# Kepler + CometCloud: Dynamic Scientific Workflow Execution on Federated Cloud Resources

Jianwu Wang[1], Moustafa AbdelBaky[2], Javier Diaz-Montes[2], Shweta Purawat[3], Manish Parashar[2], and Ilkay Altintas[3]

[1] Department of Information Systems, University of Maryland, Baltimore County, USA
jianwu@umbc.edu
[2] Rutgers Discovery Informatics Institute, Rutgers University, USA
moustafa.a@rutgers.edu, javidiaz@rdi2.rutgers.edu, parashar@rutgers.edu
[3] San Diego Supercomputer Center, University of California, San Diego, USA
{shpurawat, altintas}@sdsc.edu

**Abstract**
The widespread availability and variety of cloud offerings and their associated access models has drastically grown over the past few years. It is now common for users to have access to multiple infrastructures (e.g., campus clusters, cloud resources), however, deploying complex application workflows on top of these resources remains a challenge. In this paper we propose an approach that allows users to build and run scientific workflows on top of a federation of multiple clouds and traditional resources (e.g., clusters). We achieve this by integrating the Kepler scientific workflow platform with the CometCloud framework. This allows us to: 1) dynamically and programmatically provision and aggregate resources, 2) easily compose complex workflows, and 3) dynamically schedule and execute these workflows based on provenance and overall objectives on the resulting federation of resources. We demonstrate our approach and evaluate its capabilities by running a bioinformatics workflow on top of a federation composed of a campus cluster and two clouds.

*Keywords:* scientific workflow, federated clouds, dynamic workflow scheduling, provenance-based

## 1 Introduction

A scientific workflow is a common approach to construct and manage computational processes with complicated data and control dependencies and automate their executions on proper resources [12]. However, as the complexity of the scientific workflow grows, individual components within such workflows may exhibit heterogeneous behaviors or require dynamic resources. Therefore, ensuring the appropriate levels of quality of service (QoS) requires elastically combining available resources to meet the application demands at any time.

Nowadays, cloud computing is gaining traction as an on-demand and elastic computing resource for executing scientific workflows [9]. Quite often, and in addition to traditional resources (e.g., local clusters), an individual researcher has access to multiple cloud resources, and therefore needs to decide on how to utilize/combine them to execute a scientific workflow on top of them. In this context, cloud federations are being explored as means for extending as-a-service models to offer on-demand access to computing utilities, an abstraction of unlimited resources, customizable environments, and a pay-as-you-go business model. Like traditional clouds, these federations also have the capability of scaling up, down or out as needed, with the particularity that they can scale beyond the boundaries of a single cloud provider. Consequently, it is possible to construct hybrid federated infrastructures that integrate private/public clouds, local data centers, campus clusters, and supercomputers. This creates the potential for interesting marketplaces where users and applications can take advantage of different types of resources, QoS, geographical locations, and pricing models.

In this paper, we propose an approach that (1) allows users to easily describe data-driven workflows, including data sources of interest and QoS requirements; (2) is able to assimilate application requirements to transparently provision the appropriate blend of resources to meet application needs; and (3) is able to autonomously adjust at runtime the provisioned resources, considering changes in the application requirements and/or resource availabilities. Our approach is powered by CometCloud, which enables the autonomic software-defined federation of heterogeneous and distributed resources [6]. The federation is then delivered to users through Kepler, which enables the efficient design and execution of scientific workflows through its graphical user interface (GUI) and provenance recording [2]. As a result, our approach delivers an end-to-end solution that allows users to focus on their science while delegating systems-related tasks to the lower layers within Kepler and CometCloud.

The rest of the paper is organized as follows. Section 2 provides a brief background on Kepler and CometCloud. Section 3 details the architecture of our approach and the workflow scheduling mechanism. Experimental evaluation of our approach is presented in Section 4 using a bioinformatics workflow which was executed on a federation of two clouds and one traditional cluster. Section 5 presents the related work. Section 6 presents future work and the paper concludes in Section 7.

# 2 Background

## 2.1 Kepler

The Kepler project [1] aims to produce an open source scientific workflow system that allows scientists to design and efficiently execute scientific workflows. Kepler provides an intuitive GUI and an execution engine to help scientists edit and manage the execution of scientific workflows. Kepler adopts the actor-oriented modeling [12] paradigm for scientific workflow design and execution. Each actor is designed to perform a specific independent task. Actors can be implemented as atomic or composite, whereby composite actors (i.e., sub-workflows), are composed of atomic actors bundled together to perform complex operations. Actors in a workflow can contain ports to consume or produce data and communicate with other actors in the workflow through communication channels.

