Enabling Re-executions of Parallel Scientific Workflows Using Runtime Provenance Data*

Flávio Costa[1], Daniel de Oliveira[1], Kary A.C.S. Ocaña[1], Eduardo Ogasawara[1,2], and Marta Mattoso[1]

[1] COPPE, Federal University of Rio de Janeiro, Rio de Janeiro, Brazil
[2] CEFET, Rio de Janeiro, Brazil
{flscosta,danielc,kary,ogasawara,marta}@cos.ufrj.br

Abstract. Capturing provenance data in scientific workflows is a key issue since it allows for reproducibility and evaluation of results. Many of these workflows generate around 100,000 tasks that execute in parallel in High Performance Computing environments, such as large clusters and clouds. SciCumulus is a workflow engine for parallel execution in clouds. Activity failure is almost inevitable in clouds where virtual machine failures are a reality rather than a possibility. We present SciMultaneous, a service architecture that manages re-executions of failed scientific workflow tasks using runtime provenance. Experimental results on clouds showed that SciMultaneous considerably increases the workflow completion and reduces the total execution time of the workflow (considering executions and re-executions) up to 11.5%, when compared to *ad-hoc* approaches.

1 Introduction

Scientific workflows became a *de facto* standard for modeling scientific experiments that are based on computer simulations [1]. Experiments in different domains of science demand high processing power. Scientific workflows can generate 100,000 or more parallel tasks in High Performance Computing (HPC) environments. Recently, clouds [2] emerge as an attractive alternative environment for scientific applications. By using clouds, scientists may avoid acquiring expensive HPC machines by instantiating a multi-processor environment composed by several virtual machines (VM). In addition, scientists benefit by paying only according to the effective use of HPC resources (pay *per* use model [2]). In this paper, we aim at improving reliability of workflow completion by managing workflow re-execution, in the presence of many types of failures. Previous experiments [3] show that failures occur very often in clouds, where the environment is constantly changing. It is a top priority to provide reliability so that scientists do not have to manage the potentially large numbers of failures. To illustrate the effective need of improving the reliability of parallel scientific workflows, we have explored, as case study, a workflow from the bioinformatics domain, SciPhy [3]. Typically, one execution of SciPhy on clouds,

* This work was partially sponsored by CNPq, FAPERJ and CAPES.

P. Groth and J. Frew (Eds.): IPAW 2012, LNCS 7525, pp. 229–232, 2012.

consuming 50 multi-fasta files, generates 1,250 parallel tasks and demands approximately 4.19 days using 16 virtual cores. Mainly due to performance fluctuations on the cloud environment, a typical execution of SciPhy presents from 2% to 9% of task failures, which may demand up to 10 hours of processing [3].

2 SciMultaneous

SciMultaneous enables re-executions of parallel tasks generated from workflow activities in case of failures in clouds. SciMultaneous benefits from SciCumulus, a cloud workflow engine [4], to obtain runtime provenance data [5] in parallel executions. Differently from the current mainstream, SciMultaneous can detect and manage failures during workflow execution, by using runtime provenance. SciCumulus distributes scientific workflow activities (or even entire scientific workflows) dispatched from a scientific workflow management system (*e.g.* Kepler [6]) into clouds such as Amazon EC2. There are several approaches focused on handling failures on scientific workflows. Ferreira *et al.* [7] introduce a representation of the workflow defining *a priori* the several optional paths to follow when a failure occurs. Crawl and Altintas [8] propose the Scientific Workflow Doctor, a component for Kepler to use prospective provenance data, guaranteeing fault tolerance. However, none of them focus on parallel executions neither provide runtime provenance data.

SciMultaneous architecture follows the Software as a Service (SaaS) model [2]. Three main services are part of SciMultaneous: Task Mapper (TM), VM Configurator (VMC) and Task Executor (TE). The TM searches for any task failure indicative by submitting queries to the workflow engine provenance repository. TM analyzes the generated runtime provenance data and decides which heuristic to use. With the heuristic chosen, TM informs VMC that there is a demand for re-executions. VMC analyzes if there is any idle VM to be used or if VMC has to instantiate new VMs for re-execution. With the environment set, VMC invokes TE that effectively re-executes tasks in each VM created by VMC. TE is also responsible for capturing and storing provenance data (related to the re-executions) in the provenance repository.

