PROCEEDINGS OF SCIENCE

PoS(ISGC2019)008

Building an elastic batch system with private and public clouds

Wataru Takase*
High Energy Accelerator Research Organization (KEK)
E-mail: wataru.takase@kek

Tomoaki Nakamura
High Energy Accelerator Research Organization (KEK)
E-mail: tomoaki.nakamura@kek.jp

Koichi Murakami
High Energy Accelerator Research Organization (KEK)
E-mail: koichi.murakami@kek.jp

Takashi Sasaki
High Energy Accelerator Research Organization (KEK)
E-mail: takashi.sasaki@kek.jp

The Computing Research Center at High Energy Accelerator Research Organization (KEK) provides a Linux cluster consisting of 350 physical servers with 10000 CPU cores for the scientific data analysis and numerical simulation. All of the computing resources are shared by using a batch system among the various projects supporting at KEK. We have adopted IBM Spectrum LSF as the batch system for many years, and users in KEK are well familiar with the system. However, when the computing cluster is becoming congested by the lack of resource, user jobs make a long stay in a job queue. As a result, the turnaround time often becomes longer for each job to be finished. Furthermore, we have to consider the different requirements for processing environments depending on users/groups.

Providing flexible resources both in terms of computing power and environment, we have investigated the possibility of cloud computing technology. IBM Spectrum LSF has an optional functionality so-called Resource Connector which enables us to utilize additional computing resources in external cloud providers. By using that functionality, we have succeeded to integrate our batch system with on-premise OpenStack cloud and Amazon Web Services (AWS) by the collaboration with National Institute of Informatics (NII), Japan.

Users' jobs submitted into the dedicated job queue are going to be dispatched to dynamically provisioned instances on on-premise OpenStack, AWS, as well as static local computing node. Any kind of job processing environments can be utilized by choosing a different virtual machine according to the user's request. The OpenStack based private cloud has been integrated with our LDAP service and GPFS for user authentication and data sharing on our batch system, respectively. Amazon Simple Storage Service is utilized for the data exchange between KEK and AWS. Both the physical servers on the batch system at KEK and provisioned instances on AWS mount the S3 bucket via FUSE for transparent data access. We conducted some performance tests on our hybrid batch system for investigation of the scalability and succeeded to execute a Deep Learning job using 3500 cores on AWS.

In this talk, we would like to present the detailed configuration of the hybrid system and some results of the performance tests.

International Symposium on Grids & Clouds 2019, ISGC2019
31st March - 5th April, 2019
Academia Sinica, Taipei, Taiwan

*Speaker.

1. Introduction

A batch system makes it possible to share large computing resources among multiple users. A user submits a set of programs (a batch job) to a job queue. After waiting until the requested resources become available, it is processed on dispatched computers.

The Computing Research Center at High Energy Accelerator Research Organization (KEK) provides a batch system for the various projects supporting at KEK. It consists of 350 physical servers with 10000 CPU cores for the scientific data analysis and numerical simulation [1]. All the Operating Systems are Scientific Linux 6. We have used IBM Spectrum LSF [2] as the batch scheduler for many years, and users in KEK are well familiar with the system.

There are two challenges in the system. One is to deal with piled up waiting jobs due to the resource limitation. When the system is becoming congested, jobs make a long stay in job queues. As a result, turnaround times become longer. The other is to react to requests for various environments. We have received several requirements for specific job processing environments from our users.

In this paper, we provide a solution to resolve the challenges by integration of the batch system and cloud computing technology. This paper starts with a description of how we integrate the batch system with private and public clouds. Section 3 describes some performance test results on AWS. An automatic offloading to the AWS cloud is demonstrated in Section 4, and finally in Section 5, we provide a summary of the work.

2. Cloud-integrated batch system

Cloud computing enables flexible provisioning of computing resources by utilizing virtual machines on demand and can provide various data analysis environments. In order to resolve the two challenges in our batch system, we have investigated the possibility of using the following technology.

IBM Spectrum LSF has an optional functionality so-called Resource Connector [3] which enables us to utilize additional computing resources in external cloud providers, such as Amazon Web Services (AWS), Microsoft Azure, OpenStack. By using that functionality, we have succeeded to integrate our batch system with on-premise OpenStack cloud and AWS by the collaboration with National Institute of Informatics (NII), Japan. The system enables to expand its computing resources and provide various job processing environments dynamically.

