Pattern-based Decentralization and Run-time Adaptation Framework for Multi-site Workflow Orchestrations

Selim Kalayci, S. Masoud Sadjadi
School of Computing and Information Sciences
Florida International University
Miami, FL, USA
{skala001, sadjadi}@cs.fiu.edu

***Abstract*—Scientific applications keep getting more complex, resulting in the need for more computational resources than may be available to scientists locally. As a result, for many scientists utilization of remote and heterogeneous computational resources has become a standard practice. However, effective utilization of these resources, especially for large-scale workflow applications, necessitates the employment of software tools that are efficient and adaptive. In this study, we propose a generic framework for the decentralization and run-time adaptation for the execution of large-scale workflow applications that span across diverse and heterogeneous resource domains. By exploiting the recurring DAG patterns, we come up with corresponding decentralization and adaptation patterns to be employed by peer workflow orchestration tools to cope with various resource-based challenges in the execution environment. Our framework adopts separation of concerns and consequently does not alter the business logic of the application. We provide a prototype implementation of our framework on a standard workflow orchestration tool. But, our framework is generic enough and can be easily incorporated by other orchestration tools.**

Keywords: *workflow, DAG, orchestration, adaptation, pattern*

I. INTRODUCTION

Scientific workflows are abstractions that capture the business logic of many complex applications in different scientific disciplines. More specifically, these workflows encapsulate and represent all of the various tasks and data artifacts associated with the application lifecycle. Regardless of the size and complexity of the specific scientific workflow, Directed-Acyclic-Graphs (DAGs) are a powerful and well-established method used by many scientists/developers.

Lifecycle of a typical scientific workflow begins with the specification of individual tasks and artifacts, and dependencies among them. This specification is free of some concrete run-time specific details, and it is mostly referred as the abstract workflow. Abstract workflows more often than not do not address run-time specific details, such as the exact names and locations associated with data artifacts and details about the exact mapping of tasks on physical resources. For such an abstract workflow to run successfully to completion, those details need to be determined prior to or during the execution of the workflow. This process can be referred to as the concretization of the workflow. However, the concretization of the workflow should not be the responsibility of the scientist, or even the application developer; as a matter of fact, it has to be automated as much as possible based on the criteria provided by the user.

During the concretization process, one essential step is the mapping of workflow tasks onto physical resources. The main goal here is to achieve the minimum makespan possible for the execution of the whole workflow. As such, characteristics of tasks (e.g. estimated runtime) and data artifacts (e.g. estimated size), as well as the availability and characteristics of physical resources play a major role during this mapping process. Based on the availability of resources, the resulting concrete workflow may span across multiple sites of resources. Such multi-site resources may be made available to the usage of a specific workflow application perhaps through research collaboration among multiple partners or through a national/international cyberinfrastructure platform (e.g. XSEDE). The key common attributes of such multi-site resources is the heterogeneity and dynamicity of the resources in terms of size and capability, as well as heterogeneous access and priority rights assigned to the users by local administrators. All these factors pose many challenges to the successful execution of workflows conforming to the makespan requirements of the user.

In this paper, we propose a generic framework that utilizes the common DAG patterns to alleviate some of the problems stemming from the multi-site execution of workflows. First of all, taking advantage of DAG patterns, we transform a concrete workflow in a manner that makes it possible to be executed in a peer-to-peer fashion. This transformation has the potential to improve the efficiency of the execution of the workflow by having local interactions to be managed by local workflow execution managers rather than having a single central workflow execution manager orchestrating the whole workflow. Also, employment of local managers improves the accuracy and efficiency involved within the decision-making and adaptation process to the changes in the execution environment at run-time. Second major contribution of this paper is to propose a generic framework for the adaptation of the workflow execution in response to the dynamic changes at run-time. This adaptation framework also utilizes common

DAG patterns and suits well with the peer-to-peer execution framework devised in the previous step.

A very important aspect of our proposed framework is the separation of concerns. The modifications and adaptations to the execution logic of the workflow due to our approach, do not affect the business logic of the workflow. We showcase the validity and generality of our approach via a prototype implementation on a standard workflow execution manager. Some of the implementation details are also discussed briefly.

