4

Adaptive Execution of Scientific Workflow Applications on Clouds

Rodrigo N. Calheiros, Henry Kasim, Terence Hung, Xiaorong Li, Sifei Lu, Long Wang, Henry Palit, Gary Lee, Tuan Ngo, and Rajkumar Buyya

CONTENTS

Summary

Many e-science applications can be modeled as workflow applications. In this programming model, scientific applications are described as a set of tasks that have dependencies between them. Clouds are natural candidates for hosting such applications. This is because some of their core characteristics, such as rapid elasticity, resource pooling, and pay per use, are well suited to the nature of scientific applications that experience variable demand, spikes in resource (i.e., of the central processing unit [CPU] or disk) utilization, and sometimes, urgency for generation of results. As current workflow management systems (WfMSs) cannot support efficient and automated execution of workflow in clouds that support adaptive execution, fault tolerance, and data privacy, in this chapter we detail the requirements of a WfMS that supports these requirements, its architecture, and an application scenario involving simulation of Singapore's public transport system.

4.1 Introduction

Many e-science applications can be modeled as *workflow applications.* In this programming model, scientific applications are described as a set of tasks that have dependencies between them. Normally, this dependency is expressed in the form of input and output (I/O) files. It means that, before one task can execute, it needs the tasks it depends on to have completed their execution and the files they generate to already be available as input. Well-known application domains where workflow applications are used include astrophysics, bioinformatics, and disaster modeling and prediction, among others.

Scientists have been successfully executing this type of application on supercomputers, clusters, and grids. Recently, with the advent of clouds, scientists started investigating the suitability of this infrastructure for workflow applications.

Clouds are natural candidates for hosting workflow applications. This is because some of their core characteristics, such as rapid elasticity, resource pooling, and pay per use, are well suited to the nature of scientific applications that experience variable demand, spikes in resource (i.e., of the central processing unit [CPU], disk) utilization, and sometimes, urgency for generation of results. Furthermore, recent offerings of high-performance cloud computing instances make it even more compelling for scientists to adopt clouds as the platform of choice for hosting their scientific workflow applications.

The execution of workflow applications is a demanding task. Tasks, sometimes in the order of hundreds, need to have their execution coordinated. They have to be submitted for execution in a specific virtual machine (VM), and the required input files need to be made accessible for the application. This may require the transfer of huge amounts of data between computing hosts. Reception of user input, data transfers, task executions, and VMs can fail; in this case, some action has to be carried out to reestablish the execution of the application. Examples of such actions are retrying the data transfer, rescheduling the task, or starting a new VM to execute the remaining tasks. These activities are carried out by software called *workflow management systems* (WfMSs). Examples of well-know, WfMSs are Pegasus [1], Taverna [2], Triana [3], and Cloudbus Workflow Engine [4].

At the same pace that infrastructures and platforms evolve, so do the scientific applications using such infrastructures and platforms. The amount of data generated by scientific experiments is reaching the order of terabytes per day, and huge capacity is required to process this data to enable scientific discoveries. Therefore, WfMSs also need to evolve to support huge data sets and the complex analytics required to extract useful insights from the generated data. Even more important, if data are continuously generated, WfMSs need to support real-time capabilities. This has to be achieved at the same time that other nonfunctional requirements, such as data privacy, are enabled.

Although this information is truth regardless of the specific infrastructure hosting the workflow application, even more complexity is added to the system when the applications are executed in clouds. This is because extra capabilities are required to enable the WfMS to select the right number of resources of the right type so that the computational task is performed within a user-defined time frame and budget.

As current WfMSs cannot support efficient and automated execution of workflow in clouds that support adaptive execution, fault tolerance, and data privacy, we developed extensions to a workflow engine [4] to support such features. In this chapter, we detail the requirements of such a system, its architecture, and the application scenario explored, along with an evaluation of the system and a discussion of lessons learned during its development.

4.2 Workflow Applications

The workflow programming model is undoubtedly one of the most prominent programming models in e-science, being used in a range of domains, including bioinformatics, astrophysics, and disaster modeling, to name a few. In this model, one application (job) is composed of a number of tasks that have execution dependencies between them. Typically, the dependency is related to I/O: One task depends on the output of another (or other) task(s) as its input; therefore, it cannot be executed until such data are available (normally, after the execution of the original task is completed).

