ABCI 2.0: Advances in Open AI Computing Infrastructure at AIST

Shinichiro Takizawa[1,a)] Yusuke Tanimura[1] Hidemoto Nakada[1]
Ryousei Takano[1] Hirotaka Ogawa[1]

Abstract: ABCI is the world ' s first large-scale Open AI Computing Infrastructure for both developing AI technologies and bridging them into the industry, operated by AIST, Japan since August 2018. It delivers 19.88 petaflops of HPL performance and achieves 70 seconds for training ResNet-50 model in MLPerf Training v0.6. Last November we achieved world ' s fastest records for CosmoFlow and DeepCAM in MLPerf HPC benchmarks. ABCI was the fastest supercomputer in Japan until Fugaku made a spectacular debut, however, it soon became short of computing capacity and I/O performance due to the rapid expansion of its usage. This forced us to make a major upgrade to ABCI. With this upgrade, we have added 120 compute nodes and a stroage system with a capacity of 11 PBytes. We named the whole system which includes both existing ABCI and the newly added equipments as *ABCI 2.0*. ABCI 2.0 provides the same software environment that ABCI provided. It enables that existing ABCI users can easily use the newly equipments in a similar way they used ABCI. We compared the performance of existing and new compute nodes and found that new nodes had 4.1 times higher performance than existing nodes in training ResNet-50 model using PyTorch. We expect that the new nodes largely contributes to increase the system throughput.

1. Introduction

AIST starts operating ABCI (AI Bridging Cloud Infrastructure) [1] from August 2018 as a platform for promoting the development of AI technologies. Machine learning and big data processing are two important computations required by AI training tasks and ABCI has high performance GPUs and large capacity storage systems to process such computation at high speed and high throughput. ABCI's peak performance are 37.2 PFlops in double precision, 75 PFlops in single precision and 550 PFlops in half precision. Its LINPACK performance are 19.88 PFlops and 14.423 GFlops/W, which were fifth in June 2018 list of Top500 and third in June 2019 of Green500. ABCI also ranked in fifth in June 2018 list of HPCG. As of November 2020, ABCI is still ranked 14th, 13th and 11th in Top500, Green500 and HPCG list.

ABCI is used by more than 2,000 users and 300 research projects and used in the following areas: image and voice recognition, natural language processing, robotics, bioinformatics, Go, etc. Moreover, ABCI is used in studying distributed training algorithms for deep neural networks and achieves the world's top level of performance [2–4]. As a result of such a wide spread of use, utilization of ABCI is constantly over 90% and more than 1,000 jobs occasionally waited, especially in busy season. There were also a shortage of storage capacity and I/O performance problems where responses to I/O requests became worse because of concurrent executions of large number of jobs. because of concurrent executions of large number of jobs and a shortage of storage capacity. To solve the problems, we continuously monitored the system and tuned system parameters, however, we concluded that the root cause of the problems is a lack of computing and storage capacity to meet demands from users.

To solve the above problem, we extended ABCI's computing and storage capacity. We added the following two equipments to ABCI: 120 compute nodes equipped with eight NVIDIA A100 GPUs and 11 PB storage system. We named the whole system which includes both existing ABCI and the newly added equipments as *ABCI 2.0* and started its operation from May 2021. Peak performance of ABCI 2.0 is 56.5 PFlops in double precision and 225 PFlops in single precision, which are 1.52 times and 3 times higher than existing ABCI. ABCI 2.0 provides the same software environment that ABCI provided. Users can use new compute nodes and storage in a similar way they used ABCI. We evaluated the performance of ABCI 2.0 in terms of comparing performance of new and old compute nodes in typical use patterns of ABCI. From a performance result of training ResNet50 model using PyTorch, we confirmed that new nodes had up to 2.1 times by GPU and 4.1 times by nodes higher performance than old nodes. We can extrapolate

[1] National Institute of Advanced Industrial Science and Technology (AIST)
[a)] shinichiro.takizawa@aist.go.jp

that the throughput of processing this task on ABCI 2.0 is 1.46 times increased against ABCI from this performance improvement ratio. We expect that such a throughput gain can be observed in other AI tasks and hope that ABCI 2.0 contributes to meet more demands from users.

The paper is organized as follows. We describe hardware of ABCI 2.0 in Sec. 2 and software in Sec. 3. In Sec. 4, we show performance results of ABCI 2.0 and conclude in Sec. 5.

2. ABCI 2.0 Hardware

Figure 1 shows the hardware architecture of ABCI 2.0. Components highlighted in light blue are new one introduced as part of ABCI 2.0.

2.1 Compute Resources

ABCI 2.0 consists of three major types of compute nodes. Two are compute nodes equipped with GPUs and focus on accelerating training of machine learning tasks. They are named as *Compute Node (A)* and *Compute Node (V)*, and the former has eight NVIDIA A100 GPUs and the later has four NVIDIA V100 GPUs. Newly added Compute Node (A) is fatter than Compute Node (V) in terms of performance. We increase number of GPUs and, proportionally, the amount of memory, number of InfiniBand HCAs, etc. to increase data transfer performance of multi GPU training tasks. Figure 2 shows the block diagram of Compute Node (A) and Figure 3 shows that of Compute Node (V). We adopted the latest off-the-shelf hardware which can be acquired at the procurement to construct ABCI, and then combine them so that the best performance can be achieved for machine learning tasks. Although devices used in these two types of compute nodes are different, we designed both nodes symmetric in terms of CPU, GPU and InfiniBand HCA so that there was no load imbalance in data transfer within a node.