Another unique property of Kepler is that the order of execution of actors in the workflow is specified by an independent entity called *director*. The director defines how actors are executed and how they communicate with each other. The execution model defined by the director

[1]Kepler Project: http://www.kepler-project.org, 2016

is called the Model of Computation [12]. Since the director is decoupled from the workflow structure, a user can easily change the computational model by using the Kepler GUI to replace the director. As a result, a workflow can be executed either in a sequential manner, e.g., using the Synchronous Data Flow director, or in a parallel manner, e.g., using the Process Network director.

## 2.2 CometCloud

CometCloud [6] is an autonomic framework designed to enable highly heterogeneous, dynamically federated computing and data platforms that can support end-to-end applications with diverse and dynamic changing requirements. CometCloud uses a federation approach to aggregate heterogeneous and geographically distributed resources. These resources are exposed to users as a seamless elastic pool of resources. This federation is created dynamically and collaboratively, where resources/sites can join or leave at any point, identify themselves (using security mechanisms such as public/private keys), negotiate terms of federation, discover available resources, and advertise their own resources and capabilities [7].

### 2.2.1 Application Management in CometCloud

In order for CometCloud to manage the execution of applications, they need to be integrated with CometCloud [8]. New applications can be easily integrated by developing two simple components, namely a task generator and a worker. The task generator uses a simple API, which is used to define the properties of all tasks that need to be generated by an application in a programmable way. A set of tasks compose a single stage, and a set of stages compose the entire application. The idea is to provide users with the ability to define dynamic applications, where the tasks for each stage are created at runtime depending on previously obtained results. Results of all stages are accessible through the API. This provides tremendous flexibility as an application can evolve in different ways depending on the observed data.

On the other hand, the worker's sole responsibility is to execute tasks. The workers can execute tasks directly or through third-party, perhaps closed-source, software. In such cases where a user might be interested in executing third-party software, the resulting worker becomes a mere proxy that acts as a facade for the target software. The third-party code and workers can be installed directly on the resources (e.g., a local data center cluster), or encapsulated using VMs (e.g., in case of a cloud resource), or leveraging software container services such as Docker containers. This significantly simplifies the migration from traditional environments to our federation [1]. The worker component and any other third-party code are made available at each sites of the federation and are exposed using a federation agent. An agent can support a variety of workers and applications, each of which is identified by an application name – for simplicity we currently assume uniform naming across sites.

# 3 Kepler + CometCloud Architecture

## 3.1 Architecture

The architecture of our Kepler and CometCloud integration is shown in Figure 1. At the front end, users interact with the Kepler GUI to compose their workflow. In our approach, the execution environment is completely abstracted from the users, allowing them to focus on the details of their applications, hence they only need to specify the actors, data/control dependencies, and the QoS requirements of their workflow. At the backend, the execution

environment is managed by CometCloud. CometCloud aggregates all available resources in the federated environment and exposes them as a single pool of resources, regardless of their location and particularities. CometCloud is deployed by starting the CometCloud Workflow Manager and distributed agents that manage resources across different sites. The CometCloud workflow manager orchestrates stage executions, where each CometCloud stage corresponds to a Kepler actor. The CometCloud workflow manager only supports DAGs and loops, whereas Kepler supports more complicated workflow structures including pipeline, split, condition, loop and merge. Therefore, complex workflows can be constructed using Kepler and internally transformed into CometCloud workflows before running on the proper resources.

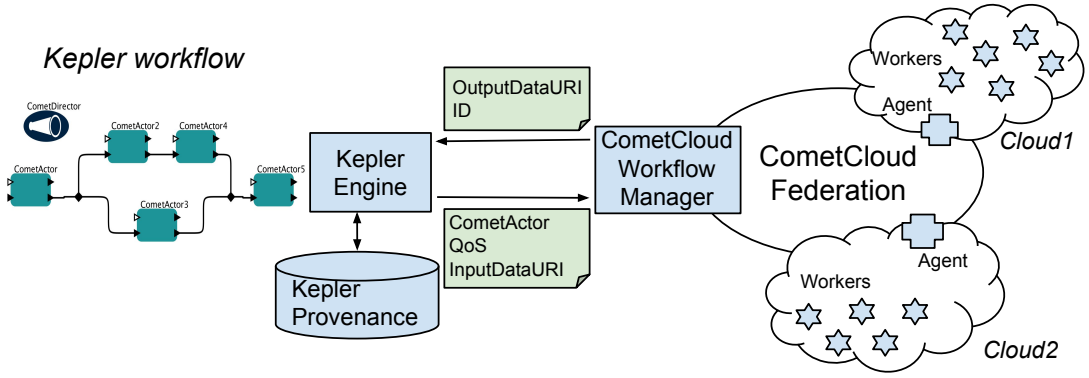



Figure 1: Architecture of the Kepler and CometCloud Integration

During the execution phase, users specify the URI for CometCloud as well as the QoS requirements for each actor in their workflow via parameter settings. The Kepler execution engine then resolves the dependencies among actors and sends each ready-to-execute actor with its QoS requirement to the CometCloud Workflow Manager in order to start its execution. CometCloud returns an ID (one per actor) to Kepler to allow Kepler to keep track of execution dependencies. This ID can also be used by downstream actors to identify the location of the data generated by upstream actors. CometCloud generates tasks for each CometCloud stage, and schedules them using the appropriate mix of resources. Moreover, CometCloud autonomously manages data movement and exploit their location to minimize data transfer overheads. Finally, to minimize communication cost, only data URIs, not data content, are transferred between Kepler and CometCloud.

