CONTAINERS & KUBERNETES

Using Google Kubernetes Engine's GPU sharing to search for neutrinos

Author: Igor Sfiligoi
Lead Scientific Software Developer and Researcher, San Diego Supercomputer Center

***Editor's note**: Today we hear from the San Diego Supercomputer Center (SDSC) and University of Wisconsin-Madison about how GPU sharing in Google Kubernetes Engines is helping them detect neutrinos at the South Pole with the gigaton-scale IceCube Neutrino Observatory.*

IceCube Neutrino Observatory is a detector at the South Pole designed to search for nearly massless subatomic particles called neutrinos. These high-energy astronomical messengers provide information to probe events like exploding stars, gamma-ray bursts, and cataclysmic phenomena involving black holes and neutron stars. Scientific computer simulations are run on the sensory data that IceCube collects on neutrinos to pinpoint the direction of detected cosmic events and improve their resolution.

The most computationally intensive part of the IceCube simulation workflow is the photon propagation code, a.k.a. ray-tracing, and that code can greatly benefit from running on NVIDIA GPUs. The application is high throughput in nature, with each photon simulation being independent of the others. Apart from the core data acquisition system at the South Pole, most of IceCube's compute needs are served by an aggregation of compute resources from various research institutions all over the world, most of which use the Open Science Grid (OSG) infrastructure as their unifying glue.

GPU resources are relatively scarce in the scientific resource provider community. In 2021, OSG had only 6M GPU hours vs 1800M CPU core hours in its infrastructure. The ability to expand the available resource pool with cloud resources is thus highly desirable.

The SDSC team recently extended the OSG infrastructure to effectively use Kubernetes-managed resources to support IceCube compute workloads on the Pacific Research Platform (PRP). The

service manages dynamic provisioning in a completely autonomous fashion by implementing horizontal pilot pod autoscaling based on the queue depth of the IceCube batch system.
Unlike on-premises systems, Google Cloud offers the benefits of elasticity (on-demand scaling) and cost efficiency (only pay for what gets used). We needed a flexible platform that can avail these benefits to our community. We found Google Kubernetes Engine (GKE) to be a great match for our needs due to its support for auto-provisioning, auto-scaling, dynamic scheduling, orchestrated maintenance, job API and fault tolerance, as well as support for co-mingling of various machine types (e.g. CPU + GPU and on-demand + Spot) in the same cluster and up to 15,000 nodes per cluster.

While IceCube's ray-tracing simulation greatly benefits from computing on the GKE GPUs, it still relies on CPU compute for feeding the data to the GPU portion of the code. And GPUs have been getting faster at a much higher rate than CPUs have! With the advent of the NVIDIA V100 and A100 GPUs, the IceCube code is now CPU-bound in many configurations. By sharing a large GPU between multiple IceCube applications, the IceCube ray-tracing simulation again becomes GPU-bound, and therefore we get significantly more simulation results from the same hardware. GKE has native support for both simple GPU time-sharing and the more advanced A100 Multi-Instance GPU (MIG) partitioning, making it incredibly easy for IceCube — and OSG at large — to use.
To leverage the elasticity of the Google Cloud, we fully relied on GKE horizontal node auto-scaling for provisioning and de-provisioning GKE compute resources. Whenever there were worker pods that could not be started, the auto-scaler provisioned more GKE nodes, up to a set maximum. Whenever a GKE node was unused, the auto-scaler de-provisioned it to save costs.

Performance results

Using Google Cloud GPU resources was very simple through GKE. We used the same setup we were already using on the on-prem PRP Kubernetes cluster, simply pointing our setup to the new cluster.

After the initial setup, IceCube was able to efficiently use Google Cloud resources, without any manual intervention by the supporting SDSC team beyond setting the auto-scaling limits. This was a very welcome change from other cloud activities the SDSC team has performed on behalf of IceCube and others, that required active management of provisioned resources.

Autoscaling

The GKE auto-scaling for autonomous provisioning and de-provisioning of cloud resources worked as advertised, closely matching the demand from IceCube users, as seen in Fig. 1. We were particularly impressed by GKE’s performance in conjunction with GPU sharing; the test run shown used seven A100 MIG partitions per GPU.

