Resilience Issues for Application Workflows on Clouds

Toàn Nguyên
Project OPALE
INRIA Grenoble Rhône-Alpes
Grenoble, France
Toan.Nguyen@inria.fr

Jean-Antoine-Désidéri
Project OPALE
INRIA Sophia-Antipolis Méditerranée
Sophia-Antipolis, France
Jean-Antoine.Desideri@inria.fr

***Abstract*-Two areas are currently the focus of active research, namely cloud computing and high-performance computing. Their expected impact on business and scientific computing is such that most application areas are eagerly uptaking or waiting for the associated infrastructures. However, open issues still remain. Resilience and load-balancing are examples of such areas where innovative solutions are required to face new or increasing challenges, e.g., fault-tolerance. This paper presents existing concepts and open issues related to the design, implementation and deployment of a fault-tolerant application framework on cloud computing platforms. Experiments are sketched including the support for application resilience, i.e., fault-tolerance and exception-handling. They also support the transparent execution of distributed codes on remote high-performance clusters.**

Keywords-workflows-fault-tolerance; resilience; simulation; cloud computing; high-performance computing.

I. INTRODUCTION

The future of computing systems in the next decade is sometimes advertised as a combination of virtual labs running large-scale application workflows on clouds that operate exascale computers [40][41]. Although this vision is attractive, it currently carries some inherent weaknesses. Among them are the complexity of the applications, e.g., multi-scale and multi-disciplines, the technical layers barrier, e.g., the network infrastructures, the multicore HPC clusters and finally the overwhelming technicalities that rely on experts that are not the final users. The consequences are that important challenges still lay ahead of us, among which are error management, fault-tolerance and application resilience.

Error recovery has long been a difficult challenge for both the computer science engineers and the application users. Approaches dealing with errors, failures and faults have mostly been designed by system engineers [20]. The characterization of faults and failures is indeed made inside software systems [37]. The emergence and widespread use of high-performance multi-core systems is also increasing the concerns for error-prone infrastructures where the mean-time between failures is decreasing [44].

Operating and communication systems have long addressed the failure detection and recovery problems with sophisticated restart and fail-safe protocols, from both the theoretical and implementation perspectives [39][42].

However, the advent of high-performance computing systems and distributed computing environments provide opportunities for new challenging applications to be deployed and run in order to solve unprecedented complex problems, e.g., full 3D aircraft flight dynamics simulation [2]. This stimulates the design of large-scale and long-running multi-discipline and multi-scale applications. They are expected to be standard within the next decade.

This induces expectations from the designers and users of such applications, e.g., better application accuracy, better performance, high-level and user-friendly interfaces, and resilience capabilities.

Consequently, rising concerns appear questioning the characterization, tracing and recovery from errors in such complex applications [43].

Indeed, the number and variety of components invoked during the execution of these applications are increasing:

- Operating system components (system libraries)
- Network components (virtual nodes, servers, backbones, protocols, messaging, duplication, etc.)
- Middleware components (resource allocation, authentication, authorization, load-balancing, etc.)
- Application components (software libraries for synchronization, results storage and migration, computation, user interfaces, etc.).

This results in several layers of software where the early detection of errors and their effective recovery are crucial with respect to resource allocation, usage cost, performance, system survivability, application consistency and user satisfaction [6][8]. Therefore, the software stack includes several different logics that must be carefully taken into account, i.e., identified and coordinated, in case of errors [20][44].

This paper explores the design, implementation and use of cloud infrastructures from the application perspective. It proposes specific techniques to handle application errors and recovery. The cloud infrastructure includes heterogeneous hardware and software components. Further, the application codes must interact in

a timely, secure and effective manner. Additionally, because the coupling of remote hardware and software components is prone to run-time errors, sophisticated mechanisms are necessary to handle unexpected failures at the infrastructure, system and application levels [19][25]. Consequently, specific management software is required to handle unexpected application and system behaviors [9][11][12][15][45].

The paper is focused on reactive approaches to occurring errors. It does not address error prevention and proactive approaches, e.g., preventive data and code migration and duplication [44]. Neither does it address prevention issues based on statistical evaluation and prediction of error occurrences and log analysis.

Indeed, the paper follows the position mentioned in [44]: "*This limited comprehension of root causes makes fault effect avoidance (the capability to avoid the effects of faults) difficult. Without a good understanding of root causes, it seems illusory to design and validate fault prediction mechanisms. Without good fault prediction systems, research on proactive actions is almost useless. In addition, even if at some point, we are capable of predicting errors accurately, we still have to find: 1) acceptable solutions to handle false negatives, and 2) how to handle predicted software errors (process or virtual machine migration is not a response for software errors)*".

This paper focuses on application resilience, i.e., survivability mechanisms to ensure the consistent termination of the applications, in the case of unexpected faulty behavior. Section II is an overview of related work. Section III is a description of open issues and gives an overview of running testcases. Section IV is a conclusion.