## 3.2 Comet Director and Actor in Kepler

To enable the communication between Kepler and CometCloud, we have developed a *Comet Director* to manage the whole workflow execution in Kepler and a *Comet Actor* to interact with CometCloud. In addition to triggering actor executions based on their dependencies, the Comet Director can also dynamically set a deadline for each Comet Actor based on the execution history and the current execution status. The configuration of the Comet Director is shown in Figure 2. The last five parameters are CometCloud specific. They specify the locations of the Kepler engine and CometCloud server. The last parameter specifies the deadline for the whole workflow. Figure 3 shows the configuration of a Comet Actor. We can specify different

QoS for each Kepler actor by specifying the ObjectiveType and value for each actor [2]. AppGenerateClass and AppGenerateClassMethod provide the information needed for CometCloud to execute the CometCloud single-stage corresponding to the Kepler actor. For instance, a parallel AppGenerateClassMethod in CometCloud would run a stage in parallel based on its input files. The input ports of a Comet actor contain information of the input/output data URIs and stage execution ID from CometCloud.

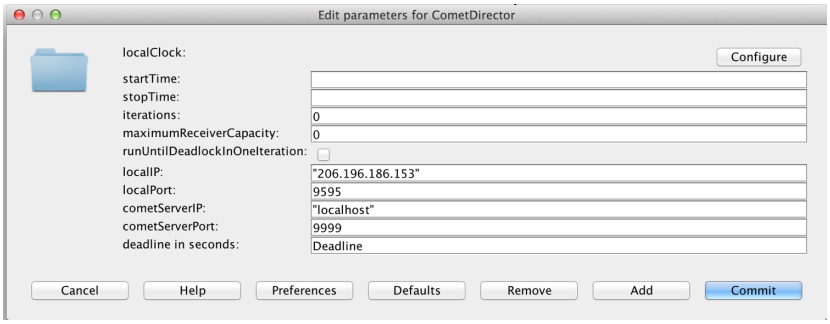



Figure 2: Comet Director

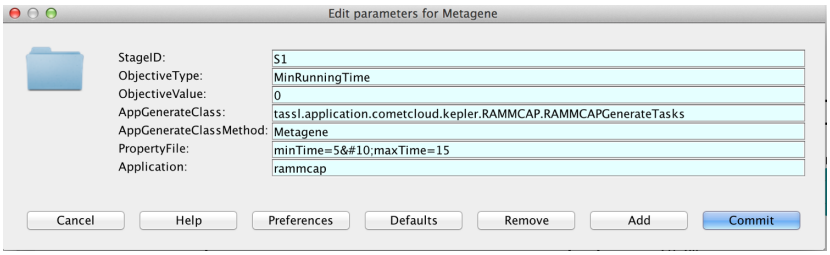



Figure 3: Comet Actor

## 3.3 Dynamic Workflow Scheduling based on Provenance

The Comet Director takes the following four steps to dynamically schedule a workflow based on provenance. The Kepler provenance module is used to mainly record the execution time of each Kepler actor.

**Step 1**: Actor Execution Time Weighting. For each workflow execution, get each actor's start time and finish time and the workflow's finish time based on provenance. The weight of the actor to the remaining workflow execution time can be calculated by $(actorFinishTime - actorStartTime)/(workflowFinishTime - actorStartTime)$.

**Step 2**: Weight Averaging. Get the arithmetic average weight of each actor from all executions for the workflow based on provenance. This calculation does not use the shortest or longest path workflow structure based logic used by many related work [24]. Instead, it uses execution statistics to get the average weight. For a workflow with a conditional fork where the condition cannot be determined until the execution reaches the fork, our approach does not try to infer/predict the condition's value for a new execution. Instead, it predicts which branch

[2] http://cometcloud.bitbucket.org/cometworkflow/workflowdefinition.html, 2016

of the fork a new execution will take based on the statistical probability calculated from its execution history.

**Step 3**: Dynamic Actor Deadline Setting. Pending actors are started once all their dependencies are solved (i.e. upstream actors are finished). The deadline set for the actor is set as $(wfDeadline - currentWFExeTime) * avgWeight(actor)$. The first part calculates the remaining deadline by subtracting current workflow execution time from overall workflow deadline. By knowing the remaining deadline and the average weight of this actor to the remaining workflow execution time, it dynamically calculates this actor's deadline.

