A Resilience Approach to High-Performance Workflows

Toan Nguyen, Laurentiu Trifan

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

A Resilience Approach to High-Performance Workflows

Toàn Nguyên – Laurentiu Trifan

N° 7421

October 2010

THÈME 1

Rapport de recherche

ISSN 0249-6399

INSTITUT NATIONAL
DE RECHERCHE
EN INFORMATIQUE
ET EN AUTOMATIQUE

INRIA

centre de recherche GRENOBLE - RHÔNE-ALPES

A Resilience Approach to High-Performance Workflows

Toàn Nguyên [1] – Laurentiu Trifan [2]

Thème 1 – Computational Models and Simulation
Projet Opale

Rapport de recherche n° 7421 – October 2010 - 17 pages

Abstract: This report presents an approach to design, implement and deploy resilient distributed workflows. It supports the smooth integration of existing software for simulation applications, e.g. Matlab, Scilab, Python, OpenFOAM, Paraview and application programs. The contribution of the report is a new feature which supports resilience, i.e., application-level fault-tolerance and exception-handling. Connections with exascale computing requirements are also made. An overview of a prototype implementation based on the YAWL workflow management system is given.

Keywords: workflows; fault-tolerance; resilience; simulation; distributed systems.

[1] INRIA Grenoble Rhône-Alpes – Toan.Nguyen@inrialpes.fr
[2] INRIA Grenoble Rhône-Alpes – trifan@inrialpes.fr

Centre de recherche INRIA Grenoble - Rhône-Alpes
655, Avenue de l'Europe, Montbonnot, 38334 Saint-Ismier (France)
Téléphone : +33 4 76 61 52 00 – Télécopie : +33 4 76 61 52 52

Une approche de la résilience pour les workflows à haute performance

Résumé: Ce rapport présente une approche pour concevoir, réaliser et déployer des workflows distribués et résilients. Elle permet l'intégration des logiciels de simulation numérique, par exemple Matlab, Scilab, Python, OpenFoam, ParaView et des codes d'applications. La contribution de ce rapport est une nouvelle fonctionnalité qui permet la résilience, c'est-à-dire la tolérance aux pannes des applications et le traitement d'exceptions. Le lien est également fait avec les besoins des futures applications exascale. On décrit un prototype basé sur le système de worflow YAWL.

Mots clés: workflows ; tolérance aux pannes ; résilience ; simulation ; systèmes distribués.

1 Introduction

This paper explores the design, implementation and use of fault-tolerant and resilient simulation platforms. It is based on distributed workflow systems and distributed computing resources [3]. Aiming petascale computing environments, this infrastructure includes heterogeneous distributed hardware and software components. Further, the application codes interact in a timely, secure and effective manner. Additionally, because the coupling of remote hardware and software components are prone to run-time errors, sophisticated mechanisms are necessary to handle unexpected failures at the infrastructure, system and application levels [20]. This is also critical for the coupled software that contribute to exascale frameworks [19]. Consequently, specific approaches, methods and software tools are required to handle unexpected faults and errors fir large-scale distributed applications.

As mentioned in the Exascale IESP report [19], current checkpoint/restart and rollback recovery techniques will not fulfill the exascale computing requirements, due in part to their large overhead: “Because there is no compromise for resilience, the challenges it presents need to be addressed now for solutions to be ready when Exascale systems arrive” (Section 4.4.1 Resilience in [19]).

More precisely (Section 4.5 Summary of X-stack priorities in [19]): “Resilience is an issue for many efforts. Historically, resilience has not required applications to do anything but checkpoint/restart. Presently, there is a general agreement that the entire software stack, including user and library code, will need to explicitly address resilience beyond the checkpoint/restart approach. We believe this is a uniquely exascale concern and of critical importance.”

Among the targets emphasized by the report [19] are (Section 4.4.1 Resilience):

- fault confinement and local recovery
- avoid global coordination towards more local recovery
- reducing checkpoint size
- language support and paradigm for resilience
- dynamic error handling by applications
- situational awareness
- fault oblivious applications

This paper addresses three of these issues:

- promoting situational awareness using high-level error handlers defined by the users inside the application workflows
- significantly reducing checkpoint size used for application recovery, using appropriate heuristics
- dynamic error handling by executing ad-hoc workflow components that can be dynamically added to the workflow original definitions

Section II is an overview of related work. Section III is a general description of a sample application, infrastructure, systems and application software. Section IV addresses resilience and asymmetric checkpointing. Section V gives an overview of the implementation, extending the YAWL workflow management system for distributed resilient computations [4]. Section VI is a conclusion.

2 Related Work

Simulation is nowadays a prerequisite for product design and scientific breakthroughs in most application areas, ranging from pharmacy, meteo, biology to climate modeling, that all require extensive simulations and testing [6, 8]. They often need large-scale experiments, including long-lasting runs in the orders of weeks, tested against petabytes volumes of data and will soon run on exascale supercomputers [10, 11, 19].

In such environments, various teams usually collaborate on several projects or part of projects. Computerized tools are shared and tightly or loosely coupled. Some codes may be remotely located and non-movable. This requires distributed code and data management facilities. Unfortunately, this is also prone to unexpected errors and breakdowns, e.g., communications, hardware and systems failures.

Data replication and redundant computations have been proposed to prevent from random hardware and communication failures, as well as deadline-dependent scheduling [9].

Hardware and system level fault-tolerance in specific programming environments are also proposed, e.g. Charm++ [5]. Also, middleware and distributed computing systems usually support mechanisms to handle fault-tolerance. They call upon data provenance [12], data replication, redundant code execution, task replication and job migration, e.g., ProActive [17], VGrADS [15].

However, erratic application behaviors are seldom addressed, due to programming errors, bad application specifications, poor accuracy and performance. They also needs to be taken into account and handled. This implies evolutions of the simulation processes in the event of unexpected data values or unexpected control flows. Little has been done in this area. The primary concern of the application designers and users has been indeed on efficiency and performance. Therefore, application erratic behavior is usually handled by re-designing and re-programming pieces of code and adjusting parameter values and bounds. This usually requires the simulations to be stopped and rebuilt [15]. This approach is inadequate when simulation runs last several days and weeks.