The last type of compute node is *Memory-Intensive Node* each of which equipped with 2.6 TiB large capacity memory. The memory consists from 768 GiB traditional DDR memories and two Intel Optane SSDs. We enabled Intel Memory Drive Technology (IMDT) to transparently use the SSDs as a part of memory. This type of compute node was introduced for pre/post processing of large amount of training data.

As other types of compute nodes, ABCI 2.0 has *Interactive Nodes* and *Gateway Nodes*. Interactive Nodes are servers to log in and used for compiling programs and submitting jobs to the above three types of nodes. Gateway Nodes are NAT servers located between compute nodes and the Internet. As the three major compute nodes are not directly connected to the Internet, their access to the Internet is relayed by Gateway Nodes.

Peak performance of ABCI 2.0 is 56.5 PFLOPS in double precision, 225 PFLOPS in single precision and 850 PFLOPS in half precision. The performance of Compute Node (A) and Compute Node (V) largely contribute to the entire ABIC 2.0 performance.

2.2 Storage

ABCI 2.0 has two types of storage systems. One is *Shared File Systems* which are mounded by compute nodes described in Sec. 2.1. Data used and generated by computation run on ABCI 2.0 is basically stored in this storage area. The other is an Amazon S3 compatible object storage system named as *ABCI Cloud Storage*. ABCI Cloud Storage supports several security features, such as end-to-end in-transit data encryption, at-rest data encryption and AWS IAM compatible access control. It can be used for publishing research deliverables, such as generated data and trained models, and deliverables can be cataloged on ABCI Datasets [5]. The detail about ABCI Cloud Storage is described in our previous research paper [6].

Shared File Systems consist of three types of area for different objectives. The first is storage area for system data and users' home. We assumed that large capacity was not required on this storage because system data would not be so large and we set quota on users' home. In contrast, we assumed that high random I/O throughput was required for accommodating frequent installing / loading user-installed libraries on / from home area. Under these assumptions, we constructed this storage as a 1 PByte Lustre file system using SSDs for both meta data and data storage.

The second is storage area for storing project data. A project of ABCI 2.0 is a research project where several researchers and developers who have the same goals are involved. ABCI offers a directory for each project on the project area. In contrast to data in home where only owner can access, data in a project directory can be shared among the project members. We assumed that large capacity is highly demanded for project storage to store large amount of projects' deliverables. As a result, we constructed several file systems using Lustre and IBM Spectrum Scale whose total capacity is 32.4 PB. [*1] 10.8 PBytes of HDD-based storage added as ABCI 2.0 (DDN ES7990X) is used for this purpose. To overcome the I/O performance problems, we designed throughputs of meta data and data of the new storage were higher than the existing storages.

The last is a high-speed storage area, which is newly introduced as a part of ABCI 2.0. This area is constructed as a 0.3 PB Lustre file system using SSDs for both meta data and data storage. This area is expected to be used as a temporal directory for grand challenge applications.

2.3 Network

ABCI 2.0 uses InfiniBand for communication between compute nodes. There are two InfiniBand networks: one is for Compute Node (A) and the other is for other compute nodes. They are connected using 40 links of InfiniBand EDR between spine switches of both networks. Here, we call the former *Compute Network (A)* and the later *Compute Net-*

[*1] At the time of writing this paper, we are migrating file system from IBM Spectrum Scale to Lustre.