**Step 4**: CometCloud Stage Scheduling. Based on the actor information and deadline setting, CometCloud gets a single-stage to execute. Scheduling tasks within that stage involves the autonomic scheduler performing four sub-steps: (i) retrieving the information of the available resources (i.e., resource availability, relative performance, cost); (ii) retrieving information related to the tasks to be executed (i.e., data location and task complexity); (iii) identifying the QoS objective policy selected by the user; and (iv) creating and implementing a plan to decide which resources to provision, from which site, for how long, and where to execute each task. The autonomic scheduler monitors the progress of the execution to adapt the plan to changes in the environment, failures, or any other deviations [8]. Once an actor/stage is complete, CometCloud informs Kepler to trigger the execution of the next available actors/stages until the end of Kepler workflow is reached. Kepler monitors the workflow execution and dynamically schedules the next available actors (based on step 3).

One advantage of using weighted averages for actor's execution time is that it is more accurate for workflow executions with varying input data. For example, it is common that the execution time of the tasks within an actor increases when processing larger input data. However, the weight of such actor's execution time to the overall workflow execution time remains relatively consistent. Therefore, our approach can still get a reasonable deadline for each actor even if the input data of the new workflow execution is very different from those in the historical executions. Further, our approach can also be applied to budgetary cost related objectives. Kepler supports this policy by obtaining the cost of each finished actor/stage from CometCloud and compares it to the historical cost information from its provenance.

Another advantage of using these adaptive mechanisms of setting the deadline for each actor independently, is that while each actor can have a different QoS policy, the overall workflow deadline can still be enforced. For example, if the QoS objective for an upstream actor/stage is to optimize cost (which results in a violation of the specified deadline for that stage), CometCloud would instead execute this stage using more resources or more powerful ones. However, if the deadline for that stage was violated due to external factors (e.g., resource failure or unavailability), Kepler will adjust the deadlines for the remaining stages so that the overall workflow would still be executed within the allocated overall deadline.

# 4 Evaluation

To evaluate our approach, we applied the above integration to a bioinformatics workflow called RAMMCAP (Rapid Analysis of Multiple Metagenomes with a Clustering and Annotation Pipeline) [11] using a federation of resources from three different providers.

## 4.1 Use Case

The RAMMCAP workflow addresses the computational challenges imposed by the huge size and large diversity of metagenomic data. RAMMCAP includes many bioinformatics tools for dif-

ferent functions, including function annotation, clustering, and open reading frame prediction. Some of the tools can be parallelized via data parallelism.

Figure 4 shows a simplified RAMMCAP workflow in Kepler using Comet director and Comet actors. The workflow includes nine bioinformatics tools where three of them can be parallelized, namely, tRNAscan-SE, rpsblast_for_COG and rpsblast_for_KOG. For these three tools, we set their AppGenerateClassMethod parameters so that CometCloud would run these actors in parallel based on the number of input files. In our experiments, these parallel actors generated 36 jobs each, while the rest of the stages in the workflow generated a single job each for a total of 114 jobs. Data movement was also considered and both input and output files were in the range of 50MB to 100MB.

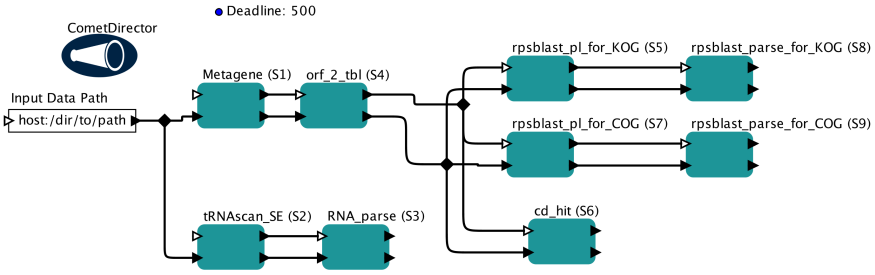



Figure 4: RAMMCAP workflow using Comet Director and Comet Actors

## 4.2 Experimental Results

We conducted a set of experiments to execute the RAMMCAP workflow across a federated multi-cloud environment with different QoS requirements. The federated environment is composed of three independent sites: an OpenStack community cloud from the Chameleon project[3], located at TACC in Texas; a public cloud (region us-west-2) from Amazon Web Services[4], located in Oregon; and a campus cluster located at Rutgers University in New Jersey. Detailed characteristics of the resources used at each infrastructure are presented in Table 1. The performance of the resources is represented by the speedup and has been experimentally calculated as a function of the performance of the Chameleon_Medium instance using the unix benchmark, which was used to characterize the workload.

