ISSN:1738-6764, eISSN:2093-7504

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

Jung-Lok Yu [1,*], Du-Seok Jin [2], Il-Yeon Yeo [3] and Hee-Jun Yoon [4]

[1] Korea Institute of Science and Technology Information (KISTI); Principle Researcher; junglok.yu@kisti.re.kr
[2] Korea Institute of Science and Technology Information (KISTI); Principle Researcher; dsjin@kisti.re.kr
[3] Korea Institute of Science and Technology Information (KISTI); Principle Researcher; ilyeon9@kisti.re.kr
[4] Korea Institute of Science and Technology Information (KISTI); Principle Researcher; k2@kisti.re.kr
* Correspondence

https://doi.org/10.5392/IJoC.2020.16.4.016

Manuscript Received 25 May 2020; Received 9 November 2020; Accepted 11 November 2020

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 data reduction system as an example application of our workflow execution system.*

Keywords: workflow; astronomy; data analysis; dynamic resource allocation; cloud; virtualization

1. 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, KMTNet (Korea Microlensing Telescope Network) 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 = 1012 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 briefly overviews background and related works. Section 3 describes the details of the proposed workflow execution system. Section 4 provides the experimental results on the dynamic resource allocation and the RESTful endpoints, as well as KMTNet data reduction system as an example application of our workflow execution system. Finally, we conclude in Section 5.

2. Background and Related Works

2.1 HTCondor and OpenStack

HTCondor is a batch workload manager for bulk and/or distributed jobs, which provides primitive functions like job queueing/scheduling, resource monitoring, etc. To do 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 interacting with 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 Daemon	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 (Infrastructure as a Service) 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 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

Component	Description
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 Related Works

There are a few efforts of providing computing environments to analyze huge astronomy data in the literature. G. B. Berriman et. al. [10] described a workflow management system on top of Pegasus [11] to create new datasets of Galactic Plane Atlas, integrating commercial Amazon EC2 cloud computing resources. To some extents, this idea is similar to our approach in terms of utilizing virtual clusters (a set of virtual machines), however, it does not consider elastic provisioning of backend computing resources according to the requested data processing ratio yet, just focusing on how to build a customized virtual cluster over Amazon EC2.

M. Johnson, et. al. [12] proposed data management and processing frameworks using geographically distributed HTCondor infrastructure for the Dark Energy Survey (DES) project – to probe the origin of the accelerating universe and help uncover the nature of dark energy. It provides the idea of glide-in data processing jobs on Virtual Organizations (VOs) HTCondor pool via flocking mechanism. However, like [10], it still has the lack of automating the provision of resources with data processing ratio, and also simply depends on HTCondor's DAGMan to build data reduction pipelines without a workflow execution system of providing higher level of abstraction like workflow templates and APIs.

3. Proposed Workflow Execution System

3.1 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 layered block diagram of the proposed WEM. It consists of four subsystems: Model, DAO (Data Access Object), Service and Controller as described follows:

- Model: Model represents database schemas which define and describe the information of workflow instances, e.g., MODP, ASAP, Resources Pool
- DAO: DAO provides CRUD (Create-Read-Update-Delete)-style access methods for the database objects (using SQLAlchemy [13]).
- Service: Service plays a key role in executing workflow instances and managing their lifecycles using HTCondor DAGMan based workflow templates.
- 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.

