2012 9th International Conference on Ubiquitous Intelligence and Computing and 9th International Conference on Autonomic and Trusted Computing

Autonomic Activities in the Execution of Scientific Workflows: Evaluation of the AWARD Framework

Luis Assuncao[1,2] ; Carlos Goncalves[1,2] ; Jose C. Cunha[2]
[1]*Instituto Superior de Engenharia de Lisboa*
[2]*CITI/Dept. Informática, Fac. Ciências e Tecnologia, Universidade Nova de Lisboa*
lass@isel.ipl.pt ; cgoncalves@deetc.isel.pt; jcc@fct.unl.pt

Abstract*—Workflows have been successfully applied to express the decomposition of complex scientific applications. However the existing tools still lack adequate support to important aspects namely, decoupling the enactment engine from tasks specification, decentralizing the control of workflow activities allowing their tasks to run in distributed infrastructures, and supporting dynamic workflow reconfigurations. We present the AWARD (Autonomic Workflow Activities Reconfigurable and Dynamic) model of computation, based on Process Networks, where the workflow activities (AWA) are autonomic processes with independent control that can run in parallel on distributed infrastructures. Each AWA executes a task developed as a Java class with a generic interface allowing end-users to code their applications without low-level details. The data-driven coordination of AWA interactions is based on a shared tuple space that also enables dynamic workflow reconfiguration. For evaluation we describe experimental results of AWARD workflow executions in several application scenarios, mapped to the Amazon (Elastic Computing EC2) Cloud.

Keywords – scientific workflows; parallel execution, distributed processing; Cloud; Tuple space

I. INTRODUCTION

Workflows have been used for the development of scientific applications in a diversity of domains [1], [2], [3], [4]. Such efforts have been supported by multiple workflow tools [5], such as Triana [6], Taverna [7] and Kepler [8]. Due to the increasing complexity of the applications and the diversity of the execution infrastructures there is still a need for improving the support by the workflow tools. In our previous work on a geostatistics application [9], as well as in other works, such as [4], some difficulties were identified regarding existing tools: i) Concerning the application development there is a need for a more clear separation of the specification of the application logic from the details of the workflow enactment. Although there are multiple proposals addressing this issue [4], the application developer is often faced with the need to understand and implement details at the level of workflow engine. For example, in Kepler to develop a new Actor, the programmer needs to implement methods difficult to understand and dependent on the enactment engine, e.g. *preinitialize()*; *initialize()*; *prefire()*; *fire()*; *postfire()* and *wrapUp()* [8]; ii) Concerning the workflow computation, several models allow parallel execution, e.g. by following the Process Networks (PN) model [10], with a well known semantics [11] as in Kepler (PN Director) [8] and in [12]. However, in such systems activities are executed as threads within the same process, which acts as a monolithic workflow engine. Furthermore, a unique central entity, as the Kepler Director, handles the global coordination. There is a clear need for a more decentralized control and coordination of the workflow activities, each designed as an autonomic component with separate control and behavior. This enables a more flexible workflow management, and eases the mappings of the workflow activities across Clusters, Grids, and Clouds; iii) Concerning flexibility in the workflow management, which is an issue addressed in business workflows [13], [14], [15], it has also been a trend towards enabling more complex scientific experiments. For example, allowing separate parts of a workflow to be launched or controlled individually by different users, and dynamic reconfiguration of the workflows, in response to changes in the application logic and behavior, e.g., as required in computational steering experiments, or in interactive workflows. Although several approaches have been proposed [16], [17], there are still insufficient reports on experimentation illustrating the feasibility of the above dimensions in real applications. We also note that there is still a lack of available, operational working prototypes of easy-to-use workflow tools supporting the above experiments.

In order to address the above issues, we present the AWARD (Autonomic Workflow Activities, Reconfigurable and Dynamic) model. Our goal is to provide a framework supported by a tool that can be used to support the practical evaluation of solutions to the above concerns, in distinct application scenarios. The paper illustrates the characteristics of the AWARD framework and the implementation of a working prototype, which may run stand-alone in a single machine, or be launched in a distributed Cloud infrastructure. We describe results of experiments on workflows execution, used to evaluate the AWARD benefits: i) AWARD follows the Process Networks (PN) model, extended with autonomic processes for modeling the workflow activities (AWA), with decentralized control and allowing flexible mappings to distributed infrastructures such as the Cloud (Amazon EC2); ii) the AWARD shared space for data-driven communication and coordination of AWA activities as well as enabling dynamic reconfigurations; iii) End-users do not need to know and implement details related to an enactment engine for developing the application algorithms (*Tasks*), which are developed as Java classes with a generic interface that encapsulates calls to Web services, or/and run local or remote processes, allowing legacy applications in other languages, e.g., C or Fortran languages.

978-0-7695-4843-2/12 $26.00 © 2012 IEEE
DOI 10.1109/UIC-ATC.2012.14

In section 2 of the paper we describe the concepts of the AWARD model. In section 3 we describe the implementation. In section 4, AWARD is compared with related work. In section 5 we describe the experimentation and evaluation of AWARD on different scenarios. In section 6 we present conclusions and future work.

