CloudQ: A Secure AI / HPC Cloud Bursting System

Shinichiro TAKIZAWA† Masaaki SHIMIZU‡ Hidemoto NAKADA† Hiroya MATSUBA‡ Ryousei TAKANO†
† *National Institute of Advanced Industrial Science and Technology* ‡ *Hitachi, Ltd.*
Tokyo, Japan Tokyo, Japan

***Abstract*—As a method to optimize the investment for computational resources, cloud bursting is collecting a lot of attention, where the organizations utilize the cloud computing environment in on-demand fashion, while preserving the minimum amount of on-premise resources for sensitive data processing. For the practical cloud bursting, we need to achieve 1) secure job / data sharing, 2) uniform job execution environment for on-premise and cloud, and 3) on-demand automatic deployment of the execution environment on the cloud. To enable these items, we propose a meta-scheduling system called CloudQ. CloudQ 1) uses cloud object storage for data sharing, 2) utilizes container images to provide uniform job execution environment, and 3) automatically deploys an execution environment on the cloud.**

***Index Terms*—cloud bursting, public cloud, auto scaling**

I. Introduction

Public cloud environments are widely adopted as computational resources for academic/industrial research activities. On the other hand, each organization still has to preserve an on-premise environment to process sensitive data which cannot be shipped outside of the organization. To fully utilize both of them, a research area called **cloud bursting** is actively explored recently [1] [2] [3]. With cloud bursting, the on-premise environment and cloud environment are seamlessly integrated, and while the users use the on-premise environment as a primal choice, they also can use cloud environment as a resort when the on-premise environment is crowded.

To enable cloud bursting, the following three issues have to be addressed:

1) **Secure communication**:
   We have to transfer input/output data and the job submission information between the client and the cloud.
2) **Abstract away the difference between the on-premise and cloud environments**:
   In general, these environments tend to differ in several aspects, such as the library versions or batch queueing systems. To integrate these systems seamlessly, we need to abstract away such differences.
3) **Execution environment deployment on the cloud**:
   The cloud environment has to be dynamically allocated and deployed on demand to avoid excessive cloud cost.

We propose a meta-scheduling system for cloud bursting called **CloudQ** which addresses the issues above.

CloudQ 1) uses cloud-object storage as a communication channel to enable secure communication, 2) introduces meta job script which abstract away the difference between the two environments, and 3) has automatic deployment capability to establish a computing environment on the cloud in on-demand fashion.

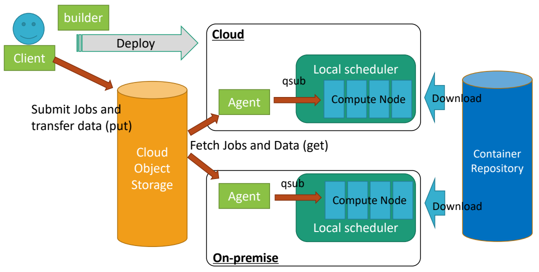

Fig. 1. The overview of CloudQ.

CloudQ is mainly composed of the following three modules: the **builder** which deploys computation environment including an autoscale local batch queueing system on the cloud, the **agent** which runs on the deployed computing environment and receives jobs from the user and submits them to the local batch scheduler, and the **client** which helps users to submit their jobs, monitor the jobs, and retrieve the job results. Figure 1 shows the overview of CloudQ.

In the following sections, we briefly introduce the techniques we employed in CloudQ.

II. Job Submission via Cloud Storage

User authentication and authorization are the most important thing for secure job submission to the cloud environment. While ssh is broadly adopted for user authentication in the HPC area, it is not adequate for our purpose. Once authenticated, ssh users are allowed to do anything, while we want to restrict the user to submit jobs only.

While it is relatively easy to implement our own server module that does authentication based on public key technology, maintaining such a server safely will be quite a pain.

We employed the S3-compatible cloud storage servers for job submission. The client and agent will poll on the pre-determined storage backet for communication. The S3-compatible cloud storages support user ID and token-based authentication. The token can have expiration time enabling secure management of the tokens. Capabilities assigned to the user can be precisely controled by the agent.

Note that both the client and the agent do not use inbound ports for communication; both of them use outbound only. This will contribute to the secure management of the agent.

III. Abstraction of the Computing Environment with Container and Meta Job Script

CloudQ employed docker containers for the job execution environment. Users specify the environment in the job script.

```
#$ run_on:ANY
#$ project:project01
#$ resource:type1
#$ n_resource:1
#$ walltime:1:00:00
#$ shell:bash
#$ container_image: img0=\
   docker://nvcr.io/nvidia/tensorflow:19.07-py3

wget https://script.is/here train.py
cloudq_cs_cp s3://myobjs/data ./data
cloudq_container_run $IMG0 python ./train.py data
```

Fig. 2. An example of meta job script file.

The local scheduler is in charge of downloading the container images from the repositories and running them. The containers encapsulate the environment and present uniform execution environment to the job, abstracting away the difference between on-premise and cloud.