II. RELATED WORK

A. Definitions

Application resilience uses several notions that need to be detailed:

- Errors
- Faults
- Failures
- Exceptions
- Recovery
- Fault-tolerance
- Robustness
- Resilience

The generic term *error* usually encompasses different types of abnormal situations and behaviors. These might originate in system, middleware and application unexpected discrepancies.

In systems such as Apache's ODE [37], system *failures* and application *faults* address different types of errors.

A *failure* to resolve a DNS address is different from a process fault, e.g., a bad expression. Indeed, a system failure does not impact the correct logics of the application process at work, and should not be handled by it, but by the system error-handling software instead: "*failures are non-terminal error conditions that do not affect the normal flow of the process*" [37].

However, an activity can be programmed to throw a *fault* following a system *failure*, and the user can choose in such a case to implement a specific application behavior, e.g., a number of activity retries or its termination.

Application and system software usually raise *exceptions* when faults and failures occur. The exception handling software then handles the faults and failures. This is the case for the YAWL workflow management system [46][47][48], where specific *exlets* can be defined by the users [4]. They are components dedicated to the management of abnormal application or system behavior (Figure 1). The extensive use of these exlets allows the users to modify the behavior of the applications in real-time, without stopping the running processes. Further, the new behavior is stored as a component workflow which incrementally modifies the application specifications. The latter can therefore be modified dynamically to handle changes in the user requirements.

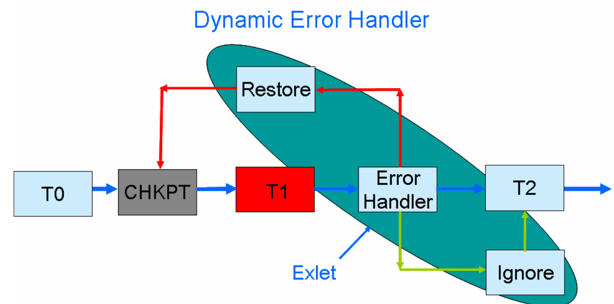

Figure 1. Error-handler.

Fault-tolerance is a generic term that has long been used to name the ability of systems and applications to handle errors. Transactional systems for example need to be fault-tolerant [38]. Critical business and scientific applications need to be fault-tolerant, i.e., to resume consistently in case of internal or external errors.

Therefore *checkpoints* need to be designed at specific intervals to backtrack the applications to consistent points in the application execution, and *restart* be enabled from there. They form the basis for *recovery* procedures.

Application *robustness* is the property of software that are able to survive consistently from data and code errors. This area is a major concern for complex numeric software that deal with data *uncertainties*. This is particularly the case for simulation applications [24].

This is also a primary concern for the applications faced to system and hardware errors. In the following, we include both (application external) fault-tolerance and (internal) robustness in the generic term *resilience* [1].

Therefore we do not follow here the definition given in [44]: "*By definition a failure is the impact of an error itself caused by a fault.*"

But, we fully adhere to the following observation: "*the response to a failure or an error depends on the context*

Copyright (c) IARIA, 2012. ISBN: 978-1-61208-186-1

and the specific sensitivity to faults of the usage scenarios, applications and algorithms" [44].

B. Overview

Simulation is a prerequisite for product design and for scientific breakthrough in many application areas ranging from pharmacy, biology to climate modeling and aircraft design [25]. They all require extensive simulation and testing. This requires often large-scale multidiscipline experiments, including the management of petabytes volumes of data and large multi-core supercomputers [10].

In such application environments, various teams usually collaborate on several projects or part of projects. Computerized tools are often shared and tightly or loosely coupled [23]. Some codes may be remotely located and non-movable. This is supported by distributed code and data management facilities [29]. And unfortunately, this is prone to a large variety of unexpected errors and breakdowns [30].

Data replication and redundant computations have been proposed to prevent from random hardware and communication failures [31], as well as failure prediction [32], sometimes applied to deadline-dependent scheduling [12].

System level fault-tolerance in specific programming environments is also proposed, e.g., CIFTS [15]. Also, middleware usually support mechanisms to handle fault-tolerance in distributed job execution, usually calling upon data replication and redundant code execution [9][15][22][24].

Also, erratic application behavior needs to be supported [34]. This implies evolution of the application process in the event of such occurrences. Little has been done in this area [33][35]. The primary concerns of the application designers and users have so far focused on efficiency and performance [36]. Therefore, application unexpected 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 restarted.

The concerns focus therefore on application resilience, although intra-node fault-tolerance is also a major concern [39].

Studies have focused on reducing checkpoint sizes and frequency, as well writing overheads [40]. Examples are diskless checkpointing [43], compressed checkpoints [44] and incremental checkpointing [41].