To illustrate our ideas, first we introduce the concept of DAG patterns in Section 2. In Section 3, we introduce our decentralization framework utilizing DAG transformation patterns. In Section 4, we discuss the need for run-time adaptation and explain how we incorporate this behavior into our decentralized execution environment. Section 5 illustrates and discusses prototype implementation issues. In Section 6 we overview related literature and Section 7 provides a summary of the paper and a brief discussion about further issues.

II. DAG PATTERNS

In this section, we introduce the concept of recurring DAG patterns that form the basic building blocks for the steps that follow. The key point here is that all possible scientific workflows in DAG form can be represented using a proper combination of these DAG patterns.

Fig. 1 illustrates three DAG patterns, namely: Sequence pattern, Fork/Branch pattern and Join pattern. In these graphs, vertices correspond to the computational tasks, whereas directed edges correspond to the control and/or data dependencies between tasks.

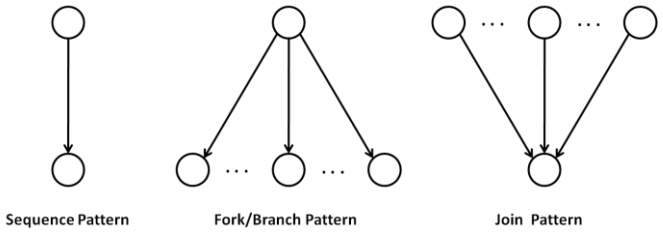

Figure 1. DAG patterns

III. DECENTRALIZATION

Regardless of being in an abstract or concrete form, a scientific workflow is usually crafted and enacted at a single central location. This means, except for a few proprietary solutions, the execution logic of the workflow is handled by a single, central workflow execution manager. This central manager keeps track of the progress of tasks and coordinates the timely execution of each task based on the specifications of the workflow.

The central workflow manager handles the execution of the workflow even if the mapped tasks of the workflow span across multiple sites. Especially in such a scenario, employment of a single central workflow execution manager raises several efficiency and decision-making accuracy issues. First of all, employing a single central manager necessitates each individual activity to be monitored and orchestrated by this central manager. For a large-scale workflow (i.e. comprised of a large number of tasks) that is mapped on multiple and potentially long-distance sites, orchestration efficiency becomes a real issue. Another important issue is the level of information sharing among partnering sites. The more detailed resource and workload information is shared among partners, the better decisions can be made by the workflow execution manager(s) to take actions in response to changing conditions. However, due to administrative and technical reasons (e.g. size of information, delay); it is not possible for a remote workflow execution manager to have the same level of information about a certain site resources compared to its local counterpart. This can cause the central workflow execution manager to make non-optimal decisions during run-time adaptation.

To overcome these issues, our proposal is to transfer the responsibility of orchestration of the whole workflow from a single workflow execution manager to several collaborating workflow execution managers. According to this, each local workflow execution manager is going to be responsible for the orchestration of the tasks that are mapped locally. At the same time, peer local workflow execution managers synchronize among each other when necessary, specifically, to fulfill the requirements of those control/data dependencies that span across them. By collaboratively carrying out these activities, the orchestration of the whole workflow is achieved without affecting the business logic of the workflow. Through this peer-to-peer orchestration approach, we are able to provide (i) improved efficiency for large-scale workflow executions, (ii) better results from run-time adaptation.

We propose a systematic and generic framework for transforming the centralized orchestration to a collaborative orchestration via the utilization of DAG transformation patterns. These patterns are built on basic DAG patterns introduced in Section 2, and illustrate the transformations on the original DAG specifications to meet the needs of the collaborative orchestration style. Each local workflow execution manager individually performs these transformations appropriate to its circumstances.