Variations of the model exist in which the workflow also contains conditional branches (i.e., particular tasks that compose the workflow may or may not be executed depending on the results of previous tasks), loops (for which execution of specific sections of the workflow is repeated), and when tasks are allowed to start execution before predecessors complete execution.

Without loss in generality, a workflow application can be formally represented by a directed acyclic graph (DAG) whose vertices represent tasks and the directed edges represent the dependencies between tasks: An edge A $\rightarrow$ B indicates that task B depends on task A for its execution. Such a representation of workflow applications is also known as DAG. A simple workflow is depicted in Figure 4.1.

Traditionally, workflow applications have been extensively deployed in high-performance infrastructures such as supercomputers and clusters [5]. When deployed on such infrastructures, emphasis was given in reducing the execution time of the workflow by optimizing the utilization of the resources available for the workflow. When grids became available, they were also used for workflow execution [6, 7]. This added complexity to the scheduling process because it was possible that resources available for execution

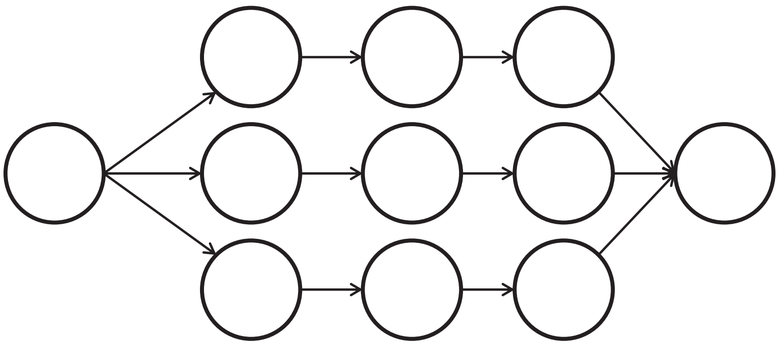

FIGURE 4.1
Graphical representation of a simple scientific workflow.

were distributed, and thus data movement across wide distances might be necessary. Even in this case, focus was still on execution time minimization.

Cloud computing adds a new dimension for workflow execution related to the financial cost of using a virtually infinite amount of resources for workflow execution. This means that the only limitations to the available resources, and consequently the improvements in execution time, are the available *budget* for workflow execution and the *structure of the workflow* itself, which determines the maximum amount of tasks that can be executed in parallel in the infrastructure. Clouds also brought other challenges for workflow management and execution. They are discussed in the next section.

4.3 Requirements for Adaptive Execution of Workflows on Clouds

Although modern WfMSs already support clouds as the platform supporting the execution of workflow applications, many desirable features are still absent in the WfMSs. This is because current WfMSs for clouds are derived from projects in the area of grid computing. Therefore, many of their features are optimized for grids and thus are unable to obtain the most key aspects of clouds, such as rapid elasticity.

In this sense, clouds add extra complexity to WfMSs because the amount of resources that WfMSs can provision for executing the workflow is virtually infinite, as long as there is budget available to spend on the workflow execution process. Thus, different from existing algorithms and approaches that operated with the goal of obtaining the most from the resources available for the application, cloud-enabled WfMSs can assume that the main restriction of the system is the budget rather than resources, and its goal is balancing utilization, cost, and reduction of execution time [8].

Li et al. [8] also identified the following requirements for cloud-enabled workflows:

1. *Dynamic resource provisioning and deadlines:* This is the capability of acquiring and releasing resources as required to accommodate the tasks of the workflow and to enable their completion within a user-specified deadline. This is an important feature because it enables execution of *mission-critical* workflow applications that need to be completed before the deadline for the computation to have value. An example of such mission-critical workflows is disaster management workflows. Consider, for example, the architecture depicted in Figure 4.2. A disaster management workflow application suite may support management of many types of natural disasters, such as floods, cyclones, and bushfires. When one such disaster strikes, the corresponding management application needs to be executed in public and private clouds to provide information that will be used by disaster mitigation and rescue teams. If the application takes too long to execute, the teams will not have time to act based on the information provided, which results in wasted time (and money) invested in the execution of the workflow in the cloud and even further losses in terms of lives and property damage that would have been prevented if the rescue and mitigation teams had access to the information in appropriate time.
2. A*daptive task/workflow/user scheduling:* This relates to the capability of reacting to conditions faced during workflow execution to maintain the balance between cost, utilization, and execution time. In the context of this requirement, a change in conditions means adapting to changes in user requirements at runtime (e.g., increased/reduced budget, increased/reduced application deadline).
3. *Fault tolerance:* This is the capacity to automatically react to changes in the available number of resources or tasks to be processed because of failures and the capability to adapt to situations if the performance delivered by cloud resources is below that contracted or historically observed.
4. S*ecurity-conscious data migration and data privacy:* Given that the data being processed by the WfMS can be sensitive, mechanisms for protection of the data, either during transfer or once stored in a public cloud, must be available. The applied method should also enable auditing of accesses and modifications in the data.
5. *Application management:* This requirement involves the capability to collect and process information about the system status and monitor the platform and the application in real time. This requirement also includes a capacity for presentation of comprehensive information to users about the resources (utilization, performance, etc.) and