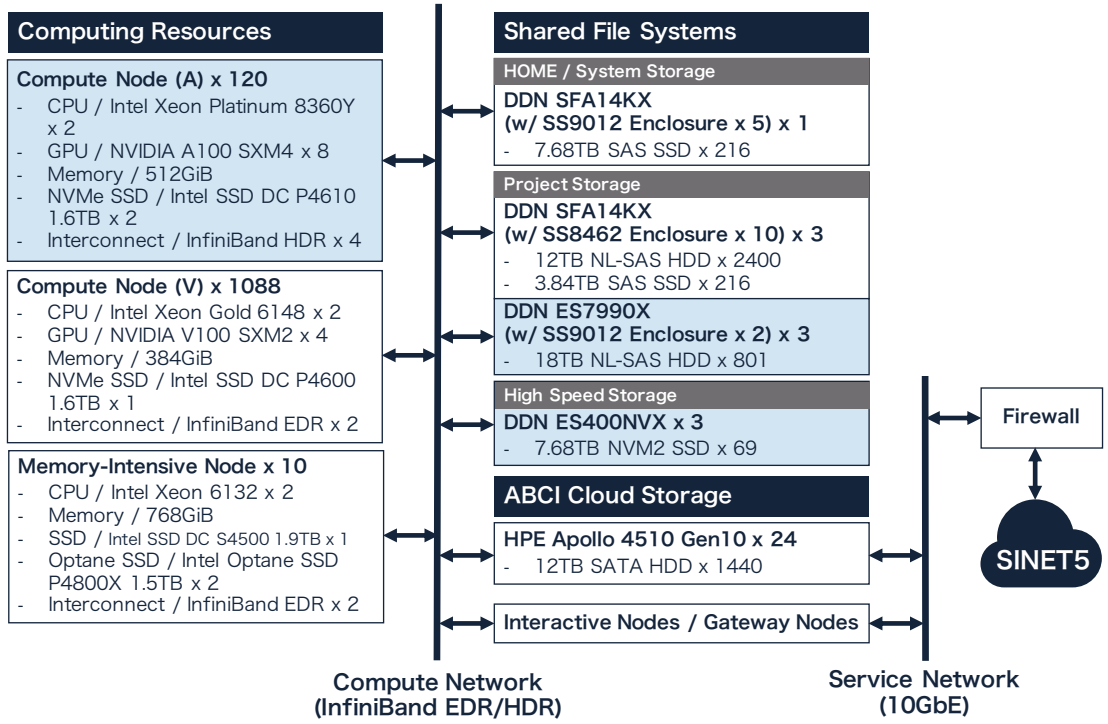


Fig. 1: Architecture of ABCI 2.0

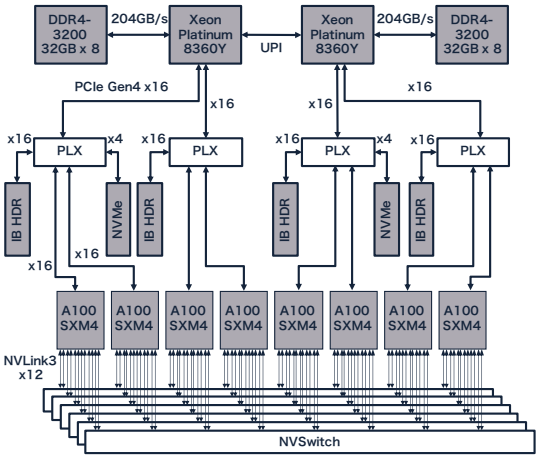


Fig. 2: Block diagram of Compute Node (A)

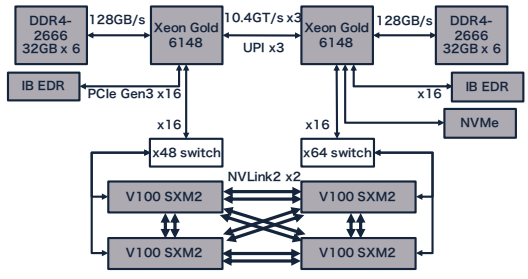


Fig. 3: Block diagram of Compute Node (V)

work (V), and we explain the later first.

Compute Network (V) is a two-level fat tree topology network in which fat trees are organized within racks and inter-racks. There are 34 nodes in each rack, and nodes are connected to a fat tree network with full-bisection bandwidth within each rack. There are 32 racks, and they are also connected to a fat tree network, however, bandwidth between each rack and spine switches is one third of aggregated bandwidth within a rack. In this network architecture, contention can occur when all 34 nodes in a rack simultaneously communicate with nodes in different racks. However, we adopted this architecture to increase cost effectiveness. The reason is that we could expect that degree of parallelism of machine learning tasks were low as previous work reported [7,8]. Actually, our analysis results of ABCI machine learning jobs showed that multi node jobs are only 2.4% [9].

Compute Network (A) also adopts a two-level fat tree topology. However, it is different from Compute Network (V) in the following points.

- All 120 Compute Node (A) are connected with full-bisection bandwidth.
- There are 5 nodes in a rack, and there are 24 racks in total. The first level fat tree is constructed using 4 racks, that is 20 nodes (We call the group as *node group*). A node in a node group can reach to other nodes in the same group by single hop.
- The second level fat tree is constructed for inter node group communication. Bandwidth between a node group and spine switches is equals to the aggregated bandwidth of nodes in a node group. There are three hops on a path between nodes in different node groups.

Compute Network (A) supports Mellanox SHARP to accelerate collective communications.

As the other network, ABCI 2.0 has *Service Network*. Most servers consist of ABCI 2.0 are isolated from the Internet. Only servers that has to be exposed to the Internet for serving their services are connected to this network.

3. ABCI 2.0 Software

We describe software for main compute resources, that is Compute Node (A) and Compute Node (V). Although some minor versions of software used on them are different, users can use both types of nodes in the same way.

3.1 System Software

System software for Compute Node (A) and Compute Node (V) is different because the former was installed three years after the latter was installed. The largest difference is OS (operating system). We intended to use CentOS 8 for both types of compute nodes, however, the end of life of CentOS 8 was announced [10]. As a result, we decided to use OS as follows: CentOS 7.5 for Compute Node (V) to sustain the same environment as previous, and Red Hat Enterprise Linux 8.2 for Compute Node (A). Users have to compile their programs for each compute node because types and versions of OS and libraries are different. As we describe in Sec. 3.3, ABCI 2.0 supports Singularity container platform, and it has potential to absorb differences in system software of two types of compute nodes.