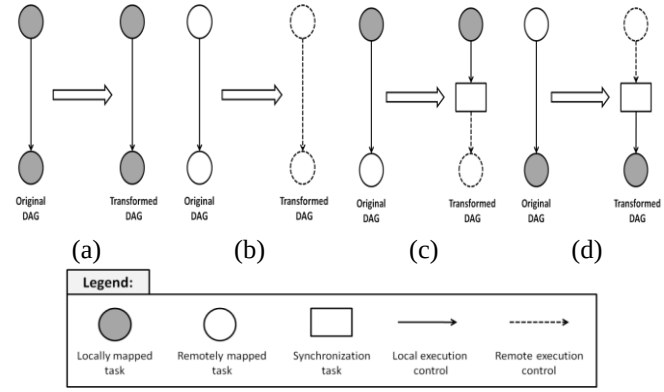

Figure 2. DAG Transformations for the Sequence DAG Pattern

Fig. 2 illustrates the set of DAG transformations on the Sequence DAG pattern corresponding to four possible mapping scenarios. If both tasks comprising the Sequence DAG pattern are mapped locally, as in Fig. 2(a), then no transformation is

necessary. If both tasks comprising the Sequence DAG pattern are mapped remotely, as in Fig. 2(b), then these tasks are marked to indicate that they will be orchestrated by another manager. Fig. 2(c) illustrates the case where the parent task is mapped locally, whereas the child task is mapped remotely. In this case, the transformation incorporates a synchronization task between these tasks. The purpose of this synchronization task is to basically inform the remote workflow execution manager of the completion of the parent task. Fig. 2(d) illustrates the case where the parent task is mapped remotely, and the child task is mapped locally. In this case, the transformation again incorporates a synchronization task between these tasks. However, in this case the local workflow execution manager waits to be informed by its remote partner about the completion of the parent task.

Fig. 3 illustrates the DAG transformations on the Fork/Branch DAG pattern corresponding to two possible mapping scenarios. We skip the two other possible mapping scenarios, in which all the set of tasks are either mapped locally or remotely, as the transformations will be very limited and done similar to the ones in Fig. 2(a) and Fig. 2(b). Fig. 3(a) illustrates the case where the parent task is mapped locally, whereas the children tasks are mapped remotely. In this case, the transformation incorporates a synchronization task after the parent task. This synchronization task will be informing the remote workflow execution manager(s) of the completion of the parent task. Fig. 3(b) is similar to the previous scenario, but this time parent task is mapped remotely and children tasks are mapped locally. This time, the local workflow execution manager is at the receiving side of the synchronization operation, so it has to wait for its remote partner to perform the associated synchronization task.

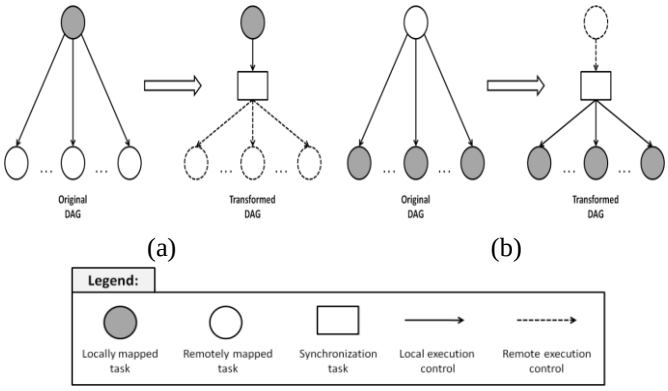

Figure 3. DAG Transformations for the Fork/Branch DAG Pattern

Fig. 4 illustrates the DAG transformations on the Join DAG pattern corresponding to two possible mapping scenarios. As in Fig. 3, we skip the two other possible mapping scenarios. Fig. 4(a) illustrates the case where the parent tasks are mapped locally, and the child task is mapped remotely. In this case, the transformation incorporates a single synchronization task before the child task to inform the completion of the parent tasks. According to the scenario in Fig. 4(b), parent tasks are mapped remotely and the child task is mapped locally. This time, the local workflow execution manager is at the receiving side of the synchronization operation, so it has to wait for its remote partner(s) to perform the associated synchronization task.