Figure 1 shows the overview of the cloud-integrated system. We prepared cloud-specific batch queues for each cloud. If a user submits a job to the OpenStack queue, Resource Connector creates instances on the on-premise OpenStack cloud based on the user's request. Then, the job is dispatched to them. After a certain period, unnecessary instances are deleted automatically by Resource Connector. If a user submits a job to the AWS queue, it is going to be executed on instances on AWS in a similar fashion.

For the integration a cloud administrator needs the following actions:

1. Create an instance on the target cloud. Then, install the necessary software and configure on it for job processing. Finally, create an instance image from the configured instance.

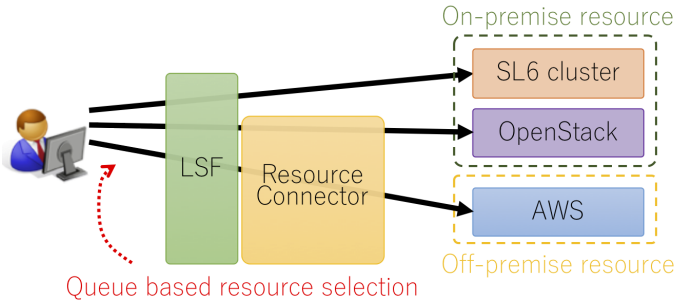

Figure 1: Overview of the cloud integrated batch system.

2. Create a Resource Connector template containing the target cloud information such instance image ID created previous step, instance type, subnet (see Figure 2).

3. Create a batch queue and link to the template.

If a job is submitted to the queue, the linked template is referred. Instances are provisioned on the target cloud based on the template information.

```
{
  "templates": [
    {
      "templateId": "Template-AWS-4cores",
      "attributes": {
        "type": ["String", "X86_64"],
        "ncpus": ["Numeric", "4"],
        "awshost": ["Boolean", "1"],
        "gpu": ["Boolean", "0"],
        "ami": ["String", "0123456789abcdef0"]
      },
      "imageId": "ami-abcdefabcdefabcde",
      "subnetId": "subnet-01234567",
      "vmType": "m4.xlarge",
      "maxNumber": "20",
      "keyName": "key-pair",
      "securityGroupIds": ["sg-abcdef01"],
      "instanceTags": "role=lsf",
      "userData": "ami=abcdefabcdefabcde"
    },
    {
      "templateId": "Template-AWS-4cores-GPU",
      "attributes": {
```

Figure 2: An example of a Resource connector template.

2.1 Integration with OpenStack

We have provided OpenStack based private cloud pre-production service at KEK. The cloud has been integrated with the batch system. We prepared some OpenStack images based on the following Operation Systems: Scientific Linux 6, CentOS 7, and Ubuntu 16.04. Managers of each

project supported at KEK are granted permission to create custom ones based on those images for various job processing environments.

The OpenStack cloud has also been integrated with our LDAP service and IBM GPFS (General Parallel File System). We configured the OpenStack to use the LDAP for the authentication and Linux account management as with the batch system. GPFS is used by the batch system to share some directories such user's home, and project shared spaces. We configured OpenStack instances to share the GPFS file system with the Scientific Linux 6 physical servers. We did not make instances to mount the GPFS file system directly because GPFS requires to register a client machine information to be connected. In order to avoid the additional GPFS operation at every instance creation, the OpenStack physical compute nodes mount the GPFS file system and expose necessary Linux directories to instances via NFS.

2.2 Integration with AWS

An LSF batch scheduler requires communication with each job processing server. We only assign a private IP address to each instance on AWS and avoid public IP address allocation for security reasons. For communication between KEK and AWS machines, we set up a VPN server on AWS. The server is only accessible from KEK. The batch scheduler in KEK can connect to instances on AWS through the server.

Although instances share an NFS file system inside of the AWS network for coordinated job processing, there was no path to share a file system between KEK and AWS. Therefore, we needed to consider the way of the data sharing.

2.2.1 Data sharing between KEK and AWS

The batch system and the on-premise OpenStack share the GPFS file sytem in KEK. Therefore, an instance on the OpenStack can read input data from the file system and write output to it as the same manner of the Scientific Linux 6 cluster. Instances on AWS cannot mount the same GPFS file system because the file system doesn't export any mount point to outside of KEK for security.