II. THE AWARD MODEL

A workflow in AWARD is defined as a graph of interconnected nodes, where each node is a configurable software component, AWA (Autonomic Workflow Activity) that encapsulates a workflow activity for executing a specific task. A *Task* is any software component implementing a generic interface that encapsulates an algorithm to solve a problem. This interface specifies the *Task* execution entry point, *EntryPoint(Parameters, Arguments)* → *Results*, where *Parameters* is a list of initial parameters, *Arguments* is a list of items from the activity inputs and *Results* is a list of items to be mapped to activity output tokens. Any AWA *Task* implementing that interface can be executed as a local thread, or as an operating system process in the local host computer, or as a job submitted to a remote cluster, or as an invocation of a Web Service. Furthermore, one workflow can itself be encapsulated inside an AWA *Task*, supporting workflow hierarchies. Besides the *Task* element, a given configuration of an AWA includes the following elements: i) A set of input and output ports, where each port has an associated data type, and a state that can be enabled or disabled. For each output port, a list of input ports to where the tokens will be sent; ii) A configurable input mapping, defining how the *Arguments* to be passed to *Task* invocation are obtained from the input ports; iii) A configurable output mapping, defining how the *Results* from the *Task* execution are forwarded to the output ports; and iv) An *Autonomic controller*, which controls the life-cycle of the workflow activity.

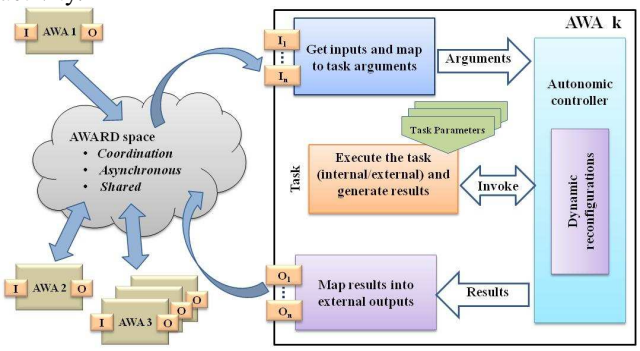


Fig. 1. Model of an AWA and interactions through the AWARD Space

The flows from the output ports to the input ports of the AWA activities are represented by tuples stored in a global shared space (AWARD Space) that indirectly supports all interactions between the workflow activities. The model assumes that the AWA nodes and their ports have unique logical names (string IDs), as keys used to transparently index and retrieve tuples from the AWARD Space. As shown in **Fig. 1**, firstly, the AWA autonomic controller gets the matching tuples from the shared space for each enabled input port and passes them to the *Task Arguments*. Secondly, the software component with the *Task* implementation is dynamically loaded and its *EntryPoint* is called. Thirdly, the *Results* from *Task* execution are mapped to the enabled output ports of the AWA activity, and the corresponding output tuples are generated and put into the shared space.

The workflow activities (AWA) communicate asynchronously through the AWARD Space where data tokens produced by an activity are stored until the destination activity consumes them. The execution order, when required the FIFO order, is easily insured by the identification of each token using a sequential iteration number. Therefore the AWARD workflows are data-driven allowing each activity to start, run and terminate separately without any centralized enactment engine, and supporting mappings to distributed architectures for instance Cloud (Amazon AWS).

Fig. 2 depicts the internals of the *Autonomic controller*: i) A state machine controls the execution of the AWA controller; ii) A rules engine with a knowledge base (a set of facts and rules) evaluates rules that produce new facts; and iii) Requests for dynamic reconfiguration and for information on the current context of AWA activities are injected as tuples in the AWARD Space and handled by event handlers. The rules engine allows to specify conditions to configure the A*utonomic controller*: i) Flexible state machines to control the life-cycle of the AWA autonomic controllers including special states for dynamic reconfigurations; ii) Configurable input and output mappings; iii) Support for multiple iterations of the workflow activities, ensuring that input and output tuples for each iteration are handled in the AWARD Space as distinct.

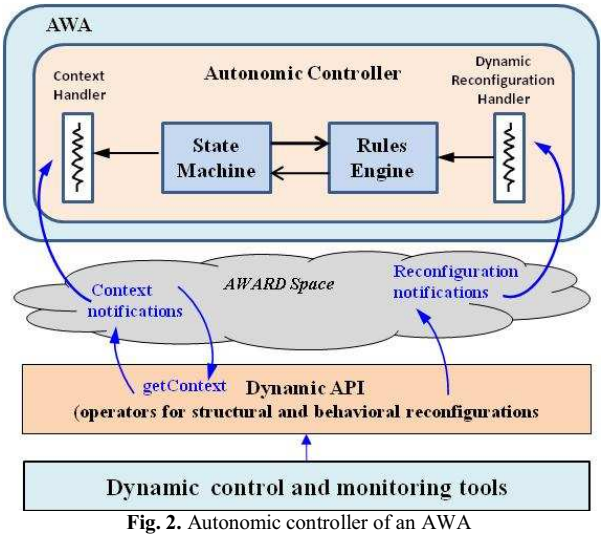


Fig. 2. Autonomic controller of an AWA

The AWARD model supports dynamic reconfiguration: i) Each AWA is autonomous, with the workflow activities being launched individually; ii) Each AWA has an independent execution control, allowing its reconfiguration while the others continue running; iii) Each AWA is configurable in terms of number and state of its ports and

their links to other activities; iv) The data flow or control flow communication between AWAs allow different rates and data granularities. The AWARD model supports operators for dynamic reconfigurations. The dynamic API allows any application tool to perform workflow reconfigurations, which is useful in many application scenarios where the workflow structure or the behavior must be changed, e.g., create, delete, enable or disable ports, modify links from output to input ports, change *Task* and its parameters, and suspend/resume activities. A detailed discussion of the operators is not included, due to space limitations.