Departing from these solutions, a dynamic approach is presented in the following sections. It supports the evolution of the application behavior using the introduction of new exception handling rules at run-time by the users, based on the observed (and possibly unexpected) data values. The running workflows do not need to be aborted, as new rules can be added at run-time without stopping the executing workflows [13]. At worst, they need to be paused.

This allows on-the-fly management of unexpected events. It allows also a continuous evolution of the applications, supporting their adaptation to the occurrence of unforeseen situations. As new situations arise and new data values appear, new rules can be added to the workflows that will permanently be taken into account in the future. These evolutions are dynamically plugged-in to the workflows, without the need to stop the running applications [13]. The overall application logics is therefore unchanged. This guarantees a continuous adaptation to new situations without the need to redesign the existing workflows, thus promoting situational awareness.

Further, because exception-handling codes are themselves defined by new specific workflows plug-ins, the user interface to the applications remains unchanged [14].

Also, checkpoint/restart procedures are addressed here by reducing significantly the number of necessary checkpoints, using a new scheme called “asymmetric checkpoints”. This addresses the critical concern for the checkpoint sizes in large-scale and exascale applications [19] (Section IV. D.)

3 Application Testcase

3.1 Example testcase

An overview of a running tescase is presented here. It deals with the optimization of a car air-conditioning duct. The goal is to optimize the air flow inside the duct, maximizing the throughput and minimizing the air pressure and air speed discrepancies inside the duct. This example is provided by a car manufacturer and involves industry partners, e.g., software vendors, as well as optimization research teams (Figure 1).

The testcase is a dual faceted 2D and 3D example. Each facet involves different software for CAD modeling, e.g. CATIA and STAR-CCM+, numeric computations, e.g., Matlab and Scilab, and flow computations, e.g., Open FOAM and visualization, e.g., ParaView (Figure 1).

The testcase is deployed using the YAWL workflow management system [4]. The goal is to distribute the testcase on various partners' locations where the different software are running (Figure 2). In order to support this distributed computing approach, an open source middleware is used, namely: ProActive [17].

A first prototype was achieved using extensively the virtualization technologies (Figure 3), in particular Oracle VM VirtualBox®, formerly called Sun VirtualBox® [7]. This allowed experiments connecting virtual guest computers running heterogeneous software. These include Linux Fedora Core 12, Windows® 7 and Windows® XP on a range of local workstations and laptops (Figure 2).

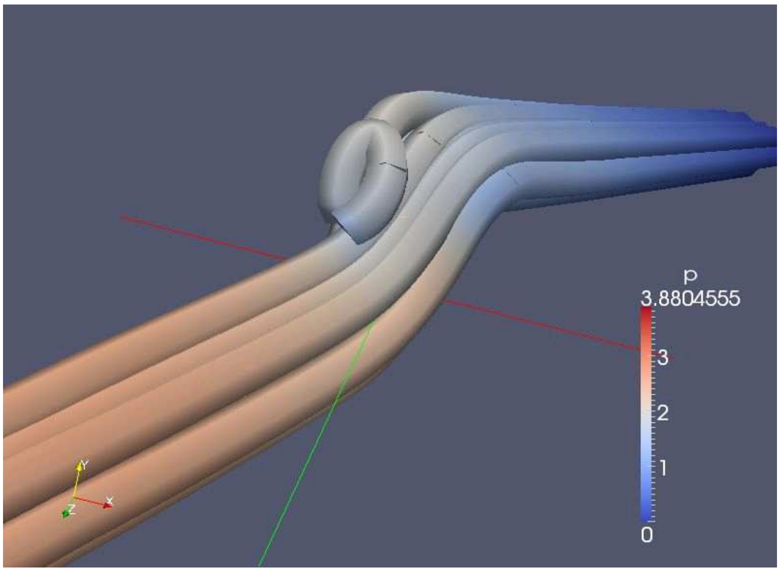

Figure 1. Pressure flow in an air-conditioning duct (ParaView screenshot).

3.2 Application workflow

In order to provide a simple and easy-to-use interface to the computing software, the YAWL workflow management system is used (Figure 2). It supports high-level graphic specifications for application design, deployment, execution and monitoring. It also supports the modeling of business organizations and interactions among heterogeneous software components. Indeed, the example testcase described above involves several codes written in Matlab, OpenFOAM and displayed using ParaView. The 3D testcase facet involves CAD files generated using CATIA and STAR-CCM+, flow calculations using OpenFOAM, Python scripts and visualization with ParaView. Future testcases will also require the use of the Scilab toolbox [16].

This work is performed for the OMD2 project, an acronym for Optimisation Multi-Disciplinaire Distribuée, i.e., Distributed Multi-Discipline Optimization, supported by the French National Research Agency ANR.

Because proprietary software are used, as well as open-source and in-house research codes, a secured network of connected computers is made available to the users, based on the ProActive middleware (Figure 5).

This network is deployed on the various partners' locations throughout France. Web servers accessed through the ssh protocol are used for the proprietary software running on dedicated servers, e.g., CATIA v5 and STAR-CCM+.

A powerful feature of the YAWL workflow system is that composite workflows can be defined hierarchically [4]. They can invoke external software, i.e., pieces of code written in whatever language is used by the users. They are called by custom YAWL services or local shell scripts.

Web Services can also be invoked. Although custom services need Java classes to be implemented, all these features are natively supported in YAWL.