3.2 Resource Management

To achieve high performance and high utilization, ABCI 2.0 uses a batch job scheduling system for resource management. To use ABCI, user describes a job script that calls programs he wants to run and submit it to the scheduler as a job. When resources the job requests are available, the scheduler starts the job. ABCI 2.0 uses Altair Grid Engine as the job scheduler and adopts first-come-first-serve and backfill [11] as its scheduling policy. Tasks for machine learning may be executed in many forms: interactive execution, batch execution, bag-of-tasks execution by changing training parameters. In order to support these execution forms, the scheduler supports interactive job, batch job, array job and advance resource reservation.

Each compute node in ABCI 2.0 has four or eight GPUs, however, machine learning tasks may use only 1 GPU or may not use GPUs. To efficiently schedule such jobs and increase utilization, ABCI 2.0 allocates the same compute node to multiple jobs without resource oversubscription. When user submits a job to ABCI, he has to specify one of resource types described in Table. 1. When a resource type whose amount of resource is smaller than a compute node is specified, the scheduler allocation partial resources of a node to the job by dynamically separating the node using cgroups and disk quota. When there are idle resources in a node and there are jobs whose resource requirements can fit in the idle resources, the job is scheduled to the node and share the node with other jobs.

For jobs that use multiple nodes, communication performance can be changed depending on the use of second-level fat tree, as we described in Sec. 2.3. To maximize communication performance for multi node jobs, the scheduler allocates nodes to jobs by using number of nodes within a full bisection bandwidth group as a threshold as follows.

Compute Node (V) If number of requested nodes is less than or equals to 34, unused nodes in a rack are assigned to the job. If the number is larger than 34, any unused nodes are assigned to the job.

Compute Node (A) If number of requested nodes is less than or equals to 20, unused nodes in a node group assigned to the job. If the number is larger than 20, any unused nodes are assigned to the job.

In addition to the node allocation policy, multi node jobs can enjoy a feature of temporal shared file system using local SSDs of allocated nodes. The feature uses BeeOND, BeeGFS on Demand, and performance of the created file system is not affected by I/O of other jobs and is proportional to the number of allocated nodes.

3.3 Development Environment

In addition to software provided by the OS distribution, ABCI 2.0 provides various software for development (hereafter, we call it *ABCI Software*). ABCI Software includes compilers and interpreters (e.g. Java, Python, R), GPU development tools (e.g. CUDA, cuDNN, NCCL) and MPI libraries. These ABCI Software is provided using Environment Modules [12] to enable users easily use various kinds and versions of software. Environment Modules on ABCI 2.0 is configured so that appropriate libraries are loaded depending on the loaded libraries. For example, cuDNN depends on CUDA version, and Environment Modules on ABCI 2.0 loads cuDNN that depends on an already loaded version of CUDA. The following example loads cuDNN that depends on CUDA 10.2 as CUDA 10.2 was already loaded.

```
$ module load cuda/10.2
$ module load cudnn/8.2
```

The next example loads cuDNN that depends on CUDA 11.0.

```
$ module load cuda/11.0
$ module load cudnn/8.2
```

In case of OpenMPI, if CUDA was already loaded before loading OpenMPI, a CUDA-aware version of OpenMPI that uses the CUDA is loaded.

Users can construct their development environments on ABCI 2.0 by loading ABCI Software using Environment Modules and then installing necessary libraries by their own to execute their programs. Python libraries for machine learning can be installed by Anaconda, pip or similar tools. Libraries for high performance computing can be installed using Spack [13, 14].

ABCI 2.0 supports running Linux containers using Singularity [15]. Singularity enables packaging a development environment as a container file which can be used on any Linux systems including ABCI 2.0. Singularity also supports running containers compatible with Docker on ABCI 2.0. As it supports running Docker container images as they are, ABCI 2.0 users do not have to care about container im-

Table 1: ABCI 2.0 Resource Types

Resource Type	Node Type	#CPU cores	#GPUs	Memory (GiB)	Local SSD (GiB)	Max Resources / Job
V Full	Compute Node (V)	40	4	360	1440	512
G.large	Compute Node (V)	20	4	240	720	1
G.small	Compute Node (V)	5	1	60	180	1
C.large	Compute Node (V)	20	0	120	720	1
C.small	Compute Node (V)	5	0	30	180	1
A Full	Compute Node (A)	72	8	480	3440	64
A G.small	Compute Node (A)	9	1	60	390	1

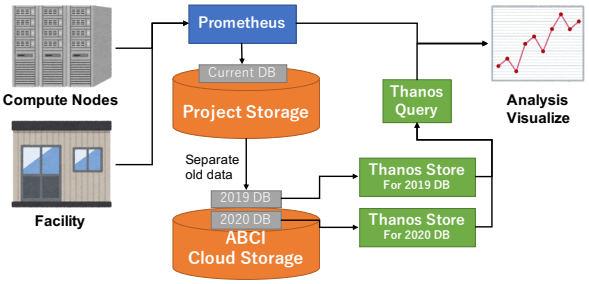


Fig. 4: Resource Monitoring for Long Term Storage

age format. However, if a Docker container is provided only in Dockerfile format, users have to convert it to a Singularity recipe before building a container image. An open source software, named Singularity Python [16], simplifies the conversion.

