Fine-Grained Attribute Level Locking Scheme for Collaborative Scientific Workflow Development

Golam Mostaeen, Banani Roy, Chanchal K. Roy, Kevin A. Schneider
Department of Computer Science
University of Saskatchewan, Canada
{golam.mostaeen, banani.roy, chanchal.roy, kevin.schneider}@usask.ca

***Abstract*—Scientific Workflow Management Systems are being widely used in recent years for data-intensive analysis tasks or domain-specific discoveries. It often becomes challenging for an individual to effectively analyze the large scale scientific data of relatively higher complexity and dimensions, and requires a collaboration of multiple members of different disciplines. Hence, researchers have focused on designing collaborative workflow management systems. However, consistency management in the face of conflicting concurrent operations of the collaborators is a major challenge in such systems. In this paper, we propose a locking scheme (e.g., collaborator gets write access to non-conflicting components of the workflow at a given time) to facilitate consistency management in collaborative scientific workflow management systems. The proposed method allows locking workflow components at a granular level in addition to supporting locks on a targeted part of the collaborative workflow. We conducted several experiments to analyze the performance of the proposed method in comparison to related existing methods. Our studies show that the proposed method can reduce the average waiting time of a collaborator by up to 36.19% in comparison to existing descendent modular level locking techniques for collaborative scientific workflow management systems.**

I. Introduction

A Scientific Workflow Management System (*SWfMS*) is a system that supports the specification, modification, execution, failure handling, and monitoring of scientific workflows using workflow logic to control the order of executing workflow tasks [1]. SWfMSs have gained much popularity in recent years for accelerating the analysis, visualization, and discoveries of important information from a sheer amount of heterogeneous data generated on a daily basis by different areas of modern science [2], [3]. With the increase in complexity, dimension, and volume of such scientific data, their effective analysis process is often beyond the scope of an individual and requires a collaboration of a research group instead [4]. Besides, some scientific domains essentially require collaboration as they are highly correlated among multiple research disciplines [5].

As a result of extensive research over the last decade, several data-intensive SWfMSs have been proposed and developed [1]. A SWfMS provides techniques for modeling re-usable modular scientific data processing steps and their dependency relations as Directed Acyclic Graph (DAG) [1]. Some of the recent popular SWfMSs are: Taverna [6], Galaxy [7], Kepler [8], Pegasus [9], VisTrails [10], Triana [11], VIEW [12] and so on. However, these workflow management systems generally operate in single user mode and do not support any collaboration among the users [13], [5]. It requires manual effort for any necessary collaboration among the users. For example, a common way for scientific workflow collaboration in recent years is by sharing the workflows on some shared social spaces, like *myExperiment* [14]. However, this manual collaboration process is time-consuming, does not support real-time editing or any management system for considering different updates by the collaborators [4], [15], [16], [5], [3].

As a result of this compelling need for collaborative scientific workflow management system, several methods have been proposed and developed in recent years [4], [16], [15], [5], [17], [18], [3]. One of the main challenges of such a collaborative system is *consistency management* - in the face of conflicting concurrent operations by the collaborators [19], [20]. The existing research works use locking techniques - where only a single collaborator gets exclusive *Write* access to a part of the workflow to facilitate the consistency management [16], [5] by preventing concurrent conflicting operations on the same workflow component. For example, some related studies are: the entire workflow object locking in turns [4], descendant modules locking [3], [5], multiple variants of module locking [16], [15] and so on (details in Section VI). However, as all of these existing locking methods work on modular levels, the collaboration concurrency is dropped significantly as the workflow complexity grows over time with an increased number of modules and complicated datalink (in SWfMSs datalink represents the dataflow dependency relation among the modular tasks as discussed in Section III) relations among them [3], [5], [18].

In order to overcome these problems, in this paper we propose a novel approach by further extending the workflow module locking to a more fine-grained attribute level. From our investigation on several recent workflow engines and their corresponding modules or tools, we found that imposing strict locks on descendent modules can often be a strong or redundant restriction in a collaborative scientific workflow development environment. A large amount of redundant descendent workflow module locks imposed by the existing methods can be significantly avoided or minimized by the proposed method of attribute level locking.