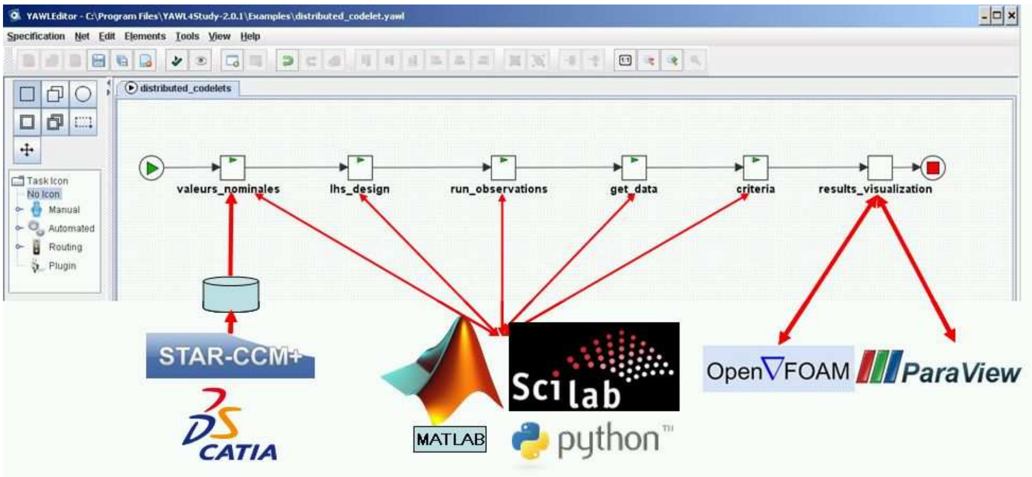

Figure 2. The YAWL workflow for the 2D testcase.

YAWL thus provides an abstraction layer that helps users design complex applications that may involve a large number of distributed components (Figure 3). Further, the workflow specifications allow alternative execution paths which may be chosen automatically or manually, depending on data values, as well as parallel branches, conditional branching and loops. Also, multiple instance tasks can execute in parallel for different data values. Combined with the run-time addition of code using the corresponding dynamic selection procedures, as well as new exception handling procedures (see Section IV), a very powerful environment is provided to the users [4].

4 Resilience

4.1 Fault tolerance

The fault-tolerance mechanism provided by the underlying middleware copes with job and communication failures. Job failures or time-outs are handled by reassignment of computing resources and re-execution and of the jobs. Communication failures are handled by re-sending appropriate messages. Thus, hardware breakdowns are handled by re-assigning running jobs to other resources, which imply possible data movements to the corresponding resources. This is standard for most middleware [17].

4.2 Resilience

Resilience is commonly defined as “the ability to bounce back from tragedy” and as “resourcefulness” [18]. It is defined here as the ability for the applications to handle correctly unexpected run-time situations, possibly – but not necessarily – with the help of the users.

Usually, hardware, communication and software failures are handled using hard-coded fault-tolerance software [15]. This is the case for communication software and for middleware that take into account possible computer and network breakdowns at run-time. These mechanisms use for example data and packet replication and duplicate code execution to cope with these situations [5].

However, when unexpected situations occur at run-time, which are due to unexpected data values and application erratic behavior, very few options are offered to the users: ignore them or abort the execution, analyze the errors and later modify and restart the applications.

Optimized approaches can be implemented in such cases trying to reduce the amount of computations to be re-run, or anticipating potential discrepancies by multiplying some critical instances of the same computations. This latter approach can rely on statistical estimations of failures. Another approach for anticipation is to prevent total loss of computations by duplicating the calculations that are running on presumably failing nodes [9].

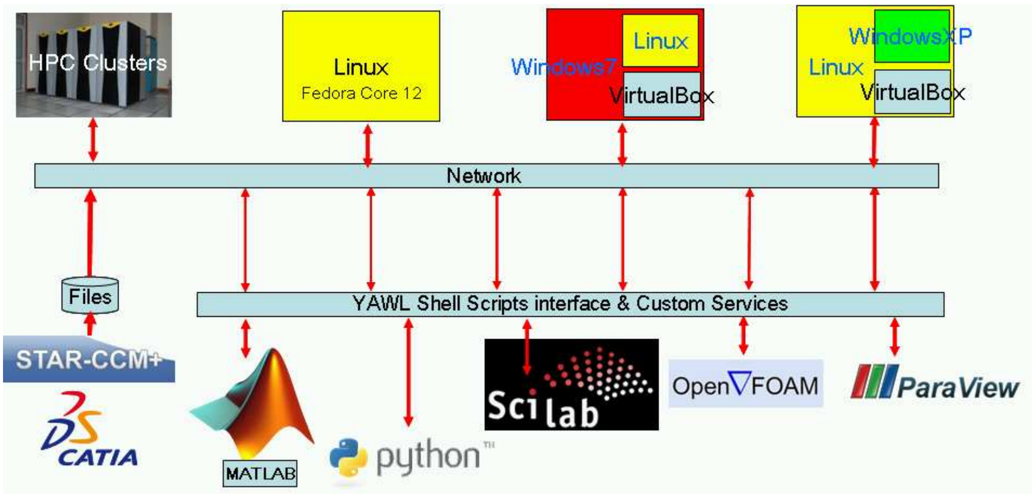

Figure 3. The virtualized distributed infrastructure.

While these approaches deal with hardware and system failures, they do not cope with application failures. These can originate from:

- Incorrect or incomplete specifications.
- Incorrect or hazardous programming.
- Incorrect anticipation of data behavior, e.g., out-of-bounds data values.
- Incorrect constraint definitions, e.g., approximate boundary conditions.

To cope with this aspect of failures, we introduce an application-level fault management that we call resilience. It provides the ability for the applications to survive, i.e., to restart, in spite of their erroneous prevailing state. In such cases, new handling codes can be introduced dynamically by the users in the form of specific new component workflows.

This requires a roll-back to a consistent state that is defined by the users at critical checkpoints.

In order to do this efficiently, a mechanism is implemented to reduce the number of necessary checkpoints. It is based on user-defined rules. Indeed, the application designers and users are the only ones to have the expertise required to define appropriate corrective actions and characterize the critical checkpoints. No automatic mechanisms can be substituted for them, as is the case in hardware and system failures. It is generally not necessary to introduce checkpoints systematically, but only at specific locations of the application processes, e.g., only before parallel branches of the applications. We call this approach *asymmetric checkpoints*. This is described in Section D, below.