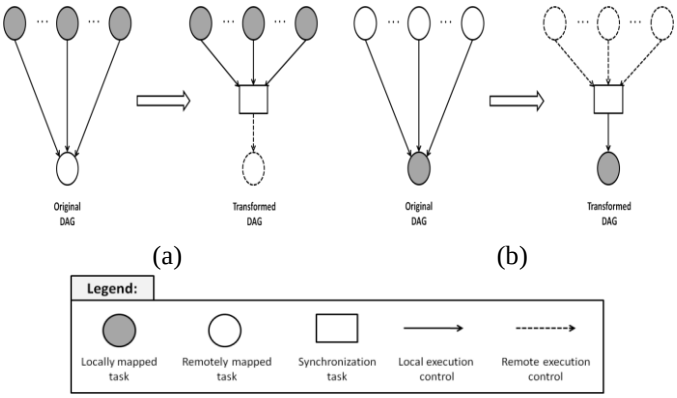

Figure 4. DAG Transformations for the Join DAG Pattern

One thing to note for the transformation of Fork/Branch and Join DAG patterns is that we incorporate a single synchronization task among the parent and children tasks, regardless of the number of parent and children tasks in the original DAG. This design choice significantly reduces the number of synchronization activities (hence overhead) among peers, and also simplifies the transformation process.

IV. Run-time Adaptation

Due to the dynamic nature of the execution environment, certain changes may need to be made to the original workflow execution plan at run-time to meet users' QoS requirements. The most common and obvious dynamic change in the execution environment is the availability of hardware resources for the utilization of workflow tasks. The availability of these resources may change basically due to hardware failures, increased workload, and higher priority tasks being deployed in the system. Especially long-running and large-scale workflows are highly susceptible to this kind of changes in the execution environment. Under these circumstances, to be able to execute a workflow successfully within QoS requirements, proper changes have to be made to the original execution plan.

There are two main issues involved within the run-time adaptation process. First major issue is the planning phase for the run-time adaptation. This phase includes the continuous monitoring of workflow progress and resources, and detecting a situation that necessitates the run-time adaptation process. After the detection, an appropriate corrective action has to be planned to cope with the situation. Second major issue in the run-time adaptation process is the enactment of the proposed adaptation plan to the ongoing workflow execution process in an efficient and non-intrusive manner. In this paper, we focus on the second aspect of the run-time adaptation process.

A standard run-time adaptation plan basically makes changes to the original execution plan by modifying the mapping (hence, the execution) site of tasks. Modifications to the mapping site of tasks needs to be reflected and implemented accordingly by the workflow execution manager(s). Here, we provide a run-time adaptation framework that integrates with the decentralization framework explained

in the Section 3. One key aspect of our framework is the low-level of intrusiveness to carry out the adaptation process. Our pattern-based framework has little effect on the ongoing workflow execution process. The peer workflow execution managers implement the re-mapping of tasks without any disruption to the orchestration of the rest of the whole workflow.

We utilize DAG adaptation patterns at peer workflow execution managers to implement the re-mapping of tasks at run-time. Through this adaptation, the responsibility of orchestration of the set of tasks being re-mapped (S_T) is transferred from the originating site to the destination site(s).

At the originating site, DAG adaptation patterns transforms S_T from being local tasks to remote tasks. Also, if there is a synchronization task between S_T and child(ren) task(s), it is removed. Fig. 5(a) illustrates this scenario for the Sequence pattern, Fig. 6(a) illustrates the same scenario for the Fork/Branch pattern, and Fig. 7(a) illustrates the same scenario for the Join pattern. But, if the child(ren) task(s) of S_T is (are) mapped locally, then a synchronization task is incorporated after S_T. Fig. 5(b) illustrates this scenario for the Sequence pattern, Fig. 6(b) illustrates the same scenario for the Fork/Branch pattern, and Fig. 7(b) illustrates the same scenario for the Join pattern.