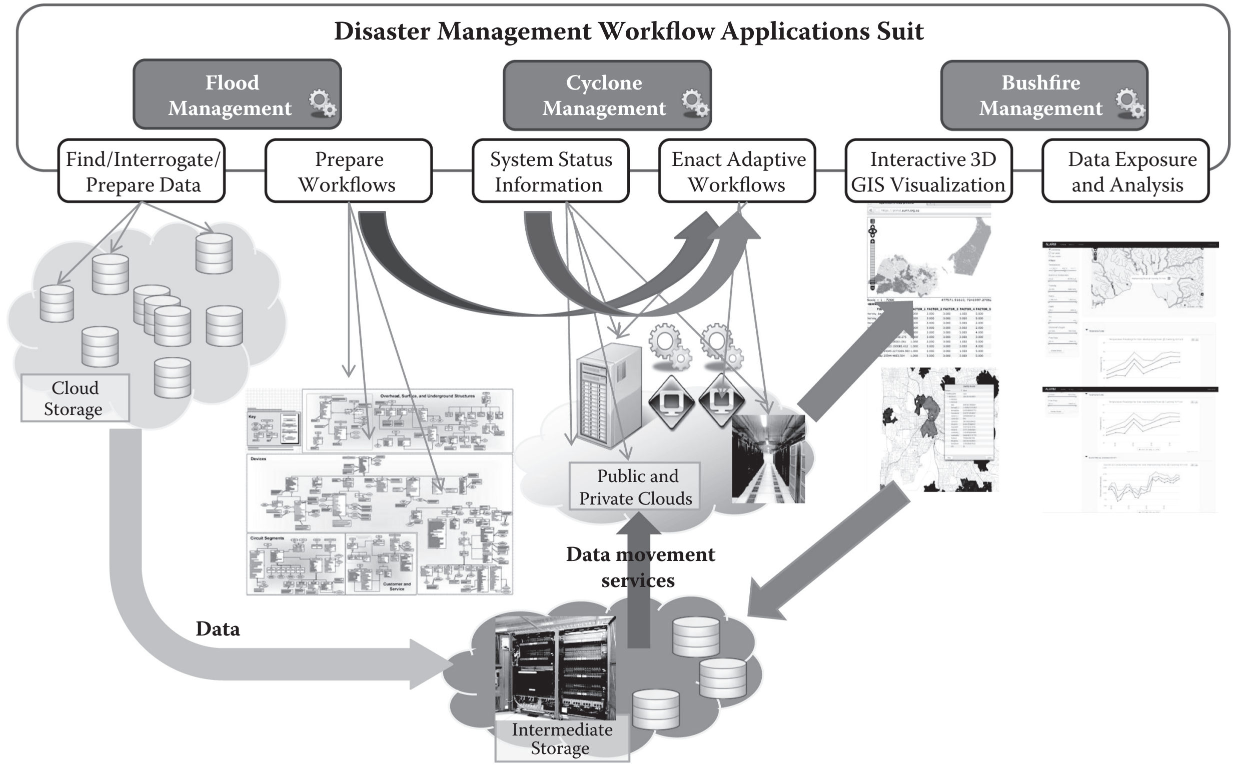
Disaster Management Workflow Applications Suit
Flood Management
Cyclone Management
Bushfire Management
Find/Interrogate/ Prepare Data
Prepare Workflows
System Status Information
Enact Adaptive Workflows
Interactive 3D GIS Visualization
Data Exposure and Analysis
Cloud Storage
Public and Private Clouds
Data movement services
Data
Intermediate Storage

FIGURE 4.2
Architecture for workflow-enabled disaster management applications.

tasks so the cost-benefit analysis of utilization of the cloud can be undertaken and the utilization of cloud computing for workflows can be justified.