4.3 Exception handling

The alternative used proposed here to cope with unexpected situation is based on the dynamic selection and exception handling mechanism featured by YAWL [13].

It provides the users with the ability to add at run-time new rules governing the application behavior and new pieces of code that will take care of the new situations.

For example, it allows for the runtime selection of alternative workflows, called worklets, based on the current (and possibly unexpected) data values. The application can therefore evolve over time without being stopped. It can also cope later with the new situations without being altered. This refinement process is therefore lasting over time and the obsolescence of the original workflows reduced.
The new worklets are defined and inserted in the original application workflow using the standard specification approach used by YAWL (Figure 2).
Because it is important that monitoring long-running applications be closely controlled by the users, this dynamic selection and exception handling mechanism also requires a user-defined probing mechanism that provides with the ability to suspend, evolve and restart the code dynamically.
For example, if the output pressure of an air-conditioning pipe is clearly off limits during a simulation run, the user must be able to suspend it as soon as he is aware of that situation. He can then take corrective actions, e.g., suspending the simulation, modifying some parameters or value ranges and restarting the process immediately. These actions can be recorded as new execution rules, stored as additional process description and invoked automatically in the future.
These features are used to implement the applications erratic behavior manager. This one is invoked by the users to restart the applications at the closest checkpoints after corrective actions have been manually performed, if necessary, e.g., modifying boundary conditions for some parameters. Because they have been defined by the users at critical locations in the workflows, the checkpoints can be later chosen automatically among the available asymmetric checkpoints available that are closest to the failure location in the workflow.

4.4 Asymmetric checkpoints

Asymmetric checkpoints are defined by the users at critical execution locations in the application workflows. They are used to avoid the systematic insertion of checkpoints at all potential failure points. They are user-defined at specific locations, depending only on the application logic. Clearly, the applications designers and users are the only ones that have the domain expertise necessary to insert appropriately these checkpoints. In contrast with middleware fault-tolerance which can re-submit jobs and resend data packets, no automatic procedure can be implemented here. It is therefore based on a dynamically evolving set of heuristic rules.
This approach significantly reduces the number of necessary checkpoints to better concentrate on only those that have an impact on the applications runs [3].
For example (Figure 4):

- The checkpoints can be chosen by the users among those that follow long-running components and large data transfers.
- Alternatively, those that precede sequences of small components executions.

The base rule set on which the asymmetric checkpoints are characterized is the following:

- 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.

This rule set can be evolved by the user dynamically, at any time during the application lifetime, depending on the specific application requirements. This uses the native rule mechanism in YAWL [13].

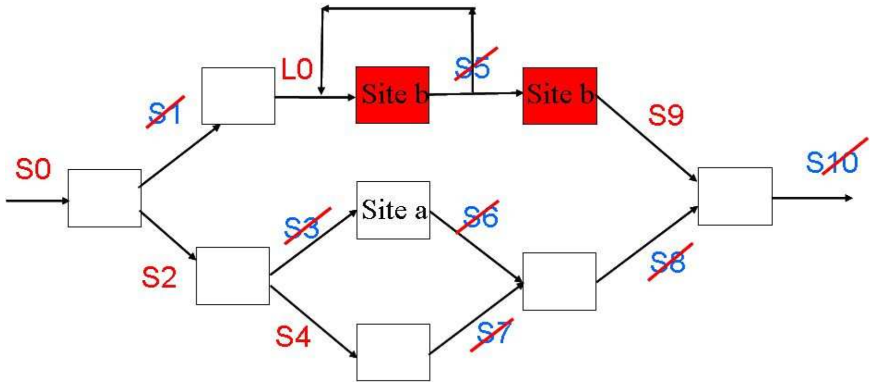

Figure 4. Asymmetric checkpoints example.

5 Implementation

5.1 The YAWL workflow management system

Workflows systems are the support for many e-Science applications [1, 6, 8]. Among the most popular systems are Taverna, Kepler, Pegasus, Bonita and many others [11, 15]. They complement scientific software environments like Dakota, Scilab and Matlab in their ability to provide complex application factories that can be shared, reused and evolved. Further, they support the incremental composition of hierarchic composite applications. Providing a control flow approach, they also complement the usual dataflow approach used in programming toolboxes. Another bonus is that they provide seamless user interfaces, masking technicalities of distributed, programming and administrative layers, thus allowing the users and experts to concentrate on their areas of interest.

The OPALE project at INRIA (http://www-opale.inrialpes.fr) is investigating the use of the YAWL workflow management system for distributed multidiscipline optimization [4]. The goal is to develop a resilient workflow system for large-scale simulation applications. It is based on extensions to the YAWL system to add resilience and remote computing facilities for deployment on high-performance distributed infrastructures. This includes large-PC clusters connected to broadband networks. It also includes interfaces with the Scilab scientific computing toolbox [16] and the ProActive middleware [17]. A prototype implementation is underway for the OMD2 project ("*Optimisation Mutlidiscipline Distribuée*") supported by the French *Agence Nationale de la Recherche* (ANR).

Provided as an open-source software, YAWL is implemented in Java. It is based on an Apache server using Tomcat and Apache's Derby relational database system for persistence. YAWL is developed by the University of Eindhoven (NL) and the University of Brisbane (Australia). It runs on Linux, Windows and MasOS 32 and 64-bits platforms. It allows complex workflows to be defined and supports high-level constructs (e.g., XOR- and OR-splits and joins, synchronized merge, loops, conditional control flow based on application variables values, composite tasks, parallel execution of multiple instances of tasks, etc) through high-level user interfaces (Figure 5). It supports over forty built-in datatypes and user-defined complex data types for application-specific requirements.

Formally, YAWL is based on a sound and proved operational semantics using extended "workflow patterns" of the Workflow Management Coalition [21] and implemented by colored Petri nets. This allows deep syntactic and semantic verifications on the workflow processes defined by the users.