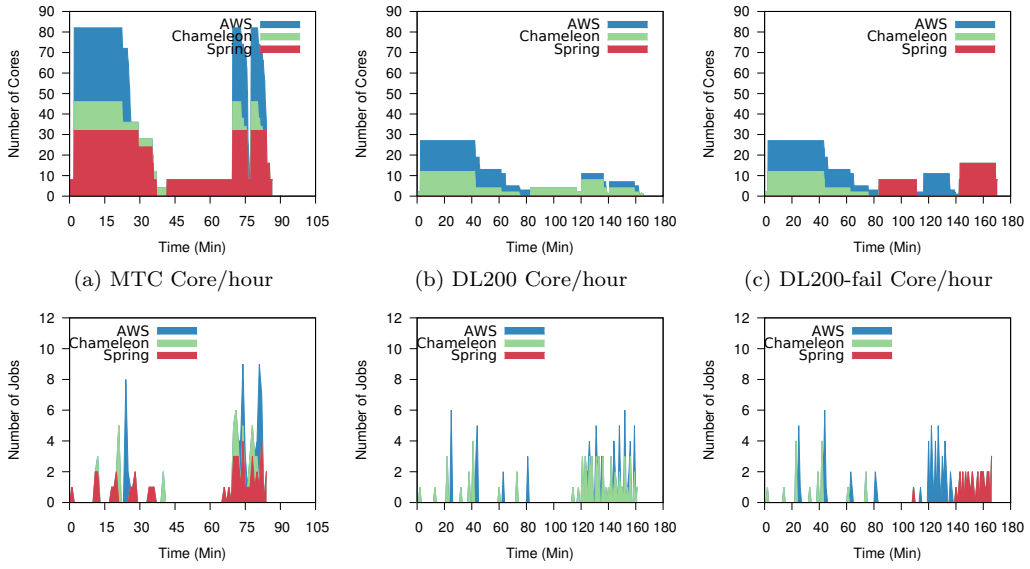
Table 1: Resources available at each site and their characteristics.

| VM type† | #Cores | Memory | Max. VMs‡ | Speedup | Cost ($)◇ |
|---|---|---|---|---|---|
| Chameleon_Large | 4 | 8 GB | 2 | 1.99 | 0.24 |
| Chameleon_Medium | 2 | 4 GB | 3 | 1 | 0.12 |
| Chameleon_Small | 1 | 2 GB | 2 | 0.54 | 0.06 |
| AWS_Large | 2 | 8 GB | 6 | 1 | 0.25 |
| AWS_Medium | 2 | 4 GB | 6 | 0.99 | 0.13 |
| AWS_Small | 1 | 2 GB | 3 | 0.5 | 0.07 |
| Spring | 8 | 24 GB | 4 | 2.5 | 0.6 |

Note: † – Name of the site followed by the type of VM.
‡ – Maximum number of available VMs per type.
◇ – Real cost per hour for AWS, simulated cost per hour for Chameleon and Spring.

[3]Chameleon Cloud: https://www.chameleoncloud.org, 2016
[4]Amazon Elastic Compute Cloud: https://aws.amazon.com/ec2/, 2016

Using this environment, we first executed the workflow indicating that we wanted the result at the earliest possible time, i.e. using minimum time of completion (MTC) as the workflow objective type. The total time taken for the MTC execution was about 86.5 minutes. Next, we executed the workflow indicating that we wanted to execute the workflow within a given deadline. We used a deadline that is a 200% larger than the time required for the workflow to be executed under the MTC policy, which amounts to 173 minutes. In this deadline case, we performed two experiments: a) the execution environment does not change during the experiment, labeled DL200; and b) Chameleon site goes down around minute 83 of our experiment and AWS site becomes unavailable around minute 140 of our experiment, labeled DL200_fail. The results are shown in Figure 5 and in Figure 6.




(a) MTC Core/hour (b) DL200 Core/hour (c) DL200-fail Core/hour

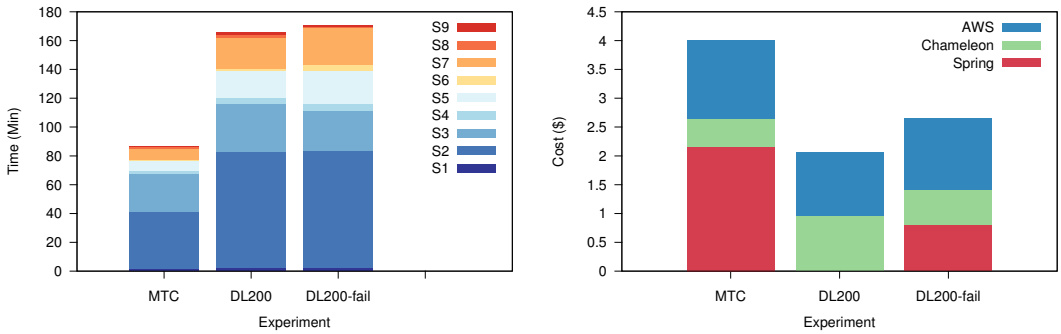
(d) MTC Throughput (e) DL200 Throughput (f) DL200-fail Throughput

Figure 5: Detailed experimental results.