We decided to use AWS Simple Storage Service (S3) for the data sharing. It provides object storage service and accessible from the Internet and also from instances on AWS. There are a Web user interface and a specific command line interface for S3 data uploading and downloading. However, we preferred to provide transparent remote file access to our users. S3FS [4] or Goofys [5] allows Linux to mount an AWS S3 bucket via FUSE. A user can upload and download data with Linux *cp* command by using those tools. Job submission servers on KEK and instances on AWS have been configured to mount the same S3 bucket for the data sharing. The data exchanging workflow is as follows: At first, a user puts input data to the S3 bucket. Secondly, the user submits a job which executes the user's application and writes output to the bucket. Lastly, the user get output from the bucket.

We compared upload and download speeds between using S3FS and Goofys. Table 1 shows four datasets used for the tests. *cp* command was used to upload and download each of the datasets. The results are shown in Figure 3 and Figure 4. The Goofys upload performance is better than the S3FS. Especially, uploading speed of more than 100 MB files is 1.7 times faster. On the other hand, in terms of POSIX compatibility, S3FS has more than Goofys; for example, S3FS enables

Table 1: Prepared datasets for uploading/downloading tests.

Dataset	Number of files	Each file size
Dataset 1	1000	1 MB
Dataset 2	100	10 MB
Dataset 3	10	100 MB
Dataset 4	1	1000 MB

random access and better metadata handling. In the case of requiring less POSIX compatibility, we consider using Goofys is a good solution.

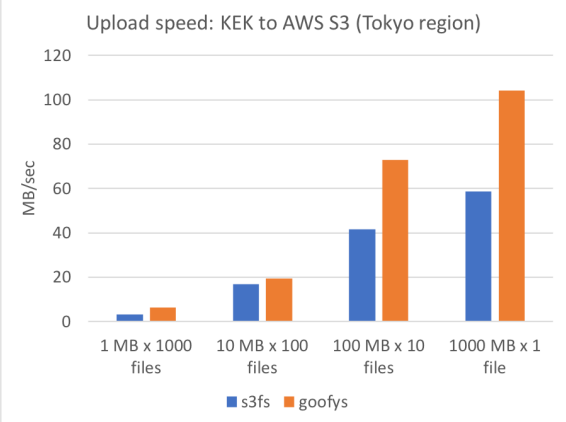

Figure 3: Upload speed from KEK to AWS.

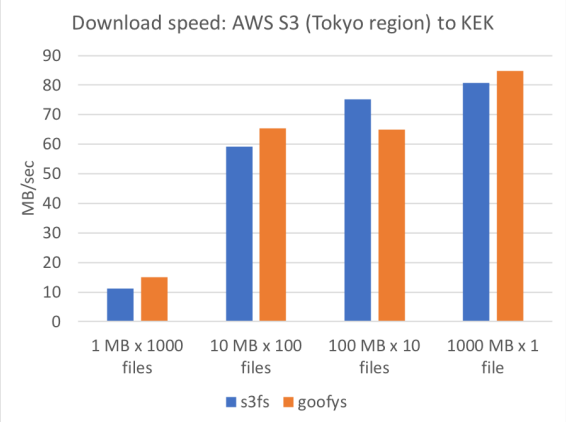

Figure 4: Download speed from AWS to KEK.

2.2.2 Monitoring resource transition on AWS

AWS offers a pay-as-you-go service. It's important for us to monitor resource utilization on AWS. We set up a monitoring system using Elasticsearch, Logstash, Kibana, and Beats. Launched instances send own system information to an Elasticsearch server by Beats and Logstash. The batch system administrator can check resource utilization on a Kibana dashboard. In the dashboard (see Figure 5), we can check number of running instances (the upper left), number of available CPU cores (the upper middle), a transition of using the number of cores (the upper right), CPU utilization (the lower left), and workload (the lower right). The upper right chart describes that available CPU cores were increased by instances creation based on users' requests. Moreover, it shows that number of CPU cores were decreased by unnecessary instances deletion after jobs finished.

3. Evaluation of the AWS cloud performance

We measured scalability of the AWS environment we set up by executing two kinds of batch jobs. The results are described the following sub-sections.