An extensible approach for petascale and future exascale systems is proposed in [45], based on a multi-level checkpointing scheme called *Scalable Checkpoint/Restart* (SCR) which proves to be effective. It provides an explicit checkpoint model to compute the optimal number of checkpoint levels and frequency of checkpoints at each level. The model and strategy are used to predict the checkpointing overhead and performance of the systems targeted. They are assessed by experiments on thousands of run hours on several production HPC clusters. This results in a thorough analysis of the impact of checkpoint intervals on overall system efficiency with respect to failure rate, compute intervals and file systems costs.

A dynamic approach is presented in the following sections. It support the evolution of the application behavior using the introduction of new exception handling rules at run-time by the users, based on occurring (and possibly unexpected) events and data values. The running workflows do not need to be suspended in this approach, as new rules can be added at run-time without stopping the executing workflows.

This allows on-the-fly management of unexpected events. This approach also allows a permanent evolution of the applications that supports their continuous adaptation to the occurrence of unforeseen situations [35]. As new situations arise and data values appear, new rules can be added to the workflows that will permanently take them into account in the future. These evolutions are dynamically hooked onto the workflows without the need to stop the running applications. The overall application logic remains therefore unchanged. This guarantees a constant adaptation to new situations without the need to redesign the existing workflows. Further, because exception-handling codes are themselves defined by dedicated component workflows, the user interface remains unchanged [14].

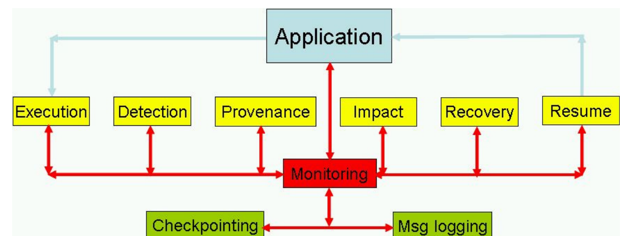

Figure 2. Architecture of a resilience sub-system.

III. OPEN ISSUES

A. Error management

Many open issues are still the subject of active research concerning application resilience. The paradigm ranges from code and data duplication and migration, to the monitoring of application behavior, and this includes also quick correctness checks on partial data values, the design of error-aware algorithms, as well as hybrid checkpointing-message logging features (Figure 2).

The baseline is:

- The early detection of errors,
- Root cause characterization,
- Characterization of transient vs. persistent errors,
- The tracing and provenance of faulty data,
- The identification of the impacted components and their associated corrupted results,

Copyright (c) IARIA, 2012. ISBN: 978-1-61208-186-1

- The ranking of the errors (warnings, fatal, medium) and associated actions (ignore, restart, backtrack),
- The identification of pending components,
- The identification and purge of transient messages,
- The secured termination of non-faulty components,
- The secure storage of partial and consistent results,
- The quick recovery of faulty and impacted components,
- The re-synchronization of the components and their associated data,
- The properly sequenced restart of the components.

Each of these items needs appropriate implementation and algorithms in order to orchestrate the various actions required by the recovery of the faulty application components.

B. Error detection

The early characterization of errors is difficult because of the complex software stack involved in the execution of multi-discipline and multi-scale applications on clouds. The consequence is that errors might be detected long after the root cause that initiated them occurred. Also, the error observed might be a complex consequence of the root cause, possibly in a different software layer.

Similarly, the exact tracing and provenance data may be very hard to sort out, because the occurrence of the original fault may be hidden deep inside the software stack.

Without explicit data dependency information and real-time tracing of the components execution, the impacted components and associated results may be unknown. Hence, there is a need for explicit dependency information [38].

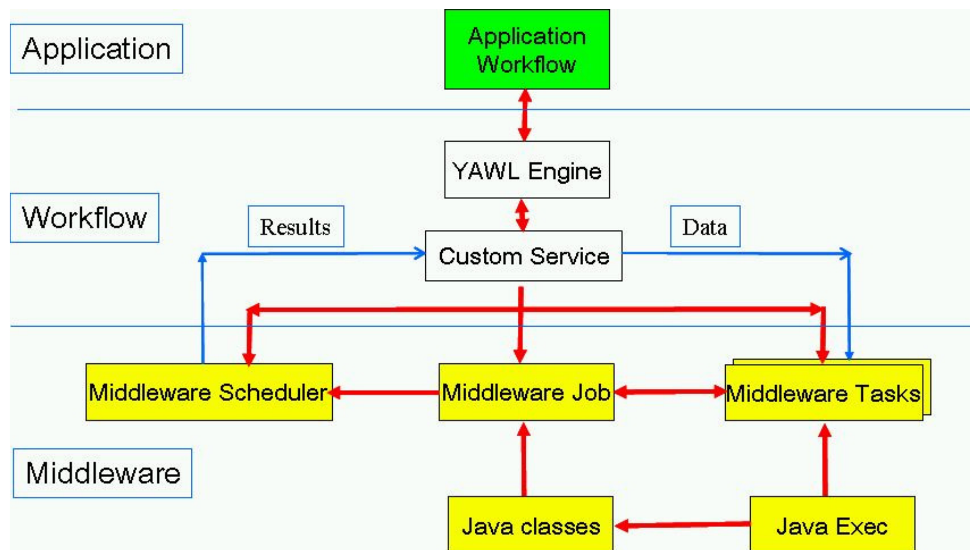

Figure 3. The YAWL workflow and middleware interface.