Figure 5 shows the amount of resources used over time for each one of the experiments, at the top, and the throughput represented as the number of jobs completed per unit of time, at the bottom. Since the parallelism of our workflow is limited to three actors, we can observe a similar distribution of resources with an increase on resource usage and throughput at the time of executing such parallel actors. We can also observe, since the deadline in DL200 was large enough, CometCloud was able to conserve the more expensive resource (Spring) and use only Chameleon and AWS to execute the entire workflow. We can also observe that in the DL200_fail case, after Chameleon fails (minute 83), CometCloud fault tolerance mechanisms react by reinserting failed tasks, rescheduling the workload, and provisioning the appropriated resources (which includes Spring) to satisfy the deadline. Furthermore, when AWS becomes unavailable (minute 140), the remaining tasks are all rescheduled and executed on Spring.

Figure 6 shows the overall execution time of the workflow for each experiment (Figure 6a) as well as the cost per experiment and infrastructure (Figure 6b). We can observe that when increasing the deadline the execution time increases and the cost of execution decreases.

From the experimental results, we can draw the following conclusions: 1) MTC and




(a) Execution Time

(b) Workflow Execution Cost

Figure 6: Summary of experimental results

DL200_fail executions utilize all three Cloud resources, often in parallel, whereas DL200 uses only two Cloud resources to minimize cost; 2) the first execution is able to finish with minimal time among the three executions; 3) the execution with longer deadline (namely DL200) finishes with less cost; 4) the third execution is able to finish the workflow by mitigating some executions to the other Cloud resource and to Spring with the additional execution time and budgetary cost.

# 5 Related Work

Efficient workflow execution on federated (cloud) infrastructure remains an active research topic. Workflow scheduling algorithms produce a mapping of workflow-tasks to resources on the cloud. Cloud service providers have APIs that enable users to automatically scale resources to meet workload demands. The effect of cloud's dynamic elasticity on performance criteria such as data-transfer rates or make span is too complex to model for most workflows. This complexity has piqued the interests of the community, resulting in numerous heuristics and approximations. [5] demonstrates that different schedules of the same application can result in significantly different costs over the cloud. It establishes that right resource allocation can significantly reduce the overall cost while maintaining performance.

[13] utilizes auto scaling to dynamically adapt to cost effective configurations to accommodate changing workload and resource problems while meeting deadlines. It abstracts a resource as a Virtual Machine (VM) and characterizes each machine using size and cost metrics. The optimization criteria comprise overall cost minimization and soft deadlines. [23] presents an approach for dynamic and autonomous resource allocation handling the constraints and limitations imposed by the resource allocation problem.

[20] uses a predictive model to generate performance estimate for each task and dynamically finds the best resource configuration. It utilizes an iterative control process that uses data mining to continuously map cloud resources while meeting performance and cost constraints. [17] performs clustering of sub-tasks and allocates formed clusters to different resources using a heuristic while taking into account the QoS metrics (cost, time and reliability). The approach utilizes resource indexing to find available resources.

[4] investigates the performance and cost implication of extending local infrastructure by elastically allocating additional resources from the cloud. [15] presents an Adaptive Heuristic

for a hybrid cloud environment that considers workflow level optimization to minimize the cost of execution while meeting other QoS metrics such as budget, deadline and data placement. [16] formulates the cloud outsourcing problem as a binary integer program and analyzes the cost of running a deadline constrained application in a hybrid cloud environment.

[10] presents a priority based fault tolerant scheduling approach that deploys redundancy and re-execution of failed tasks to meet performance criteria. [25] presents an Improved Genetic Algorithm that maximizes resource utilization in the cloud by launching Virtual Machines (VMs) at economical sites. [21] approaches the cloud based workflow scheduling problem in a bottom-up manner. It takes a hierarchical scheduling approach and finds an optimal task to virtual machine mapping while maintaining the QoS requirements. [22] proposes a particle swarm optimization based approach that uses both data transfer and computation cost in consideration for scheduling workflows in the cloud.

[3] presents a Multiple Criteria Decision Analysis approach and formulates the task of finding the right type and size of resources required for a computation as a resource allocation problem with multiplicity. [14] presents a dynamic scheduler that allocates resources on multiple cloud providers with different cost models while maintaining a user-defined budget constraint.