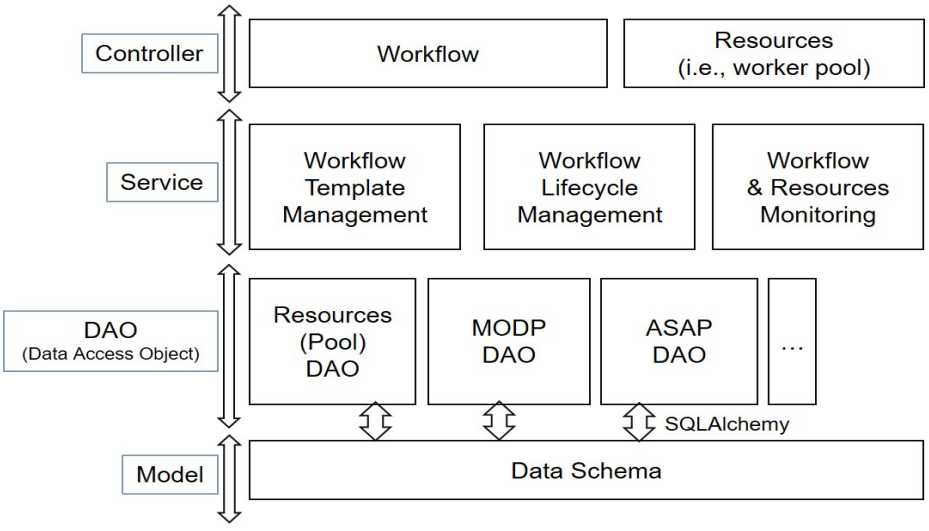

Figure 1. Layered block diagram of workflow execution manager

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.)

Table 3. RESTful endpoints of workflow execution manager

Function	Endpoint URL
Submit MODP workflow	http://-/WEM/api/v1/workflow?category=MODP&PID={...}&SITE={...}&DATE={...}&TF= {...}&TRIPLET= {...}&targetDir= {"/x/y/z"}&splitNum= {256}
Submit ASAP workflow	http://-/WEM/api/v1/workflow?category=ASAP&PID={...}&SITE={...}&DATE={...}&OBJECT={...}&TF={...}&targetDir={"/x/y/z"}&CHIP={...}&objectNum= {...}
Monitor workflow status	http://-/WEM/api/v1/workflow/ {executionID}/status?category={MODP|ASAP}
Suspend workflow execution	http://-/WEM/api/v1/workflow/ {executionID}/suspend?category={MODP|ASAP}
Resource workflow execution	http://-/WEM/api/v1/workflow/ {executionID}/resume?category={MODP|ASAP}
Cancel workflow execution	http://-/WEM/api/v1/workflow/ {executionID}/stop?category={MODP|ASAP}
List available resources	http://-/WEM/api/v1/pool/list?category={MODP|ASAP}

3.2 Elastic Resource Manager

3.2.1 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, 16 GBytes main memory, 400 GBytes 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 [14] clients for authentication/authorization, mount clients for shared storages (e.g., scratch volumes with Lustre [15] 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 $1 \sim (N-1)^{th}$ worker VMs and N^{th} worker VM separately to share all the same cluster configurations. First, ERM calls an OpenStack IaaS toolkit function to create $1 \sim (N-1)^{th}$ worker VMs. Then, all the worker VMs are booted using a shared VM image and then a user-script file is executed on each worker VM. After finishing the creation of $1 \sim (N-1)^{th}$ worker VMs, ERM can obtain hostnames and IPs mapping information of the worker VMs.

Second, ERM creates N^{th} worker VM with those mapping information. When N^{th} worker VM completes its booting procedure, N^{th} worker VM merges previous mapping for the existing VC and newly created $1 \sim (N-1)^{th}$ worker VMs mapping information, and finally N^{th} VM propagates all the mappings to all the cluster members. Through this way, ERM can add *N* worker VMs to the existing VC.

