A Workflow Execution Technique for Analyzing Large-scale Astronomy Data on Virtualized Computing Environments

[1]Jung-Lok Yu, Il-Yeon Yeo, Seo-Young Noh, Hee-Jung Byun, Du-Seok Jin

Abstract: ***The size of observation data in astronomy has been increasing exponentially with the advents of wide-field optical telescopes. This means the needs of changes to the way used for large-scale astronomy data analysis. The complexity of analysis tools and the lack of extensibility of computing environments, however, lead to the difficulty and inefficiency of dealing with the huge observation data. To address this problem, this paper proposes a workflow execution system for analyzing large-scale astronomy data efficiently. The proposed system is composed of two parts: 1) a workflow execution manager and its RESTful endpoints that can automate and control data analysis tasks based on workflow templates and 2) an elastic resource manager as an underlying mechanism that can dynamically add/remove virtualized computing resources (i.e., virtual machines) according to the analysis requests. To realize our workflow execution system, we implement it on a testbed using OpenStack IaaS (Infrastructure as a Service) toolkit and HTCondor workload manager. We also exhaustively perform a broad range of experiments with different resource allocation patterns, system loads, etc. to show the effectiveness of the proposed system. The results show that the resource allocation mechanism works properly according to the number of queued and running tasks, resulting in improving resource utilization, and the workflow execution manager can handle more than 1,000 concurrent requests within a second with reasonable average response times. We finally describe a case study of DEEP-South data reduction system as an example application of our workflow execution system.***

Keywords: astronomy, data analysis, workflow management, elastic resources, virtualized computing, experiments

I. INTRODUCTION

Astronomy is facing a data tsunami – the size of observation data has been increasing exponentially with the advents of wide-field optical telescopes [1]. For example, DEEP-South project [2] aiming to provide a system for analyzing Near-Earth Objects (NEOs) in the southern hemisphere sky, has installed three 1.6 meters wide-field optical telescopes in South-Africa, Australia, and Chile. The data from these telescopes, already amounting to hundreds of terabytes (1 terabyte = 10^{12} bytes), will be more critical in size in the near future. This means the needs of changes to the way used for scientific data analysis i.e., the gathered huge observation data need to be handled in a timely manner to fully utilize the telescopes resources in detecting supernovae, asteroids, external galaxies, etc. However, traditional image processing tools like MODP (Moving Object Detection Program) and ASAP (Asteroid Spin Analysis Packages) are composed of complex internal pipelines to be processed (i.e., the complexity of analysis tools) [3]. Furthermore, those tools also have no glues to bridge underlying computing resources (i.e., the lack of extensibility with regard to computing environments), which lead to the difficulty and inefficiency of dealing with the huge observation data.

To address this problem, this paper proposes a workflow execution system for analyzing large-scale astronomy data efficiently. The proposed system is composed of two parts: 1) a workflow execution manager (WEM) and its RESTful endpoints [4] that can automate and control data analysis tasks based on workflow templates and 2) an elastic resource manager (ERM) as an underlying resource provisioning mechanism of WEM that can dynamically add/remove virtualized computing resources (i.e., virtual machines) according to the analysis requests. Specifically, because WEM's workflow templates which represent the sequences of tasks with their dependencies are based on HTCondor DAGMan (Directed Acyclic Graph MANager) [5,6] engine, it makes WEM domain-agnostic and highly extensible. With ERM, our system can also elastically scale-out or scale-in the resources according to the workload status of system, leading to the improvement of both resource utilization and job throughput. To realize our workflow execution system, we implement it on a testbed using OpenStack [7] IaaS (Infrastructure as a Service) toolkit and HTCondor workload manager. We also exhaustively perform a broad range of experiments with different resource allocation patterns, system loads, etc. to show the effectiveness of the proposed system.

The rest of the paper is organized as follows. Section 2 describes the details of the proposed workflow execution system with the brief overviews of HTCondor and OpenStack. Section 3 provides the experimental results on the dynamic resource allocation and the RESTful endpoints, as well as a case study of DEEP-South data reduction system as an example application of our workflow execution system. Finally, we conclude in Section 4.

Revised Manuscript Received on January 03, 2019.

Jung-Lok Yu, Il-Yeon Yeo, Seo-Young Noh, Supercomputing Center, Korea Institute of Science and Technology Information (KISTI), Korea.

Hee-Jung Byun, Department of Information and Telecommunications Engineering, Suwon University, Korea.