Our approach distinguishes itself by giving users the capability to define an abstract resource structure over a heterogeneous and distributed set of resources. This software defined federation transparently performs resource provisioning, keeping the user focused on solving the main scientific problem rather than hunting for proper infrastructure configurations and managing costs.

# 6 Future Work

We are currently working on improving the integration between Kepler and CometCloud to support more complicated scenarios and provide smarter scheduling. These efforts are discussed below.

## 6.1 Data Mining based Actor Objective Setting

To achieve more accurate actor objective setting, we plan to replace the arithmetic weight average with data mining-based prediction. It is very common that the same workflow runs many times with different input data sizes and parameter values. Although the current weighted averaging approach is more reasonable than using absolute time/cost value averaging, it still does not consider the unique configurations of each execution. We have found that the execution time/cost weight of an actor to the whole workflow time/cost might vary a lot because of factors including input data sizes and parameter values [18]. With enough execution history data in provenance, we could utilize data mining techniques, such as decision tree to train a model on which factors affect actor execution time/cost weights [19]. When a new actor execution starts, we can use the trained model to get the execution time/cost weights.

## 6.2 Science-as-a-Service Platform

We envision enhancing our current approach to create a Science-as-a-Service Platform, where scientists define their QoS requirements in terms of science. In this way, they can define different configurable parameters in their workflows and the expected results. For example, there may exist some operations that have certain degrees of freedom in their configuration or they may potentially use different methods with similar outcome. In such cases, a scientist could define

different levels of accepted QoS for the solution, e.g., accuracy, error margins, etc., together with other requirements such as budget or deadline. As a result, we could enable an organic platform that can consider all these variables or "knobs" offered by the application and try to allocate the workload in the best possible way by initiating a bidirectional negotiation between the workflow manager and the autonomic scheduler. In this way, not only the execution environment adapts to meet the application needs, but the application also adapts by modifying previously defined "knobs" to facilitate finding a solution that is feasible given the current status of the available resources in our execution environment.

# 7 Conclusion

Although cloud computing provides a wide variety of on-demand resources, finding a proper execution plan for a scientific workflow on a set of multiple cloud resources is non-trivial. In this paper, we presented an integration of the Kepler scientific workflow system and the CometCloud software-defined federation framework to achieve dynamic workflow execution on federated cloud resources. The proposed approach enables users to build scientific workflows that are agnostic to the execution environment and underlying resources, namely only describing workflow dependencies and the overall QoS requirements. Kepler and CometCloud work together to get the sub-objectives for each workflow stage/actor based on provenance, find the proper resource for executing a stage/actor, and recalculate sub-objectives for downstream actors based on the current workflow status. Our approach is evaluated using a bioinformatics workflow on resources from three resource providers to show its functionality and advantages.

**Acknowledgments:**The research presented in this work is supported in part by National Science Foundation (NSF) via grants numbers ACI 1339036, ACI 1310283, ACI 1441376. This project used resources from Chameleon supported by NSF OCI-1419152. The research at UMBC was supported by a startup fund. The research at Rutgers was conducted as part of the Rutgers Discovery Informatics Institute (RDI2).

# References

[1] M. Abdelbaky, J. Diaz-Montes, M. Parashar, M. Unuvar, and M. Steinder. Docker containers across multiple clouds and data centers. In *2015 IEEE/ACM 8th International Conference on Utility and Cloud Computing (UCC)*, pages 368–371, Dec 2015.

[2] I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludaescher, and S. Mock. Kepler: An extensible system for design and execution of scientific workflows. In *Intl. Conference on Scientific and Statistical Database Management (SSDBM)*, Santorini Island, Greece, 2004.

[3] F. Chang, J. Ren, and R. Viswanathan. Optimal Resource Allocation in Clouds. In *2010 IEEE 3rd International Conference on Cloud Computing (CLOUD)*, pages 418–425, July 2010.

[4] M. D. de Assuncao, A. di Costanzo, and R. Buyya. Evaluating the cost-benefit of using cloud computing to extend the capacity of clusters. In *18th ACM International Symposium on High Performance Distributed Computing*, HPDC '09, pages 141–150, New York, NY, USA, 2009.

[5] E. Deelman, G. Singh, M. Livny, B. Berriman, and J. Good. The cost of doing science on the cloud: The Montage example. In *High Performance Computing, Networking, Storage and Analysis, 2008. SC 2008. International Conference for*, pages 1–12, Nov. 2008.

[6] J. Diaz-Montes, M. AbdelBaky, M. Zou, and M. Parashar. Cometcloud: Enabling software-defined federations for end-to-end application workflows. *Internet Computing, IEEE*, 19(1), 2015.

[7] J. Diaz-Montes, Y. Xie, I. Rodero, et al. Federated computing for the masses - aggregating resources to tackle large-scale engineering problems. *CiSE Magazine*, 16(4):62–72, 2014.