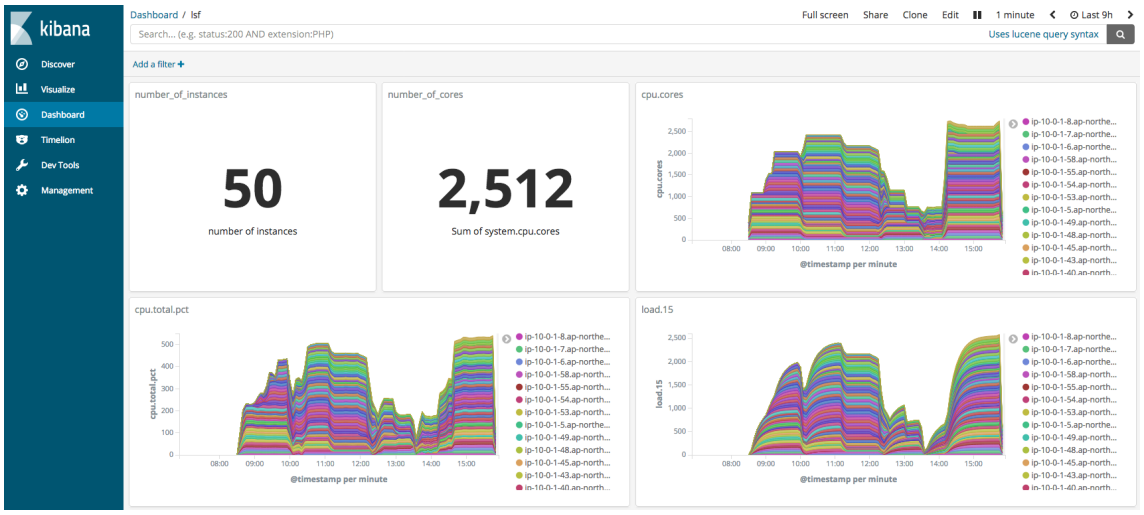

Figure 5: Monitoring resource usage on AWS by Elastic Stack.

3.1 Submission of Monte Carlo simulation jobs to AWS

We submitted Geant4 [6] based Particle therapy Monte Carlo simulation [7] jobs to the AWS queue. The simulation shoot 2 million protons in total against a treatment head with patient data obtained from CT images and outputs simulated dose distribution. The simulation workload can be distributed on multiple CPU cores. We measured the simulation times by changing the number of using CPU cores. The same jobs were submitted to a KEK queue which only consumes the KEK resource for the comparison.

Figure 6 shows measured simulation times to be finished. The blue and orange correspond to the results on KEK and AWS respectively. Although the AWS simulation times took longer than the result on KEK, we observed the same tendencies that the simulation times become shorter by increasing the number of CPU cores used.

We consider the NFS leads to degrading the performance on AWS. In the simulation, each CPU core reads input data from a shared directory and outputs the result to it. KEK local environment uses GPFS which enables users to access data with high throughput. On the other hand, the single NFS server we prepared on AWS may downgrade read and write performances at the time of parallel data access.

3.2 Submission of Deep Learing jobs to AWS

We also submitted TensorFlow [9] based Deep Learning training jobs to the AWS queue. The job builds Convolutional Neural Network, then trains to classify CIFAR-10 [8] images into ten categories. TensorFlow enables distributed training by constructing a cluster. A cluster consists of two kind of servers, parameter servers and worker hosts. Parameter servers store and update the network parameters. Worker hosts caluculate network loss. As increasing the number of worker hosts, the training speed becomes faster.

We set up a cluster consisting of a single parameter server and several worker hosts. Then, we measured training time by changing the number of worker hosts. The result is shown in Figure 7. With 64 CPU cores, it took more than 23,000 seconds to train. As using CPU cores doubled,