We conducted several simulated studies of different existing locking schemes for a comparative analysis with the proposed method. The experimental studies were conducted considering

different dimensions of collaborative workflow development environment, such as varying workflow tree structures, varying topology of the incoming lock requests and so on (Section IV). All of the studies were repeated a number of times, and their average values were used to mitigate any possible biasness in the result. The experimental results show that the proposed locking scheme can reduce the average waiting time of a collaborator by up to 36.19% in comparison to existing descendent module locking schemes, which is promising in the context of collaborative workflow development system (details in Section IV).

The rest of the paper is organized as follows: Section II outlines technical preliminaries and challenges for consistency management in collaborative workflow development environment. We then present our proposed method in Section III. Section IV contains several experimental evaluations of the proposed method. Section VI presents the related research works. Finally, Section VII reports our future works and draws the conclusion.

II. Consistency Management in Collaborative Scientific Workflow Management Systems

Two of the most important requirements for an effective collaborative systems are [19]:

- *High Responsiveness*: The effect of any user's actions must take place with least amount of delay even with the non-deterministic communication latency.
- *High Concurrency*: Multiple users must be able to work and concurrently edit the same shared object collaboratively.

To ensure high responsiveness, most of the collaborative systems follow a *replicated architecture* [19]. For example, the existing research works on collaborative scientific workflow management systems maintain a local copy of the workflow in all of the users' local workflow engine [4], [5], [3]. Collaboration among the users are then maintained by simple message passing among the workflow engines for the corresponding local user actions for better responsiveness [19]. However, maintaining *High Concurrency* while preserving consistent workflow state is comparatively more challenging in the face of conflicting concurrent operations by collaborators [19], [20].

For example, Figure 1 illustrates the workflow collaboration between two users. The target workflow has been replicated to both the local workflow engines. At any given time, we assume the consistent existence of the workflow object in both the local ends with version 0 (represented as workflow state 1.0 and 2.0 for the User 1 and 2 respectively for demonstration). In this case two concurrent conflicting operations - $O_1 = updateDatalink(m_m, \boldsymbol{m_i})$ and $O_2 = updateDatalink(m_m, \boldsymbol{m_k})$, target the same workflow module - m_m. As the design follows a replicated architecture, the operations immediately update the workflow in the local ends to give a quick action response, creating the workflow state 1. The operation information is later passed to the other collaborating workflow engine for the corresponding update.

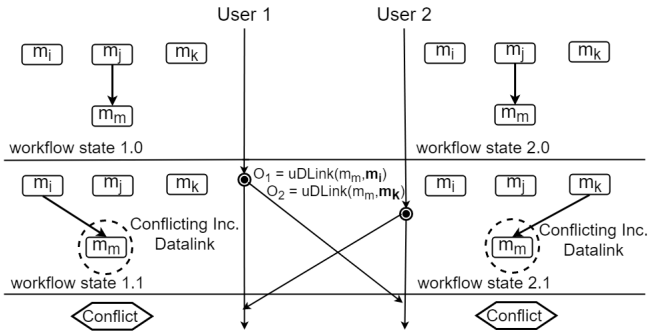

Fig. 1. Workflow Version Conflict in Collaborative Development.

Since O_1 and O_2 target the same workflow module - m_m, but with different attribute values (e.g., $m_i \neq m_k$), it is impossible to preserve the consistency of the workflow object by simple execution of the operations [19], [20] as depicted in the figure (e.g., the resulting incoming datalink relation to module m_m are different for the two collaborating users). Several locking schemes have been proposed [5], [4], [16], [15], [3] to facilitate preventing such inconsistency problems in the context of collaborative SWfMSs, where only a single user is allowed to work on any particular part of the collaborative workflow at any given time. However, the existing methods show poor concurrency in terms of modern scientific workflows having complex dependency relations and hence, to mitigate the problems we propose our techniques as discussed in the following section.