These requirements were addressed while we developed an adaptive system for execution of a workflow for agent-based simulation in hybrid clouds. The application is detailed next.

4.4 Case Study

A city is sustainable only if it can accommodate economic and population growth while ensuring the well-being of its people and environment [9]. Therefore, reaching sustainability becomes harder when the growth of a population is high or when the growth occurs in areas of high density, such as Singapore.

Singapore's land area has increased from 581 km^2 in the 1960s to 716 km^2 in 2012; its population in the same period has grown from 1.6 million to 5.3 million [10]. To maintain reasonably good economic growth, the Singapore government has projected a need for the population to reach 6.9 million by 2030. However, the land area is only slated to grow to about 800 km^2 in the same period. The disparity in the growth rate of population versus land area means that there is increasing strain on space and the service infrastructure. It is crucial for the planning agencies to adopt a scientific approach to understanding the urban fabric and how it can adapt to social, economic, and environmental changes.

One key aspect to improve the quality of living of city inhabitants is public transport. There is a need for efficient transport covering the biggest extension of the city as possible and running with enough frequency so people are motivated to use it rather than using cars. In this sense, Singapore's public transport network is ranked among the best in the world. Its Mass Rapid Transit (MRT) train network comprises 102 stations distributed over four main lines, with a total of almost 150 km of rail lines. It currently serves around 2.5 million commuters per day, which represents more than 75% of the total public transport users [11].

The number of commuters and the high frequency of trains (running in intervals as short as 90 seconds) make it a complex system. Furthermore, even a minimal disruption in the operation of one train can cascade over several lines, affecting hundreds of thousands of commuters.

This complex and sensitive system will be subject to even further pressure as the population increases. Therefore, tools are needed to help planners evaluate the effects that disruptions would have over the whole system.

This fact motivated us to adopt a data-driven approach to understanding the dynamics of the public transport system in Singapore. To achieve that, a scalable complex system modeling for a sustainable city (S^3) has been developed to study how the city will behave under different planning scenarios.

The goal of S^3 is to provide insights to users on what-if scenarios for a day-to-day public transport system by leveraging on a synthetic journey function that generates agent-based models for public transport dynamics simulation. This insight will provide information on the future public transport infrastructure preparedness to handle the growing population and the preparedness for emergencies in cases of breakdowns in the public transport system.

Scaling areas that we address in this context are (1) the extract-transform-load (ETL) or preprocessing that is required to train the synthetic journey function that generates the agent-based model; (2) the agent-based generation required to generate millions of agents that represent the increasing population and public transport infrastructure; and (3) the large-scale agent-based simulation that is required to handle, track, and process each of the agents and to support complex interactions between agents to provide insight on what-if scenarios for the public transportation system in Singapore.

We tackled the large-scale computation requirements by designing agent-based complex system modeling supported by an adaptive cloud WfMS [12] for workflow scheduling and handling big data and dynamic resource scaling on public and private clouds.

The S^3 application has three phases: preprocessing, data analysis, and agent-based simulation. Figure 4.3 shows our S^3 application architecture, which comprises an adaptive cloud WfMS, ETL or preprocessing algorithm, data analysis algorithm, and agent-based simulation.

ETL or preprocessing. The synthetic data set for the application is based on the studies of trends and random sampling of daily public commuters' activities in Singapore. It consists of 1-second time granularities for 7 days' duration with approximately 3 million journeys per day. Based on the synthetic data set, we extract and transform the data for travel duration for each origin-station to destinations-station (OD-pair) of 90 x 90 by three different route choices. The order of complexity in this phase is $O(n^2)$, where n represents the number of stations.

Data analysis. The objective of this phase is to understand commuter demand and, based on data analysis results, create or improve the journey function of all possible OD pairs, possible routes for each OD pair, and temporal travel demand. The order of complexity in this phase is also $O(n^2)$, where n represents the number of stations.