The ranking of errors is dependent on the application logic and semantics (e.g., default values usage). It is also dependent on the logics of each software layer composing the software stack. Some errors might be recoverable (Unresolved address, resource unavailable, etc.), some others not (Network partition, etc.). In each case, the actions to recover and resume differ: ignore, retry, reassign, suspend, abort.

In all cases, resilience requires the application to include four components:

- A monitoring component for (early) error detection,
- A (effective) decision system, for provenance and impact assessment,
- A (low overhead) checkpointing mechanism,
- An effective recovery mechanism.

Further, some errors might be undetected and transient. Without explicit data dependency information and real-time tracing of the components execution, the impacted components and associated results may be unknown. Hence there is a need for explicit dependency information between the component executing instances and between the corresponding result data [38].

A sub-system dedicated to application resilience includes therefore several components in charge of specific tasks contributing to the management of errors and consistent resuming of the applications (Figure 2). First, it includes an intelligence engine in charge of the application monitoring and of the orchestration of the resilience components. This engine runs as a background process in charge of event listening during the execution of the applications. It is also in charge of triggering the periodic checkpointing mechanism, depending on the policy defined for the applications being monitored. It is also in charge of triggering the message-logging component for safekeeping the messages exchanged

Copyright (c) IARIA, 2012. ISBN: 978-1-61208-186-1

between tasks during their execution. This component is however optional, depending on the algorithms implemented, e.g., checkpointing only or hybrid checkpoint-message logging approaches. Both run as background processes and should execute without user intervention. Should an error occur, an error detection component that is constantly listening to the events published by the application tasks and the operating system raises the appropriate exceptions to the monitoring component. The following components are then triggered in such error cases: an optional provenance component which is in charge of root cause characterization, whenever possible. An impact assessment component is then triggered to evaluate the consequences of the error on the application tasks and data, that may be impacted by the error. Next, a recovery component is triggered in charge of restoring the impacted tasks and the associated data, in order to re-synchronize the tasks and data, and restore the application to a previous consistent state. A resuming component is finally triggered to deploy and rerun the appropriate tasks and data on the computing resources, in order to resume the application execution.

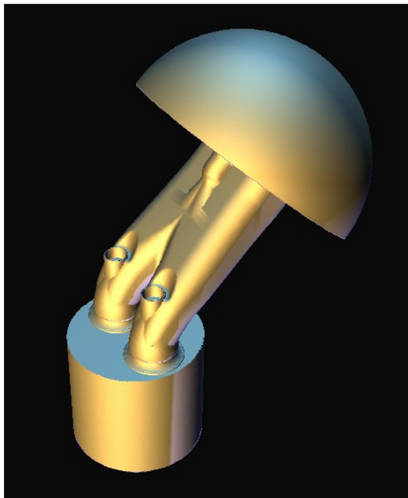

Figure 4. Cylinder optimized input pipes – courtesy Lab. Roberval, Université de Technologie de Compiègne (France).

In contrast with approaches designed for global fault-tolerance systems, e.g., CIFTS [15], this functional architecture describes a sub-system dedicated to application resilience. It can be immersed in, or contribute to, a more global fault-tolerance system that includes also the management of system and communication errors.

C. *Experiments*

A distributed platform featuring the resilience capabilities described above is developed [30], based on the YAWL workflow management system [46][48]. Experiments are connecting the platform to the FAMOSA optimization suite [24] developed at INRIA by project OPALE [29].

The experiments are deployed on the Grid5000 infrastructure [13]. This involves five different locations throughout France (Figure 5), including two locations near Paris for CAD data and mesh generation. In addition, another location near Nantes involves CFD calculations, and another one in Sophia-Antipolis near Nice is dedicated to optimization. The last location in Grenoble is for application deployment, monitoring and result visualization (Figure 5).

The first experiments simulated this deployment scenario by duplicating the application with two identical parallel sequences running on Lyon and Grenoble clusters respectively, then on Sophia-Antipolis and Grenoble respectively.

This allowed for performance assessment of the various clusters implied on Lyon, Sophia-Antipolis and Grenoble.

Because Grid5000 infrastructure does not currently serve Nantes, a further experiment will invoke the clusters in Rennes instead, Lille instead of Paris2, and Orsay instead of Paris1.

An extension will invoke one more cluster in Lyon instead of one of the Sophia-Antipolis instances. A total of six remote HPC clusters will therefore be invoked (Figure 6). The reason for this is that most application codes are proprietary and are located at the various partners offices.

Data transfers between clusters use a 10 Gbps IP network infrastructure dedicated to Grid5000 (Figure 10).

All the locations involve HPC clusters and are invoked from a remote workflow running on a Linux workstation in Grenoble.