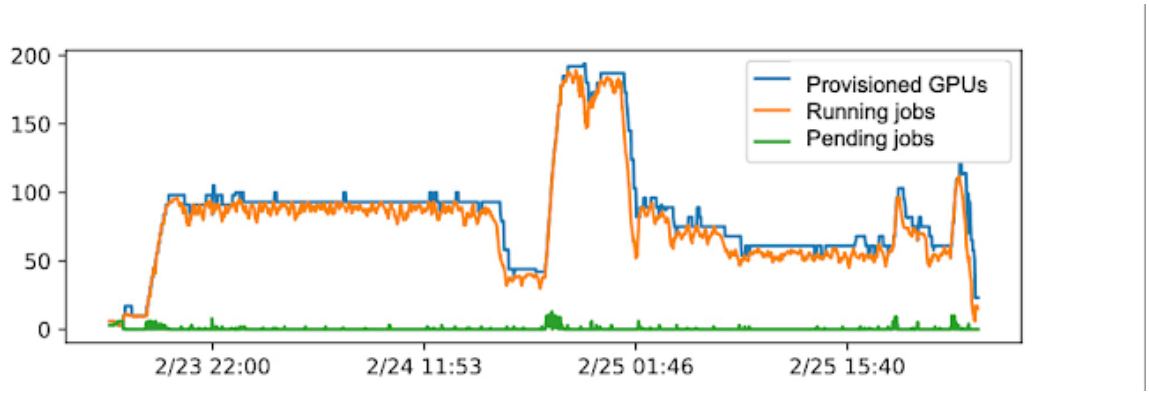

Fig. 1: Monitoring snapshot of the unconstrained GKE auto-scaling test run.

NVIDIA GPU sharing

Both full-GPU and shared-GPU Kubernetes nodes with NVIDIA A100, V100 and T4 Tensor Core GPUs were provisioned, but IceCube jobs did not differentiate between them, since all provisioned resources met the jobs' minimum requirements.

We assumed that GPU sharing benefits would vary based on the CPU-to-GPU ratio of the chosen workflow, so during this exercise we picked one workflow from each extreme. IceCube users can choose to speed up the GPU-based ray-tracing compute of some problems by, roughly speaking, increasing the size of the target for the photons by some factor. For example, setting oversize=1 gives the most precise simulation, and oversize=4 gives the fastest. Faster compute (of course) results in a higher CPU-to-GPU ratio.

The fastest oversize=4 workload benefitted the most from GPU sharing. As can be seen from Fig. 2, IceCube oversize=4 jobs cannot make good use of anything faster than a NVIDIA T4. Indeed, even for the low-end T4 GPU, sharing increases the job throughput by about 40%! For the A100 GPU, GPU sharing gets us a 4.5x throughput increase, which is truly transformational. Note that MIG and "plain" GPU sharing provide comparable throughput improvements, but MIG comes with much stronger isolation guarantees, which would be very valuable in a multi-user setup.

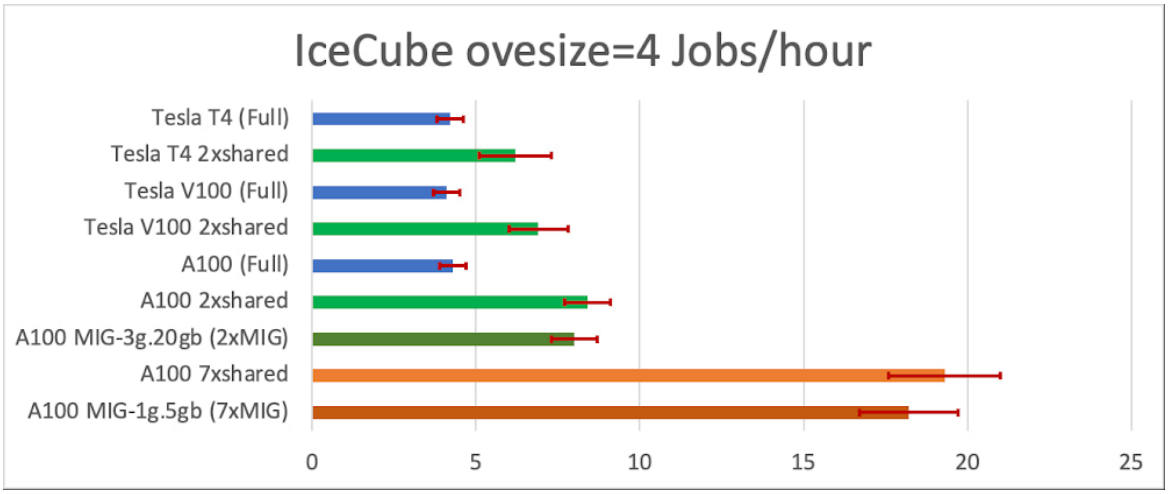

Fig. 2: Number of IceCube oversize=4 jobs per hour, grouped by GPU setup.