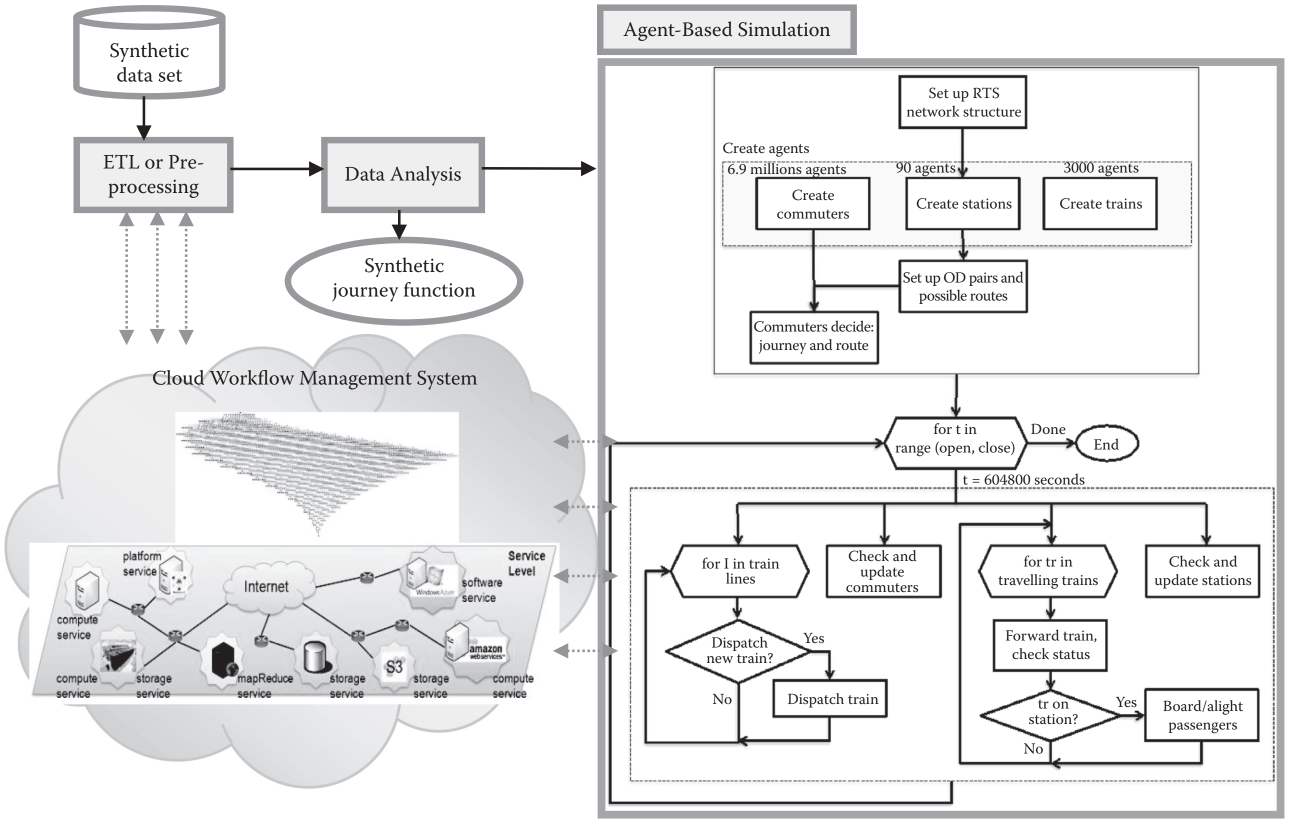

FIGURE 4.3
S^3: architecture, concepts, and technologies.

Agent-based simulation. In this phase, we simulate the actions and interactions of autonomous agents. This agent-based simulation consists of agent granularity, adaptive agent process, decision-making heuristics, and agent interactions. Agent granularity refers to the number of agents specified at various scales. The adaptive agents process refers to the action that an agent takes when a situation occurs (redefining the decision-making heuristics). Decision-making heuristics refer to rules or behaviors of an agent. Agent interaction refers to the complexity of communications or interactions between agents.

There are three types of agents in the S^3 application: commuters, stations, and trains. Each of these agents has its own attributes, adaptive agent process, decision-making process, and agent interactions, as summarized in Table 4.1. The order of complexity in this phase is $O(n^3)$ due to the interactions between agents on simulation time interval or $O(tn^2)$, where n represents the number of agents and t represents the simulation time steps.

Data requirements. The size and quantity of the data set that is generated is large. The size of the data can easily take up a few gigabytes each day. For example, the data set consists of 7 days of public transportation journeys for each individual, with approximately 3 million journeys per day. As for the agent-based simulation, we simulate the growing population as 6.9 million. This translates into approximately 14 million journeys (travel and return) performed for each simulated day.