Fig. 3 presents the state machine and the main transitions with conditions especially those who are involved in dynamic reconfigurations. When new activities are dynamically added in some reconfiguration scenarios they eventually need immediate configuration. Thus all activities start in the ***Init*** state where an evaluation is made if the activity goes to normal execution or if it goes to the ***Wait*** state, waiting for an immediate configuration.

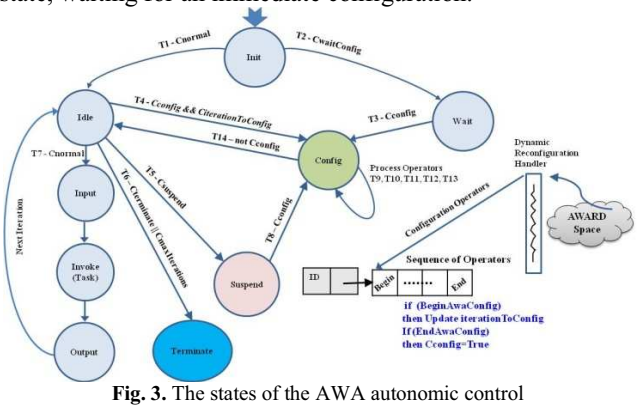


Fig. 3. The states of the AWA autonomic control

Normally without reconfiguration, the sequence of states per iteration is as follows:

Idle – In this state a set of rules related to dynamic reconfigurations are evaluated as well as the control of iteration number verifying if the last iteration has been reached. This is the only state where the state machine enters in the configuration state to apply dynamic configurations.

Input – This state defines the point where the activity waits and gets the data from AWARD space to its inputs and then maps this data to the arguments that will be passed to *Task* invocation.

Invoke – This state creates an instance of the *Task* (dynamic binding) according to the type (class) defined in AWA specification, and invokes the *EntryPoint()* method, passing the arguments prepared in the *Input* state as well as eventual static parameters defined for the activity.

Output – In this state the result returned from *Task* invocation is mapped to the activity Outputs and stored in AWARD space. After this state the state machine goes to the ***Idle*** state to start the next iteration.

Config – In this state the sequences of tuples describing requests for dynamic reconfigurations are processed. When the execution starts, each *AWA Autonomic controller* subscribes to an asynchronous notification service of the AWARD Space in order to receive the special reconfiguration tuples through the *Dynamic Reconfiguration Handler*. These tuples carry information to insert facts and rules into the rules engine of that *AWA Autonomic controller* and force the state machine to the reconfiguration state where the corresponding actions are handled. Each AWA dynamic reconfiguration is composed by a sequence of operators enclosed by special *BeginAwaConfig* and *EndAwaConfig* operators ensuring the atomicity of each reconfiguration.

In order to specify an AWARD workflow, the information that must be provided corresponds to the 2nd column of table in **Fig. 4**. As indicated in 3rd column several items have default definitions.

AWA	What user needs to specify	Default
Inputs	Name	
	State (enable or disable)	Enable
	Token Data Type	
Outputs	Name	
	State (enable or disable)	Enable
	Token Data Type	
	List of Input names to send tokens	
Inputs Mappings	Function to map input tokens to Task Arguments	Array of tokens
Output Mappings	Function to map Task Results to output tokens	All Outputs receive results
Autonomic controller	Facts and Rules to the state machine	Basic AWA rules
	Number maximum of Iterations	1
Task	Any Java class implementing the generic interface	Utility Task library

Fig. 4. The specification of an AWA

Although this is out of the scope of our work, we note that any high level tool that is able to generate a XML file with a schema as required by AWARD can be used to specify the corresponding workflows. **Fig. 5** illustrates parts of the XML representation of an AWA with a *Task* to invoke a Web service and the corresponding definition of the input and output ports.

```
<Awa>  <!-- Activity to invoke a Web Service (WS) -->
  <name>InvokeWS</name> <!-- Name of the activity  -->
  <ControlUnit>...</ControlUnit>
  <input>  <!--Access ID to WS-->
    <name>In1</name> <!--Name of the input-->
    <state>Enabled</state>  <!--Initial state of the input-->
    <dataType>java.lang.String</dataType> <!--object Type received-->
  </input>
  <output>
    <name>WS-O1</name>  <!--Name of the output-->
    <state>Enabled</state>  <!--Initial state of the output-->
    <dataType>java.lang.String</dataType> <!--object Type produced-->
    <sendTo>Out-In</sendTo> <!--Name of destination input activity-->
  </output>
  <Task>
    <parameters> <!-- Url of the Web Service-->
      <par>http://WSserver/WS/WebService.asmx</par>
    </parameters>
    <LocalComponent>
      <mappingArgs>...</mappingArgs>
      <!-- Type that implements generic interface -->
      <!-- and invoke the WS in the URL parameter -->
      <taskType>awardutils.TaskInvokeWService</taskType>
      <mappingResults>...</mappingResults>
    </LocalComponent>
  </Task>
</Awa>
```

Fig. 5. The main parts of a xml AWA specification

III. AWARD IMPLEMENTATION

The AWARD model is currently supported by a working prototype that is being used to enable experimentation with real applications. It is developed in Java and uses JESS [18] as the rule engine. It allows AWA activities to spread over several computers in a distributed environment. The AWARD Space is a standalone server application based on IBM TSpaces API [19] that can be executed in any computer accessible from other computers where the AWA activities are running. The AWARD Space also supports the tools to monitor workflow execution and to manage workflow reconfigurations at execution time. The prototype offers a set of tools to launch one or more AWA per computer, to browse and edit workflows definition XML files, and to manage the AWARD space for debugging and monitoring workflow execution.