There are other kinds of differences which cannot be hidden by the container only, such as local scheduler types, job queue names, and commands that are unique to the specific environment. We introduced meta job script which gives canonical names for these things. Users uses the canonical names in the scripts, and the agent will replace them to the local environment specific names. Figure 2 shows an example of meta job script file. Commands with prefix `cloudq_` are the canonical commands which will be replaced with local commands.

IV. Execution environment deployment on the cloud

To deploy the job execution environment on the cloud we need to take care of security, user management, and account management. To fullfill these requirements, we came up with two approaches. One approach is to deploy a multi-user environment that is similar to the on-premise environment. Another approach is to deploy a single-user environment for each user. In this paper, we call the former **multi-user style** and the latter **single-user style**.

In the multi-user style, we set up a shared cluster on the cloud that have the same user name space as the on-premise cluster. The cluster is permanently maintained while the computation nodes are launched/stopped based on the user requirement. The user name space has to be synchronized with the on-premise cluster. Accounting has to be addressed using the local batch scheduler log files.

In the single-user style, each user launches his/her own dedicated job execution environment on the cloud, using the builder command we provide. We do not need to manage the user namespace since each environment effectively has just one user. Accounting is also easy since each user have his/her own cloud account. We can rely on the cloud accounting service.

We implement both of these styles to compare. We employ AWS for both of them. We call the multi-user style implementation CloudQ/M, and single-user style implementation CloudQ/S.

A. *CloudQ/S: Single user style implementation*

We utilized AWS ParallelCluster [4] for cluster deployment, however we had to also use AWS CloudFormation [5] for VPC creation since ParallelCluster cannot meet our security requirement. ParallelCluster provides us with the autoscaling capability, which automatically launches and stops the job execution nodes. We manage the logs from the agent and batch queueing system with Amazon CloudWatch Logs [6]. All the system logs and daemon logs can be browsed and searched on the AWS console.

This means that CloudQ/S does not have to provide access from the external network at all. Jobs will be submitted via cloud storage and the results will be available on the same place. CloudWatch Logs provides us with enough information on the status and health of the nodes. Therefore, we can set up the environment as a completely closed system that does not have any globally accessible addresses. The enviroment is disposable. When the cluster is stopped, all the virtual machines are terminated and any state in the cluster will be completely disposed. Even if the environment is secretly compromised, the effect does not last long, since everything starts from scratch when the user relaunches the environment.

We confirmed that accounting for the single-user style is quite straightforward as expected.

B. *CloudQ/M: Multi-user style implementation*

For CloudQ/M, we deploy an execution environment on AWS, employing PBS as a job scheduler. To manage the whole structure, we utilize CloudFormation. For user namespace management, we use LDAP server and sync the namespace with the on-premise cluster.

For home file system, we use AWS FSx for Lustre [7], which is a fully managed Lustre service. The FSx for Lustre supports HSM(Hierarchical Storage Management) [8], which enables to use S3 as backend storage and seamless access to S3. This means that the home directory is effectively exposed as S3 bucket for the outside world, and S3 file system can be accessed with POSIX interface. It is also useful to reduce the storage cost to maitain the Lustre file system, since we can reduce the amount of genuine Lustre files, which is much more expensive than S3. HSM is quite useful for cloud storage-based communication. Thanks to HSM, we do not need to use S3 API directly to implement the agent. Instead, we can use the POSIX API which is easy to use.

Accounting for CloudQ/M is not simple as in CloudQ/S. We implemented an accounting monitoring system which monitors batch queueing system logfile and calculate the usage of all the users.

V. Conclusion

In this paper, we proposed a meta-scheduling system for secure cloud bursting. CloudQ/M is already deployed on our site and practically utilized. CloudQ/S is publically available now.

References

[1] S. K. Nair, S. Porwal, T. Dimitrakos, A. J. Ferrer, J. Tordsson, T. Sharif, C. Sheridan, M. Rajarajan, and A. U. Khan, "Towards Secure Cloud Bursting, Brokerage and Aggregation," in *2010 Eighth IEEE European Conference on Web Services*, dec 2010, pp. 189–196.

[2] T. Guo, U. Sharma, T. Wood, S. Sahu, and P. Shenoy, "Seagull: Intelligent Cloud Bursting for Enterprise Applications," in *Presented as part of the 2012 USENIX Annual Technical Conference (USENIX ATC 12)*, 2012, pp. 361–366.

[3] S. Date, H. Kataoka, S. Gojuki, Y. Katsuura, Y. Teramae, and S. Kigoshi, "First Experience and Practice of Cloud Bursting Extension to OCTOPUS," in *10th International Conference on Cloud Computing and Services Science, CLOSER2020*, 2020, pp. 448–455.

[4] "AWS ParallelCluster," https://aws.amazon.com/hpc/parallelcluster/.

[5] "AWS CloudFormation," https://aws.amazon.com/cloudformation/.

[6] "What is Amazon CloudWatch Logs?" https://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/WhatIsCloudWatchLogs.html.

[7] "Amazon FSx," https://aws.amazon.com/fsx/.

[8] A. Degrémont and T. Leibovici, "Lustre hsm project," https://wiki.lustre.org/images/4/4d/Lustre_hsm_seminar_lug10.pdf, 2009.