The various errors that are taken into account by the resilience algorithm include run-time errors in the solvers, inconsistent CAD and mesh generation files, and execution time-outs [3].

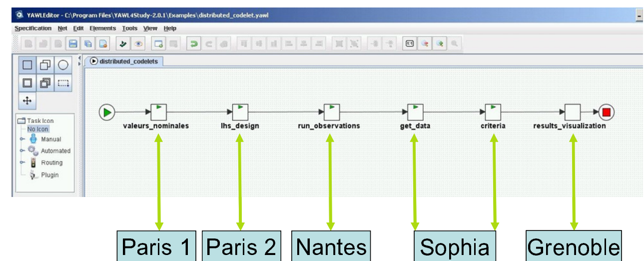

Figure 5. The workflow experiment schema.

FAMOSA is currently tested for car rear mirrors optimization (Figure 7) and by ONERA (the French National Aerospace Research Office) for aerodynamics optimization.

FAMOSA is an acronym for "*Fully Adaptive Multilevel Optimization Shape Algorithms*" and includes C++ components for:

- CAD generation,
- Mesh generation,
- Domain partitioning,
- Parallel CFD solvers using MPI, and
- Post-processors.

The input is a design vector and the output is a set of simulation results. The components also include other software for mesh generation, e.g., Gmsh [26], partitioning,

Copyright (c) IARIA, 2012. ISBN: 978-1-61208-186-1

e.g., Metis [27] and solvers, e.g., Num3sis [28]. They are remotely invoked from the YAWL application workflow by shell scripts [30].

The FAMOSA components are triggered by remote shell scripts running for each one on the HPC cluster. The shell scripts are called by YAWL custom service invocations from the user workflow running on the workstation [30].

Other testcases implemented by academic and industry partners include the optimization of cylinder input pipes and valves for car engines (Figure 4 and 8) and vehicle aerodynamics (Figure 9).

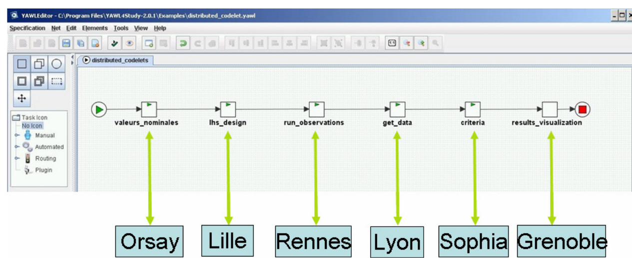

Figure 6. The final distributed workflow.

IV. CONCLUSION

The requirements for large-scale simulations make it necessary to deploy various software components on heterogeneous distributed computing infrastructures [10][33]. These environments are often remotely located among a number of project partners for administrative and collaborative purposes.

An overview of resilience is given with open issues. A workflow distributed platform and running testcases are briefly described. The underlying interface to the distributed components is a middleware providing resource allocation and job scheduling [13]. Besides fault-tolerance provided by the middleware, which handles communication and hardware failures, the users can define and handle application errors at the workflow level. Application errors may result from unforeseen situations, data values and boundary conditions. In such cases, user intervention is required in order to modify parameter values and application behavior. Complex error characterization is then invoked to assess the impact on the executing tasks and the data involved. The approach presented uses dynamic rules and constraint enforcement techniques, combined with asymmetric checkpoints. It is based on the YAWL workflow management system.

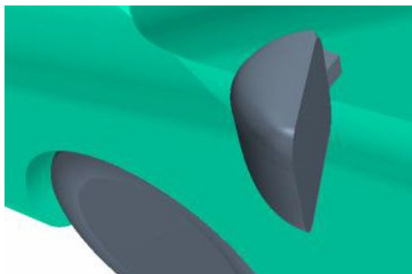

Figure 7. Optimized rear mirror (courtesy CD-adapco).

ACKNOWLEDGMENTS

This work is supported by the European Commission FP7 Cooperation Program “Transport (incl. aeronautics)”, for the *GRAIN* Coordination and Support Action *(“Greener Aeronautics International Networking”)*, grant ACS0-GA-2010-266184. It is also supported by the French National Research Agency ANR (*Agence Nationale de la Recherche*) for the OMD2 project (*Optimisation Multi-Discipline Distribuée)*, grant ANR-08-COSI-007, program COSINUS (*Conception et Simulation*).

The authors wish to thank Laboratoire Roberval (Université de Technologie de Compiègne, France), and also CD-adapco, for the testcase implementations and the images.

REFERENCES

[1] T. Nguyên, L. Trifan and J-A Désidéri . “A Distributed Workflow Platform for Simulation”. Proc. 4th Intl. Conf on Advanced Engineering Computing and Applications in Sciences (ADVCOMP2010). pp. 375-382. Florence (I). October 2010.