IV. RELATED WORK

Both AWARD and Kepler (PN Director) [8], follow the semantics of Process Networks (PN) [10]. There are important differences between Kepler and AWARD. In Kepler the parallelism is based on the execution of each Actor by threads within the same monolithic process, and actors communicate using first-in-first-out memory buffers. The order of the execution (actors firing) is controlled by a centralized PN Director that can be very inefficient, as it must keep looking for actors with sufficient data to fire: If one actor fires at a much higher rate than another, the actors´ memory buffers may overflow, causing workflow execution to fail [8]. In addition the PN Director does not manage iterations, possibly leading to non-determinism and undefined termination of the execution. For instance in composite actors with workflow hierarchies, if two actors have computation threads, it is ambiguous which actor should be allowed to perform computation [20]. Instead, AWARD activities are encapsulated in parallel processes (AWA) and can execute in distributed environments without a centralized control. Each AWA activity has autonomic control and communicates through a shared tuple space by producing/consuming tokens at different rates without overflow problems. AWARD has the notion of iterations allowing determinism and well defined termination of the AWA activities. Using tuples for communication improves flexibility, e.g. supporting different granularities of complex data types and dynamic workflow changes are easily enabled by injecting reconfiguration tuples into the shared space. Although with different objectives other works also rely on Tuple Spaces. For example the Workflow Enactment Engine (WFEE) [21] uses a tuple space to provide an event-based notification for just-in-time scheduling. Based in Comet, a decentralized tuple space [22], Rudder [23], [24] provides a software agents framework for dynamic discovery of services, enactment and management of workflows, where an interaction space is used to coordinate task scheduling among a set of workers. Similar to AWARD space the task tuples contain data items among workflow nodes. As the AWARD model is orthogonal to tuple space implementation, we argue that it is possible to map the AWARD Space to the Comet space.

Closer to AWARD in [12] a framework is based on persistent queues to support flow between activities, but the workflow engine is monolithic. Like in Kepler tasks are executed as threads, not allowing the execution of workflow activities on distributed infrastructures. Additionally, authors in [12] claim that persistent queues can easily support provenance storage. This claim also applies to AWARD, as far as tuple spaces, including IBM TSpaces, also support persistence. In [25] a workflow system is presented with a decentralized architecture with an external storage (Multiset) as a shared space between workflow activities encapsulating a Chemical engine. Unlike AWARD Space, the Multiset contains coordination information and the workflow definition. Although the goal to support flexibility in business workflows has been a concern since the nineties [13], [14], many issues still remain open, regarding the support for dynamic changes [15]. Such issues are also important in scientific workflows, and some are supported by AWARD. Namely, supporting dynamic behavioral changes, e.g. changing the execution Task and its parameters at runtime, are important in scientific experiments where the behavior of algorithms and their parameters are not known in advance. A Kepler workflow [8] is static and must be completely specified before starting the execution, not allowing changes during run-time. However a prototype implementation based on Kepler [16], proposes a frame abstraction as a placeholder for actors to be instantiated at runtime (dynamic embedding), according to rules defined at design time. This approach can be compared to our proposal to change dynamically the algorithm of an activity. However AWARD is more flexible because we do not need to specify the alternative algorithms at design time as we allow to dynamically changing them by injecting tuples with the new Tasks. In the GridBus workflow engine [17] the user can either specify the location of a particular service at design time, or leave it open until the enactment engine identifies service providers at run-time. This is easily supported in AWARD: If the location of the service (URL) is a parameter we can dynamically reconfigure the parameter by changing the service provider, or we can inject rules into the control unit of an AWA activity in order to search for service providers in any service directory.

V. AWARD EVALUATION

AWARD allows executing workflows on heterogeneous environments. For instance we executed workflows in local area networks involving Windows and Linux computers. We describe our experiments on Amazon's EC2 infrastructure [26]. The mapping of AWARD components (AWA and AWARD Space) onto the Amazon EC2 infrastructure is presented in **Fig. 6**.

There is one dedicated EC2 instance to host the AWARD Space server, accessed from other EC2 instances through the TCP protocol. The AWARD Space server can be monitored and controlled from anywhere outside the Amazon cloud using the HTTP protocol and using Web interfaces to allow an end-user to follow the execution of the workflows.

End-users access the EC2 instances using any SSH client (we used PuTTY) and all data files (xml workflow definition files, input/output data files) are transferred to the shared storage using any ftp client (we used WinSCP). The shared storage is an EC2 volume mounted in all EC2 instances.