Designed as a open platform, YAWL supports interactions with external and existing software and application codes written in any programming languages, through shell scripts invocations, as well as distributed computing through Web Services (Figure 6).

Besides a native Web Services interface, YAWL supports custom services invocations through "codelets", as well as rules, powerful exception handling facilities, and monitoring of the workflow executions [13].

Further, it supports dynamic evolution of the applications by extensions to the existing workflows through "worklets", i.e., on-line inclusion of new workflow components during execution [14].

It supports also automatic and step-by-step execution of the workflows, as well as persistence of (possibly partial) executions of the workflows for later resuming, using its internal database system. It also features extensive event logging for later analysis, simulation, configuration and tuning of the application workflows.

Additionnally, YAWL supports extensive organizations modeling features allowing complex collaborative projects and user teams to be defined with the appropriate access rights and granting capabilities to the various members on the projects, workflows and processing tools by the project administrators.

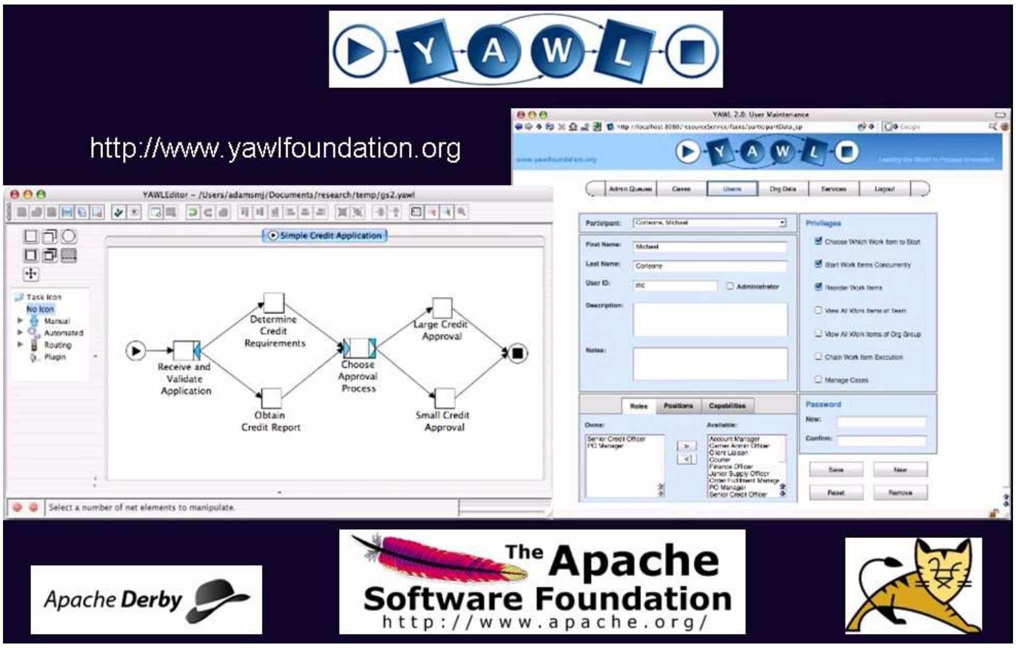

Figure 5. YAWL interfaces.

5.2 Resilience

Resilience is the ability for applications to handle unexpected behavior, e.g., erratic computations, abnormal result values, etc. It is inherent to the applications logic and programming. It is therefore different from systems or hardware errors and failures. The usual fault-tolerance mechanisms are therefore inappropriate here. They only cope with late symptoms, at best.

New mechanisms are therefore required to handle logic discrepancies in the applications, most of which are only discovered incrementally during the applications life-time, whatever projected exhaustive details are included at the application design time.

It is therefore important to provide the users with powerful monitoring features and to complement them with dynamic tools to evolve the applications specifications and behavior according to the future erratic behavior that will be observed during the application life-time.

This is supported here using the YAWL workflow system so-called "dynamic selection and exception handling mechanism" [4]. It supports:

- Application update using dynamically added rules specifying new worklets to be executed, based on data values and constraints.
- The persistence of these new rules to allow applications to handle correctly the future occurrences of the new cases.
- The dynamic extension of these sets of rules.
- The definition of the new worklets to be executed, using the native framework provided by the YAWL specification editor: the new worklets are new component workflows attached to the global composite application workflows [13].
- Worklets can invoke external programs written in any programming language through shell scripts, custom service invocations and Web Services [14].