The more demanding oversize=1 workload makes much better use of the GPUs, so we observe no job throughput improvement for the older T4 and V100 GPUs. The A100 GPU, however, is still too powerful to be used as a whole, and GPU sharing gives us almost a 2x throughput improvement here, as illustrated in Fig. 3.

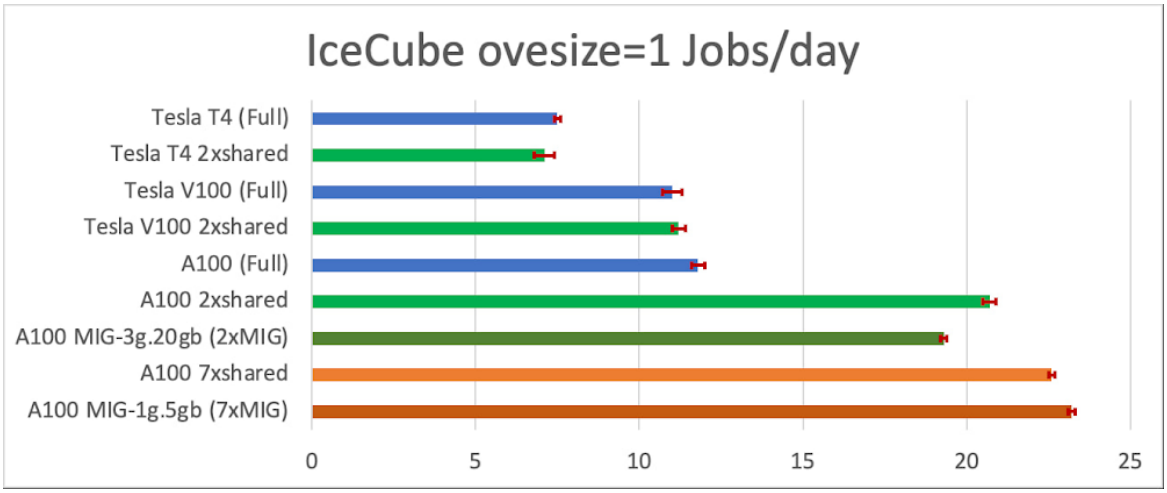

Fig. 3: Number of IceCube oversize=1 jobs per day, grouped by GPU setup.

GPU sharing of course increases the wallclock time needed by any single job to run to completion. This is however not a limiting factor for IceCube, since the main objective is to produce the output of thousands of independent jobs, and the expected timeline is measured in days, not minutes. Job throughput and cost effectiveness are therefore much more important than compute latency.

Finally, we would like to stress that most of the used resources were provisioned on top of Spot VMs, making them significantly cheaper than their on-demand equivalents. GKE gracefully handled any preemption, making this mode of operation very cost effective.

Lessons learned

GKE with GPU sharing has proven to be very simple to use, given that our workloads were already Kubernetes-ready. From a user point of view, there were virtually no differences from the on-prem Kubernetes cluster they were accustomed to.
The benefits of GPU sharing obviously depend on the chosen workloads, but at least for IceCube it seems to be a necessary feature for the latest GPUs, i.e. the NVIDIA A100. Additionally, a significant fraction of IceCube jobs can benefit from GPU sharing even for lower-end T4 GPUs. When choosing the GPU-sharing methodology, we definitely prefer MIG partitioning. While less flexible than time-shared GPU sharing, MIG's strong isolation properties make management of multi-workload setups much more predictable. That said, "plain" GPU sharing was still more than acceptable, and was especially welcome on GPUs that lack MIG support.

In summary, the GKE shared-GPU experience was very positive. The observed benefits of GPU sharing in Kubernetes were an eye-opener and we plan to make use of it whenever possible.

Want to learn more about sharing GPUs on GKE? Check out this user guide.

Related Article

Turbocharge workloads with new multi-instance NVIDIA GPUs on GKE