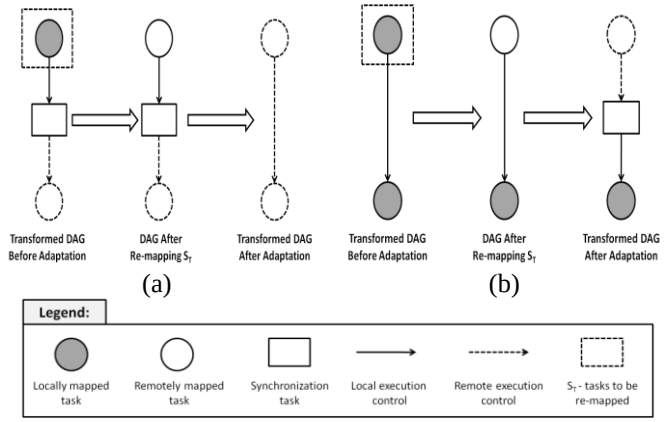

Figure 5. DAG Adaptations for the Sequence DAG Pattern

The destination site(s) basically captures the re-mapped tasks (S_T) and orchestrates them in accordance with the resulting DAG structure. However, attempting to capture and integrate this DAG structure with the transformed DAG specification at destination site(s) proves to be cumbersome to design and implement. For this reason, we propose the orchestration of these DAG structures in isolation from the transformed DAG structure(s) at destination site(s). In fact, we refer to these DAG structures that result from run-time adaptation processes and orchestrated in isolation as "patch DAGs". The combined orchestration of patch DAGs with the transformed DAGs provide the same business logic as the orchestration of the whole original DAG.

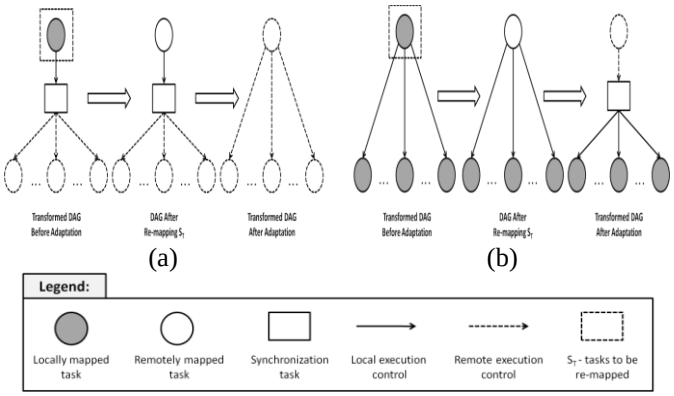

Figure 6. DAG Adaptations for the Fork/Branch DAG Pattern

The essential duty of a destination site involved in the run-time adaptation process is to capture S_T and compose the corresponding patch DAG. Once the patch DAG is ready, destination site orchestrates the patch DAG in isolation.

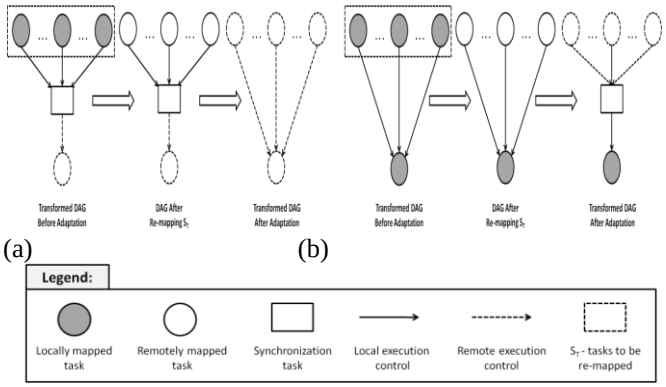

Figure 7. DAG Adaptations for the Join DAG Pattern

Fig. 8 illustrates the corresponding patch DAGs for the adaptation of Sequence patterns in Fig. 5. In both Fig. 8(a) and Fig. 8(b), the corresponding patch DAG has the same structure. The difference between them is the inner-workings of the synchronization task. In Fig. 8(b), the synchronization task synchronizes back with the originating site for the execution of child task. On the other hand, synchronization task in Fig. 8(a) synchronizes with a third site other than the originating site. In fact, this third site is the one that is responsible for the execution of the child task, and it is agnostic to the changes made during the adaptation.