III. Fine-Grained Workflow Component Locking for Workflow Collaboration

A scientific workflow can be represented as a Directed Acyclic Graph (DAG), $W = (M, E)$, where M is a set of workflow modules representing different tasks and E is a set of directed edges representing the dataflow dependencies among the modular tasks [3], [21], [22], [23]. A valid scientific workflow contains n finite number of modules [21], $m_i \in M, (1 \leq i \leq n)$ depending on the specific data analysis or manipulation tasks. A workflow module m, is responsible for performing a specific modular task on a given dataset. A workflow module thus is generalized as tuple $m =< id, C, S >$, where id is the unique identifier of the module in a workflow that is used for several purposes like workflow provenance, workflow task scheduling and so on, C is a set of $\mathcal{P}$ different parameter settings or configurations, $c_i \in C, (1 \leq i \leq \mathcal{P})$ of the module that determines the execution order (e.g., input dataset in a dataflow oriented workflow architecture [24]) and behaviour of the modular task execution (e.g., threshold value for a dataset filtering task and so on) and finally, S is the modular source code that executes based on the configuration set, C. The execution order of those workflow modules are defined in E, which is a set of directed edges, $e_{ij} = (m_i, m_j), (1 \leq i \leq n, 1 \leq j \leq n, i \neq j)$ representing dataflow link from module m_i to module m_j [21].

A. Hierarchical Sub-Workflow Locking

Unlike collaborative text or graphics editing systems the scientific workflows are more structured where one module can be highly dependent on another forming a hierarchical relation among them [3]. We formally define the dependency relation as following:

Definition III.1 (Module Dependency Relation). *For a workflow $W = (M, E)$, a workflow module $m_t \in M$ is dependent on the workflow module $m_s \in M$, if there exists a sequence of workflow modules $m_0 = m_s, m_1, m_2, \cdots, m_k = m_t$, such that datalink $(m_{i-1}, m_i) \in E$, for all, $1 \leq i \leq k$. Here, the workflow module m_t is a descendant of workflow module m_s, represented as relation $\mathcal{D}(m_s, m_t)$, and cannot begin its execution until all of its such ancestor modules finish their executions.*

The workflow structure being a DAG [3], [16], the dependency relation is anti-symmetric. That is, if two workflow modules m_s and m_t hold dependency relation as $\mathcal{D}(m_s, m_t)$, then it must not hold the relation $\mathcal{D}(m_t, m_s)$, where $m_s \neq m_t$. Holding both the relation $\mathcal{D}(m_s, m_t)$ and $\mathcal{D}(m_t, m_s)$, would create a cycle in the Graph (e.g., given, $m_s \neq m_t$) causing deadlock in the execution order of the module m_s and m_t.

Definition III.2 (Hierarchical Descendant Module Lock). *For a workflow $W = (M, E)$, a hierarchical descendant module lock, mLOCK(m_l) on any module $m_l \in M$, grants Write access to m_l and any other module $m_x \in M$, where the relation $\mathcal{D}(m_l, m_x)$ holds. The lock recursively applies on any datalink $(m_{i-1}, m_i) \in E$, in the module sequence $m_0 = m_l, m_1, m_2, \cdots, m_k = m_x$, $(1 \leq i \leq k)$ for the dependency relation $\mathcal{D}(m_l, m_x)$.*

A hierarchical descendant module lock on any module $m_l =< id_l, C_l, S_l >$, *mLOCK(m_l)*, thus allows *Write* access to any parameter settings or configuration, $c_i \in C_l, (1 \leq i \leq \mathcal{P}_{C_l})$, where $\mathcal{P}_{C_l}$ is the number of parameter settings or configuration available in workflow module, m_l.

Definition III.2 for descendant module locking is applicable for any simpler linear workflows to any hierarchical scientific workflows where a workflow is generally composed of several branched (e.g., dataflow dependency) smaller sub-workflows recursively. Based on these definitions, a central locking algorithm can be designed for managing the concurrent sub-workflow lock/unlock requests by different collaborators.