Du-Seok Jin, Corresponding author, Supercomputing Center, Korea Institute of Science and Technology Information (KISTI), Korea.

II. PROPOSED WORKFLOW EXECUTION MANAGEMENT

2.1. Background: HTCondor and OpenStack

HTCondor is a workload manager for bulk and/or distributed jobs, which provides primitive functions like job queueing/scheduling, resource monitoring, etc. For this, HTCondor executes different kinds of service daemons – collector, negotiator, scheduler, starter– (see Table 1) and all the information (e.g., slot(core), job, queue, etc.) for controlling these service daemons is expressed by ClassAd [8] and can be manipulated through Python API bindings. HTCondor also provides a DAGMan [9] that can execute a series of jobs in sequence according to their dependencies.

Table 1. HTCondor Service Daemons

Service Daemons	Description
collector	Manage all the information of other service daemons
negotiator	Decide whether to assign jobs to slots using data from collector daemon
scheduler	Manage queues and assign a job to the slots according to the decision of negotiator daemon
starter	Manage the slots (cores) for executing a job

As a well-established IaaS Cloud toolkit, OpenStack is composed of a lot of components (Compute, Storage, Network, etc.) which can be used to configure large-scale pooled computing resources, and interoperates with outside 3rd party applications by exporting OpenAPIs functions for each component (as shown in Table 2). Especially, OpenStack provides the features to customize virtualized computing environments using user-defined scripts (or using virtual machine images including those scripts), for instance, setting hostname, setting the mount point of shared network storage, etc. when virtual machines (VMs) are created.

Table 2. OpenStack Components

Components	Functions
Keystone	Provide user authentication/ authorization service
Neutron	Provide network functions for virtual machines
Glance	Manage the OS images for virtual machines
Nova	Manage virtual machines lifecycles (create, delete, reboot, etc.)

2.2. Workflow Execution Manager

Workflow execution manager (WEM) transforms a workflow into a series of HTCondor jobs and performs the control of execution (submit, monitor, cancel, suspend/resume, etc.) of each job. It also provides HTTP(S)-based RESTful endpoints for the incorporation of external applications. Figure 1 shows the architectural diagram of the proposed WEM. It consists of four layered subsystems: Model, DAO (Data Access Object), Service and Controller as described follows:

(1) Model: Model represents database schemas which define and describe the information of workflow instances, e.g., MODP, ASAP, Resources Pool

(2) DAO: DAO provides CRUD (Create-Retrieval-Update-Delete)-style access methods for the database objects (using SQLAlchemy [10]).

(3) Service: Service plays a key role in executing workflow instances and managing their lifecycles using HTCondor DAGMan based workflow templates.

(4) Controller: Controller provides HTTP(S)-based RESTful endpoints for the execution of workflow instances and for the retrieval of available computing resources allocated to the specific workflows.

Basically, MODP and ASAP are the image processing tools used in astronomy field which consist of complex internal analysis pipelines, which generally expressed as directed acyclic graphs. WEM exploits split-merge model for MODP because MODP tool divides each image file (18K x 18K size) into 256 independent tiles for 3 consecutive image files (TRIPLTET) and merges the results to find moving objects. After the execution of MODP, ASAP uses MODP's results to perform asteroid spin/features analysis with its light curve. Because ASAP is composed of three main steps to do this analysis, WEM implements ASAP workflow templates with linear model. Table 3 shows WEM's RESTful endpoints to execute workflows defined by workflow templates and check the available computing resources. Using these endpoints, applications (or users) can submit workflow instances with the resulting executionID, and manage the workflow lifecycles (stop/suspend/resume, etc.)

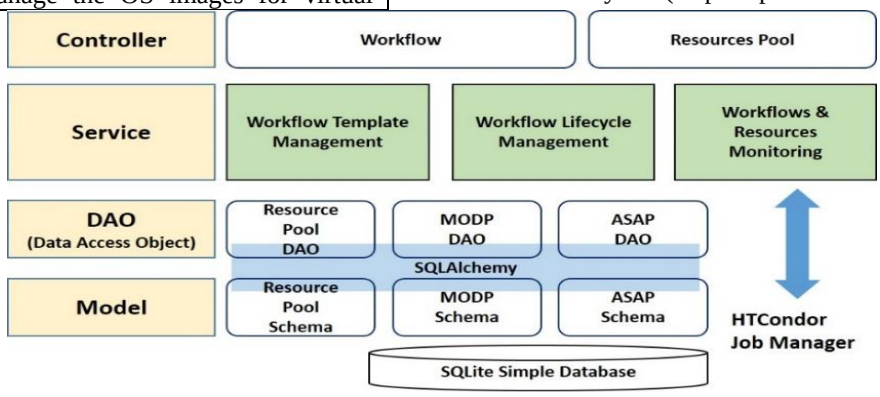


Figure 1. Architectural Diagram of Workflow Execution Manager (WEM)