Patch DAGs corresponding to the adaptation of Fork/Branch patterns look and operate the same way as in the adapted Sequence patterns. The only difference between these two patterns is the number of children tasks the patch DAG synchronization task enables for progression.

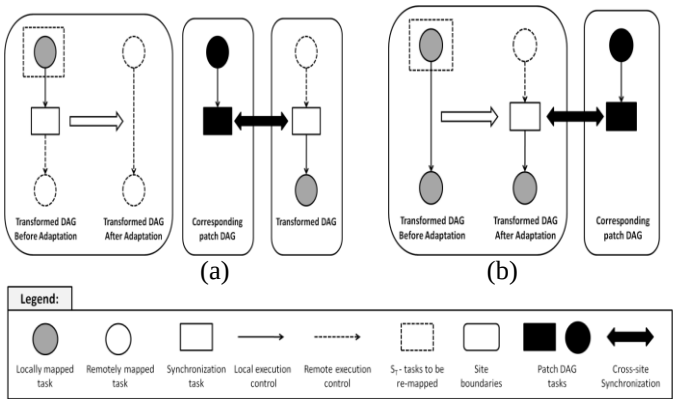

Figure 8. Patch DAG Patterns corresponding to the Adapted Sequence Patterns

The corresponding patch DAG structures to the adaptation of Join Pattern in Fig. 7(a) and Fig. 7(b) are also identical. As in the Sequence pattern, the difference between them is the inner-workings of the synchronization task. However, in a Join pattern, it is possible for more than one site to be involved in the re-mapping of S_T, due to the multiplicity of the number of tasks in S_T. To illustrate this point, we provide two alternative scenarios for patch DAG composition and operation corresponding to the case in Fig. 7(b). Fig. 9 illustrates the corresponding patch DAG structure in a scenario where only one site gets to re-map all the tasks in S_T. For this scenario, the composition and the operation of the patch DAG is quite similar to the previous patterns.

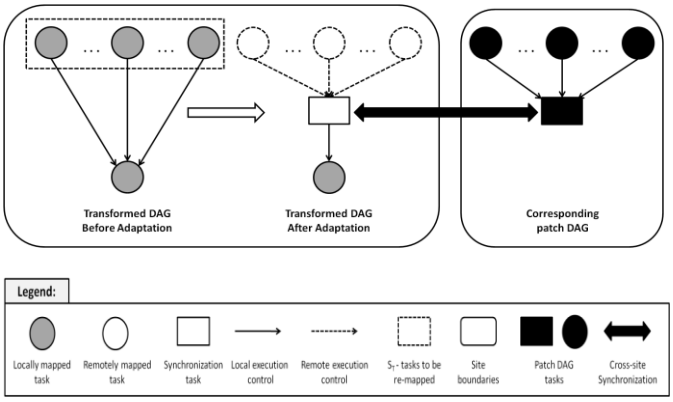

Figure 9. Patch DAG Pattern corresponding to the Adapted Join Pattern in a scenario where a single site re-maps S_T

Fig. 10 illustrates the corresponding patch DAG structure in a scenario where two destination sites are involved in re-mapping of the tasks in S_T. In this scenario, two patch DAGs need to composed, and one of those DAGs is designated as the primary patch DAG. Primary patch DAG is responsible for handling the synchronization with the originating site. Also, in this scenario, an additional layer of synchronization is needed between the destination sites.

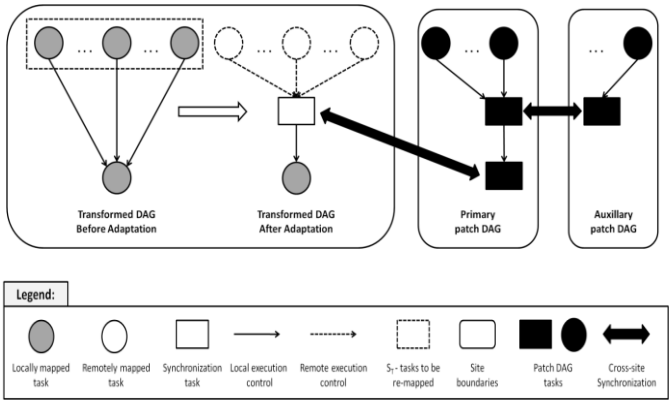

Figure 10. Patch DAG Pattern corresponding to the Adapted Join Pattern in a scenario where two sites re-map the tasks in S_T

V. PROTOTYPE IMPLEMENTATION

Our implementation is based on Condor DAGMan [5] workflow execution engine. Condor DAGMan is a widely used workflow execution tool that is available in most High-Performance and High-Throughput Computing environments. The original orchestration architecture is centralized and it does not provide native run-time adaptation support.

