Future Generation Computer Systems 46 (2015) 114–126

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Data-centric iteration in dynamic workflows

Jonas Dias[a], Gabriel Guerra[a], Fernando Rochinha[a], Alvaro L.G.A. Coutinho[a], Patrick Valduriez[b], Marta Mattoso[a,*]

[a] *COPPE - Federal University of Rio de Janeiro, Rio de Janeiro, Brazil*
[b] *INRIA and LIRMM, Montpellier, France*

HIGHLIGHTS

- Algebraic operators support data-centric iteration in dynamic workflows.
- Runtime data lineage, a concept inspired by provenance enables dynamic loops.
- Two algorithms support runtime adaptation of the workflow based on user input.
- Real-life experiment for Uncertainty Quantification in the Oil & Gas domain.
- A novel iterative workflow for Uncertainty Quantification is steered by users.

ARTICLE INFO

Article history:
Received 15 February 2014
Received in revised form
4 October 2014
Accepted 10 October 2014
Available online 22 October 2014

Keywords:
Scientific workflows
Dynamic workflows
Iteration
Steering

ABSTRACT

Dynamic workflows are scientific workflows to support computational science simulations, typically using dynamic processes based on runtime scientific data analyses. They require the ability of adapting the workflow, at runtime, based on user input and dynamic steering. Supporting data-centric iteration is an important step towards dynamic workflows because user interaction with workflows is iterative. However, current support for iteration in scientific workflows is static and does not allow for changing data at runtime. In this paper, we propose a solution based on algebraic operators and a dynamic execution model to enable workflow adaptation based on user input and dynamic steering. We introduce the concept of iteration lineage that makes provenance data management consistent with dynamic iterative workflow changes. Lineage enables scientists to interact with workflow data and configuration at runtime through an API that triggers steering. We evaluate our approach using a novel and real large-scale workflow for uncertainty quantification on a 640-core cluster. The results show impressive execution time savings from 2.5 to 24 days, compared to non-iterative workflow execution. We verify that the maximum overhead introduced by our iterative model is less than 5% of execution time. Also, our proposed steering algorithms are very efficient and run in less than 1 millisecond, in the worst-case scenario.

1. Introduction

Large-scale scientific experiments tend to explore big datasets, searching for confirmation (or not) of given hypotheses. The process typically involves multiple computing steps supported by programs and chained with scripts. Scientific Workflow Management Systems (SWfMS) [1] have been successful in helping scientists modeling this process as a workflow of activities that operate on the datasets. SWfMS implement a workflow specification model and its execution model. The specification model is supported by a workflow programming language that allows for workflow composition. The execution model is supported by an execution engine, which also provides for result analysis, based on provenance [2]. SWfMS should support specific needs of large-scale workflows. For instance, to reduce execution time, data intensive workflows need to run in parallel in high performance computing (HPC) environments. They may also need specialized features such as iteration.

Iteration is a basic concept that is supported by most programming languages. Because it is intuitive, it has been used extensively in the development of algorithms and computational models. Scientists usually program their iterative methods directly within

* Correspondence to: COPPE, Federal University of Rio de Janeiro, P.O. Box 68511, 21941-972 Rio de Janeiro, RJ, Brazil. Tel.: +55 21 2562 8694; fax: +55 21 2562 8080.
E-mail addresses: jonasdias@cos.ufrj.br (J. Dias), gguerra@mecsol.ufrj.br (G. Guerra), faro@mecanica.coppe.ufrj.br (F. Rochinha), alvaro@nacad.ufrj.br (A.L.G.A. Coutinho), Patrick.Valduriez@inria.fr (P. Valduriez), marta@cos.ufrj.br (M. Mattoso).

http://dx.doi.org/10.1016/j.future.2014.10.021
0167-739X/© 2014 Elsevier B.V. All rights reserved.

compiled applications or scripts. However, when modeling an iterative application as a scientific workflow, the workflow language has to support iterative constructs. Current support for iteration in scientific workflows is designed for simpler loop behaviors [3–7]. They typically are control-flow-based and rely on three iteration types [8]: (1) counting loops without dependencies, (2) counting loops with dependencies and (3) conditional loops. However, these iteration types are static and scientists cannot change the loop specification at runtime.

Workflow configuration is likely to change during its life cycle, as the scientist may need to refine a numerical model, try other algorithms or explore different slices of the parameter space to find better results. This exploratory nature of large-scale scientific experimentation makes workflows iterative. The workflow specification needs to be evaluated and explored several times by scientists until a solution is found. Thus, scientists evaluate the results they get, and as adjustments are needed to achieve better results, they must resubmit the workflow execution after doing the necessary changes. Waiting for these results may take a very long time. Scientists usually do not wait until the end of the workflow execution. They try to analyze partial results and, if they can infer that the execution is not on the right track, they abort it, refine data parameters on the specification, and restart. Such manual support of iteration makes it hard to manage the evolution of scientific data analysis. For example, scientists may find it hard to analyze the converged error at runtime or discover which parameter combination they are exploring has produced the best outcome.