Table 3. WEM's RESTful Endpoints

Function	HTTP Method	Endpoint URL
Submit MODP workflow	GET	http://-/WEM/api/v1/workflow?category=MODP&PID= {...}&SITE={...}&DATE= {...}&TF= {...}&TRIPLET= {...}&targetDir= {"/x/y/z"}&splitNum= {…}
Submit ASAP workflow	GET	http://-/WEM/api/v1/workflow?category=ASAP&PID= {...}&SITE={...}&DATE= {...}&OBJECT= {...}&TF= {...}&targetDir= {"/x/y/z"}&CHIP={...}&objectNum= {...}
Monitor workflow	GET	http://-/WEM/api/v1/workflow/{*executionID*}/status?category={MODP|ASAP|…}
Suspend workflow	GET	http://-/WEM/api/v1/workflow/{*executionID*}/suspend?category={MODP|ASAP|…}
Resume workflow	GET	http://-/WEM/api/v1/workflow/{*executionID*}/resume?category={MODP|ASAP|…}
Cancel workflow	GET	http://-/WEM/api/v1/workflow/{*executionID*}/stop?category={MODP|ASAP|…}
List available computing resources	GET	http://-/WEM/api/v1/pool/list?category={MODP|ASAP|…}

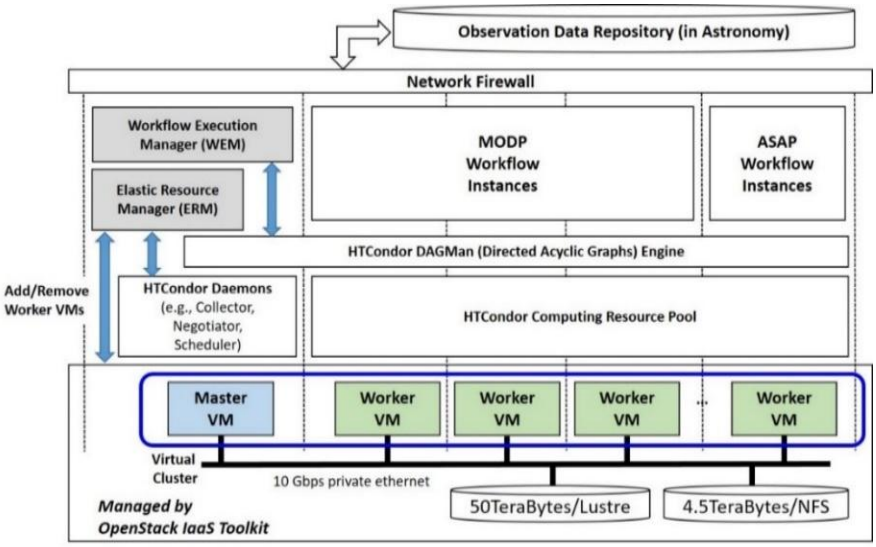


Figure 2. Configuration of a Virtual Cluster

2.3. Elastic Resource Manager

Elastic resource manager (ERM) is an underlying resource management tool supporting WEM, which can dynamically (de-)allocate virtualized computing resources (i.e., virtual machines; VMs) according to the WEM's analysis requests.

2.3.1. Virtual Cluster

ERM initially configures a virtual cluster (VC)environment, which consists of VMs to provide compute/storage elements that can be utilized to perform large-scale pipelined data analysis tasks in WEM. Figure 2 shows the configuration of a VC that can be created by ERM in order to process distributed data analysis tasks. A VC consists of a master VM and number of worker VMs, where each VM has 16 CPU cores, 16GBytes main memory, 400GBytes local storage and interconnections with other VMs by 10Gbps private Ethernet. A master VM commonly has a role of managing worker VMs with executing HTCondor related daemons (e.g., collector, negotiator, scheduler, etc.) as well as WEM and ERM. Unlike master VM, worker VMs as elements for HTCondor computing resources pool, can execute MODP/ASAP workflow instances. All the worker VMs have the same setup configurations, for example, OpenLDAP [11] clients for authentication/authorization, mount clients for shared storages (e.g., scratch volumes with Lustre [12] parallel file systems, home volumes with NFS, etc.).