Computation requirements. For agent-based simulation, millions of agents are created to simulate the future infrastructure and dynamics of the transportation system in Singapore. In total, the system manages 7 million agents that have their own attributes, adaptive agent process, decision-making process, and interactions with other agents. Furthermore, there is complexity of agent interactions and tracking for the simulation interval at 1-second granularity.

TABLE 4.1
Agent-Based Simulation Characteristics

	Commuter Agents	Station Agents	Train Agents
Agent granularity	6.9 million agents	90 agents	Approximately 200 agents
Attributes	12 attributes	9 attributes	16 attributes
Adaptive agent process	1 adaptive process	—	2 adaptive processes
Decision-making heuristics	5 decision-making heuristics	2 decision-making heuristics	5 decision-making heuristics
Agent interactions	• Station • Train	• Commuter • Train	• Commuter • Station

To support not only these requirements for data and computation but also the requirements listed in the previous section, we proposed and developed a workflow middleware whose architecture is described next.

4.5 System Architecture

The requirements presented previously are addressed by software middleware comprising a WfMS augmented with capabilities for data analytics integrated as a second layer above the WfMS. The overall organization of the system is depicted in Figure 4.3. It shows the S^3 application architecture, which consists of the adaptive cloud WfMS, the ETL or preprocessing algorithm, the data analysis algorithm, and the agent-based simulation.

Cloud WfMS system. The cloud WfMS is responsible for workflow scheduling, big data handling, and dynamic resource scaling on hybrid clouds. The Cloud WfMS comprises the workflow engine, task dispatcher, and resource management. The workflow scheduling coordinates the execution of tasks, handles communication between components, implements the scheduling algorithm, and manages the execution of applications on distributed resources. The task dispatcher component submits tasks to resources for execution. The resource management component interacts with the cloud infrastructure to enable resource allocation.

Preprocessing and data analysis. This component is responsible for managing preprocessing and data analysis activities that are required to train the synthetic journey function that generates the synthetic journey. It tackles the scalability challenge by dynamically scaling up the number of VM instances; thus, the preprocessing processes are executed in parallel. Since this is a computationally intensive task with a long duration and the total number of origin-station and destination-station pairs is large (composed of more than 8,000 pairs), VM instances are pooled from a hybrid cloud where each VM instance processes the travel duration for each origin-station and destination-station pair.

Agent-based simulation. There are three phases of agent-based simulation: agent creation, attribute definition, and simulation execution. Our module is able to scale the process of agent-based generation in orders of magnitude of up to millions of agents. Further in this chapter, we demonstrate the process for 6.9 million commuter agents, 90 station agents, and 200 train agents. The activities of the process of simulation execution are (1) time series simulation with 1-second intervals; (2) tracking of each agent, which includes checking and updating each agent's state; (3) a decision-making process for each agent (e.g., dispatch the train at simulation time *t*); (4) adaptive agent process that allows agents to adapt to different situations (e.g., when a train arrives at a station, commuter agents need to board or leave the train); (5) interactions between agents (e.g., communication between train agents and station agents when the train arrives at the station, communication between commuter agents and train agents when the commuter boards the train) and management of tasks and data flows on the hybrid cloud utilizing the cloud WfMS.

A discussion of the implementation aspects of the architecture and its performance is presented next.

4.6 Discussion and Lessons Learned

The agent-based simulation is based on three phases: create agents, define attributes, and run the simulation. To test the scalability of the model, we evaluated two different setups. The first one uses the ZeroMQ (ZMQ) technology [13] in our hybrid cloud. ZMQ is a low-latency asynchronous message-passing library that is used in scalable distributed or concurrent applications. The second one is a hybrid cloud test bed. The private cloud component of the hybrid cloud is composed of 64 cores (hyperthreaded) and a 2.2-GHz processor with 128 GB of memory. On top of this infrastructure, we deployed 50 VMs, with each VM an Ubuntu 12.04 with 1 core and 4 GB of memory. The public cloud is composed of 1,000 Amazon EC2 small instances (1 core with 1 ECU and 1.7 GB of memory).

Scaling of the “create agents” and “define attributes” phases is achieved through the division of the workload, with each process handling a group of agents. For example, in a simulation with 7 million commuters running on an infrastructure containing 1,000 VMs, creation of commuter agents was split among the VMs in such a way that each VM handled the creation of 7,000 agents.