B. Lock Extension to Granular Module Attribute Level

The hierarchical descendant module locking as presented above can alone facilitate the consistency management in the face of concurrent conflicting workflow operations. However, allowing only hierarchical descendant module locking can often be too restrictive in terms of modern collaborative workflow development (as demonstrated in Section IV). From these considerations, we further extend the lock to a more granular level as defined in the following:

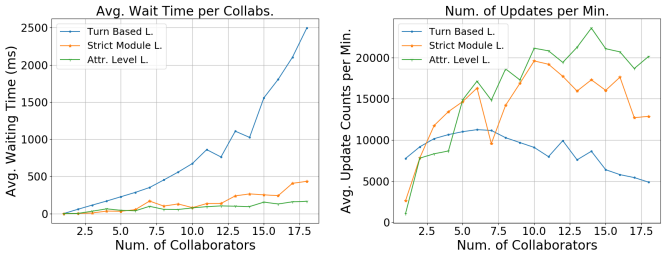

Fig. 2. Throughput and Average Waiting Time Comparison of Locking Algorithms for Collaborative Workflow Composition.

Definition III.3 (Granular Attribute Locking). *For a workflow module, $m_l =< id_l, C_l, S_l > \in M$ of a workflow, $W = (M, E)$, an attribute lock, aLOCK(m_l, c_i), grants Write access to attribute, $c_i \in C_l$ of the workflow module m_l.*

This granular locking allows additional controls in conjunction to hierarchical descendant module locking for explicit sub-workflow locking (e.g., Definition III.2). Attaining an attribute level lock by a user ensures only single operation execution on the modular attribute at any given time to facilitate consistency management. This adds the scope for higher concurrency as it does not necessarily impose possible redundant locks to all its descendant modules. Besides, by Definition III.3, the granular lock can be expanded to include any workflow module, $m_x \in w^U$ by iteratively imposing locks to all its attributes, $c_i \in C_x, (1 \leq i \leq \mathcal{P}_{C_x})$.

IV. Experiments and Evaluation

A. Experimental Setup

For our experiments, we considered four basic workflow operations: i) Adding a new module to the workflow, ii) Adding a new datalink to/from a module, iii) Updating an attribute of a module and iv) Updating source/destination of a datalink. Workflow collaborators were simulated using independent threads. To simulate *short-read, long-thinking* pattern [5], [4], as adopted by related works [5], [3], we considered random *thinking time* [5] interval ranging from 10 ms to 15 ms in between any basic workflow operation execution by a collaborator. If the next thinking time is relatively longer (e.g., considered, >10 ms), the corresponding collaborator releases any accessed object, making it available for other collaborators of the group. To mitigate any possible biasness from the results, the experiments were repeated multiple times, and their average values were used for convergence.

B. Evaluation

1) Analysis Study on Throughput and Collaborative Composition Time: Figure 2 shows the comparison of the locking schemes in terms of two different dimensions for varying number of collaborators. All of the three locking schemes illustrate similar behaviour regarding *average waiting time* (i.e., average delay time in between an access request and its granting) and *average update counts* (i.e.,

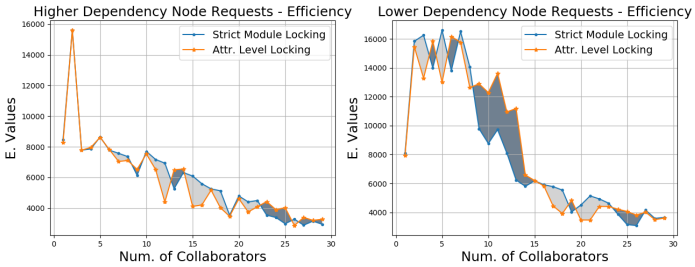

Fig. 3. Algorithm Performances on *'Best'* and *'Worst'* Case Scenario of Collaborative Node Access Requests.

throughput) when the collaborative group size is relatively smaller (i.e., around two collaborators). However, as the collaborative group size increases the graphs show significant differences in performance among the locking schemes. For example, the average waiting time for the proposed attribute level locking scheme with 18 collaborators is around 165 ms in comparison to 2495 ms and 433 ms for turn based [4] and strict descendant module locking schemes [3], [5], [16] respectively. A significant improvement is also noticeable in terms of average update count per minute as the collaborative group size increases. For example, the workflow update count per minute for turn based, strict and attribute level locking schemes are around 4886, 12880 and 20143 respectively for a collaborative group size of 18.