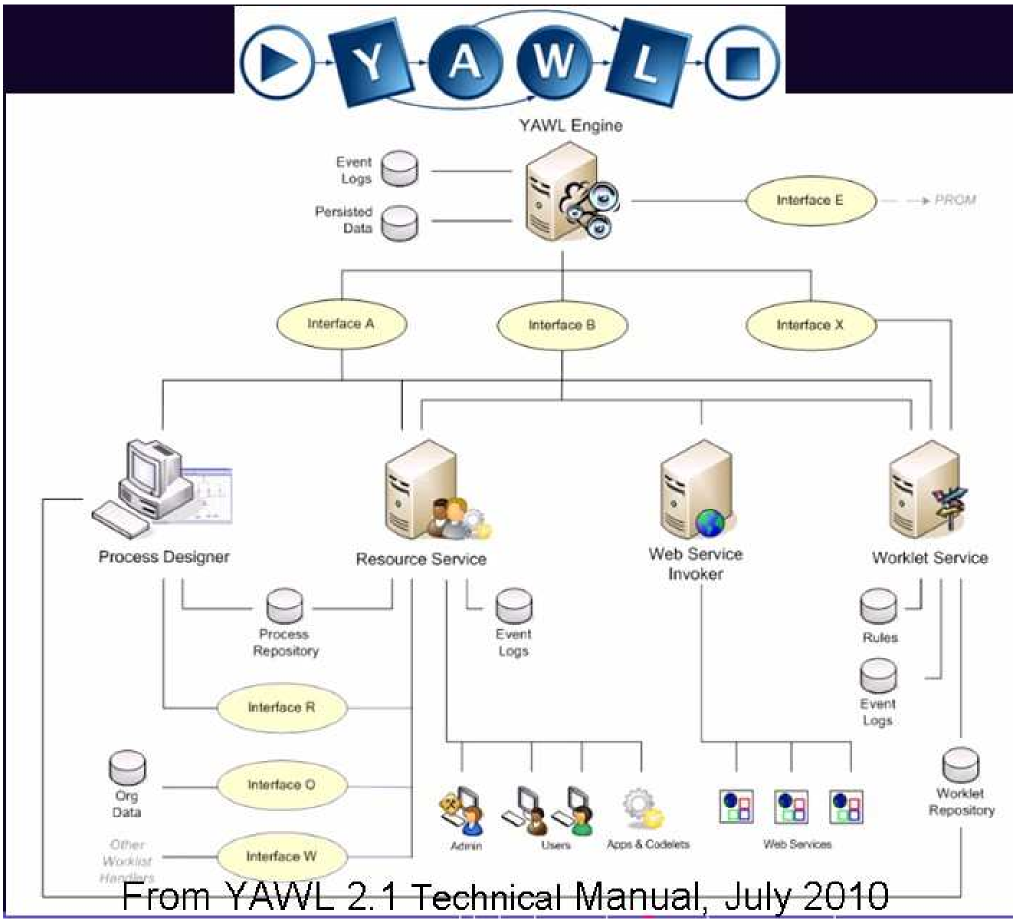

Figure 6. YAWL archiitecture.

5.3 Distributed workflows

The distributed workflows rely on the interface between the YAWL engine and the ProActive middleware (Figure 7). Users provide a specification of the simulation applications using the YAWL Editor. It supports a high-level abstract description of the simulation processes (Figure 2).

These processes are decomposed into components which can be other workflows or basic workitems. The basic workitems invoke executable tasks, e.g., shell scripts or so-called "custom

services". These custom services are specific execution units that call user-defined YAWL services. They support interactions with external and remote codes. In this particular platform, the remote external services are invoked through the ProActive middleware interface (Figure 8).
This interface delegates the distributed execution of the remote tasks to the ProActive middleware [17]. The middleware is in charge of the distributed resources allocation to the individual jobs, their scheduling, and the coordinated execution and result gathering of the individual tasks composing the jobs. The scheduler default policy is "best-effort". However, users can implement their own policy, if desired. The middleware also takes in charge the fault-tolerance related to hardware, communications and system failures. The resilience, i.e., the application-level fault-tolerance is handled using the rules described in the previous sections.

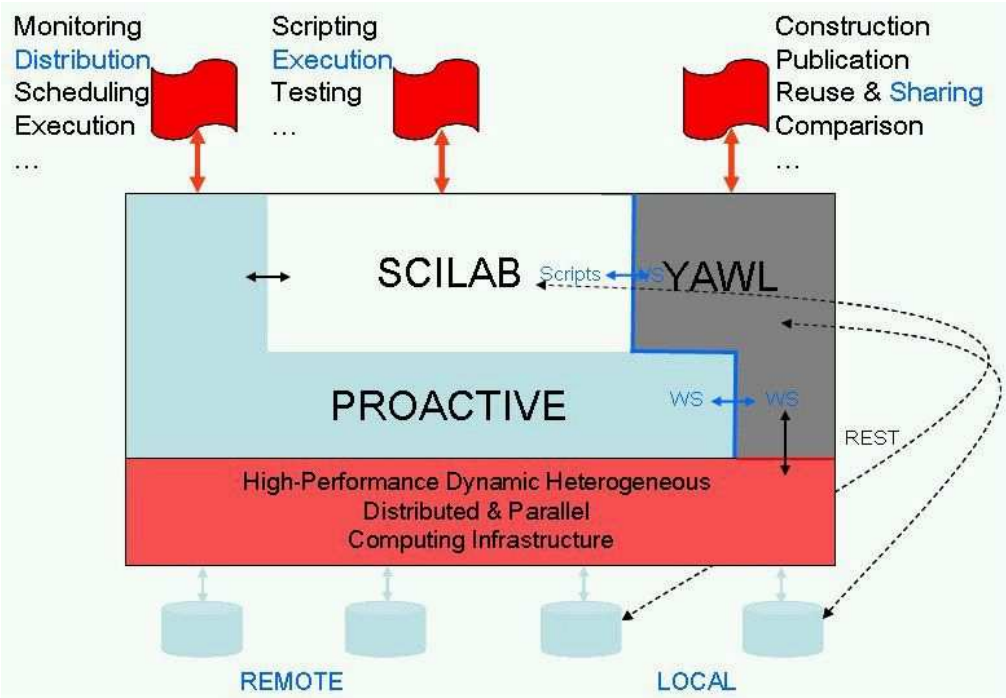

Figure 7. OMD2 distributed simulation platform.

The remote executions invoke the middleware functionalities through ProActive's Java API. The various modules invoked are the ProActive Scheduler, the Jobs definition module and the Tasks which compose the jobs. The jobs are allocated to the distributed computing resources based upon the scheduler policy. The tasks are dispatched based on the job scheduling and resource allocation. They invoke Java executables, possibly wrapping code written in other programming languages, e.g., Matlab, Scilab, Python, or calling other software, e.g., CATIA v5, STAR-CCM+, ParaView, etc.
Optionally, the workflow can invoke local tasks using shell scripts and remote tasks using Web Services. These options are standard in YAWL [4]. Calling the ProActive middleware is however necessary to run tasks on large multi-core clusters. ProActive is here in charge of the scheduling and resource allocation in these highly parallel environments, which YAWL does not support natively.