Whenever we need a VC to perform data analysis tasks, ERM can initially do provisioning it in dynamic and independent manner. Figure 3 shows the sequence diagram of composing a VC environment (i.e., adding N worker VMs to an existing VC). Assume that we have an existing VC that has one master VM and one worker VM. If it is required to create N worker VMs and add those VMs to the existing VC, ERM deals with 1 ~ (N-1)th worker VMs and Nth worker VM separately to share all the same cluster configurations. First, ERM calls an OpenStack IaaS toolkit function to create 1 ~ (N-1)th worker VMs. Then, all the worker VMs are booted using a shared VM image and a user-script file is executed on each worker VM. After finishing the creation of 1 ~ (N-1)th worker VMs, ERM can obtain hostnames and IP mapping information of the worker VMs.

Second, ERM creates Nth worker VM with those mapping information. When Nth worker VM completes its booting procedure, Nth worker VM merges previous mapping for the existing VC and newly created 1 ~ (N-1)th worker VMs mapping information, and finally Nth VM propagates all the mappings to all the cluster members. Through this way, ERM can allocate N worker VMs to the existing VC.

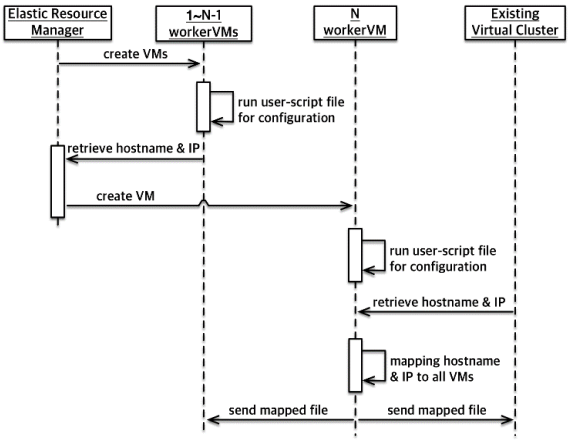


Figure 3. Sequences for composing a VC environment

2.3.2. Dynamic Resource Provisioning

Managing computing resources elastically becomes more crucial in order to improve both resource utilization and job throughput in data analytics systems. To do this, ERM can dynamically allocate (or deallocate) worker VMs to (or from) the existing HTCondor resources pool based on the information about the number of queued (thus waiting) jobs and the number of running cores (i.e., slots in terms of HTCondor), which can be done by using HTCondor APIs and OpenStack APIs.

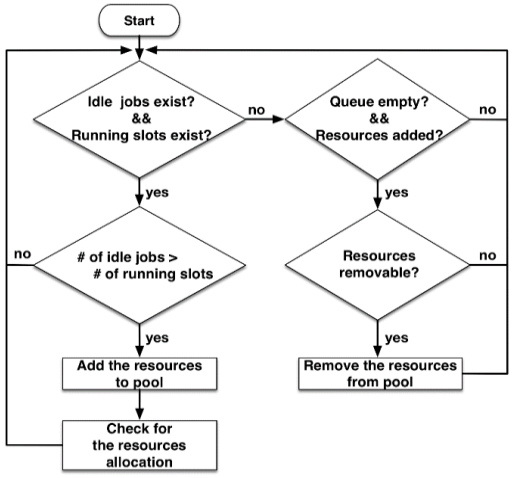


Figure 4. Flow chart of allocating or deallocating worker VMs

Figure 4 shows the flow chart of allocating or deallocating worker VMs used in our ERM. Detailed description on the dynamic resource provisioning is as follows:

(1) ERM periodically renews the information about the number of queued (waiting) jobs and the number of running slots which can be obtained from HTCondor job manager.

(2) Using the information of (1), ERM tries to add a worker VM to the existing HTCondor computing resources pool if and only if the number of waiting jobs is greater than the number of running slots (by the procedure explained in section 2.3.1).

(3) If waiting jobs do not exist, ERM finds worker VMs that do not have running slots. Then, it tries to deallocate such kind of worker VMs and updates all the cluster mapping information.