[2] A. Abbas, “High Computing Power: A radical Change in Aircraft Design Process”, Proc. 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”, Proc. of the 6th Intl. Conf. on Cooperative Design, Visualization and Engineering (CDVE2009). Luxemburg. September 2009. Springer-Verlag. LNCS 5738, pp. 341–350 (2009)

[4] W. Van der Aalst et al., Modern Business Process Automation: YAWL and its support environment, Springer (2010).

[5] E. Deelman et Y. Gil., “Managing Large-Scale Scientific Workflows in Distributed Environments: Experiences and Challenges”, Proc. of the 2nd IEEE Intl. Conf. on e-Science and the Grid. pp. 165-172. Amsterdam (NL). December 2006.

[6] M. Ghanem, N. Azam, M. Boniface and J. Ferris. “Grid-enabled workflows for industrial product design”, Proc. of the 2nd Intl. Conf. on e-Science and Grid Computing. pp. 285-294. Amsterdam (NL). December 2006.

[7] G. Kandaswamy, A. Mandal and D.A. Reed., “Fault-tolerant and recovery of scientific workflows on computational grids”, Proc. of the 8th Intl. Symp. On Cluster Computing and the Grid. pp. 415-428. 2008.

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

[9] J. Wang, I. Altintas, C. Berkley, L. Gilbert and M.B. Jones., “A high-level distributed execution framework for scientific workflows”, Proc. of the 4th IEEE Intl. Conf. on eScience. pp. 147-156. Indianapolis (In). December 2008.

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

[11] 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.

[12] L. Ramakrishna, D. Nurmi et al.., “VGrADS: Enabling e-Science workflows on grids and clouds with fault tolerance”, Proc. ACM SC’09 Conf. pp. 369-376. Portland (Or.), November 2009.

[13] Grid5000 project home. Last accessed: 23/11/2011. https://www.grid5000.fr/mediawiki/index.php/Grid5000:Home.

[14] Dongarra, P. Beckman et al. "The International Exascale Software Roadmap". Volume 25, Number 1, 2011, International Journal of High Performance Computer Applications, pp. 77-83. Available at: http://www.exascale.org/ Last accessed: 03/31/2011.

[15] R. Gupta, P. Beckman et al. "CIFTS: a Coordinated Infrastructure for Fault-Tolerant Systems", Proc. 38th Intl. Conf. Parallel Processing Systems. pp. 145-156. Vienna (Au). September 2009.

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

[17] A.Bachmann, M. Kunde, D. Seider and A. Schreiber, "Advances in Generalization and Decoupling of Software Parts in a Scientific Simulation Workflow System", Proc. 4th Intl. Conf. Advanced Engineering Computing and Applications in Sciences (ADVCOMP2010). Pp 247-258. Florence (I). October 2010.

[18] R. Duan, R. Prodan and T. Fahringer. "DEE: a Distributed Fault Tolerant Workflow Enactment Engine for Grid Computing", Proc. 1st. Intl. Conf. on High-Performance Computing and Communications. pp. 255-267. Sorrento (I). LNCS 3726. September 2005.

[19] Sherp G., Hoing A., Gudenkauf S., Hasselbring W. and Kao O., "Using UNICORE and WS-BPEL for Scientific Workflow execution in Grid Environments", Proc. EuroPAR 2009. pp. 133-148. LNCS 6043. . Springer. 2010.

[20] B. Ludäscher, M. Weske, T. McPhillips and S. Bowers, "Scientific Workflows: Business as usual ?", Proc. BPM 2009. pp. 269-278. LNCS 5701. Springer. 2009.

[21] Montagnat J., Isnard B., Gatard T., Maheshwari K. and Fornarino M., "A Data-driven Workflow Language for Grids based on Array Programming Principles", Proc. SC 2009 4th Workshop on Workflows in Support of Large-Scale Science. pp. 23-35. WORKS 2009. Portland (Or). ACM 2009.

[22] Yildiz U., Guabtni A. and Ngu A.H., "Towards Scientific Workflow Patterns", Proc. SC 2009 4th Workshop on Workflows in Support of Large-Scale Science. pp. 135-145. WORKS 2009. Portland (Or). ACM 2009.

[23] Plankensteiner K., Prodan R. and Fahringer T., "Fault-tolerant Behavior in State-of-the-Art Grid Workflow Management Systems", CoreGRID Technical Report TR-0091. October 2007. http://www.coregrid.net Last accessed: 03/31/2011.

[24] Duvigneau R., Kloczko T., and Praveen C.., "A three-level parallelization strategy for robust design in aerodynamics", Proc. 20th Intl. Conf. on Parallel Computational Fluid Dynamics (ParCFD2008). pp. 241-252. May 2008. Lyon (F).

[25] E.C. Joseph, et al. "A Strategic Agenda for European Leadership in Supercomputing: HPC 2020", IDC Final Report of the HPC Study for the DG Information Society of the EC. July 2010. Available at: http://www.hpcuserforum.com/EU/ Last accessed: 03/31/2011.

[26] Gmsh. https://geuz.org/gmsh/ Last accessed: 03/31/2011.

