Resilient Workflows for Computational Mechanics Platforms

Toàn Nguyên, Laurentiu Trifan and Jean-Antoine Désidéri
INRIA, 655, Av. de l’Europe, Montbonnot, FR-38334 Saint-Ismier (France)
Toan.Nguyen@inrialpes.fr, trifan@inrialpes.fr, Jean-Antoine.Desideri@sophia.inria.fr

Abstract. Workflow management systems have recently been the focus of much interest and many research and deployment for scientific applications worldwide [26, 27]. Their ability to abstract the applications by wrapping application codes have also stressed the usefulness of such systems for multidiscipline applications [23, 24]. When complex applications need to provide seamless interfaces hiding the technicalities of the computing infrastructures, their high-level modeling, monitoring and execution functionalities help giving production teams seamless and effective facilities [25, 31, 33]. Software integration infrastructures based on programming paradigms such as Python, Mathlab and Scilab have also provided evidence of the usefulness of such approaches for the tight coupling of multidiscipline application codes [22, 24]. Also high-performance computing based on multi-core multi-cluster infrastructures open new opportunities for more accurate, more extensive and effective robust multi-discipline simulations for the decades to come [28]. This supports the goal of full flight dynamics simulation for 3D aircraft models within the next decade, opening the way to virtual flight-tests and certification of aircraft in the future [23, 24, 29].

1. Introduction
During the last decade, the e-science sector has shown a growing interest in workflows [1, 4, 14, 16, 17]. It has extensively used a dataflow approach for the processing of large numeric data sets [5, 6].

Large-scale multiphysics applications, e.g., aircraft flight dynamics simulation that takes into account aerodynamics and structural loads, are considered today fundamental by aircraft manufacturers in order to gain leading position on highly competitive innovative markets world-wide. The same goes for mobile phones manufacturers.

Not only are organizational problems put forward, because of the risk-sharing partnerships that are often implemented, but technological and scientific challenges are addressed because verification and validation of numeric models are necessary in order for virtual prototypes to allow drastic reduction in time-to-market design [2, 8].

Multiphysics approaches are considered here in order to better combine and synchronize the intricate relationships between the various disciplines that contribute to the integration of complex new products, e.g., acoustics, electromagnetics and fluid dynamics in aircraft design [7, 5].

High-performance computing also opens new perspectives for complex products definition, design and tuning to market needs [9]. However, high-performance computing platforms also raise new challenges to computer scientists in order to fulfill the design bureaus requirements, e.g., the management of petascale volumes of data, the management of distributed teams collaborating on large and complex virtual prototypes, using various remote computerized tools and databases, etc [1, 11].
This paper focuses on the design of distributed workflows systems that are used to define, deploy, configure, execute and monitor complex simulation and optimization applications. It emphasizes the need for resilient workflows. It does not consider hardware and system-level fault-tolerance. In a way similar to [7], it focuses on a generic approach to handle application-level failures, namely the implementation of resilient workflows. In that sense, it copes with the byzantine application processes as described elsewhere in the literature [6].

Section 2 deals with resilient workflows, including fault-tolerance, resiliency, and checkpointing issues: two approaches are described, namely bracketing checkpoints and asymmetric checkpoints. Section 3 gives some implementation details in connection with OMD2, an ongoing project on distributed multidiscipline optimization platforms. Section 4 is a conclusion.

2. Resilient workflows

Also multiphysics design includes several disciplines and various tools that pertain to each particular expertise involved. This includes CAD tools, meshers, solvers, analyzers and optimizers, which in turn are used to modify the meshes in iterative and incrementally optimized design processes.

Past research in distributed systems tells us that distributed recovery algorithms are implemented using partial order among checkpoints. This guarantees that if updates can be serialized, then pause and restart mechanisms are safe, i.e., they are robust in case of failure. Further, backup and restart mechanisms restore the data and processes in a previously saved state, when correctly scheduled.

This guarantees that the executing application codes can be paused and resumed after dynamic parameter updates by the users. It also guarantees that the executing applications can be restored after system or application failures. The whole workflow systems, including the running applications, are then qualified here resilient workflows. This departs from fault-tolerance, which is restricted here only to hardware, system and communication failures. In this case, it concerns fault-tolerant workflows.