2) Performance Analysis Study for Varying Topology of Access Requests: In similar studies, Sipos *et al.* [16], [15] showed that the locking algorithm performance can be influenced by the topology of l lock requests, $R_1, R_2, R_3, \cdots R_l \in R^S$ for the same collaborative workflow $W = (M, E)$. To test the performance in two extreme cases, two types of request topologies were considered - a new access request always targets the available: i) node with lowest dependency degree, and ii) Oppositely, node with highest dependency degree, ϕ. Figure 3 illustrates the obtained results by the algorithms. The proposed method shows better result for higher number of collaborators for both the extreme cases. However, the graphs do not show any general recognizable pattern in their behaviors as they are often depended on request topology, R^S [16].

V. Threats to the Validity

In our simulated experimental studies, we adopted *short-read, long-thinking* pattern [5], [4] with a predefined thinking time range to imitate the human collaborators' working behaviour. However, human working pattern can be more diverse in nature (e.g., longer *thinking time*, inter-collaborator communications and so on) and thus it can be often challenging to exactly imitate in a simulation study. While this is a common threat for any simulation studies, the existing state of the art simulation based related techniques [4], [5], [3] used this approach for evaluating their studies with success which gave us confidence on our evaluation as well. Furthermore, in order to mitigate any biasness in the results, the exact experimental settings were applied to all of the locking schemes, the experiments were conducted on the same machine and also repeated a number of times to use their average values for convergence. We also conducted the experiments in several dimensions to validate the performance comparison studies.

VI. Related Works

SWfMSs have gained much popularity in the past few years and are widely used for data-intensive analysis, simulation, visualization and so on [13], [1]. Lu *et al.* [25] studied several motivations opportunities for collaborative SWfMSs from the perspective of large-scale and multidisciplinary research projects. In recent years, several methods have been proposed for consistency management of the shared workflow in a collaborative environment. Floor control or turn based locking schemes (e.g., the entire collaborative object is locked in turns by collaborators) are widely used for consistency management in collaborative work environment. Zhang *et al.* [4] studied the concept in the context of collaborative workflow management systems. While such turn based approach better matches with human communication protocol (e.g., Robert's Rules of Order (RRO) [26]), it has several issues such as only a single collaborator can work on workflow update at any given time (e.g., the concurrency count is significantly low), longer average waiting time even for a medium-sized collaborative group and so on.

Fei *et al.* [3] and Zhang *et al.* [5] presented locking schemes by allowing only descendent module locks (e.g., descendent nodes of the workflow DAG [1]) instead of imposing the lock on the entire workflow. Though the collaboration concurrency is improved in this case in comparison to turn based locking [4] (as discussed in Section IV), these modular locking schemes show significant reduction in the concurrency count as the workflow complexity grows over time with an increased number of modules and complicated datalink dependency relation among them. Because, a modular lock in these cases in turn locks major portions of the collaborative workflow. In an attempt to lower the redundant sub-workflow locks (e.g., any intended update on a module, strictly locks all its descendants modules due to the extension of the locking set [15]), using multiple modes of sub-workflow locks have also been proposed. Sipos *et al.* [15] used two lock modes - *User* and *System* locks. Fei *et al.* [3] proposed a lock compatibility matrix for a set of six pre-defined modes of locks. While multiple modular locks can avoid a few of the redundant locks depending on the defined compatibility relation, the improvement is almost negligible for a larger collaborative group due to their several lock conflicts. Techniques have also been studied for extending the single-user Grid portals to a collaborative environment [27], [16]. Dou *et al.* [28] studied context and role-driven scientific workflow development pattern in a collaborative environment. However, the extension or generalization of the method is challenging as defining non-conflicting roles can often be much complex in terms of consistency and depends largely on the given collaboration domain.