On the “run simulation” phase, we experienced the execution of the simulation on a time-based simulation with 1-second intervals and tracking, checking, and updating of each agent's states. The scale method in this case delegates each VM to handle a group of agents. When the ZMQ push-pull method is used, one of the VMs acts as the head node that is in charge of distributing the tasks to all the worker VMs and controlling the timekeeping process of the simulation. The timekeeping process consists of sending a message to each worker to inform them of the current simulation time so that workers can start the simulation of events scheduled for such a given time.

However, we noticed that the time-based simulation has limited scalability. When executed in a private cloud of 50 VMs, it took 35,248 seconds to complete the 2 million commuters' agent-based simulation. This happened because there were dependencies in $t + 1$ with time t (i.e., simulation at time t needs to be completed before simulation of time $t + 1$ starts). Because of this issue, we replaced the time-based simulation with an event-based simulation.

In event-based simulation, the model handles the agents' interactions, such as boarding of commuters, unboarding of commuters, train arrivals at stations, and train departures from stations. On the back end, the workload is distributed via a similar method to other phases (each process handles a group of agents). With this new technique, the execution time of the simulation in the same private cloud was completed in 1,818 seconds for the same 2-million-commuter agent-based simulation, an improvement of 19 times over the original technique.

We further scaled the agent-based simulation by executing it on 1,000 VMs. In this case, the agent-based simulation completed in 434 seconds for simulation of 2 million commuters and 963 seconds for 7 million commuters. This demonstrated that the three phases of our approach are scalable and suitable for execution on elastic cloud platforms.

To summarize, we gave preference to the cloud-enabled WfMS over the ZMQ system because of the following reasons: (1) It enabled more efficient management of the highly distributed data required by the agent-based simulation workflow; (2) it better automated the workflow process for data analytics with multiobjective optimization of performance and budget; and (3) it enabled dynamic resource allocation for adaptive services with fault tolerance.

4.7 Related Work

Given the importance of workflow applications for the scientific community, many scientific workflow platforms were developed to explore scientific computational platforms such as grids. As cloud platforms became popular among the scientific community, WfMSs where enhanced to support them.

Pegasus [1] offers a set of tools for different aspects of execution and management of workflow applications and platforms. It implements application programming interfaces (APIs) for diverse programming languages, supports submission of workflows via web portals, and integrates with external tools. On its back end, it supports multiple cloud providers and scientific infrastructures.

Taverna [2] is another widely adopted workflow engine that can explore both grid and cloud platforms. Applications running on the platform can be deployed in many modes, including "server mode," by which it supports requests from many users to execute remote workflow applications.

The Cloudbus Workflow Engine incorporates a market-oriented utility computing model that supports grids, desktops, and clouds. It supports the concept of InterCloud for allocation and management of resources for execution of workflow applications [1].

Kim et al. [14] proposed a WfMS able to deploy workflows in hybrid infrastructures composed of TeraGrid nodes and Amazon EC2 resources. Our proposed system, on the other hand, can also leverage resources from private and public cloud providers.

Gogouvitis et al. [15] proposed a WfMS for deploying workflow applications on virtualized environments that is able to utilize resources from public clouds. However, it has no dynamic provisioning capabilities to speed application execution and to meet real-time application performance requirements as does our approach.

Fernandez et al. [16] proposed a cloud WfMS that applies a concept called chemical programming for the application scheduling. The system, however, does not offer dynamic resource provisioning capabilities and autonomic self-healing features.

CometCloud [17] is a more recent tool that implements an infrastructure for autonomic management of workflow applications on clouds.

4.8 Conclusions and Future Work

Clouds became a powerful platform for e-research as they enable scientists to have access to elastic, cost-effective, and virtually infinite computing power. Because clouds provide their users the view of infinite computing capacity, the real limitations on the scalability of the applications lie in the available budget for cloud usage and limitations in the applications themselves. Therefore, it is important that scientific application developers enable their applications to get the most from the cloud.

In this chapter, we discussed recent trends for execution of workflows in clouds. The architecture we presented is composed of a platform layer and an application layer. The platform layer enables operations such as dynamic resource provisioning, autonomic scheduling of applications, fault tolerance, security, and privacy in data access. The features enabled by this layer can be explored by virtually any application that can be described as scientific workflow.

In the application layer, we discussed a data analytics application enabling simulation of the public transport system of Singapore and the effect of abnormal events in the transport network. The application consists of an agent-based simulation of the public transport system of Singapore, and it allows evaluation of effects of incidents (such as train delays) in the flow of passengers in the country.