In case of erratic application behavior, it is also clear that the users can invoke these services to abort them as well as pause and later update the execution parameters in order to restart the simulation processes.

The infrastructure required to implement these services can benefit from the appropriate functionalities developed in existing grid middleware, e.g., Globus, UNICORE, gLite [12, 13]. Also, high-performance visualization tools like parallel display walls can be interfaced with the workflow systems, e.g., CUDA programming tools on GPU-clusters, in order to compare various design alternatives in real-time.

2.1. Fault-tolerant workflows

Because distributed systems are potentially faced with unexpected hardware and software failures, adequate mechanisms have been devised to handle recovery of running systems, software and applications.

Checkpoint and restart mechanisms are usually implemented using the local ordering of the running processes. This implies that the safe execution of all the running processes is not guaranteed, i.e., there is no way a randomly aborted distributed process can be restored in a consistent state and resume correctly. The solution would be to use a global synchronization and clock, which is practically unfeasible and very constraining.

On the application side, this is used by transaction systems, e.g., airline reservation and banking systems, because compensation in case of failure is fundamental.

However, design, simulation and optimization applications bear specificities that require less stringent mechanisms than transaction systems. Design is a stepwise process that does not require global synchronizing, except when, and only when, dynamic update propagation is required. This can be executed during limited time periods and does not impair the usual stepwise approach.

The same goes for simulation and optimization, where long duration processes are executed, which can invoke many composite components. These components may be invoked by sub-workflows. They bear a similar nature: global synchronization is not required, only synchronization for composite sub-
workflows with their running components. Even so, asynchronous executions using pipelining of intermediate results can be devised.

For example, the wing optimization workflow depicted in Figure 2 can use the following checkpoints:

- C0 and C1 to save the initial parameters and the optimization results
- C2 and C3 to save the individual solutions (alternatively, C’3 can save the results)
- C4 and C5 to save the individual solutions geometry variants (the forms)
- C6 and C7 to save the various flight regimes results (C’7 if the database is saved)
- C8 and C9 to save the results of the various solvers executions (and C’9 to save the database)

They are called in the following sections *bracketing checkpoints* (Section 2.2). An example is illustrated by figure 1.

![Figure 1. Fault-tolerant workflow checkpoints.](image)

Should some random hardware and software failure occur, it is easy to see that each optimized solution (called here “Individual”) computed so far is saved, corresponding to every geometry (called here “Form”), every flight “Regime” and every “Solver” computation is saved. This minimizes the process of resuming the optimization workflow when aborted due to some external cause. This is an implementation of fault-tolerance.

For example, the checkpoint C6 supports the resuming of the composite and parallel sub-workflow “Regimes”. The latter can be restarted entirely or partially if some of its component “Solvers” resumed correctly. Their results are checkpointed by C9 and alternatively C’9 if they are stored in the database DB_Perf (Figure 1).

This implements a coarse grained approach, which can be sufficient in simulation and optimization scenarios. In case of unexpected hardware and software failure when one of these components is executing, the CPU demanding or optimization processes can be restarted, saving the whole workflow which has not to be restarted entirely. This is not adequate for resiliency, however, which requires a fine-grained approach.
2.2. **Bracketing checkpoints**

Resiliency differs from fault-tolerance because it is related to the ability of the applications to survive to unpredictable behavior.

In contrast with fault-tolerant workflows which can survive hardware and system failures, using ad-hoc bracketing by checkpointing mechanisms, resilient workflows need to be aware of the application structure in order to implement automated survival procedures. These procedures can use the bracketing of sub-workflows also, but in addition, they need specific logging of the workflow component operations and parameters to restore incrementally previous states and resume their operations (Figure 1).

Thus, checkpoints must be inserted in the workflow composite hierarchy. They can bracket critical parts of the hierarchy, e.g., the most demanding CPU components (unsteady flow calculations over a 3D wing model, for example) and the following optimization components which might be less CPU demanding, but are fundamental to the application because they allow for the comparison of various optimized solutions. This scheme is called *bracketing checkpoints*.