Decentralization

Condor DAGMan specifies a DAG structure by listing the tasks and their dependencies in a standard text file. This specification is parsed and then the execution of tasks is orchestrated by submitting them to local and/or remote resources by Condor DAGMan.

In our decentralized framework, each site has its own deployment of Condor DAGMan tool, and each engine submits computational tasks only to their local resources. Synchronization among peer engines is accomplished via transferring light-weight sync files.

Each local Condor DAGMan tool needs to orchestrate a transformed copy of the original DAG specification. The transformation process is implemented by performing the following basic actions. Remotely mapped tasks are labeled as DONE, so that Condor DAGMan will not attempt to submit them to local resources. DAG transformations illustrated in Fig. 2(c) and Fig. 3(a) is accomplished via insertion of a POST Script accompanying the parent task. DAG transformations illustrated in Fig. 2(d) and Fig. 3(b) is accomplished via insertion of a PRE Script accompanying the child(ren) task(s). DAG Transformations for the Join Patterns illustrated in Fig. 4 is accomplished via incorporation of a light-weight sync task in the original DAG specification.

Run-time Adaptation

As a fault-tolerance feature, Condor DAGMan provides a checkpoint-style recovery mechanism via the creation of rescue DAGs. The rescue DAG represents the specification of a DAG that was not run to completion, but specifies the completed tasks as DONE so that they will not need to be re-run. At the originating site, we utilize the rescue DAG mechanism to halt the DAG specification prior to the adaptation and then to re-enact the corresponding DAG specification after adaptation. To

generate the proper corresponding DAG specification, adaptation decisions should be reflected on the standard rescue DAG specification. This is achieved via labeling re-mapped tasks as DONE and also performing similar DAG transformation activities as was done for Decentralization purposes.

At the destination site, corresponding patch DAG specification needs to be generated. DAG pattern of the task(s) received from the originating site gives enough information to accomplish this. The destination site can then perform the same kind of DAG transformation activities to generate the corresponding patch DAG specification. Following this, destination site parses and starts orchestration of the patch DAG at its local site in isolation.

For more details about the prototype implementation and system design issues, please refer to [13].

VI. Related Work

Pegasus workflow management system [2] uses the Condor DAGMan [5] as the underlying workflow execution engine. Other DAG-based workflow management systems like Taverna [10] and GrADS [11] also employ centralized execution model. ASKALON workflow management system [4] is comprised of a hierarchical architecture where a master engine provides all enactment decisions whereas multiple slave engines perform the orchestration of the workflow under the administration of the master engine.

In the business process orchestration field, several studies [8,9] propose decentralization mechanisms for centralized BPEL process execution engines for scalability and performance improvement. Also, the study in [7] provides a pattern-based analysis of BPEL4WS workflows. Authors provide a comprehensive set of patterns pertaining to all aspects of BPEL4WS specification, whereas in our study the investigation of patterns is limited to the orchestration semantics of DAG-based workflow specifications.

Another pattern based study [12] investigates the usage of workflow patterns to incorporate policy-based fault-tolerant recovery mechanism at run-time.

Several studies [1, 3, 6] deal with run-time workflow adaptation problem through periodically rescheduling the workflow and re-enacting the workflow in correspondence with the new mapping layout. However all these methods necessitate the halting of the whole workflow execution in progress, hence they are highly-intrusive and not scalable for large scale workflows.