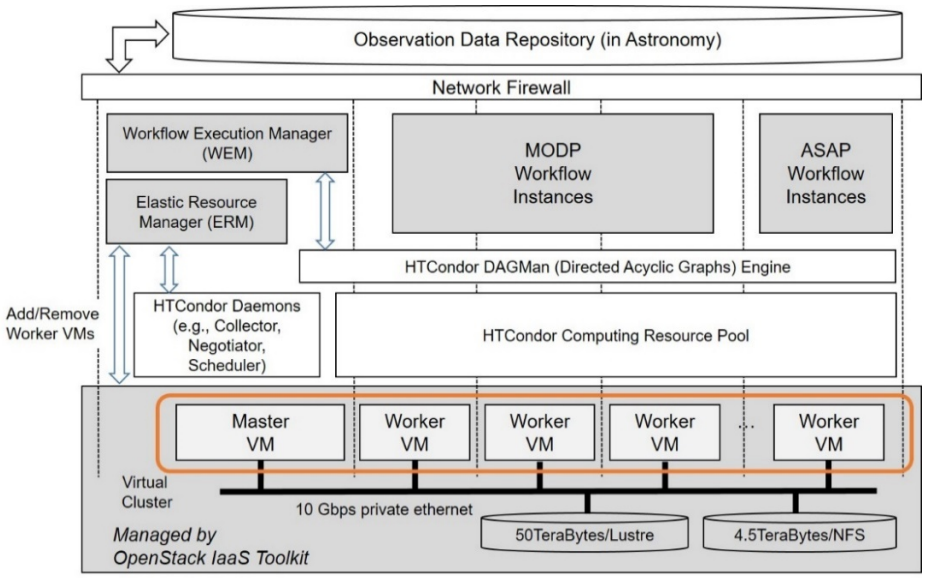

Figure 2. A configuration of virtual cluster that is created by ERM to process distributed analysis tasks

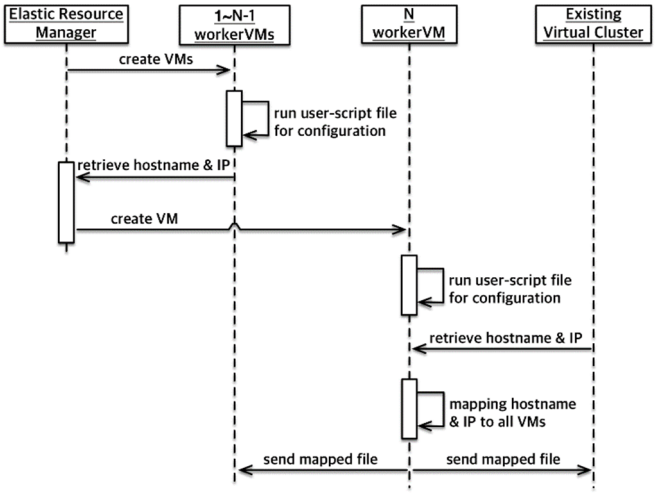

Figure 3. Sequence diagram for building a virtual cluster environment

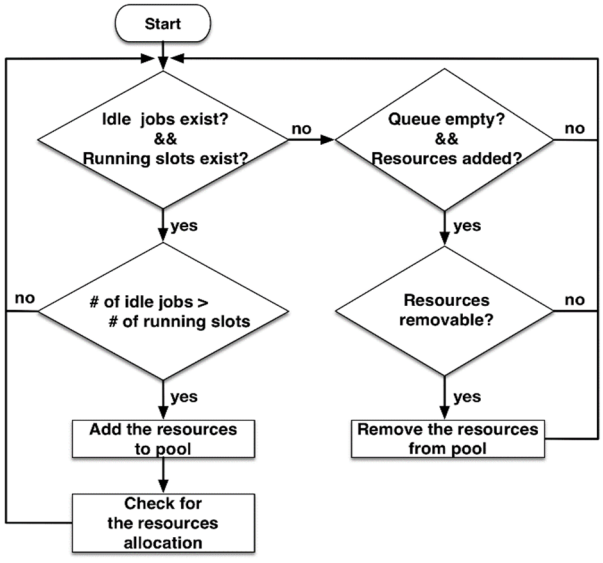

Figure 4. Flow chart of allocating or deallocating worker virtual machines

3.2.2 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 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 subsection 3.2.1).
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).

4. Experimental Results

To realize our workflow execution system described in Section 3, 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.

4.1 Dynamic Resource Allocation

4.1.1 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 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.

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

4.1.2 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).