Further, parallel branches of the workflow that failed need later to be re-synchronized with the branches that resumed correctly. This requires that the results of the successful branches are stored for further processing with the failed branches results, if they resume correctly later. Otherwise, these results are discarded if the failed branches never succeed. Because there is no awareness of the successful branches on the possible failures of parallel branches in the workflow, time-out and synchronization signals must be exchanged on a regular basis to notify each branch of the current state of the others: alive or not responding.

2.3. **Asymmetric checkpoints**

Because bracketing checkpoints might also incur a large overhead when used in composite workflows, their occurrence must be fine-tuned to each particular application workflow.

For example, the checkpoints C0, C2, C4 and C6 which store the state and data relevant to the component workflows “Optimize”, “Individuals”, “Forms” and “Regimes” in figure 2 are redundant with the checkpoints C1, C3, C5 and C7.

Indeed, should a failure occur in a component workflow, e.g. “Regime”, the preceding checkpoint C6 will be used to restore the application in a safe state. It is therefore redundant to insert the checkpoint C8, except if the “Configuration” and “Read_DB” tasks are critical (Figure 2).

An appropriate placement mechanism must therefore be implemented in order to optimize the recovery procedure and minimize the checkpoints overhead for running applications.

An asymmetric scheme has been designed to handle this problem. Opening checkpoints, e.g., checkpoints inserted prior to critical tasks (e.g., C2, C4, C6) are paired with closing checkpoints of component workflows that are not immediate children of the parent component workflow. This avoids redundant checkpoints.

This scheme is called *asymmetric checkpoints*.

It is particularly useful when multiple instances of component workflows are defined, e.g., in the example above, the “Regime” task is defined as a multiple instance composite component. This means that there may exist several instance of the “Regime” component workflow executing at the same time in parallel for a specific Form instance of workflow (and for a specific “Individual” solution). In the example above, the tasks “Forms”, Individuals are also multiple instance composite components.

If it is desired to save explicitly every possible instance of executing component, the asymmetric checkpoints scheme needs to be used. In this case, should a failure occur in the midst of a component execution, say the Forms component, all the “Regime” instances which have resumed prior to the failure are checkpointed (hence saved) by the C9 checkpoint. Potentially interesting solutions have therefore been saved this way. But this is implemented at the price of a systematic checkpointing (and logging) of the “Regime” partial results.
If this option is not retained, the asymmetric scheme will reduce drastically the overhead incurred by checkpointing. The price to pay is then to run the whole “Regime” component again which initialization parameters are checkpointed at C6.

Figure 2. Resilient workflow: iterative recovery.

2.4. Rules
We assume in the following that “join” operations are those that require several input datastreams to execute. Similarly, we assume that “fork” operations are those that output their results on several datastreams. They model generic tasks that execute application codes. We also consider “remote” and “local” operations. We do not distinguish between parallel and sequential implementations of the operations.

Further, we consider in the following that the “specified” operations are those operations or workflow tasks that are marked by the application designers or the users as requiring a specific treatment in the following heuristic procedure.

The specific characterization of the marked tasks is implemented by raising an exception that invokes a specific treatment that departs from the standard heuristic rules. An example of such exception is the backup of a particular intermediate result after processing by a large CPU intensive task or the back up of the result of a task producing petascale volumes of data. Workflow management systems usually provide powerful exception handling functionalities that can be used to implement this kind of “specified” operations management, e.g., YAWL [14].

This enables the designers and users to adapt the execution of the workflow depending on their specific knowledge and expertise.

This is a prerequisite for the effective implementation of the workflows based on previous runs and casestudies involving petaflops and petabytes of data. Some automated learning procedure could eventually be designed to support this kind of feedback.

The recovery procedure implements a heuristic approach based on the following rules:
- R1: no output backup for specified join operations
- R2: only one output backup for fork operations
- R3: no intermediate result backup for user-specified sequences of operations
- R4: no backup for user-specified local operations
- R5: systematic backup for remote inputs
In order to improve performance, these rules can be tuned by the application designers to fit their specific requirements. This includes specified operations that are deemed CPU intensive and data transfer intensive.