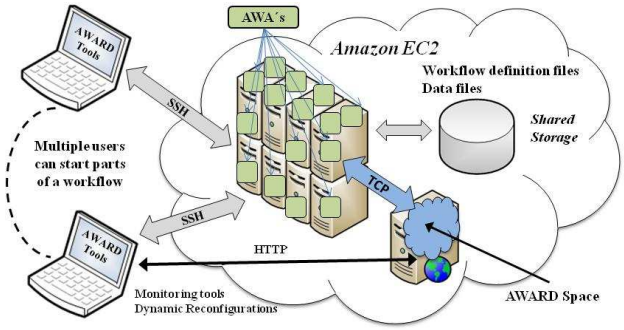
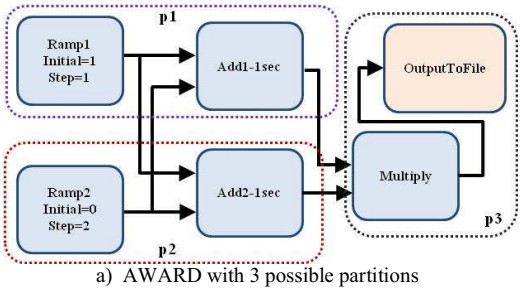


Fig. 6. Mappings to execute AWARD workflows on Amazon EC2

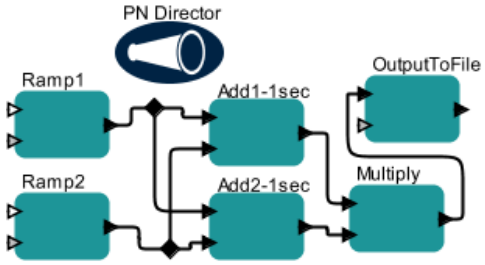
All EC2 instances used have the following characteristics: Amazon Linux AMI 64 bit, 4 CPU (2 virtual cores with 2 EC2 Compute Units each), 7.5GB memory, 8 GB root device, and a 8 GB shared volume. One EC2 Compute Unit provides the equivalent CPU capacity of a 1.0-1.2 GHz 2007 Opteron or 2007 Xeon processor [26]. To evaluate AWARD we defined a set of scenarios in order to answer the following questions: 1) What are the overheads for executing an AWARD workflow comparing with the execution of a similar workflow in Kepler, including dynamic reconfigurations to enable optimizations during the execution of long-term workflows (hundred or thousand iterations); and 2) What is the flexibility of AWARD: i) concerning parallelism in workflows with a large number of activities to be launched and controlled by different scientists without a centralized control in distributed environments e.g. as a Cloud; ii) concerning easily changing the mappings from the workflow activities to the virtual machines without any modification to the workflow specification.

First scenario: This involves the execution of a similar workflow in AWARD and Kepler as presented in **Fig. 7**. The two *Ramp* activities generate a sequence of numbers in parallel, until a maximum number of iterations. These numbers are added in parallel and the results are multiplied. The Output activity writes the results to a file. The AWARD *Add1* and *Add2* activities have a *Task* with a sleep time of 1 second to simulate a long execution time. To achieve the same behavior in Kepler we customized the necessary Actors, in order to have the same sleep time, to manage the workflow iterations, and to report the execution time per iteration in a similar way as AWARD. In AWARD we log the execution times into the AWARD Space and in Kepler we log them into a file.

In Kepler we used a single AWS EC2 instance, and considered three cases for AWARD: In case 1 we ran the workflow using a single EC2 instance, including the AWARD Space server. In case 2, two EC2 instances, one to host the AWARD Space server and the other to execute the workflow; In case 3, four EC2 instances, one to host the AWARD Space server and three EC2 instances to distribute the activities according to the workflow partitions p1, p2, p3 as shown in **Fig. 7a)**.



a) AWARD with 3 possible partitions



b) Kepler using customized Actors

Fig. 7. An equivalent workflow in AWARD and Kepler

The average execution time per iteration is shown in **Fig. 8** for executions with 1, 10, 50, 100, 500 and 1000 iterations. These results confirm that AWARD has some overheads compared to Kepler, which uses a dedicated thread for each Actor inside a monolithic process, whereas AWARD uses an independent process for each AWA. However for long-term workflows (with thousands of iterations) as shown in **Fig. 8**, AWARD has small overheads compared to Kepler allowing us to conclude that AWARD becomes adequate to execute this class of workflows.

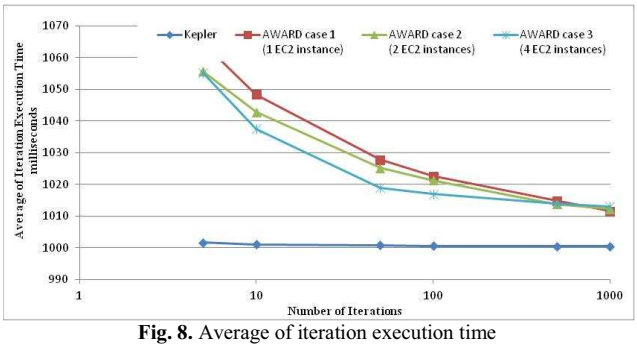


Fig. 8. Average of iteration execution time

One should note that the execution time gets smaller for executions on multiple EC2 instances, even for lower numbers of iterations, although the execution time reduction is not so strong when the iterations increase up to the thousands. This is explained by the cost due to tuples persistence and tuple matching when the size of the tuple space grows. This simple workflow also allows us to illustrate the dynamic reconfiguration capabilities related to performance optimizations. As shown in **Fig. 8** the iteration

execution time is always greater than 1 second due to the delay forced in the two *Add* activities. Unlike Kepler, in long-term workflows AWARD allows to change parameters or the Task (algorithm) of any activity. As an example we can change the Task of *Add1-1sec* and *Add2-1sec* activities to new algorithms with optimized performance.