[27] Metis. http://glaros.dtc.umn.edu/gkhome/metis/metis/overview Last accessed: 03/31/2011.

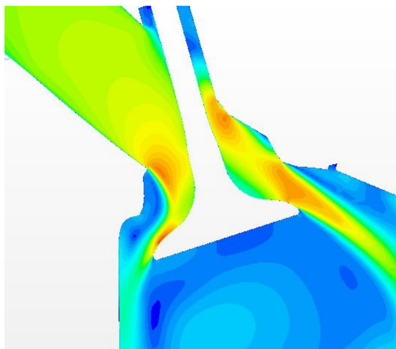

Figure 8. Cylinder flow simulation (courtesy CD-adapco).

[28] Num3sis. http://num3sis.inria.fr/blog/ Last accessed: 03/31/2011.

[29] OPALE project at INRIA. http://www-opale.inrialpes.fr and http://wiki.inria.fr/opale Last accessed: 05/03/2011.

[30] T. Nguyên, L. Trifan, J.A. Désidéri. "A Workflow Platform for Simulation on Grids", Proc. 7th Intl. Conf. on Networking and Services (ICNS2011). pp. 295-302. Venice (I). May 2011.

[31] Plankensteiner K., Prodan R. and Fahringer T., "A New Fault-Tolerant Heuristic for Scientific Workflows in Highly Distributed Environments based on Resubmission impact", Proc. 5th IEEE Intl. Conf. on e-Science. Oxford (UK). December 2009. pp 313-320.

[32] Z. Lan and Y. Li. "Adaptive Fault Management of Parallel Applications for High-Performance Computing", IEEE Trans. On Computers. pp. 45-56. Vol. 57, No. 12. December 2008.

[33] S. Ostermann, et al. "Extending Grids with Cloud Resource Management for Scientific Computing", Proc. 10th IEEE/ACM Intl. Conf. on Grid Computing. Pp. 266-278. 2009.

[34] E. Sindrilaru, A. Costan and V. Cristea. "Fault-Tolerance and Recovery in Grid Workflow Mangement Systems", Proc. 4th Intl. Conf. on Complex, Intelligent and Software Intensive Systems. pp. 162-173. Krakow (PL). February 2010.

[35] S. Hwang and C. Kesselman. "Grid Workflow: A Flexible Failure Handling Framework for the Grid", Proc. 12th IEEE Intl. Symp. on High Performance Distributed Computing. pp. 369-374. Seattle (USA). 2003.

[36] The Grid Workflow Forum. Last accessed: 06/21/2011. http://www.gridworkflow.org/snips/gridworkflow/space/start

[37] The Apache Foundation. http://ode.apache.org/bpel-extensions.html #BPELExtensions-ActivityFailureandRecovery Last accessed: 08/25 /2011.

[38] W. Zang, M. Yu, P. Liu. "A Distributed Algorithm for Workflow Recovery", Intl. Journal Intelligent Control and Systems. Vol. 12. No. 1. March 2007. pp 56-62.

[39] P. Beckman. "Facts and Speculations on Exascale: Revolution or Evolution?", Keynote Lecture. Proc. 17th European Conf. Parallel and Distributed Computing (Euro-Par 2011). pp. 135-142. Bordeaux (F). August 2011.

[40] P. Kovatch, M. Ezell, R. Braby. "The Malthusian Catastrophe is Upon Us! Are the Largest HPC Machines Ever Up?", Proc. Resilience Workshop at 17th European Conf. Parallel and Distributed Computing (Euro-Par 2011). pp. 255-262. Bordeaux (F). August 2011.

[41] R. Riesen, K. Ferreira, M. Ruiz Varela, M. Taufer, A. Rodrigues. "Simulating Application Resilience at Exascale", Proc. Resilience Workshop at 17th European Conf. Parallel and Distributed Computing (Euro-Par 2011). pp. 417-425. Bordeaux (F). August 2011.

[42] P. Bridges, et al. "Cooperative Application/OS DRAM Fault Recovery", Proc. Resilience Workshop at 17th European Conf. Parallel and Distributed Computing (Euro-Par 2011). pp. 213-222. Bordeaux (F). August 2011.

[43] Proc. 5th Workshop INRIA-Illinois Joint Laboratory on Petascale Computing. Grenoble (F). June 2011. http://jointlab.ncsa.illinois.edu/events/workshop5/ Last accessed 09/05/2011.

[44] F. Capello, et al. "Toward Exascale Resilience", Technical Report TR-JLPC-09-01. INRIA-Illinois Joint Laboratory on PetaScale Computing. Chicago (Il.). 2009. http://jointlab.ncsa.illinois.edu/

[45] Moody A., G.Bronevetsky, K. Mohror, B. de Supinski. Design, "Modeling and evaluation of a Scalable Multi-level checkpointing System", Proc. ACM/IEEE Intl. Conf. for High Performance Computing, Networking, Storage and Analysis (SC10). pp. 73-86. New Orleans (La.). Nov. 2010. http://library-ext.llnl.gov Also Tech. Report LLNL-TR-440491. July 2010. Last accessed: 09/12/2011.