They can also be altered or ignored by load-balancing strategies if appropriately authorized by the designers and users, and if global and local policies make this mandatory, e.g., preemptive local strategies.

Based on these rules, the example illustrates the asymmetric checkpoints on an unfolded workflow (Figure 3). Two remote execution sites are considered: Site a (white colored tasks) and Site b (red colored tasks).

When it is not modified and tuned by the designers, the result of the asymmetric checkpoints procedure results in seven unnecessary checkpoints which are deleted, thus leaving five remaining checkpoints: S0, S2, S4, S9 and L0.

3. Implementation

The approach implemented here uses the YAWL workflow management system. It is one of the rare workflow systems to be defined with a sound formal semantics [18]. It is designed to combine grid and distributed computing through a middleware with scientific computing using a mathematical problem solving environment. It thus provides an e-Science infrastructure as a high-performance platform for large-scale distributed data and CPU intensive applications. Validation of the platform is through industrial testcases concerning car aerodynamics and engine valves and pipes optimization.

The approach wraps the existing applications codes, e.g., optimizers and solvers, with Web services. This is extended to the checkpointing and resiliency procedures which are defined by standard YAWL workflow tasks. They are inserted appropriately in the workflow definition, in compliance with the specific scheme adopted, i.e., bracketing or asymmetric schemes (Section 2).

The invocations of the various tasks in the workflow by other tasks are specific YAWL invocations when the tasks are local. They are invoked by Web Services if the tasks are remote. Data and task parameters and descriptions are uniformly exchanged as XML schemas.

A platform supporting these features is developed for the OMD2 project [21] by a consortium that includes twelve academic and industry partners, including a major international car manufacturer leading the project. OMD2 is an acronym for Distributed Multi-Discipline Optimization.

The goal is to develop a high-performance distributed environment for simulation and multidiscipline optimization in complex design projects.

The distributed platform uses the ProActive middleware for resource allocation and scheduling of tasks [19]. The tasks include software codes that collaborate and include Scilab scripts [20], optimization software developed by the project partners as well as commercial CAD tools.

YAWL is used for defining incrementally composite workflows, as well as the sharing and reuse of the various software that form the applications. These can interact with the users through sophisticated exception handling mechanisms and interact with each other using Web services (Figure 4). This is also used for implementing the resilience and fault-tolerance features described in the previous sections (Section 2).
Because the workflow engine supports dynamic interactions with remote software through Web Services, it communicates with the ProActive engine using specific services for distributed resource allocation and scheduling of the applications. Similarly, interaction with the Scilab numeric computation software is based on script invocations for the execution of the application codes. Access to local data and codes are available for both Scilab and YAWL users.

They benefit therefore from the features provided by the three platform components:
- the Scilab scientific software package for numeric computations
- the ProActive middleware for distributed task allocation, execution and monitoring
- the YAWL workflow system for application definition, evolution and sharing

4. Experiments
The distributed optimization platform is run against industrial testcases provided by a world-class auto manufacturer. They include 2D and 3D air duct optimization for air-conditioning systems (Figure 6), input pipes optimization for diesel engines and trucks aerodynamics optimization. These testcases require CPU times in the order of minutes, tens, hundreds and thousands of minutes respectively, on conventional non-distributed hardware running commercial and inhouse software.

The workflow system is used to alleviate the burden on engineers in deploying and reusing the software components and to introduce application-level resilience in the platform, as explained in the previous sections. Local components, e.g., Scilab and Matlab scripts as well as the OpenFOAM
software, are invoked using shell scripts, while remote components are invoked using Web Services. Both shell script and Web Service invocations are natively supported in YAWL. Considering the application structure and data volumes exchanged, Web services overhead is considered here negligible considering the large CPU requirements of the testcases, ranging from tens to thousands of minutes.

![Air duct optimization testcase](screenshot Courtesy SIREHNA)

Figure 6. Air duct optimization testcase (screenshot Courtesy SIREHNA).