[8] J. Diaz-Montes, M. Zou, R. Singh, S. Tao, and M. Parashar. Data-driven workflows in multi-cloud marketplaces. In *IEEE Cloud 2014*, 2014.

[9] C. Hoffa, G. Mehta, T. Freeman, E. Deelman, K. Keahey, B. Berriman, and J. Good. On the use of cloud computing for scientific workflows. In *eScience, 2008. eScience'08. IEEE Fourth International Conference on*, pages 640–645. IEEE, 2008.

[10] S. M. S. B. Jayadivya S K, S. Jaya Nirmala. Fault tolerant workflow scheduling based on replication and resubmission of tasks in cloud computing. pages 996–1006, June 2012.

[11] W. Li. Analysis and comparison of very large metagenomes with fast clustering and functional annotation. *BMC Bioinformatics*, 10:359, 2009.

[12] B. Ludaescher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger-Frank, J. Jones, M.and Lee, and Y. Zhao. Scientific workflow management and the Kepler system. *Concurrency and Computation: Practice & Experience, Special Issue on Scientific Workflows*, 18(10):1039–1065, 2006.

[13] M. Mao and M. Humphrey. Auto-scaling to minimize cost and meet application deadlines in cloud workflows. In *High Performance Computing, Networking, Storage and Analysis (SC), 2011 International Conference for*, pages 1–12, Nov. 2011.

[14] A. Oprescu and T. Kielmann. Bag-of-Tasks Scheduling under Budget Constraints. In *2010 IEEE Second International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 351–359, Nov. 2010.

[15] M. Rahman, X. Li, and H. Palit. Hybrid Heuristic for Scheduling Data Analytics Workflow Applications in Hybrid Cloud Environment. In *2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW)*, pages 966–974, May 2011.

[16] R. Van den Bossche, K. Vanmechelen, and J. Broeckhove. Cost-Optimal Scheduling in Hybrid IaaS Clouds for Deadline Constrained Workloads. In *2010 IEEE 3rd International Conference on Cloud Computing (CLOUD)*, pages 228–235, July 2010.

[17] P. Varalakshmi, A. Ramaswamy, A. Balasubramanian, and P. Vijaykumar. An Optimal Workflow Based Scheduling and Resource Allocation in Cloud. In A. Abraham, J. L. Mauri, J. F. Buford, J. Suzuki, and S. M. Thampi, editors, *Advances in Computing and Communications*, number 190 in Communications in Computer and Information Science, pages 411–420. Springer Berlin Heidelberg, July 2011. DOI: 10.1007/978-3-642-22709-7_41.

[18] J. Wang, D. Crawl, I. Altintas, K. Tzoumas, and V. Markl. Comparison of distributed data-parallelization patterns for big data analysis: A bioinformatics case study. In *Proceedings of the Fourth International Workshop on Data Intensive Computing in the Clouds (DataCloud)*, 2013.

[19] J. Wang, D. Crawl, S. Purawat, M. Nguyen, and I. Altintas. Big data provenance: Challenges, state of the art and opportunities. In *IEEE Big Data*, pages 2509–2516, 2015.

[20] L. Wang, R. Duan, X. Li, S. Lu, T. Hung, R. Calheiros, and R. Buyya. An Iterative Optimization Framework for Adaptive Workflow Management in Computational Clouds. In *2013 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pages 1049–1056, July 2013.

[21] Z. Wu, X. Liu, Z. Ni, D. Yuan, and Y. Yang. A market-oriented hierarchical scheduling strategy in cloud workflow systems. *The Journal of Supercomputing*, 63(1):256–293, Mar. 2011.

[22] Z. Wu, Z. Ni, L. Gu, and X. Liu. A Revised Discrete Particle Swarm Optimization for Cloud Workflow Scheduling. In *2010 International Conference on Computational Intelligence and Security (CIS)*, pages 184–188, Dec. 2010.

[23] Y. Yazir, C. Matthews, R. Farahbod, S. Neville, A. Guitouni, S. Ganti, and Y. Coady. Dynamic Resource Allocation in Computing Clouds Using Distributed Multiple Criteria Decision Analysis. In *IEEE 3rd International Conference on Cloud Computing (CLOUD)*, pages 91–98, July 2010.

[24] J. Yu, R. Buyya, and K. Ramamohanarao. Workflow scheduling algorithms for grid computing. In *Metaheuristics for scheduling in distributed computing environments*, pages 173–214. Springer, 2008.

[25] H. Zhong, K. Tao, and X. Zhang. An Approach to Optimized Resource Scheduling Algorithm for Open-Source Cloud Systems. In *ChinaGrid Conference (ChinaGrid), 2010 Fifth Annual*, pages 124–129, July 2010.