3.4 Monitoring System

ABCI 2.0 has two monitoring systems. One is *Resource Monitoring* which monitors and collects metrics from servers and services that construct ABCI. The other is *Usage Monitoring* which collects information about how users use ABCI 2.0.

We use Zabbix and Prometheus [17] for Resource Monitoring. Zabbix monitors all ABCI 2.0 components and mainly used for alerting. Incidents recognized as faults are manually recorded in a fault history database. However, Zabbix is not configured to store raw metrics for long term for a use of later analyses. We use a Prometheus based system for this purpose. To create a database that stores operation status of ABCI 2.0, Prometheus is configured to collect metrics from compute nodes and power and cooling facilities in one minute interval. Most of the metrics come from compute nodes. Specifically, Prometheus collects metrics about CPU cores, GPUs and various sensors from a compute node, which results in 1393 points for a Compute Node (V) and 2411 points for a Compute Node (A). That means there are more than 1.5 million metrics collected every minute. As the amount of annual data exceeds 1.5 TB, we separate the database as depicted in Fig. 4. Only latest metrics of compute nodes and metrics of facilities are recorded in Prometheus. Old metrics of compute nodes are removed from Prometheus and archived in ABCI Cloud Storage. To use the metrics in ABCI Cloud Storage as the same interface with Prometheus, we use Thanos [18].

For the purpose of Usage Monitoring, we collect two types of data. One is attributes of all jobs submitted to the scheduler which include followings: User ID, Project ID, Resource Type, Amount of Resources, Request Walltime, Actual Walltime, Used CPU Core IDs, Used GPU IDs, etc. The other is ABCI Software used by each job. The log system logs all "module load"ed ABCI Software with its name and version for each job and then associates them with job attributes. We analyzed usage of ABCI 2.0 by using these information [9]. We feed the results of our analyses back to operation as new job scheduling parameter settings or kind and versions of software provided as ABCI Software.

Table 2: Software Used for Performance Measurement

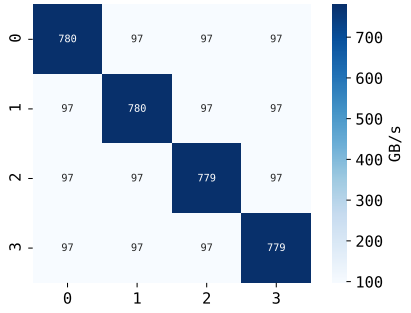
Software	Compute Node (A)	Compute Node (V)
OS	RHEL 8.2	CentOS 7.5
gcc	8.3.1	7.4.0
Singularity Pro	3.7	←
CUDA	11.2.2	←
GDRCopy	2.1	2.0
OpenMPI	4.0.5	←
UCX	1.9.0	1.7.0
NCCL	2.8.4-1	←
Horovod	0.22.0	←
OSU Micro-Benchmarks	5.7	←
NCCL Tests	Commit:0b30de5	←
fio	3.26	←
NVIDIA PyTorch Container Image	21.04-py3	←

4. Performance Evaluation

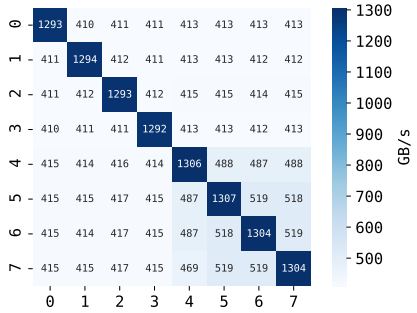
We evaluated performance of both compute nodes of ABCI 2.0 to compare their performance. We evaluated computation and communication where GPU is involved. As our previous work showed [9], most of ABCI jobs were single node or single GPU jobs. As a result, we focused on showing performance of general use of ABCI 2.0 and evaluated using small number of nodes. Specifically, the maximum number of nodes used in the evaluation is limited by number of nodes in the first level of the two-level fat tree: 16 nodes for Compute Node (A) and 32 nodes for Compute Node (V). Table. 2 shows software and versions used in the evaluation. We used software already installed in ABCI 2.0 or published on the Internet. We plan to publish procedures and scripts used for the evaluation and all results.