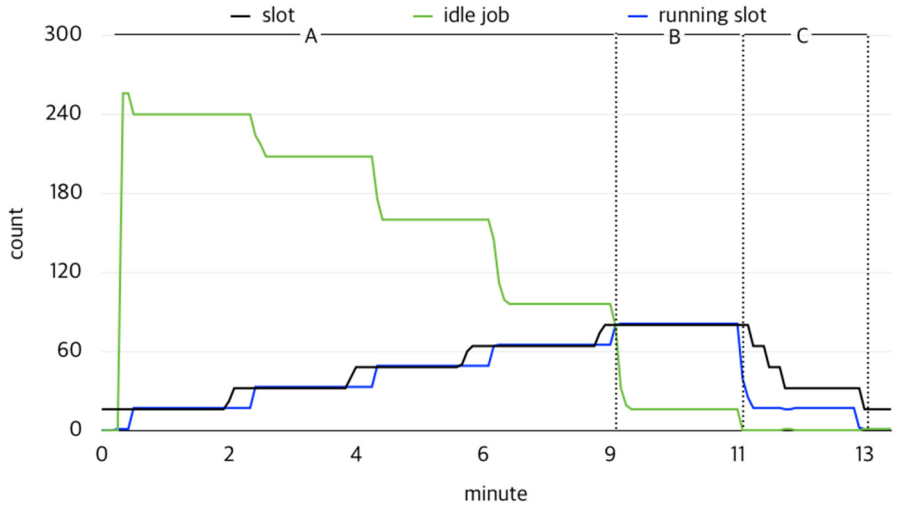

Figure 6. Dynamic resource allocation for the MODP workflow

4.2 Performance Test of RESTful Endpoints

Figure 7 shows the results for the measurement of throughput of our WEM's RESTful endpoints using httperf [16] 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 equipped with more vCPU (virtual CPU) cores, main memory and high-speed network interface card enough to perform lots of HTTP requests in multi-threaded way) and optimized settings.

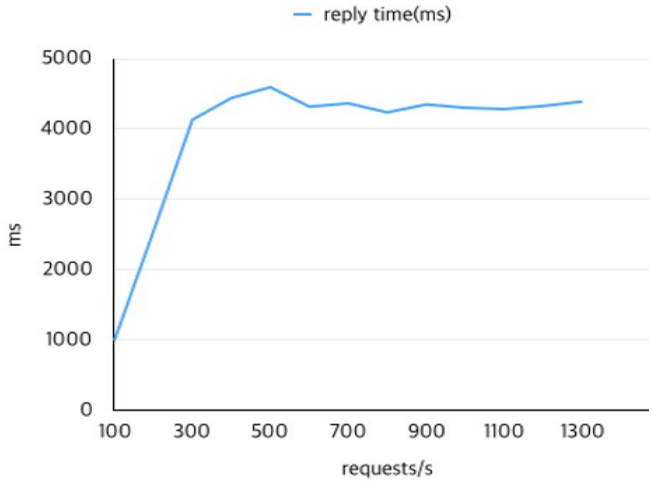

Figure 7. Average response time with varying the number of endpoints requests per second

4.3 Case Study : KMTNet Data Reduction

KMTNet [2, 3] 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. All the raw CCD (Charge-Coupled Device) images from the three sites are transferred to the KMTNet headquarter in South Korea via network communication. About 200 GBytes raw data is collected from each site per night (more than hundreds of terabytes per year) and should be processed in near real-time before researchers can access the processed image data to detect supernovae, asteroids, external galaxies, etc. and analyze their features.

For processing those large-scale observation data in near real-time, we deployed our workflow execution system as an underlying computing environment of KMTNet data reduction. Figure 8(a) and 8(b) show KMTNet controller GUI and its data reduction screenshots, respectively. 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 as we mentioned before. Then, the KMTNet 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.

(a) **(b)**

Figure 8. KMTNet controller GUIs for monitoring, searching and analyzing Near-Earth Objects (NEOs). **(a)** Telescope controller and **(b)** Data reduction

5. Conclusions

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 KMTNet reduction system as the best practical application example. For the further works, we will compare our experimental results with those of prior data analysis systems focusing on the performance, more specifically in terms of data processing rate and API's response time, and also discuss our pros and cons. We also 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.

Acknowledgments: This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (No. NRF-2010-0018156).

Conflicts of Interest: The authors declare no conflict of interest.

References

[1] Y. Zhang and Y. Zhao, "Astronomy in the Big Data Era," Data Science Journal, vol. 14, no. 11, 2015, doi: http://doi.org/10.5334/dsj-2015-011.

[2] S. L. KIM, C. U. LEE, B. G. PARK, D. J. KIM, S. M. CHA, Y. S. LEE, C. H. HAN, M. Y. CHUN, and I. S. YUK, "KMTNET: A NETWORK OF 1.6 M WIDE-FIELD OPTICAL TELESCOPES INSTALLED AT THREE SOUTHERN OBSERVATORIES," Journal of The Korean Astronomical Society, vol. 49, no. 1, pp. 37-44, Feb. 2016, doi: https://doi.org/10.5303/JKAS.2016.49.1.37.