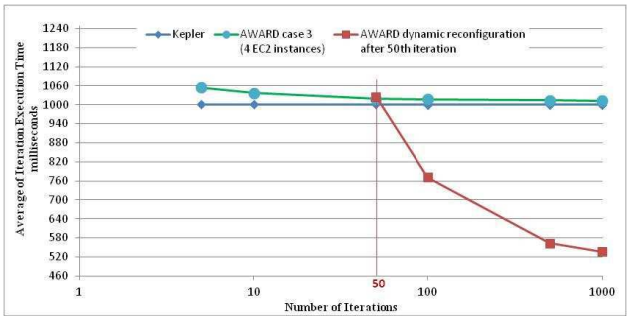


Fig. 9. Dynamic reconfiguration to reduce execution times

Fig. 9, illustrates the result of workflow execution when we dynamically change the two *Add* tasks with a new algorithm that is optimized for a delay of only 0.5 seconds after the 50th iteration. As shown, the average iteration execution time decreases after the 50th iteration leading to a reduction of the overall execution time. This is easily achieved by a simple command that injects the operator *ChangeTask* as a tuple in the AWARD Space. This tuple is handled by the *Dynamic Reconfiguration Handler* of the autonomic controller of the *Add* AWAs. Although this example is minimal and very simple it clearly demonstrates the AWARD capabilities to perform dynamic reconfigurations.

Second scenario: This uses a simulation of a Montage workflow as in [3] and [27]. For our experiments we used the workflow in **Fig. 10** with 24 activities and the execution time of each activity based in [3]. This shows how AWARD supports scenarios where multiple users can execute a large workflow interactively executing their specialized activities.

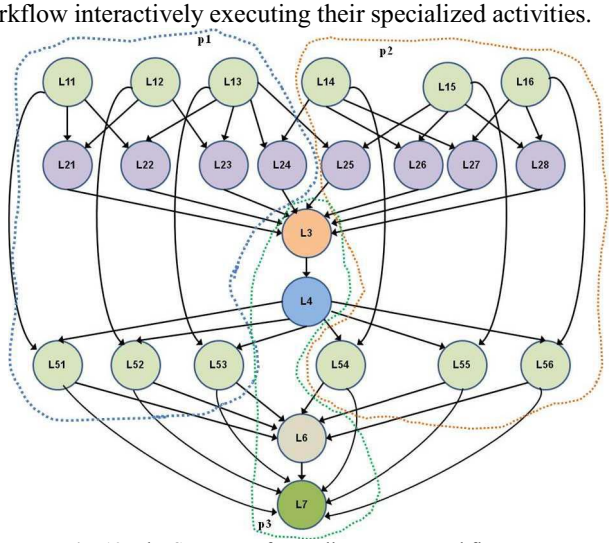


Fig. 10. The Structure of a small Montage Workflow

We experimented several ways to launch this workflow: To launch all activities in the same EC2 instance or launch different partitions by different users on multiple EC2 instances. **Fig. 10** shows three possible partitions (p1, p2, p3) launched by 3 users in 3 different EC2 instances concluding on the AWARD flexibility to execute workflows with large number of activities that can be separately launched. These partitions are explicitly defined by the user according to a decomposition strategy depending on the application knowledge.

Furthermore in the second scenario: We experimented with a Text Mining application to extract relevant expressions (RE) from text documents. Such expressions, called n-grams, where n is the number of words, are sequences of words with strong semantic meaning, such as "parallel processing", "global finance crisis". The *LocalMaxs* algorithm supports a language independent approach for extracting RE [28]. Apart its good precision, it has a low performance in terms of execution time, when executed sequentially, so a parallel implementation of this application has been developed using distributed tasks. For a problem instance with n-grams where n=6 the algorithm is modeled as the workflow presented in **Fig. 11**. The partitions shown (p6-1…p6-6) illustrate parts of the workflow that can be executed separately. These partitions are also defined by the Text mining application developer.

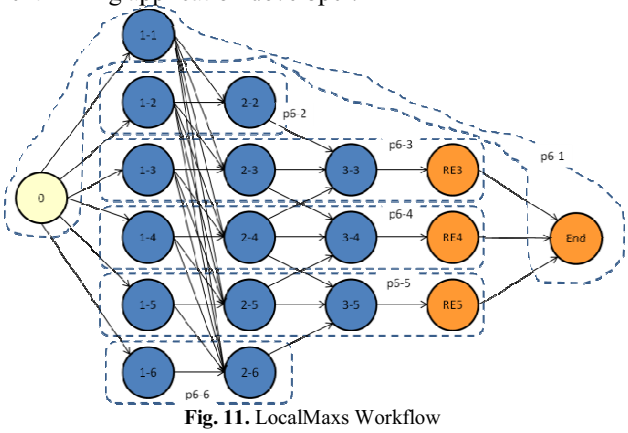
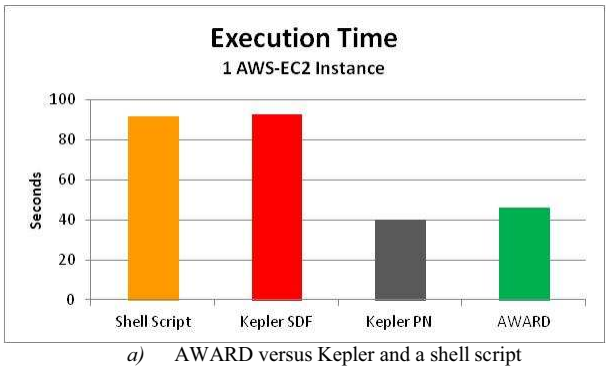