4.1 Intra-Node GPU Point-to-Point Communication

We measured bandwidth of GPU memory access and inter-GPU communication using `p2pBandwidthLatencyTest` attached to CUDA. We

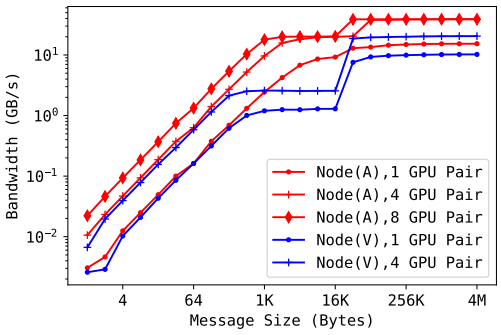


(a) Compute Node (V) - NVIDIA V100

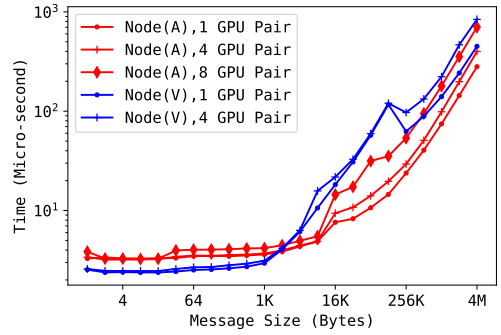


(b) Compute Node (A) - NVIDIA A100

Fig. 5: Bidirectional Bandwidth of Inter-GPU Communication



(a) Bandwidth



(b) Latency

Fig. 6: Inter-Node GPU Point-to-Point Communication

selected 10 nodes from each node type to measure performance. The average bandwidth is shown in Fig. 5. Compute Node (A) achieves 1.66 times higher memory bandwidth and 4 to 5 times higher inter-GPU communication bandwidth against Compute Node (V). In contrast, Compute Node (V) achieved higher ratios against the peak performance. Specifically, the peak memory bandwidth of Compute Node (A) is 1,555 GB/s and that of Compute Node (V) is 900 GB/s. From the evaluation results, we can find that the performance ratio against peak is 83% for Compute Node (A) and 86% for Compute Node (V). In terms of inter-GPU communication, peak performance of Compute Node (A) is 600 GB/s and that of Compute Node (V) is 100 GB/s. so, the peak performance ratios are 68 to 86% for Compute Node (A) and 97% for Compute Node (V).

4.2 Inter-Node GPU Point-to-Point Communication

We measured bandwidth and latency between two nodes by changing number of communicating GPU pairs and message size. We used a CUDA-aware OpenMPI for MPI library and used `osu_mbw_mr` of OSU Micro-Benchmarks [19, 20] to transfer between inter-GPU memories.

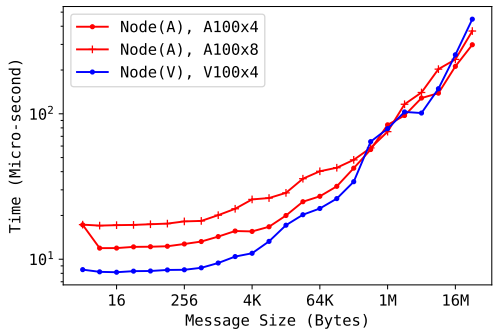
The results of bandwidth are shown in Fig. 6a. When we compare performance by fixing the number of GPU pairs, Compute Node (A) and (V) have the similar performance until 256 Bytes messages. Compute Node (V) had higher ratios against peak performance. Compute Node (A) has 100 GB/s as its peak bandwidth and its maximum measured bandwidth was 39.02 GB/s; the ratio is 39%. In contrast, Compute Node (V) has 25 GB/s as its peak bandwidth and its maximum bandwidth was 20.48 GB/s; the ratio is 82%. Fig. 6b shows the latency results. Compute Node (V) has smaller latency than Compute Node (A) until 2 KBytes messages.

4.3 Inter-GPU Allreduce

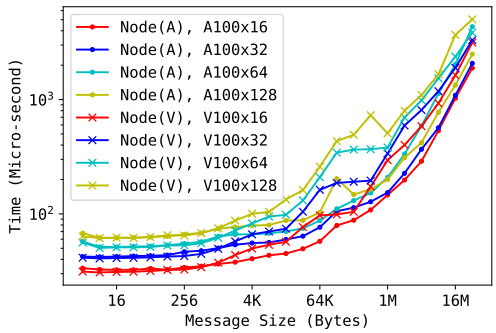
We measured Allreduce between GPUs using NCCL. We used `all_reduce_perf` from NCCL Tests [21] as the benchmark program. Fig. 7a shows the performance using a node. To compare performance using the same number of GPUs, we also show results using 4 GPUs for Compute Node (A). We can see the Compute Node (V) is superior to Compute Node (A) if message size is less than 256 KBytes. Fig. 7b shows the performance using multiple nodes. There are no performance differences between node types when message sizes are less than 1 KBytes, however, Compute Node (A) outperforms Compute Node (V) when message size becomes larger. The expected reasons are not only performance improvements of communication and memory access performance, but also that Compute Node (V) need twice count of nodes against Compute Node (A) to use the same number of GPUs.