To tackle the exploratory nature of science and the dynamic process involved in scientific analysis, dynamic workflows have been identified as an open challenge [9]. Dynamic workflows are scientific workflows that are subject to rapid reuse and exploration accompanied by continuous adaptation and improvement [9]. They need the ability of adapting the workflow based on external events such as human interaction and runtime steering. A typical user interaction would be to adjust data such as filters or domain-specific parameters based on runtime data and partial results. Similar to an iterative optimization approach, scientists may keep the workflow running in a loop, doing adjustments and refinements searching for the best solution as the workflow execution progresses. This runtime adaptation based on user input is more efficient than pre-programming all possible data combinations in a loop, which may not be viable in a scientific workflow due to inherent complexity and unexpected data transformation behavior. In fact, the expression *human-in-the-loop* has gained a lot of interest in dataflow analytics. According to [10] "there remain many patterns that humans can easily detect but computer algorithms have a hard time finding".

In a previous work [11], we showed the benefits of dynamic workflows in reduced order models for heat conduction and genetic algorithms. We then started working with bigger problems such as Uncertainty Quantification (UQ) [12]. Adaptations of the workflow at runtime in [12] were still hand-made through provenance database queries, but the performance improvements obtained, motivated us to develop specific constructs to support dynamic workflows. We identified scenarios that can make good use of dynamic workflows, such as: bioinformatics workflows involving multiple sequence alignment [13] that need to refine the similarity predefined *e-value* for a given input dataset running the same workflow several times; and UQ workflows, which can be used in several scientific domains. One may foresee a similar scenario in other application areas, e.g. in clustering algorithms, where scientists need to refine the quantity of data clusters, or in genetic algorithms, where scientists refine parameters related to crossover and mutation rates. All these examples require runtime data analysis and data adjustments to refine initial configurations. Otherwise, they may increase the experiment data size and complexity by running all possible alternatives to pick the best after the whole execution. This manual exploratory approach cannot be done even at a terabyte scale.

Resubmitting the workflow execution after manually changing its configuration, as done by scientists today, is labor-intensive and error-prone. Because of the lack of provenance data along the trials, they may lose track of what has already been explored and how the workflow has evolved. To improve this iterative experimental process, the user should be able to analyze partial results during execution and to interfere dynamically accordingly, in the next steps of the workflow. There are obvious advantages if the whole iterative process is modeled as a dynamic workflow. However, current iterative approaches do not allow for workflow configuration data to be adapted and fine-tuned during the execution. They typically have an iterative specification model that is submitted for execution as a static plan. Scientists wait until execution ends to analyze results and provenance data and only then decide if it is necessary to interfere in the workflow data configuration to be further executed.

In this paper, we characterize a fourth iteration type – dynamic loops – as a particular type of conditional loops where the condition of the loop and the data being processed in the loop may be adapted during the execution. As done today, programming the dynamic iteration with big data management tools [14] may not be viable due to legacy scientific code complexity and the requirement for sophisticated provenance querying support. Dynamic loops have a data-centric iterative specification because they need to naturally respond to data-driven events. These events may be data inserts and updates at runtime, which change the condition of the loop in its specification. In other words, conditional loops need to be dynamic to adapt the workflow based on the action of the users when they are steering the execution. To support the execution of workflows with dynamic loops, we propose a dynamic execution model guided by runtime data lineage, a concept inspired in provenance that keeps track of the dataflow changes in the loop. Adaptations made by users when they steer the workflow become part of the dataflow by two algorithms that access the data lineage structure and triggers the data-driven events. Thus, dynamic loops are iterations subjected to adaptations based on user input and we support it by means of a data-centric, provenance oriented approach.

Our dynamic loops support follows the algebraic approach for data-centric scientific workflows [15]. Nevertheless, the original algebra, defined in [15], only supports counting loops (1), which is also known as parameter sweeps. It does not support the other types of iterations such as (2) or (3) neither or adaptations based on user input in the algebraic expression specification at runtime. In order to support dynamic loops, we have defined new algebraic operators and the corresponding formalism to keep the algebraic properties. Since dynamic loops are the most general iteration type among the four, when we support dynamic loops in the algebra, we are now currently supporting all the four types of loops.

We introduce the dynamic loops support without increasing workflow complexity by following the intuitive concepts of iterative data-centric programming languages. For instance, FAD [16], a functional database programming language designed for easing optimization and parallel execution, provides a *whiledo* second-order construct for iterating over first-order function calls. Our approach is also tightly-coupled with a relational data provenance model that encompasses W3C PROV [17] and stores additional information such as execution performance and domain data altogether. In order to store domain data, our approach requires the workflow to be instrumented so that the execution engine can capture data and store it in the provenance database during runtime. Consequently, users can query real-time provenance [12,13], which is essential to support decision making during the dynamic steering. In [12], for example, a UQ workflow is improved using