Current experiments involve the dynamic connection of the Matlab and OpenFOAM software to the YAWL platform (Figure 5). The YAWL workflow system is used to define the optimization processes, include the testcases and control their execution: this includes reading input data (CATIA v5 CAD files), invocation of external software, passing control to the various application components (i.e., Matlab and Scilab scripts) as well as solver calculations and display of the results (call to the OpenFOAM and Star-CCM+ software).

Further experiments will include the deployment of remote CAD and software packages, as well as the deployment of the optimizers and solvers on large parallel clusters for high-performance computing. The YAWL workflow will invoke them, control their execution, handle possible failures with the appropriate recovery procedures (see Section 3). It will support ultimately the interactions of entire distributed workflow components for complex applications. This will allow the remote execution of component workflows in a seamless, controlled, fault-tolerant and collaborative approach.

5. Conclusion
Distributed infrastructures exhibit potential hazards to the executing processes, due to unexpected hardware and software failures. This is endangered by the use of distributed high-performance environments that include very large clusters of multi-processors nodes. This requires fault-tolerant workflows systems.

Further, erratic application behavior requires dynamic user interventions, in order to adapt execution parameters for the executing codes and to run dynamic application re-configurations. This requires resilient workflow systems.

This implies applications roll-back to appropriate checkpoints, and the implementation of survivability procedures, including fault-tolerance to external failures and resiliency to unexpected application behavior.

Asymmetric checkpoints are presented here to effectively support the resiliency procedure. In order to minimize the overhead incurred by the checkpointing and logging of the workflow operations, a heuristic is presented that uses tunable rules to adapt the resiliency procedure to the application requirements and to comply with the computing infrastructures.

Several open issues are currently under investigation:

- the impact of the rule ordering on the resilience performance in case of application restart after unexpected crashes
• the impact of user defined rules inserted in the default rule set
• the impact of application characteristics (CPU and data intensive) on the expected resilience
  overhead/application performance ratio

Although it seems arguable to use business-oriented workflow management systems for scientific
applications, it is strongly believed that they exhibit advantages over computing-oriented ones, e.g.,
VisTrails, Pegasus, Swift [32], etc. Among these advantages, our tescases have proved:
  • YAWL’s seamless interface for complex, highly-parallel, composite and hierarchically
defined applications definition, deployment and sharing
  • its strong support for component and application evolution
  • its organizational management features, which are usually lacking in other systems, including
large trans-organization and cross-domains collaborative projects involving multiple users

Last but not least, YAWL is one of the very few existing workflow management systems to have a
fully proven operational and formal semantics [18]. This is a fundamental and so far unsurpassed
advantage over other more data and scientific-computing workflow systems. It allows for complex
applications to be safely defined and run in distributed high-performance computing environments.

References
in the Cloud”. Proc. 3rd Intl. Conf. on Advanced Engineering Computing and Applications in
China-EU Workshop on Multi-Physics and RTD Collaboration in Aeronautics. Harbin (China)
April 2009.
http://www.swinflow.org/tcsc/wmsce.htm
Conference on Cooperative Design, Visualization and Engineering. CDVE2009. Luxemburg,
EU Workshop on Multi-Physics and RTD Collaboration in Aeronautics. Harbin (China) April
2009.
November 2005.
Simulation and Optimization”. Proc. 2nd China-EU Workshop on Multi-Physics and RTD
Changsha(China). http://www.swinflow.org/confs/WaGe08/WaGe08.htm
Acknowledgments
This work is the result of the authors’ contributions to:

- the OPALE project at INRIA: http://www-opale.inrialpes.fr and http://www-sop.inria.fr/opale/
- the AEROCHINA2 project of the EC, supported by the “Transport including Aeronautics” program of the FP7, contract n°ACS7-GA-2008-213599: http://www.cimne.com/aerochina2
- the OMD2 project (“Optimisation Multi-Disciplines Distribuée”) of the French National Research Agency (ANR), program COSINUS (“Conception et Simulation”), grant n°ANR-08-COSI-007.

The authors wish to thank all their partners for their support and Prof. Jacques Périaux from CIMNE (Barcelona, Spain) and Univerity of Jyväskylä (Finland) for numerous fruitful discussions.