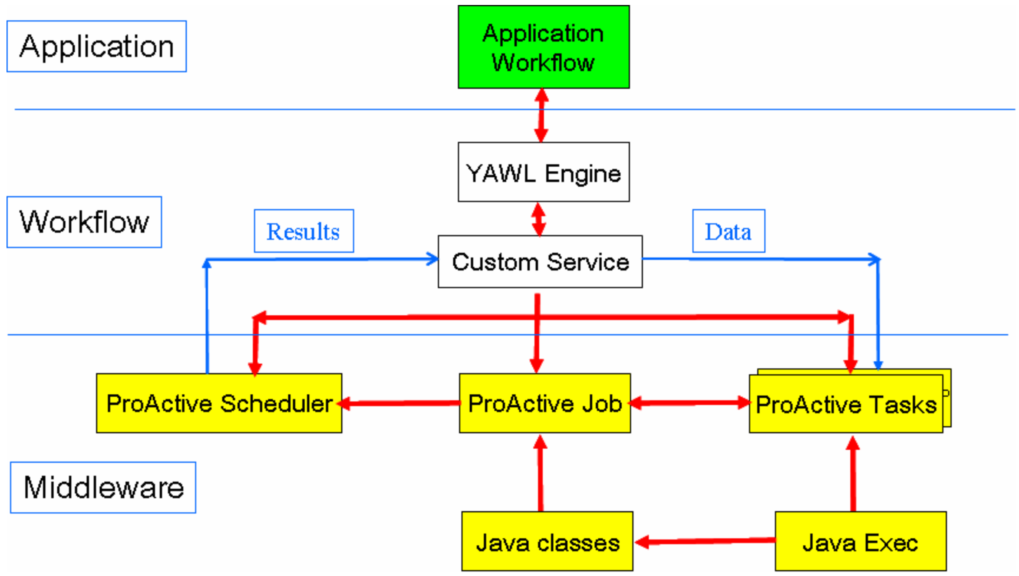

Figure 8. YAWL workflow / ProActive middleware interface.

6 Conclusion

This report presents an experiment for designing, implementing and deploying distributed simulation platforms. It uses a network of high-performance computers connected by a middleware layer. Users interact dynamically with the applications using a distributed workflow management system, i.e., YAWL. It allows them to define, deploy, evolve and control the applications.

A significant bonus of this approach is that besides fault-tolerance provided by the middleware, which handles communication, hardware and system failures, the users can define and handle dynamically, i.e., at run-time, the application failures at the workflow specification level. This adds *resilience* to the applications.

This report also addresses four major concerns that impact exascale application frameworks, as pointed out by the Exascale Software Project [19]:

- reduced checkpoint size
- language support and paradigm for resilience
- dynamic error handling
- situational awereness

A new abstraction layer is introduced to anwser the need for *situational awareness* [19], in order to cope with the application errors at run-time. Indeed, these errors do not necessarily result from programming and design errors. They may also result from unforeseen situations, data values and boundary conditions that could not be envisaged at first. This is often the case for simulations due to the experimental nature of the applications, e.g., discovering the behavior of the system being simulated, like unusual flight dynamics: characterization of the stall behavior of an aircraft for various load and balance profiles at the limits of its flight envelope [2].

To answer the requirement for *reduced checkpoint size* in [19], the approach presented here supports resilience using an asymmetric checkpoints mechanism. This feature allows for efficient handling mechanisms to restart only those parts of an application that are characterized by the users as critical for overcoming erratic and unexpected behaviors.

Further, this approach can evolve dynamically, i.e., when applications are running. This uses the native dynamic selection and exception handling mechanism in the YAWL workflow system

[4]. Should unexpected situations occur, it allows for new rules and new exception handlers to be plugged-in at run-time. This answer the need for *dynamic error handling* at run-time.
Additionally, the requirement for *language support and paradigm for resilience* [19] is also addressed , using the error handlers plugged into the application workflows by new component workflows thus providing a uniform, homogeneous and high-level user interface.
New testcases are currently being set-up that involve large-scale simulations (50x10**6 vertices CFD meshes, 1000 CPU hours), e.g., car aerodynamics, running on networks of multi-core clusters.

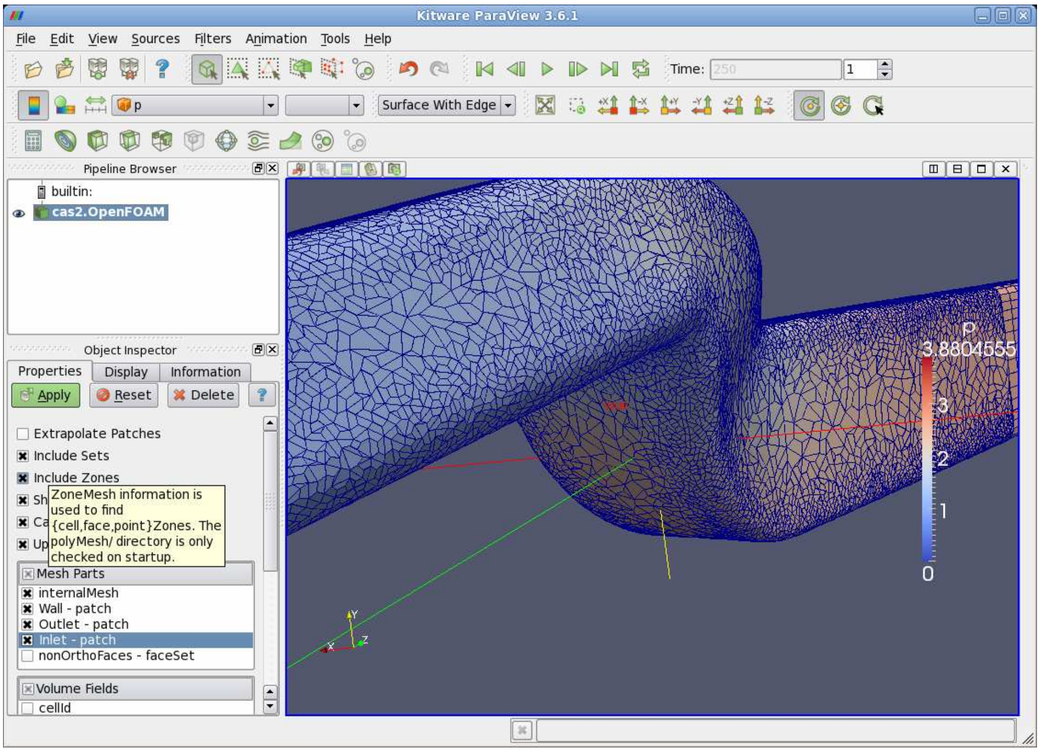

Figure 9. The 3D testcase visualization (ParaView screenshot).