[46] Adams M., ter Hofstede A., La Rosa M. "Open source software for workflow management: the case of YAWL", IEEE Software. 28(3): 16-19. pp. 211-219. May/June 2011.

Copyright (c) IARIA, 2012. ISBN: 978-1-61208-186-1

[47] Russell N., ter Hofstede A. "Surmounting BPM challenges: the YAWL story.", Special Issue Paper on Research and Development on Flexible Process Aware Information Systems. Computer Science. 23(2): 67-79. pp. 123-132. March 2009. Springer 2009.

[48] Lachlan A., van der Aalst W., Dumas M., ter Hofstede A. "Dimensions of coupling in middleware", Concurrency and Computation: Practice and Experience. 21(18):233-2269. pp. 75-82. J. Wiley & Sons 2009.

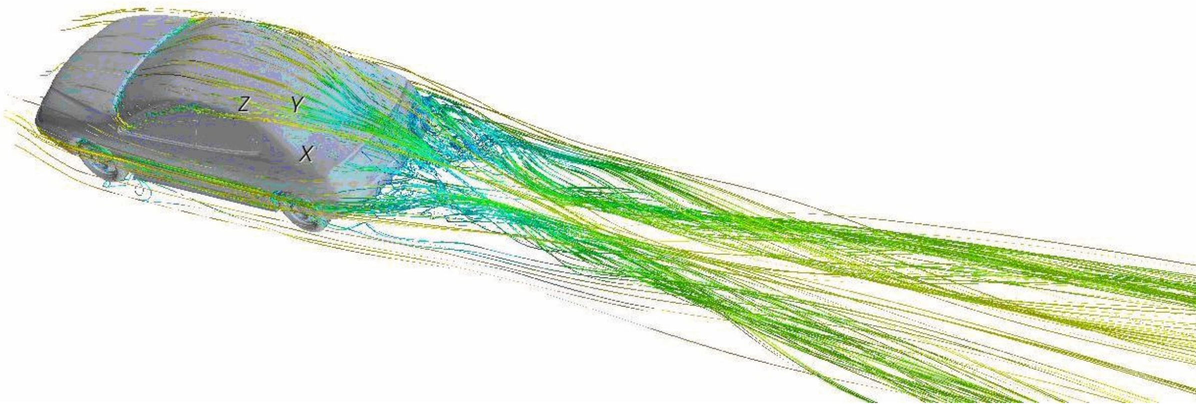

Figure 9. Vehicle aerodynamics simulation (courtesy CD-adapco).

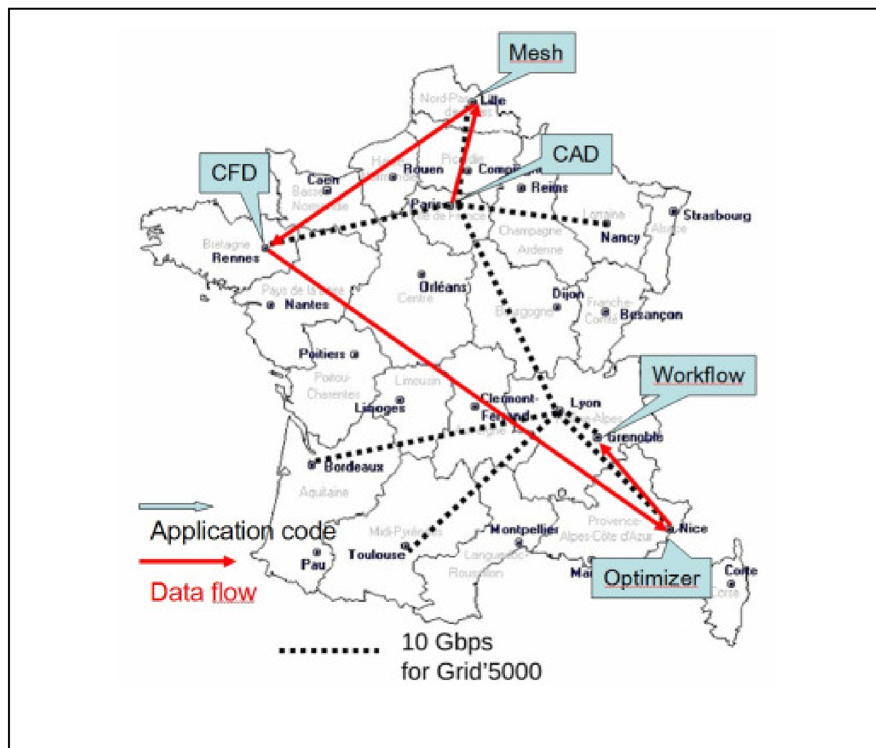

Figure 10. The testcase deployment on the Grid5000 infrastructure.