Fig. 11. LocalMaxs Workflow

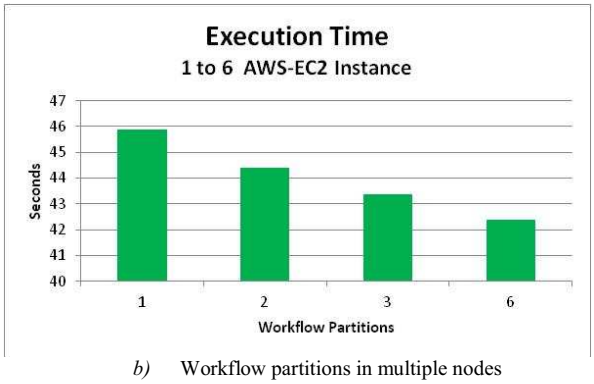
Note that the algorithm tasks that are encapsulated in the AWA activities were developed by other users previously in the C++ language, without any concern to the AWARD framework, and their integration into this workflow was performed by someone who is not a AWARD developer. As part of ongoing work we are developing a large scale workflow for this text mining case study using AWARD. This example is only briefly presented here to demonstrate the AWARD capabilities for real application scenarios.

Fig. 12a) presents the execution times of this workflow using different approaches, when considering a single EC2 instance: i) using a shell script to sequentially start the workflow activities as processes; ii) using Kepler, respectively, under SDF and PN Directors; iii) using AWARD.

Fig. 12b) shows the execution times of the AWARD workflow using 2, 3 and 6 partitions with one EC2 instance for each partition. Note that this scenario is not possible when we use Kepler or shell scripts.



a) AWARD versus Kepler and a shell script



b) Workflow partitions in multiple nodes

Fig. 12. Text Mining workflow execution time on AWS EC2

The conclusions taken from the above experiments are threefold: i) AWARD is clearly better than the sequential executions (shell script and Kepler SDF Director); ii) AWARD reveals a not significant overhead comparing to the Kepler PN Director; iii) The advantages of AWARD to execute workflow partitions in parallel using multiple computing nodes are clearly illustrated.

VI. Conclusions and Future Work

This paper presents an evaluation of the AWARD framework focusing on its benefits: i) Autonomic workflow activities with decentralized control that can run in distributed architectures; ii) A shared Tuple Space for data-driven communication and coordination; iii) Development of application algorithms, easily integrated as workflow tasks, without knowledge on the details of the enactment engine; iv) Dynamic reconfigurations of long-term workflows with multiple iterations. AWARD is supported by a working prototype deployed on a single machine as well as on a distributed infrastructure, as the Amazon EC2 Cloud. All AWARD tools are running in EC2 instances without any modification. Regarding i) we conclude that AWARD eases the experimentation with different partitions of the workflow activities and their mappings to the cloud environment. In addition, workflows with large number of iterations do not incur significant overheads. Regarding ii) we conclude that the AWARD Space is flexible to support data-driven coordination between activities. Furthermore its benefits include the possibility to logging runtime information that allows monitoring and managing workflow intermediary results. Regarding iii) we conclude from the Text Mining workflow that it is easy for other users to specify AWARD workflows and develop Tasks, involving the use of legacy code, without any knowledge of AWA internal functionality. Regarding iv) we illustrate the AWARD advantages to support dynamic reconfigurations. In this paper due to space limitations we could not cover this aspect in depth. In ongoing work we are using AWARD to experiment with workflow scenarios that can benefit from using AWARD operators to dynamically change the workflow structure and behavior, namely to control quality of service (QoS), load balancing or data filtering.

Acknowledgements. Thanks are due to PEst-OE/EEI/UI0527/2011, CITI/FCT/UNL-2011-2012.

References

[1] I. J. Taylor, E. Deelman, D. B. Gannon, and M. Shields. *Workflows for e-Science: Scientific Workflows for Grids*. London,: Springer, 2007.

[2] J. Yu and R. Buyya, "A Taxonomy of Workflow Management Systems for Grid Computing," *Journal of Grid Computing*, vol. 3, pp. 171–200, 2005.

[3] E. Deelman, G. Singh, M.-H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. B. Berriman, J. Good, A. Laity, J. C. Jacob, and D. S. Katz, "Pegasus: A framework for mapping complex scientific workflows onto distributed systems," *Scientific Programming*, vol. 13, pp. 219–237, July 2005.