4.4 Single Node Training

To measure performance of training on a single node,



(a) Single Node



(b) Multi Node

Fig. 7: Inter-GPU Allreduce

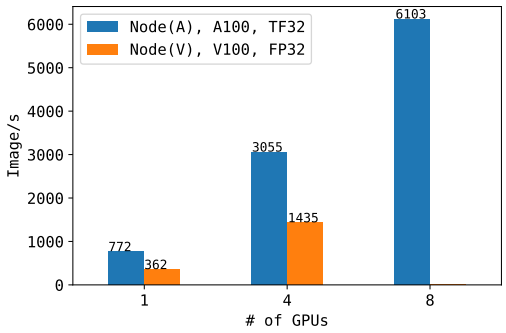


Fig. 8: Single Node Training of ResNet50

Table 3: Speedup of Node (A) against Node (V)

(a) By Changing # of Nodes

#Nodes	1	2	4	8	16
Speedup	4.10	3.89	3.68	3.56	3.09

(b) By Changing # of GPUs

#GPUs	8	16	32	64	128
Speedup	2.10	2.00	1.96	1.82	1.64

we used a training program of ResNet-50 [22] attached to NVIDIA's DeepLearningExamples [23] to measure count of processed images per second. We used a container image published on NVIDIA'S NGC Catalog that includes PyTorch optimized for NVIDIA GPUs. The image also includes the above training program so that we can get results by just running a Singularity container using the image on ABCI 2.0. The program was executed using a standard benchmark parameter set, however, we changed batch sizes per GPU as follows depending on amount of GPU memory: 256 for Compute Node (A) and 128 for Compute Node (V). We used ImageNet ILSVRC2012 [24] dataset as input and used local SSD of compute node to store data.

Fig. 8 shows the results. Compute Node (A) achieved 2.13 times higher performance than Compute Node (V) when the same number of GPUs are used. In terms of scalability, performances of both node types almost linearly scale against number of GPUs. We show that this performance is not limited by I/O performance of local SSDs as follows. As the average image size of ILSVRC2012 dataset is 114 KBytes, amount of I/O per second of Compute Node (A) and (V) can be calculated as 340 MBytes and 160 MBytes in case of 4 GPUs. In contrast, random read performance of local SSD measured by fio [25] was 1,650 MB/s and 942 MB/s for Compute Node (A) and (V), which were large enough.

By comparing the whole node performance, we can see that Compute Node (A) has 4.25 times higher performance than Compute Node (V). If we do a simple math, adding 120 Compute Node (A) is equals to adding 510 Compute Node (V) for this task. As most jobs run on ABCI uses equal to or less than 1 node, adding nodes directly contributes to improve task processing throughput. We can extrapolate the throughput increase ratio of this task is 1.46 ($\frac{1088+510}{1088}$). We expect that ABCI 2.0 also increases throughput for other tasks by equivalent or higher level.

4.5 Multi Node Training

To evaluate training performance using multiple nodes, we used PyTorch and Horovod [26,27] to measure training time of ResNet-50 in data parallel. We used a ResNet-50 model training example program of Horovod as a benchmark program and run it in a Singularity container. We created a container image where Horovod is installed using the image described in Sec. 4.5 as a base image because the image does not include Horovod. We used a standard parameter set, such as 90 epochs, except for batch size per GPU: 256 for Compute Node (A) and 96 for Compute Node (V). We used ImageNet ILSVRC2012 dataset as input and used local SSD of compute node to store data.

Fig. 9 shows training times and Table. 3 shows speedup of Compute Node (A) against Compute Node (V). We can see from Table. 3a that Compute Node (A) has 3.1 to 4.1 times higher performance than Compute Node (V) in terms of node performance. When we compare performance using the same number of GPUs, the performance improvement of Compute Node (A) is 1.6 to 2.1 as shown in Table. 3b; it is a similar trend with Sec. 4.4. Table. 4 shows scalability. Compute Node (V) has better scalability than Compute Node (A).

5. Summary

From the start of its operation on August 2018, ABCI is used in basic AI research and applied research areas that

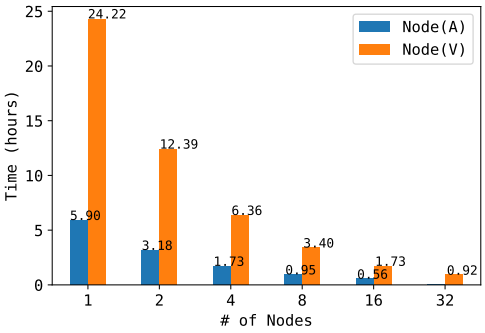


Fig. 9: Multi Node Training of ResNet50

Table 4: Relative Performance Gain

(a) Against #Node=1

#Nodes	1	2	4	8	16	32
Node (A)	1.00	1.86	3.41	6.19	10.52	-
Node (V)	1.00	1.96	3.81	7.13	13.97	26.26

(b) Against #GPU=8

#GPUs	8	16	32	64	128
Node (A)	1.00	1.86	3.41	6.19	10.52
Node (V)	1.00	1.95	3.65	7.14	13.43

use AI. Because of a lack of computing and storage capacity caused by increased demands, we added 120 compute nodes equipped with eight NVIDIA A100 GPUs and 11 PBytes storage to ABCI and started its operation as ABCI 2.0 from May 2021. To compare performance of existing and newly added compute nodes, we conducted micro-benchmarks focusing on measuring communication performance and benchmarks to train deep learning models. As a result, we confirmed that the new node outperformed existing node, and new node had up to 4.1 times higher performance than existing node when training ResNet-50 model using PyTorch. We can extrapolate that ABCI 2.0 has 1.46 times higher throughput for processing this task than ABCI, and expect that such a throughput gain can be observed in other AI tasks. We hope that ABCI 2.0 will greatly contribute to developments of future AI technologies.