However, the existing locking schemes operate in the modu-

lar level and thus often result in significantly low concurrency count in modern scientific workflow collaborations (Section III). To mitigate the similar problems, collaborative research works on other domains such as text or graphics editing systems have considered locking on finest component levels. For example, Sun *et al.* [29] studied fine-grain locking scheme in the character sequence levels for collaborative text editing systems as previous studies [30] show that finer grained locking allows higher concurrency in collaborative environments. To the best of our knowledge, our work is the first in the context of collaborative SWfMSs to consider finer attribute level locking in comparison to workflow module level locking.

VII. Conclusion and Future Works

In this paper, we presented our investigation results of existing locking schemes in terms of consistency management of modern scientific workflow collaboration in the face of concurrent conflicting operations. From our study, we found that considering module level workflow locks can often be strong assumption resulting significantly low concurrency. We proposed a fine-grained locking scheme by further extending the modular locks to attribute level. The proposed attribute level locking scheme attempts to accelerate the collaborative workflow development process by lessening redundant sub-workflow locks. We got promising results from our simulation studies on multiple collaboration scenarios with a reduction of average waiting time by up to 36.19% while an increase of average workflow update rate by up to 15.28% in comparison to existing descendent modular level locking techniques.

Our future works include conducting user studies for usability analysis of the locking schemes in terms of human collaborators. None of the existing works targeted similar studies and thus our findings can be useful for further improvement of the collaborative workflow development systems.

References

[1] Ji Liu, Esther Pacitti, Patrick Valduriez, and Marta Mattoso. A survey of data-intensive scientific workflow management. *Journal of Grid Computing*, 13(4):457–493, 2015.

[2] Gordon Bell, Tony Hey, and Alex Szalay. Beyond the data deluge. *Science*, 323(5919):1297–1298, 2009.

[3] Xubo Fei, Shiyong Lu, and Jia Zhang. A granular concurrency control for collaborative scientific workflow composition. In *Services Computing (SCC), 2011 IEEE International Conference on*, pages 410–417. IEEE, 2011.

[4] Jia Zhang. Co-taverna: a tool supporting collaborative scientific workflows. In *Services Computing (SCC), 2010 IEEE International Conference on*, pages 41–48. IEEE, 2010.

[5] Jia Zhang, Daniel Kuc, and Shiyong Lu. Confucius: A tool supporting collaborative scientific workflow composition. *IEEE Transactions on Services Computing*, 7(1):2–17, 2014.

[6] Tom Oinn, Matthew Addis, Justin Ferris, Darren Marvin, Martin Senger, Mark Greenwood, Tim Carver, Kevin Glover, Matthew R Pocock, Anil Wipat, et al. Taverna: a tool for the composition and enactment of bioinformatics workflows. *Bioinformatics*, 20(17):3045–3054, 2004.

[7] Jeremy Goecks, Anton Nekrutenko, and James Taylor. Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. *Genome biology*, 11(8):R86, 2010.

[8] Bertram Ludäscher, Ilkay Altintas, Chad Berkley, Dan Higgins, Efrat Jaeger, Matthew Jones, Edward A Lee, Jing Tao, and Yang Zhao. Scientific workflow management and the kepler system. *Concurrency and Computation: Practice and Experience*, 18(10):1039–1065, 2006.

[9] Ewa Deelman, Gurmeet Singh, Mei-Hui Su, James Blythe, Yolanda Gil, Carl Kesselman, Gaurang Mehta, Karan Vahi, G Bruce Berriman, John Good, et al. Pegasus: A framework for mapping complex scientific workflows onto distributed systems. *Scientific Programming*, 13(3):219–237, 2005.

[10] Steven P Callahan, Juliana Freire, Emanuele Santos, Carlos E Scheidegger, Cláudio T Silva, and Huy T Vo. Vistrails: visualization meets data management. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pages 745–747. ACM, 2006.