VII. Conclusion

In this paper, we propose a generic framework for the decentralization and run-time adaptation of large-scale workflow applications that span multiple sites of resources. Our framework is applicable to any scientific workflow application that is specified in a DAG form. By investigating recurring DAG patterns, we devise corresponding transformation and adaptation patterns to incorporate decentralization and run-time adaptation capabilities to standard workflow execution managers. As our framework does not alter the business logic of a workflow application, it is safe to use and can be evaluated under varying circumstances. We also provide a prototype implementation of our framework on a standard workflow execution manager.

In the future, we would like to investigate the implementation of our framework on other workflow execution managers as well. We are also interested in investigating various business and technical aspects of cloud-bursting on the orchestration and adaptation of workflow applications.

Acknowledgment

This material is based upon work supported by the National Science Foundation under Grant Nos. OISE-0730065 and HRD-0833093.

References

[1] Sakellariou, R. and Zhao, H. 2004. “A low-cost rescheduling policy for efficient mapping of workflows on grid systems.” *Sci. Program.* 12, 4 (Dec. 2004), 253-262.

[2] Deelman, E, et al., "Pegasus: A framework for mapping complex scientific workflows onto distributed systems." s.l. : Scientific Programming, 2005, Issue 3, Vol. 13.

[3] Lee, Kevin, Norman W. Paton, Rizos Sakellariou, Ewa Deelman, Alvaro AA Fernandes, and Gaurang Mehta. "Adaptive workflow processing and execution in pegasus." Concurrency and Computation: Practice and Experience 21, no. 16 (2009): 1965-1981.

[4] Wieczorek, M, Prodan, R and Fahringer, T., "Scheduling of scientific workflows in the ASKALON Grid environment." s.l. : ACM, 2005, Issue 3, Vol. 34.

[5] Condor team, “The directed acyclic graph manager”, www.cs.wisc.edu/condor/dagmang, 2002.

[6] Yu Z, Shi W. “An adaptive rescheduling strategy for grid workflow applications.” *IPDPS*, IEEE Press, 2007; 1–8.

[7] Wohed, Petia, Wil MP van der Aalst, Marlon Dumas, and Arthur HM ter Hofstede. “Pattern based analysis of BPEL4WS”. QUT Technical report, FIT-TR-2002-04, Queensland University of Technology, Brisbane, 2002.

[8] Weihai Yu. 2009. “Decentralized Orchestration of BPEL Processes with Execution Consistency.” In Proceedings of the Joint International Conferences on Advances in Data and Web Management (APWeb/WAIM '09), Qing Li, Ling Feng, Jian Pei, Sean X. Wang, Xiaofang Zhou, and Qiao-Ming Zhu (Eds.). Springer-Verlag, Berlin, Heidelberg, 665-670.

[9] Pantazoglou, Michael, Ioannis Pogkas, and Aphrodite Tsalgatidou. "Decentralized Enactment of BPEL Processes." (2013): 1-1.

[10] Oinn, Tom, Matthew Addis, Justin Ferris, Darren Marvin, Martin Senger, Mark Greenwood, Tim Carver et al. "Taverna: a tool for the composition and enactment of bioinformatics workflows." Bioinformatics 20, no. 17 (2004): 3045-3054.

[11] Berman, Francine, Andrew Chien, Keith Cooper, Jack Dongarra, Ian Foster, Dennis Gannon, Lennart Johnsson et al. "The GrADS project: Software support for high-level grid application development." International Journal of High Performance Computing Applications 15, no. 4 (2001): 327-344.

[12] Kalayci, Selim, Onyeka Ezenwoye, Balaji Viswanathan, Gargi Dasgupta, S. Sadjadi, and Liana Fong. "Design and implementation of a fault tolerant job flow manager using job flow patterns and recovery policies." Service-Oriented Computing–ICSOC 2008 (2008): 54-69.

[13] Kalayci, Selim, Gargi Dasgupta, Liana Fong, Onyeka Ezenwoye, and S. Masoud Sadjadi. "Distributed and Adaptive Execution of Condor DAGMan Workflows." 22nd International Conference on Software Engineering & Knowledge Engineering (SEKE'2010): 587-590.