[3] H. S. Yim, M. J. Kim, Y. H. Bae, H. K. Moon, Y. J. Choi, D. G. Roh, J. Park, and B. Moon, "DEEP-South: Automated Observation Scheduling, Data Reduction and Analysis Software Subsystem," *Proceedings of the International Astronomical Union*, vol. 10, no. S318, pp. 311-312, 2015, doi: https://doi.org/10.1017/S1743921315007243.

[4] G. Alonso, F. Casati, H. Kuno, and V. Machiraju, Web Services: Concepts, Architectures, Applications. Springer, 2004, doi: https://doi.org/10.1007/978-3-662-10876-5_5.

[5] W. Gropp, E. Lusk, and T. L. Sterling, Beowulf cluster computing with Linux, MIT Press, Cambridge, MA, 2003, doi: https://doi.org/10.7551/mitpress/1556.001.0001.

[6] D. Thain, T. Tannenbaum, and M. Livny, “Distributed computing in practice: the Condor experience,” Concurrency and Computation: Practice and Experience, vol. 17, no. 2-4, pp. 323-356, 2005, doi: https://doi.org/10.1002/cpe.938.

[7] P. Ivanovic and H. Richter, "OpenStack cloud tuning for high performance computing," *2018 IEEE 3rd International Conference on Cloud Computing and Big Data Analysis (ICCCBDA),* Chengdu, pp. 142-146, 2018, doi: https://doi.org/10.1109/ICCCBDA.2018.8386502.

[8] R. Raman, M. Livny, and M. Solomon, "Matchmaking: distributed resource management for high throughput computing," *Proceedings. The Seventh International Symposium on High Performance Distributed Computing (Cat. No.98TB100244)*, Chicago, IL, USA, pp. 140-146, 1998, doi: https://doi.org/10.1109/HPDC.1998.709966.

[9] P. Couvares, T. Kosar, A. Roy, J. Weber, and K. Wenger, "Workflow Management in Condor," In: I. J. Taylor, E. Deelman, D. B. Gannon, and M. Shields (eds), Workflows for e-Science, Springer, London, 2007, doi: https://doi.org/10.1007/978-1-84628-757-2_22.

[10] G. B. Berriman, E. Deelman, J. Good, G. Juve, J. Kinney, A. Merrihew, and M. Rynge, Creating A Galactic Plane Atlas With Amazon Web Services, 2013, [Online] Available: http://arxiv.org/abs/1312.6723

[11] R. E. Deelman, K. Vahi, G. Juve, M. Rynge, S. Callaghan, P. J. Maechling, R. Mayani, W. Chen, R. F. da Silva, M. Livny, and K. Wenger, “Pegasus, a workflow management system for science automation,” Future Generation Computer Systems, vol. 46, pp. 17-35, 2015, doi: https://doi.org/10.1016/j.future.2014.10.008.

[12] M. Johnson, G. Daues, and H. F. Chiang, "HTCondor in Astronomy at NCSA," HTCondor Week 2019 [Online] Available: https://agenda.hep.wisc.edu/event/1325/session/10/contribution/51/material/slides/0.pdf

[13] M. Ramm, M. Bayer, and B. Rhodes, SQLAlchemy: database access using Python, Addison-Wesley, Boston, MA, 2009.

[14] X. Wang, H. Schulzrinne, D. Kandlur, and D. Verma, "Measurement and Analysis of LDAP Performance," IEEE/ACM Transactions on Networking, vol. 16, no. 1, pp. 232-243, Feb. 2008, doi: https://doi.org/10.1109/TNET.2007.911335.

[15] J. Han, D. Kim, and H. Eom, "Improving the Performance of Lustre File System in HPC Environments," *2016 IEEE 1st International Workshops on Foundations and Applications of Self* Systems (FAS*W),* Augsburg, pp. 84-89, 2016, doi: https://doi.org/10.1109/FAS-W.2016.29.

[16] D. Mosberger and T. Jin, “Httperf-a tool for measuring web server performance,” SIGMETRICS Performance Evaluation, vol. 26, no. 3, pp. 31-37, Dec. 1998, doi: https://doi.org/10.1145/306225.306235.