[11] Ian Taylor, Matthew Shields, Ian Wang, and Andrew Harrison. The triana workflow environment: Architecture and applications. *Workflows for e-Science*, pages 320–339, 2007.

[12] Cui Lin, Shiyong Lu, Xubo Fei, Artem Chebotko, Darshan Pai, Zhaoqiang Lai, Farshad Fotouhi, and Jing Hua. A reference architecture for scientific workflow management systems and the view soa solution. *IEEE Transactions on Services Computing*, 2(1):79–92, 2009.

[13] Adam Barker and Jano Van Hemert. Scientific workflow: a survey and research directions. In *International Conference on Parallel Processing and Applied Mathematics*, pages 746–753. Springer, 2007.

[14] David De, Roure Carole, and Goble Robert Stevens. The design and realisation of the myexperiment virtual research environment for social sharing of workflows. 2008.

[15] Gergely Sipos. Protecting the consistency of workflow applications in collaborative development environments. *Future Generation Computer Systems*, 28(3):500–512, 2012.

[16] Gergely Sipos and Péter Kacsuk. Maintaining consistency properties of grid workflows in collaborative editing systems. In *Grid and Cooperative Computing, 2009. GCC'09. Eighth International Conference on*, pages 168–175. IEEE, 2009.

[17] Gergely Sipos and Peter K Kacsuk. Collaborative workflow editing in the p-grade portal. 2005.

[18] Gergely Sipos and Péter Kacsuk. Efficient graph partitioning algorithms for collaborative grid workflow developer environments. *Euro-Par 2010-Parallel Processing*, pages 50–61, 2010.

[19] Chengzheng Sun and David Chen. Consistency maintenance in real-time collaborative graphics editing systems. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 9(1):1–41, 2002.

[20] David Sun and Chengzheng Sun. Operation context and context-based operational transformation. In *Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work*, pages 279–288. ACM, 2006.

[21] Ritu Garg and Awadhesh Kumar Singh. Adaptive workflow scheduling in grid computing based on dynamic resource availability. *Engineering Science and Technology, an International Journal*, 18(2):256–269, 2015.

[22] Ritu Garg and Awadhesh Kumar Singh. Multi-objective workflow grid scheduling using\varepsilon-fuzzy dominance sort based discrete particle swarm optimization. *The Journal of Supercomputing*, 68(2):709–732, 2014.

[23] Haluk Topcuoglu, Salim Hariri, and Min-you Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE transactions on parallel and distributed systems*, 13(3):260–274, 2002.

[24] Xubo Fei and Shiyong Lu. A dataflow-based scientific workflow composition framework. *IEEE Transactions on Services Computing*, 5(1):45–58, 2012.

[25] Shiyong Lu and Jia Zhang. Collaborative scientific workflows. In *Web Services, 2009. ICWS 2009. IEEE International Conference on*, pages 527–534. IEEE, 2009.

[26] D.H. Honemann M. Robert, W.J. Evans and T.J. Balch. Robert's Rules of Order. Newly Revised, 10th Edition. Perseus Publishing Company, 2000.

[27] Gergely Sipos, Gareth Lewis, Péter Kacsuk, and Vassil Alexandrov. Workflow-oriented collaborative grid portals. *Advances in Grid Computing-EGC 2005*, pages 64–69, 2005.

[28] Wanchun Dou, Jinjun Chen, Shaokun Fan, and SC Chueng. A context- and role-driven scientific workflow development pattern. *Concurrency and Computation: Practice and Experience*, 20(15):1741–1757, 2008.

[29] Chengzheng Sun. Optional and responsive fine-grain locking in internet-based collaborative systems. *IEEE Transactions on Parallel and Distributed Systems*, 13(9):994–1008, 2002.

[30] Jonathan Munson and Prasun Dewan. A concurrency control framework for collaborative systems. In *Proceedings of the 1996 ACM conference on Computer supported cooperative work*, pages 278–287. ACM, 1996.