centralized approach, so that failure of one scheduler does not affect whole system. Each compute nodes runs a scheduler, which monitors its own resource and shares resource information with other schedulers. If the local server resource is not enough to serve the VMs, then it will ask for more resource from the nearest schedulers. These schedulers are clustered based on the hop-count information from the network discovery module. The research focuses on efficient and fault-tolerant scheduler system, which considers network topology, speed link and current load on each cluster.
III. PERFORMANCE EVALUATION

These are performance of multiple GPU benchmark applications on 1 VM and 1 GPU environment, compared to same configuration with PCI pass-through GPU virtualization. In this testing, we tended to show performance of our Cloud platform with API forwarding, compared to existing Cloud platform with PCI pass-through. We used total eight benchmark programs: four from Rodinia [2] benchmark suite and other four from Scalable HeterOgeneous Computing [3] (SHOC). In Rodinia, we chose BFS, Gaussian, NW, and Pathfinder. In SHOC, we used BusSpeedDownload, BusSpeedReadBack, MD, and GEMM. The first two programs in SHOC are to check PCI-e bus bandwidth of Cloud platform. Other six programs are separated to compute intensive (Gaussian, MD, GEMM) and data intensive (BFS, NW, Pathfinder) programs. Intensity of compute and data is measured by memory copy ratio of host to device and vice versa.

On 1 VM and 1 GPU environment, speedup ratios are measured with results of last three largest inputs. Speedup ratios of compute intensive programs vary from 0.93 to 0.99. When multiple VMs share single GPU, running time was almost directly proportional to number of VMs. With these results, we concluded our GPGPU HPC Cloud platform can be useful with compute intensive HPC applications. But with data intensive HPC applications, we still faced performance bottleneck with network, even though we used high-speed Infiniband network. When multiple VMs share single GPU, we found that rCUDA does not support Hyper-Q techniques provided by CUDA 5.5. This requires more research on API forwarding technique to make GPU sharing more efficient.

IV. REFERENCE