(4) If there are no jobs in the system, ERM maintains only two VMs (a master VM and a worker VM).

III. EXPERIMENTAL RESULTS

To realize our workflow execution system described in Section 2, we implement it on a testbed using OpenStack IaaS toolkit and HTCondor job manager. Our testbed consists of a controller node, a network node, an image node and 16 compute nodes, totally 19 nodes, which is configured as IaaS computing environment using OpenStack keystone (for ID management), neutron (for Layer3 network management), glance (for VM image management), and nova (for compute management) components, respectively. All the physical nodes have connected with 50 Terabytes and 4.5 Terabytes shared storages for scratch and user home directory. They are inter-connected with both 10Gbps private network and 1Gbps management Ethernet network, and specially the controller and the network nodes are also connected to public 1Gbps Ethernet in order to provide self-serviced multi-tenancy networks. Each node has one Intel Xeon 2.6GHz CPU(16 cores), 96GBytes main memory. On this testbed, we exhaustively perform a broad range of experiments with different resource allocation patterns, system loads, etc. to show the effectiveness of the proposed system.

3.1. Dynamic Resource Allocation

3.1.1. Analysis of Resource Allocation using synthetic workloads

First, we analyze the behavior of resources scaling-out/in while a synthetic workload has been injected to our workflow execution system. For this experiment, we used a synthetic workload profile that represents 14 groups of workloads (with 50 seconds inter-arrival time) where each workload has simultaneous 16 jobs with 300 seconds execution time.

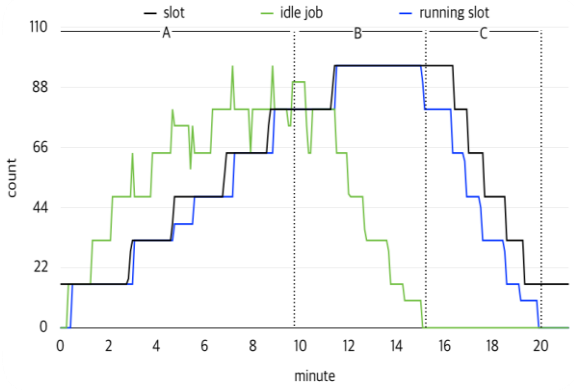


Figure 5. Dynamic resource allocation according to the number of idle jobs and running slots

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication

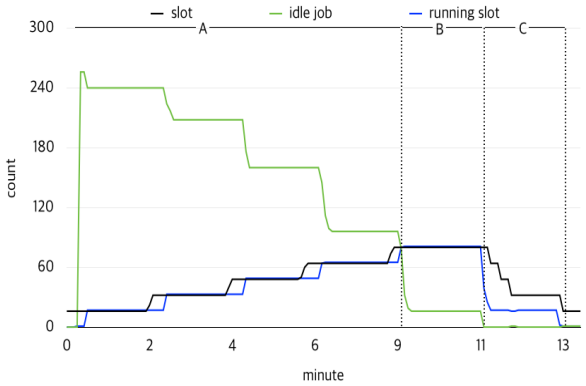


Figure 6. Dynamic resource allocation for the MODP workflow

Figure 5 shows the resource allocation graph for this workload. Before executing the workload, we initially set up HTCondor resource pool to have one worker VM (i.e., 16 slots) with the limitation of maximum of 96 slots. From the Figure 5, we confirm that our system can dynamically adjust the amount of worker VMs according to the workload status. Especially, if there are newly arrived jobs which are not able to be executed due to the lack of available resources (i.e., idle/waiting jobs, green-colored line), the ERM tries to add worker VMs to the resources pool by comparing the number of idle jobs with the number of running slots (see Figure 5. A period), resulting in the increment of total and running slots (black and blue-colored lines, respectively). After that, our system eventually reaches the point that the number of idle jobs becomes the ceiling of the number of total slots. In this case, our system decides not to allocate more worker VMs with the expectation that the idle jobs can be executed using existing slots in the near future (see Figure 5. B period). Next, if there are no idle jobs, our system can release worker VMs from the resources pool with finding worker VMs that do not have running slots, as shown in Figure 5. C period. With this way, our system can elastically scale-out or scale-in the resources according to the workload status of system, with the expectation of improving both resource utilization and job throughput.