7 Acknowledgements

This work is performed for the OPALE project at INRIA (http://www-opale.inrialpes.fr).
It is supported by the French *Agence Nationale de la Recherche* (ANR), grant ANR-08-COSI-007, program COSINUS (Conception et Simulation), for the OMD2 project (*Optimisation Multi Disciplinaire Distribuée*). The authors wish to thank all their partners for providing the testcases and for many fruitful discussions.

8 References

[1] Y.Simmhan, R. Barga, C. van Ingen, E. Lazowska and A. Szalay "Building the Trident Scientific Workflow Workbench for Data Management in the Cloud". *Proceedings of the 3rd Intl. Conf. on Advanced Engineering Computing and Applications in Science*. ADVCOMP'2009. Sliema (Malta). October 2009. pp 132-138.

[2] A. Abbas, "High Computing Power: A radical Change in Aircraft Design Process", *Proceedings of the 2nd China-EU Workshop on Multi-Physics and RTD Collaboration in Aeronautics*. Harbin (China) April 2009.

[3] T. Nguyên and J-A Désidéri, "Dynamic Resilient Workflows for Collaborative Design", *Proceedings of the 6th Intl. Conf. on Cooperative Design, Visualization and Engineering*. Luxemburg. September 2009. Springer-Verlag. LNCS 5738, pp. 341–350 (2009)

[4] A.H.M ter Hofstede, W. Van der Aalst, M. Adams and N. Russell, "Modern Business Process Automation: YAWL and its support environment", Springer (2010).

[5] D. Mogilevsky,G.A. Koenig and Y. Yurcik, "Byzantine anomaly testing in Charm++: providing fault-tolerance and survivability for Charm++ empowered clusters", *Proceedings of the 6th IEEE Intl. Symp. On Cluster Computing and Grid Workshops*. CCGRIDW'06. Singapore. May 2006. pp146-154.

[6] E. Deelman and Y. Gil., "Managing Large-Scale Scientific Workflows in Distributed Environments: Experiences and Challenges", *Proceedings of the 2nd IEEE Intl. Conf. on e-Science and the Grid*. Amsterdam (NL). December 2006. pp 26-32.

[7] Oracle Corp. "Oracle VM VirtualBox, User Manual", version 3.2.0, May 2010. Also: http://www.virtualbox.org.

[8] M. Ghanem, N. Azam, M. Boniface and J. Ferris, "Grid-enabled workflows for industrial product design", *Proceedings of the 2nd Intl. Conf. on e-Science and Grid Computing*. Amsterdam (NL). December 2006. pp 88-92.

[9] G. Kandaswamy, A. Mandal and D.A. Reed, "Fault-tolerant and recovery of scientific workflows on computational grids", Proceedings *of the 8th Intl. Symp. On Cluster Computing and the Grid.* 2008. pp 264-272.

[10] H. Simon. "Future directions in High-Performance Computing 2009-2018". Lecture given at the *ParCFD 2009 Conference*. Moffett Field (Ca). May 2009.

[11] J. Wang, I. Altintas, C. Berkley, L. Gilbert and M.B. Jones, "A high-level distributed execution framework for scientific workflows", *Proceedings of the 4th IEEE Intl. Conf. on eScience*. Indianapolis (In). December 2008. pp 156-164.

[12] D. Crawl and I. Altintas, "A Provenance-Based Fault Tolerance Mechanism for Scientific Workflows", *Proceedings of the 2nd Intl. Provenance and Annotation Workshop*. IPAW 2008. Salt Lake City (UT). June 2008. Springer. LNCS 5272. pp 152-159.

[13] M. Adams, "Facilitating Dynamic Flexibility and Exception Handling for Workflows", *PhD Thesis, Queensland University of Technology*, Brisbane (Aus.), 2007.

[14] M. Adams and L. Aldred, "The worklet custom service for YAWL, Installation and User Manual", Beta-8 Release, Technical Report, Faculty of Information Technology, Queensland University of Technology, Brisbane (Aus.), October 2006.

[15] L. Ramakrishnan et al., "VGrADS: Enabling e-Science workflows on grids and clouds with fault tolerance", *Proceedings of ACM SC'09 Conf.* Portland (Or.), November 2009.

[16] M. Baudin, "Introduction to Scilab", *Consortium Scilab*. January 2010. Also: http://wiki.scilab.org/

[17] F. Baude et al., "An efficient framework for running applications on clusters, grids and clouds", in "Cloud Computing: principles, systems and applications", Springer 2010.

[18] http://edition.cnn.com/2009/TRAVEL/01/20/mumbai.overview/ last accessed: 07/07/2010.

[19] J. Dongarra, P. Beckman et al. "The International Exascale Software Project Roadmap". *University of Tennessee EECS Technical report UT-CS-10-654*. May 2010. Available at: http://www.exascale.org/

[20] R. Gupta, P. Beckman et al. "CIFTS : a Coordinated Infrastructure for Fault-Tolerant Systems". *Proceedings of the 38th Intl. Conf. Parallel Processing Systems*. Vienna (Au). September 2009.

[21] The Workflow Management Coalition. http://www.wfmc.org

[22] D. Abramson, B. Bethwaite et al. "Embedding Optimization in Computational Science Workflows". *Journal of Computational Science 1 (2010)*. Pp 41-47. Elsevier.

Contents