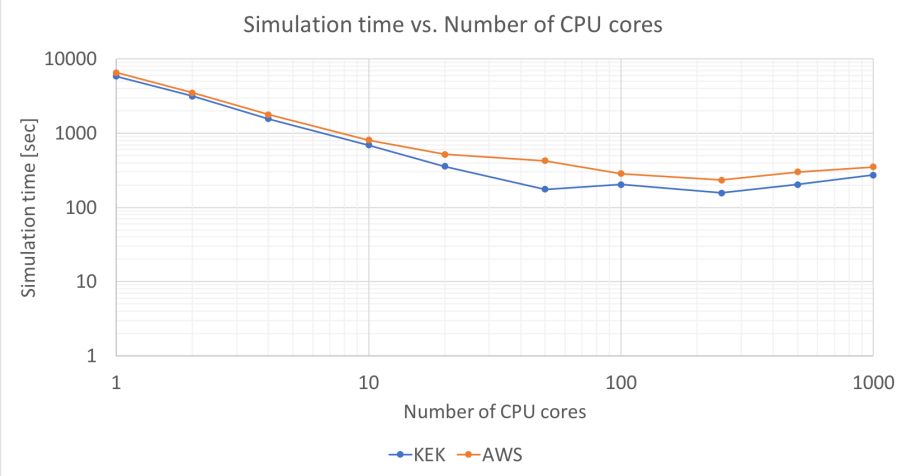

Figure 6: Comparison of Monte Carlo simulation times.

training time becomes halved. We confirmed the training speed performance is scaled well up to 1920 CPU cores. Although we increased using the number of CPU cores to 3648, the performance did not improve further. In clustered TensorFlow training, each worker host repeats to calculate the network loss and to report it to the parameter server during the training. We consider that parameter exchange leads the traffic congestion. About this thing, we need a further investigation that the increasing number of parameter servers further improves the performance or not.

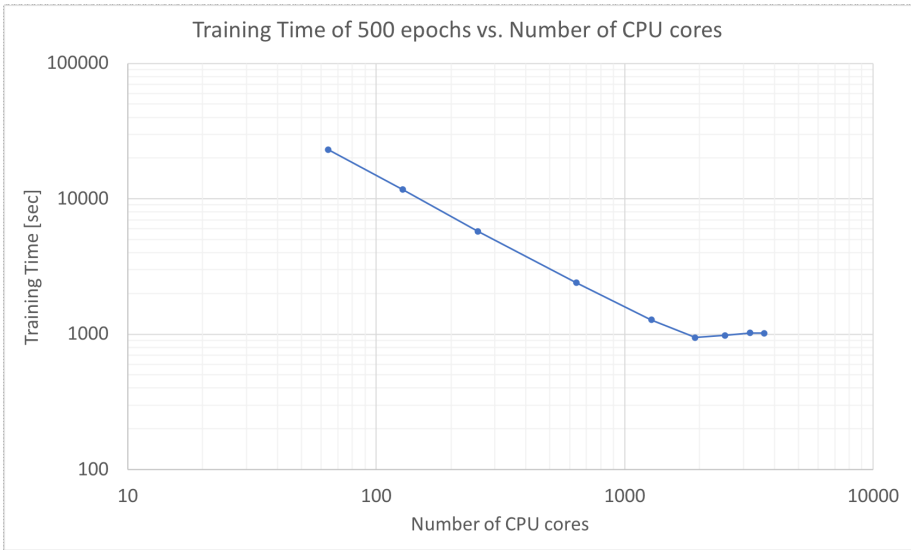

Figure 7: Deep Learning training speed.

4. Automatic offloading to AWS

The previous section described the scalability of the AWS environment by submission of jobs to the AWS queue which consumes only AWS resource. We prepared another queue which con-

sumes both KEK and AWS resources. A job submitted to the queue runs on servers in KEK if the resource is available; otherwise, it's going to be dispatched to instances dynamically launched on AWS. Using the queue, we have succeeded to offload some batch workloads to the external cloud dynamically at the time of congestion.

Figure 8 visualizes transitions of individual status of 3000 jobs submitted to the queue at the same time. First, some jobs are dispatched to KEK servers, and there is no more free resource on KEK. Then, Resource Connector launches AWS instances, and part of the other jobs run on them. The batch scheduler dispatches some jobs to a newly found resource on KEK. Finally, the others run on AWS instances.