[4] G. C. Jr., C. Sivaramakrishnan, T. Critchlow, K. Schuchardt, and A. H. Ngu, "Scientist-Centered Workflow Abstractions via Generic Actors, Workflow Templates, and Context-Awareness for Groundwater Modeling and Analysis," *IEEE Congress on Services*, pp. 176–183, 2011.

[5] G. Lazweski, [Accessed, (2011, October)]. [Online]. Available: http://wiki.cogkit.org/index.php/Scientific_Workflow_Survey

[6] Triana, [Accessed, (2011, October)]. [Online]. Available: http://www.trianacode.org/

[7] Taverna, Taverna web site, [Accessed, (2011, October)]. [Online]. Available: http://www.taverna.org.uk/

[8] Kepler project, *Kepler User Manual*, August 2011. [Online]. Available: https://kepler-project.org/

[9] L. Assunção, C. Gonçalves, and J. C. Cunha, "On the Difficulties of Using Workflow Tools to Express Parallelism and Distribution – A Case Study in Geological Sciences," Proceedings of the International Workshop on Workflow Management of the International Conference on Grid and Pervasive Computing (GPC2009). IEEE Computer Society, 2009, pp. 104–110.

[10] G. Kahn, "The semantics of a simple language for parallel programming," in *Information processing*, J. L. Rosenfeld, Ed., 1974, pp. 471–475.

[11] T. M. Parks, "Bounded Scheduling of Process Networks," Ph.D. dissertation, University of California at Berkeley, 1995.

[12] M. Agun and S. Bowers, "Approaches for Implementing Persistent Queues within Data-Intensive Scientific Workflows," in *IEEE World Congress on Services*, 2011, pp. 200 –207.

[13] P. Dadam and M. Reichert, "The ADEPT project: a decade of research and development for robust and flexible process support," *Computer Science*, vol. 23, pp. 81–97, 2009.

[14] J. Halliday, S. Shrivastava, and S. Wheater, "Flexible workflow management in the OPENflow system," in *Proceedings of Fifth IEEE International Enterprise Distributed Object Computing Conference EDOC '01*, 2001, pp. 82 –92.

[15] T. Burkhart and P. Loos, "Flexible business processes - evaluation of current approaches," in *Proceedings of MKWI 2010*, 2010, pp. 1217–1228.

[16] A. H. Ngu, S. Bowers, N. Haasch, T. Mcphillips, and T. Critchlow, "Flexible Scientific Workflow Modeling Using Frames, Templates, and Dynamic Embedding," in *SSDBM '08: Proceedings of the 20th international conference on Scientific and Statistical Database Management*. Springer-Verlag, 2008, pp. 566–572.

[17] J. Yu and R. Buyya, *Gridbus Workflow Enactment Engine*. CRC Press, 2010, ch. 5, pp. 119–146.

[18] E. Friedman-Hill, [Accessed, (2010, March)]. [Online]. Available: http://www.jessrules.com/jess/

[19] T. J. Lehman, A. Cozzi, Y. Xiong, J. Gottschalk, V. Vasudevan, S. Landis, P. Davis, B. Khavar, and P. Bowman, "Hitting the distributed computing sweet spot with TSpaces," *Computer Networks*, vol. 35, no. 4, pp. 457–472, 2001.

[20] A. Goderis, C. Brooks, I. Altintas, E. A. Lee, and C. Goble, "Heterogeneous composition of models of computation," *Future G. Comput. Syst.*, vol. 25, pp. 552–560, 2009.

[21] J. Yu and R. Buyya, "A Novel Architecture for Realizing Grid Workflow using Tuple Spaces," in *GRID '04: Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*. IEEE Computer Society, 2004, pp. 119–128.

[22] Z. Li and M. Parashar, "Comet: A scalable coordination space for decentralized distributed environments." Proceedings of the 2nd International Workshop on Hot Topics in Peer-to-Peer Systems (HOT-P2P 2005, 2005, pp. 104–112.

[23] ——, "An infrastructure for dynamic composition of grid services," in *Proceedings of the 7th IEEE/ACM International Conference on Grid Computing*, ser. GRID '06. Washington, DC, USA: IEEE Computer Society, 2006, pp. 315–316. [Online]. Available: http://dx.doi.org/10.1109/ICGRID.2006.311036

[24] ——, "An Infrastructure for Dynamic Composition of Grid Services," Rutgers University, Tech. Rep., 2007.

[25] H. Fernandez, C. Tedeschi, and T. Priol, "A Chemistry-Inspired Workflow Management System for Scientific Applications in Clouds," in *IEEE 7th International Conference on E-Science*, 2011, pp. 39 –46.

[26] Amazon.com, [Accessed, (2012, January)]. [Online]. Available: http://aws.amazon.com/ec2/

[27] G. Juve, E. Deelman, K. Vahi, G. Mehta, B. Berriman, B. P. Berman, and P. Maechling, "Data Sharing Options for Scientific Workflows on Amazon EC2," in *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, 2010, pp. 1–9.

[28] J. F. d. Silva, G. Dias, S. Guilloré, and J. G. P. Lopes, "Using LocalMaxs Algorithm for the Extraction of Contiguous and Non-contiguous Multiword Lexical Units," in *Proceedings of the 9th Portuguese Conference on Artificial Intelligence: Progress in Artificial Intelligence*, ser. EPIA '99. Springer-Verlag, 1999, pp. 113–132.

.