SciMultaneous follows two heuristics. In both of them we assume that scientists can access more than one cloud environment. In this case, one of the clouds is chosen as the reliable one with more processing power, consequently presenting higher financial costs. The main idea is to start re-executing failed tasks using the other cloud, less reliable, with lower financial costs. The reliable cloud is left to execute critical tasks or tasks that also failed in their re-executions. The first heuristic (named H1) tries to anticipate failures, by redundantly executing tasks that are considered critical (*i.e.* long-term tasks). In case of failure, a replica is available for substituting the failed task. The second heuristic (named H2) focuses on continuous task monitoring. In case of task failure, SciMultaneous immediately re-schedules it to another VM in the same cloud or even to another cloud. For example, in H2, SciMultaneous follows a hierarchical strategy: firstly it re-schedules the task to the same cloud and in the same VM, assuming that some intermittent problem may have occurred. If the failure persists, another VM is instantiated in the same cloud environment to re-schedule the specific task. Then, if the task is considered critical it is re-scheduled to a powerful VM. Finally, if the failure still persists, SciMultaneous re-schedules the task to a different cloud, using more processing power *per* VM.

3 Experimental Results and Conclusions

SciMultaneous coupled to SciCumulus was evaluated in the Amazon EC2. We executed SciPhy workflow as the case study. Our experiment uses as input, a dataset of 250 multi-fasta files (each file with 10 sequences) of protein sequences extracted from RefSeq database release 48, as detailed in Ocaña *et al.* [3]. This data set generates a total of 6,250 parallel tasks, which approximately 1.96% (thus 123 tasks) presented some kind of failure. We executed SciPhy workflow varying the number of virtual cores used in each execution. We compared the two proposed heuristics of SciMultaneous. For one of the scenarios, both H1 and H2 presented a total execution time lower than *ad-hoc* approaches in all cases. For example, using 32 cores, H1 executed in 9.90 days, H2 in 9.63 days and *ad-hoc* in 10.39 days. In the case of H1, 9.2% of the long-term tasks (*i.e.* ModelGenerator and RAxML programs [3]) were executed redundantly (creating one original task and one redundant). Then, 8% of the original long-term tasks have failed and they were automatically replaced by the redundant task, as soon as, provenance was produced, and queried by SciMultaneous. SciMultaneous reached a performance improvement up to 11.5% when using H1 compared to an *ad-hoc* re-execution approach. In addition, in larger experiments, this performance gain can be higher than the one reached in this experiment. These performance improvements led to a reduction of up to US$ 373.24 in financial costs when comparing H1 execution with *ad-hoc* re-execution approach. However, the highest SciMultaneous gain is reliability improvement by workflow completion. Another advantage of using SciMultaneous is that as it is a service architecture it can be coupled to other approaches for managing workflow execution, as long as, provenance data is provided at runtime. Querying provenance at runtime is fundamental since it allows for online execution adjustments (re-executions) that otherwise would be impossible to be pre-programmed.

References

[1] Mattoso, M., Werner, C., Travassos, G.H., Braganholo, V., Murta, L., Ogasawara, E., Oliveira, D., da Cruz, S.M.S., Martinho, W.: Towards Supporting the Life Cycle of Large-scale Scientific Experiments. International Journal of Business Process Integration and Management 5(1), 79–92 (2010)

[2] Vaquero, L.M., Rodero-Merino, L., Caceres, J., Lindner, M.: A break in the clouds: towards a cloud definition. SIGCOMM Comput. Commun. Rev. 39(1), 50–55 (2009)

[3] Ocaña, K.A.C.S., de Oliveira, D., Ogasawara, E., Dávila, A.M.R., Lima, A.A.B., Mattoso, M.: SciPhy: A Cloud-Based Workflow for Phylogenetic Analysis of Drug Targets in Protozoan Genomes. In: Norberto de Souza, O., Telles, G.P., Palakal, M. (eds.) BSB 2011. LNCS, vol. 6832, pp. 66–70. Springer, Heidelberg (2011)

[4] Oliveira, D., Ogasawara, E., Ocana, K., Baiao, F., Mattoso, M.: An Adaptive Parallel Execution Strategy for Cloud-based Scientific Workflows. Concurrency and Computation: Practice and Experience (2011) (online)

[5] Freire, J., Koop, D., Santos, E., Silva, C.T.: Provenance for Computational Tasks: A Survey. Computing in Science and Engineering 10(3), 11–21 (2008)

[6] Altintas, I., Berkley, C., Jaeger, E., Jones, M., Ludascher, B., Mock, S.: Kepler: an extensible system for design and execution of scientific workflows. In: Scientific and Statistical Database Management, Greece, pp. 423–424 (2004)
[7] Ferreira, J.E., Wu, Q., Malkowski, S., Pu, C.: Towards Flexible Event-Handling in Workflows Through Data States. In: Proc. of the 2010 IEEE 6th World Congress on Services, Miami, FL, pp. 344–351 (2010)
[8] Crawl, D., Altintas, I.: A Provenance-Based Fault Tolerance Mechanism for Scientific Workflows. In: Freire, J., Koop, D., Moreau, L. (eds.) IPAW 2008. LNCS, vol. 5272, pp. 152–159. Springer, Heidelberg (2008)