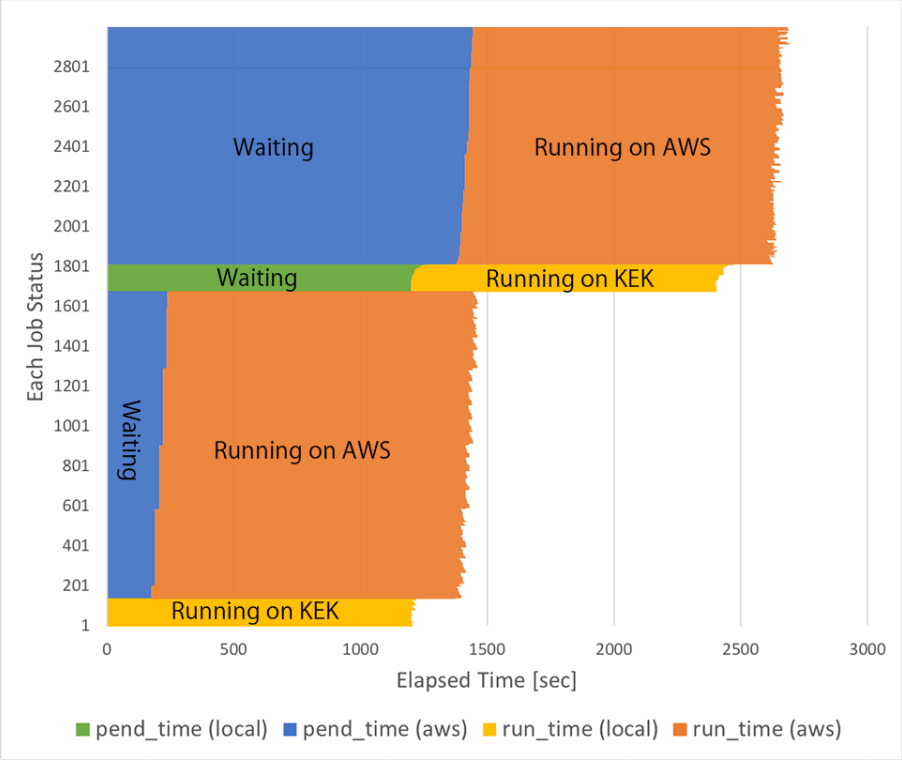

Figure 8: Visualization of automatic offloading.

5. Summary

This paper showed the integration of the IBM Spectrum LSF based KEK batch system with on-premise OpenStack and AWS clouds using the Resource Connector functionality. We prepared some cloud-specific batch queues and linked them to the Resource Connector templates. A job submitted to the cloud dedicated queue is dispatched to instances provisioned on the linked cloud.

We submitted Monte Carlo simulation jobs to the AWS queue and confirmed working well with slight performance degradation due to NFS. Deep Learning training jobs also worked well and scaled up to 1920 CPU cores. In addition, we demonstrated the automatic offloading some batch workloads to the AWS cloud dynamically.

The integrated system enables the KEK batch system to expand computing resources to the cloud, reducing turnaround times of jobs at the time of congestion. Moreover, the system provides any kind of job processing environments by choosing a difference instance image.

Acknowledgments

Cloud resources used in this work was provided in the Demonstration Experiment of Cloud Use conducted by National Institute of Informatics (NII) Japan (FY2017).

References

[1] K. Murakami, G. Iwai, T. Sasaki, T. Nakamura, and W. Takase, *System Upgrade of the KEK Central Computing System*, in *Journal of Physics: Conference Series, Volume 898, 082038*, 2017.

[2] "IBM Knowledge Center - IBM Spectrum LSF." `https://www.ibm.com/support/knowledgecenter/en/SSWRJV/product_welcome_spectrum_lsf.html`.

[3] "IBM Knowledge Center - Using the IBM Spectrum LSF Resource Connector." `https://www.ibm.com/support/knowledgecenter/en/SSWRJV_10.1.0/lsf_welcome/lsf_kc_resource_connector.html`.

[4] "s3fs-fuse/s3fs-fuse: FUSE-based file system backed by Amazon S3." `https://github.com/s3fs-fuse/s3fs-fuse`.

[5] "kahing/goofys: a high-performace, POSIX-ish Amazon S3 file system written in Go." `https://github.com/kahing/goofys`.

[6] S. Agostinelli, J. Allison, K. Amako, and et al., *Geant4, A Simulation Toolkit*, in *NIM A 506: 250-303*, 2003.

[7] T. Aso, A. Kimura, S. Kameoka, and et al., *Geant4 based simulation framework for particle therapy system*, in *Nucl. Sci. Symp. Conf. Record. NSS' 07. IEEE: N60-1*, 2007.

[8] "CIFAR-10 and CIFAR-100 datasets." `https://www.cs.toronto.edu/~kriz/cifar.html`.

[9] "TensorFlow: Large-scale machine learning on heterogeneous systems." `https://www.tensorflow.org`.