As future work, we plan to extend our platform to support a disaster decision support system (DDSS). The principles presented in this chapter will be further expanded so the DDSS will provide a dashboard for the strategic, tactical, and operational decisions arising during disaster mitigation. It will be integrated with a range of modeling and simulation tools to provide optimization models with up-to-date situational awareness and predictions to provide recommendations to authorities. This extension will support not only workflow applications but also other programming models suitable for clouds, such as MapReduce. Ideally, the platform will support not only applications that are entirely described as one of these models but also complex applications that are composed of diverse subcomponents that may be developed as different programming models.

References

1. Deelman, E., Singh, G., Su, M., et al. 2005. Pegasus: a framework for mapping complex scientific workflows onto distributed systems. *Scientific Computing* 13:219–237.
2. Oinn, T., Greenwood, M., Addis, M., et al. 2006. Taverna: lessons in creating a workflow environment for the life sciences. *Concurrency and Computation: Practice and Experience* 18:1067–1100.
3. Taylor, I., Shields, M., Wang, I., et al. 2007. The Triana Workflow Environment: Architecture and Applications. In *Workflows for E-Science*, ed. I. J. Taylor, E. Deelman, D. B. Gannon, et al., 320–339. London: Springer.
4. Pandey, S., Karunamoorthy, D., and Buyya, R. 2011. Workflow engine for clouds. In *Cloud Computing: Principles and Paradigms*, ed. R. Buyya, J. Broberg, and A. Goscinski, 321–344. New York: Wiley.
5. Kwok, Y., and Ahmad, I. 1999. Static scheduling algorithms for allocating directed task graphs to multiprocessors. *ACM Computing Surveys* 3:406–471.
6. Yu, J., Buyya, R., and Ramamohanarao, K. 2008. Workflow scheduling algorithms for grid computing. In *Metaheuristics for Scheduling in Distributed Computing Environments*, ed. F. Xhafa and A. Abraham, 173–214. Berlin: Springer.
7. Hirales-Carbajal, A., Tchernykh, A., Yahyapour, R., et al. 2012. Multiple workflow scheduling strategies with user run time estimates on a grid. *Journal of Grid Computing* 10:325–346.
8. Li, X., Calheiros, R., Lu, S., et al. 2012. Design and development of an adaptive workflow-enabled spatial-temporal analytics framework. In *Proceedings of the 2012 IEEE International Workshop on Scalable Computing for Big Data Analytics (SC-BDA 2012)*, 862–867. Piscataway, NJ: IEEE Computer Society.
9. Bryan, L. 2010. The social and psychological issues of high-density city space. In *Designing High-Density Cities for Social and Environmental Sustainability*, ed. E. Ng, 285–292. London: Earthscan.
10. Singapore Department of Statistics. 2013. Singapore in figures 2013. http://www.singstat.gov.sg/Publications/publications_and_papers/reference/sif2013.pdf.

11. Singapore Land Transport and Authority. 2013. Singapore land transport in brief 2013. http://www.lta.gov.sg/content/dam/ltaweb/corp/PublicationsResearch/files/FactsandFigures/Stats_in_Brief_2013.pdf.
12. Rahman, M., Li, X., and Veeravalli, B. 2012. Hybrid heuristic for scheduling data analytics workflow applications in hybrid cloud environments. In *Proceedings of the 2011 IEEE Symposium on Parallel and Distributed Processing Workshops and PhD Forum (IPDPSW'11)*, 966–974. Piscataway, NJ: IEEE Computer Society.
13. Hintjens, P. 2013. *ZeroMQ: Messaging for Many Applications.* Sebastopol, CA: O'Reilly.
14. Kim, H., el-Khamra, Y., Rodero, I., et al. 2011. Autonomic management of application workflows on hybrid computing infrastructure. *Scientific Computing* 19:75–89.
15. Gogouvitis, S., Konstanteli, K., Waldschmidt, S., et al. 2012. Workflow management for soft real-time interactive applications in virtualized environments. *Future Generation Computer Systems*, 28:193–209.
16. Fernandez, H., Tedeschi, C., and Priol, T. 2011. A chemistry-inspired workflow management system for scientific applications in clouds. In *Proceedings of the Seventh International Conference on e-Science (e-Science'11)*, 39–46. Piscataway, NJ: IEEE Computer Society.
17. Kim, H., el-Khamra, Y., Rodero, I., et al. 2011. Autonomic management of application workflows on hybrid computing infrastructure. *Scientific Programming* 19:75–89.