References

[1] : ABCI, `https://abci.ai/`.
[2] Yamazaki, M., Kasagi, A., Tabuchi, A., Honda, T., Miwa, M., Fukumoto, N., Tabaru, T., Ike, A. and Nakashima, K.: Yet Another Accelerated SGD: ResNet-50 Training on ImageNet in 74.7 seconds, (online), available from ⟨http://arxiv.org/abs/1903.12650⟩ (2019).
[3] Mikami, H., Suganuma, H., U-chupala, P., Tanaka, Y. and Kageyama, Y.: Massively Distributed SGD: ImageNet/ResNet-50 Training in a Flash, (online), available from ⟨https://arxiv.org/abs/1811.05233⟩ (2018).
[4] Osawa, K., Tsuji, Y., Ueno, Y., Naruse, A., Yokota, R. and Matsuoka, S.: Large-Scale Distributed Second-Order Optimization Using Kronecker-Factored Approximate Curvature for Deep Convolutional Neural Networks, *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2019).
[5] : ABCI Datasets, `https://datasets.abci.ai/`.
[6] Tanimura, Y., Takizawa, S., Ogawa, H. and Hamanishi, T.: Building and Evaluation of Cloud Storage and Datasets Services on AI and HPC Converged Infrastructure, *The 9th Workshop on Scalable Cloud Data Management, co-located with 2020 IEEE International Conference on Big Data (Big Data)*, pp. 1992–2001 (2020).
[7] Wang, M., Meng, C., Long, G., Wu, C., Yang, J., Lin, W. and Jia, Y.: Characterizing Deep Learning Training Workloads on Alibaba-PAI, *arXive*, (online), available from ⟨http://arxiv.org/abs/1910.05930⟩ (2019).
[8] Jeon, M., Venkataraman, S., Phanishayee, A., Qian, J., Xiao, W. and Yang, F.: Analysis of large-scale multi-tenant GPU clusters for DNN training workloads, *Proceedings of the 2019 USENIX Annual Technical Conference* (2019).
[9] Takizawa, S., Sakabe, A., Tanimura, Y. and Ogawa, H.: ABCI 上でのジョブ実行履歴の分析による深層学習計算の傾向把握，第 176 回ハイパフォーマンスコンピューティング研究会 (2020).
[10] : CentOS Project shifts focus to CentOS Stream, `https://blog.centos.org/2020/12/future-is-centos-stream/`.
[11] Mu'alem, A. W. and Feitelson, D. G.: Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling, *IEEE Transactions on Parallel and Distributed Systems*, Vol. 12, No. 6, pp. 529–543 (2001).
[12] : Environment Modules, `http://modules.sourceforge.net/`.
[13] Gamblin, T., LeGendre, M., Collette, M. R., Lee, G. L., Moody, A., de Supinski, B. R. and Futral, S.: The Spack package manager: bringing order to HPC software chaos, *SC '15: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–12 (2015).
[14] : Spack, `https://spack.io/`.
[15] Kurtzer, G. M., Sochat, V. and Bauer, M. W.: Singularity: Scientific containers for mobility of compute, *PLoS ONE*, Vol. 12, No. 5 (2017).
[16] : Singularity Python, `https://github.com/singularityhub/singularity-cli`.
[17] : Prometheus, `https://prometheus.io/`.
[18] : Thanos - Highly available Prometheus setup with long term storage capabilities, `https://thanos.io/`.
[19] Bureddy, D., Wang, H., Venkatesh, A., Potluri, S. and Panda, D. K.: OMB-GPU: A Micro-Benchmark Suite for Evaluating MPI Libraries on GPU Clusters, *Recent Advances in the Message Passing Interface*, pp. 110–120 (2012).
[20] : OSU Micro-Banchmarks, `http://mvapich.cse.ohio-state.edu/benchmarks/`.
[21] : NCCL Tests, `https://github.com/NVIDIA/nccl-tests`.
[22] He, K., Zhang, X., Ren, S. and Sun, J.: Deep Residual Learning for Image Recognition, *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016).
[23] : NVIDIA Deep Learning Examples for Tensor Cores, `https://github.com/NVIDIA/DeepLearningExamples`.
[24] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C. and Fei-Fei, L.: ImageNet Large Scale Visual Recognition Challenge, *International Journal of Computer Vision*, Vol. 115, No. 3, pp. 211–252 (2015).
[25] : fio, `https://github.com/axboe/fio`.
[26] Sergeev, A. and Balso, M. D.: Horovod: fast and easy distributed deep learning in TensorFlow (2018).
[27] : Horovod, `https://github.com/horovod/horovod`.