3.1.2. Analysis of Resource Allocation using MODP/ASAP workloads

In this subsection, next, we analyze the resource allocation when we apply more realistic MODP/ASAP workloads to the system. Figure 6 depicts the resource allocation graph for a MODP workflow instance when a user submits a MODP workflow execution to WEM using WEM OpenAPIs. Note that, as described in Section 2.2, a MODP workflow instance is divided into the 256 independent jobs by its workflow template. For this experiment, we also set up an initial VC to have one worker VM (i.e., 16 slots). Like the case of synthetic workloads, from the Figure 6, we confirm again that our ERM can elastically provide additional computing cores to (or release the computing cores from) HTCondor resource pool according to the resource requirements of the workload. When increased idle/waiting jobs compared to the running slots, the ERM tries to increase gradually more worker VMs to the resource pool (see Figure 6. A period). Also, it can release worker VMs from the resource pool if there are no idle jobs (see Figure 6. C period). Unlike MODP workflow, an ASAP workflow requires just one slot to do asteroid spin/features analysis (because of its linearity defined by its workflow template), thus we performed the same experiment with multiple ASAP workflow instances. From the results, we obtained the similar resource allocation behavior as that of Figure 6 (we omit the graph due to the space limitation).

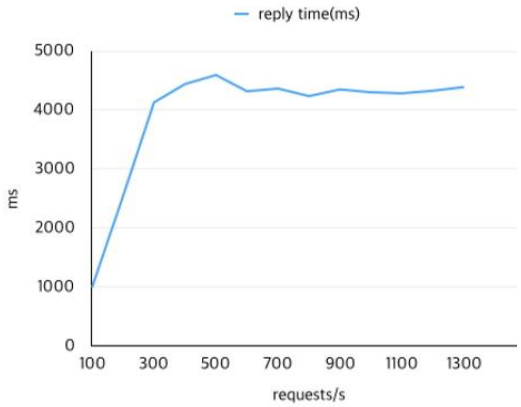
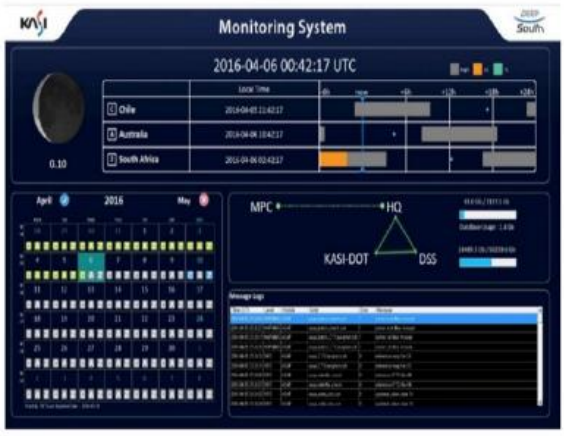


Figure 7. Average response times with varying the number of endpoints requests

3.2. Performance Test on RESTful Endpoints

Figure 7 shows the results for the measurement of throughput of our WEM's RESTful endpoints using httperf [13] benchmark tool. For this experiment, our WEM is deployed on the master VM that has 16 CPU cores, 16 GBytes main memory and 1Gbps Ethernet NIC. Specifically, Figure 7 depicts the average response times in milliseconds when we increase the number of concurrent requests. From the results, we note that with increasing the number of concurrent requests, the average response times for the requests are also increased, but they are converged to about 4500 milliseconds even with 1,000 requests per second. This means more than 1,000 concurrent requests within a second can be handled with reasonable average response times by our WEM. We also expect that the throughput of WEM's RESTful endpoints will be improved if we use more powerful VM (e.g., VMs with more CPU cores, main memory with high-bandwidth Ethernet NIC) and optimized settings.



(a) DEEP-South monitoring

(b) DEEP-South data reduction

Figure 8. DEEP-South monitoring and reduction system

3.3. Case Study : DEEP-South Data Reduction System

DEEP-South project aims to provide a system for monitoring, searching and analyzing Near-Earth Objects (NEOs) in the southern hemisphere sky using three 1.6 meters wide-field optical telescopes which are installed and operated in South-Africa, Australia, and Chile. These telescopes, in general, produce more than hundreds of terabytes per year, which have been used in many researches for detecting supernovae, asteroids, external galaxies, etc. and analyzing their features.

For analyzing those large-scale observation data, we deployed our workflow execution system as an underlying computing environment of DEEP-South data reduction system. Figure 8(a) and 8(b) show DEEP-South monitoring and data reduction systems, respectively using our workflow execution system. According to the scheduling decision for three telescopes (after monitoring and controlling the telescopes status (see Figure 8(a))), tremendous raw data is collected and transferred into the permanent repository in the headquarter (in South Korea). Then, the DEEP-South data reduction system (see Figure 8(b)) can upload the raw data to our buffer storage temporarily and can call WEM's RESTful endpoints to submit pre-defined workflows (for examples, Basic Inspection, MODP/ASAP workflows, etc.) and monitor their status. On WEM, each kind of workflow instance is transformed into a couple of distributed tasks (defined by workflow templates), which are executed on the virtualized computing resources controlled by our ERM in an elastic manner.

IV. CONCLUSION

In astronomy area, the size of observation data has been increasing exponentially with the advents of wide-field optical telescopes, which implies the changes to the way used for large-scale data analysis. However, the complexity of analysis tools and the lack of extensibility of computing environments can cause the difficulty and inefficiency of dealing with the huge data. To resolve this problem, this paper described a workflow execution system to efficiently analyze large-scale astronomy data, which is based on high extensible workflow templates and elastic resource provisioning mechanism. In order to realize the proposed workflow execution system, we also implemented it on OpenStack IaaS testbed. From the various experimental results, we clearly confirmed the effectiveness of the proposed system. We also described a case study of DEEP-South reduction system as the best practical application example. For the further works, we have a plan to design more sophisticated and/or workflow-aware resource provisioning algorithms that can be applied to other data-intensive application areas (e.g., high-energy physics, genomics, etc.) to fully utilize underlying computing/storage resources.

ACKNOWLEDGMENT

This work was supported by Global Hub for Experiment Data of Basic Science through the Ministry of Education of the Republic of Korea and the National Research Foundation of Korea (NRF-2010-0018156) and the KISTI Program (K-19-L02-C03).

REFERENCES

1. Zhang Y, Zhao Y. Astronomy in the Big Data Era. Data Science Journal. 2015;14:11. DOI: http://doi.org/10.5334/dsj-2015-011
2. Yim H-S, Kim M-J, Bae Y-H, Moon H-K, Choi Y-J, Roh D-G, et al. DEEP-South: Automated Observation Scheduling, Data Reduction and Analysis Software Subsystem. Proceedings of the International Astronomical Union. Cambridge University Press; 2015;10(S318):311–2.
3. Kim M-J, Moon H-K, Choi Y-J, Yim H-S, Bae Y-H, Roh D-G, et al. DEEP-South: Preliminary Photometric Results from the KMTNet-CTIO. Proceedings of the International Astronomical Union. Cambridge University Press; 2015;10(S318):313–6.
4. Leonard R, Sam R. RESTful Web Services: California: O'Reilly & Associates; 2007.
5. Gropp W, Lusk E, Sterling TL. Beowulf cluster computing with Linux. Cambridge, MA: MIT Press; 2003.
6. Thain D, Tannenbaum T, Livny M. Distributed computing in practice: the Condor experience. Concurrency and Computation: Practice and Experience. 2005 Feb;17(2-4):323–56.
7. Sefraoui O, Aissaoui M, Eleuldj M. OpenStack: Toward an Open-source Solution for Cloud Computing. International Journal of Computer Applications. 2012;55(3):38–42.
8. Raman R, Livny M, Solomon M. Matchmaking: distributed resource management for high throughput computing. Proceedings The Seventh International Symposium on High Performance Distributed Computing (Cat No98TB100244).
9. Couvares P, Kosar T, Roy A, Weber J, Wenger K. Workflow Management in Condor. Workflows for e-Science. 2007;:357–75.
10. Ramm M, Bayer M. SQLAlchemy: database access using Python. Boston, MA: Addison-Wesley; 2009.
11. Wang X, Schulzrinne H, Kandlur D, Verma D. Measurement and analysis of LDAP performance. IEEE/ACM Transactions On Networking 2008; 16(1): 232-43.
12. Han J, Kim D, Eom H. Improving the Performance of Lustre File System in HPC Environments. 2016 IEEE 1st International Workshops on Foundations and Applications of Self* Systems (FAS*W). 2016; DOI: http://doi.org/10.1109/FAS-W.2016.29
13. Mosberger D, Jin T. httperf---a tool for measuring web server performance. ACM SIGMETRICS Performance Evaluation Review. 1998Jan;26(3):31–7.