A Granular Concurrency Control for Collaborative Scientific Workflow Composition

Xubo Fei*, Shiyong Lu*, Jia Zhang†
*Department of Computer Science, Wayne State University, Detroit, MI, USA
{xubo, shiyong}@wayne.edu
†Department of Computer Science, Northern Illinois University, Detroit, MI, USA
jiazhang@cs.niu.edu

***Abstract*—Collaborative scientific workflow composition has recently been proposed to support collaborative scientific research projects, which require intensive collaboration among scientists with diverse expertise. A collaborative scientific workflow management system allows participating scientists to design and compose common scientific workflows concurrently. Concurrency control has become one of the key challenges in collaborative scientific workflow composition, which aims to facilitate collaboration while ensuring the correctness and consistency of the results generated from concurrent operations. We have previously proposed a locking scheme to support simple scientific workflow compositions, in which workflows are flat (not hierarchical). In this paper, we take a step forward to propose a new granular scientific workflow locking scheme, which captures dependency relationships between workflows, to support general hierarchical workflow compositions. We also conduct several experiments to evaluate the performance of the concurrency control.**

***Keywords*-collaborative scientific workflow; concurrency control; granular locking.**

I. Introduction

Scientific workflow has been recognized as a key technique to support data-intensive scientific research - from data capture and data curation, to data analysis and final data product visualization [1]. Modern data-intensive scientific research projects often require intensive collaboration among scientists with diverse expertise. Collaborative workflow composition is one important step in the lifecycle of collaborative scientific workflow management [2], [3].

Although several scientific workflow management systems (SWFMSs) have been developed, including Taverna [4], Kepler [5], Triana [6], VisTrails [7], Pegasus [8], Swift [9], and VIEW [10], they are designed mainly for single users. The collaboration between distributed scientists are not effectively supported by the existing systems. We summarize two current collaboration patterns as follows: first, scientists can collaboratively design a scientific workflow in a sequential order such that only one scientist can work on the workflow composition at one time and once finished, she can send it to the next scientist. Such a sequential pattern may become extremely time-consuming for larger scientific research projects which involve multiple scientists. Second, scientists can work separately on subworkflows and a coordinator integrates them together to form a whole workflow. This pattern is more efficient but cannot be adopted when multiple scientists intend to design or modify the same subworkflow. Therefore, there is a need to establish a new infrastructure and develop a collaborative scientific workflow composition system.

A collaborative scientific workflow management system allows participating scientists to design and compose a common scientific workflow. However, in this type of systems, *conflicts* may occur when concurrent operations are performed from multiple scientists to change the same workflow. Such conflicting operations may interfere with each other and result in *inconsistency*, which is a typical problem in transaction processing [11] [12]. Therefore, *concurrency control* has become one of the key challenges in collaborative scientific workflow composition, which aims to increase the system throughput while ensuring the correctness and consistency of the results generated from concurrent operations.

We have previously presented a database locking scheme in [13] to support simple workflow composition, which is a directed acyclic graph (DAG) consisting of a set of workflow tasks and a set of data links. All workflows are considered independent from each other. However, a hierarchical scientific workflow is usually composed of several subworkflows, each of which in turn might consist of other subworkflows. The traditional locking schemes are not efficient to support such a hierarchical organization because a lock on a hierarchical scientific workflow leads to many locks on all of its descendants. Each such access incurs computational overhead of setting and waiting for locks and the storage overhead of maintaining locks. Granular locking [14] provides a hierarchical locking scheme which allows users to set locks on arbitrary nodes of the hierarchy. If one transaction is granted with a lock to a particular node, then the transaction has access to the node and implicitly to each of its descendants. However, existing database granularity locking mechanisms are insufficient to support collaborative scientific workflow applications, because the locking granularity in relational databases and that in scientific workflow systems are different. While relational databases apply locks on pages, tables, rows, and cells, the granularity considered in scientific workflows are typically workflows and tasks with an arbitrary hierarchy. Therefore, a lock on a scientific

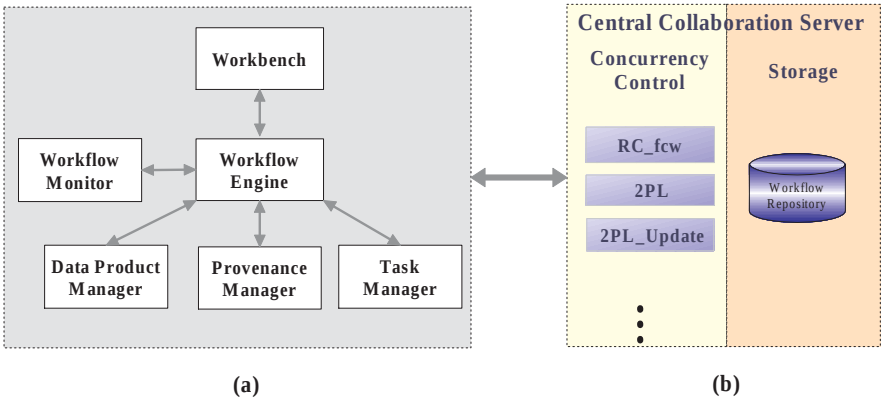


Figure 1. (a) Reference architecture for SWFMSs; (b) a central collaboration server.

workflow might affect multiple tables in different pages and cannot be implemented with an existing database granular locking scheme.

In this paper, we take a step forward to propose a new granular scientific workflow locking scheme, which captures dependency relationships between workflows, to support general hierarchical workflow compositions. The remainder of the paper is organized as follows. Section II extends the reference scientific workflow architecture to support collaborative scientific workflow composition. Section III proposes a granular scientific workflow locking scheme and its associated algorithms. Section IV presents experimental results. Section V discusses related work. Section VI draws our conclusions.

II. Collaborative SWFMSs

A. An Architecture for Collaborative SWFMSs

We have previously proposed a reference architecture for scientific workflow management systems [10]. As shown in Figure 1.(a), a scientific workflow management system consists of six loosely-coupled subsystems: a *Workflow Engine* to execute workflows, a *Task Manager* to manage and execute tasks, a *Workflow Monitor* to display system status and handle exceptions, a *Provenance Manager* to store and query workflow provenance, a *Data Product Manager* to store and manage data products, and a *Workbench* to visually design and modify workflows, present data product and provenance, and manage subsystems. Here, we extend the previous architecture, which supports only single users, to support multiple users for collaborative scientific workflow compositions.

While in the single-user mode, a scientist composes a workflow on a visual designer and the workbench translates the graphical composition into a scientific workflow specification based on a well-defined scientific workflow language; in the collaborative mode, multiple scientists are allowed to collaborate to edit a common scientific workflow application. As shown in Figure 1.(b), scientists can design, modify, and compose a common scientific workflow concurrently using distributed workbenches and their operations will be sent to a central collaboration server and applied on the same workflow specification that is stored in a workflow repository. The central collaboration server consists of two components as follows:

- A workflow repository stores workflow specifications. The workflow repository can be implemented using either relational databases or file repositories.
- A concurrency control layer to ensure the correctness and consistency of the results generated by concurrent operations.

Different concurrency control schemes can be applied to ensure different isolation levels of the system for different scientific workflow models, such as *strict two-phase locking (2PL_wait0 or 2PL in short)*, *strict two-phase locking with update lock (2PL_update)* and *READ COMMITTED with first-committer-win (RC_fcw)* [15]. However, existing locking schemes cannot effectively capture dependencies between workflows. Therefore, we propose a new granular scientific workflow locking scheme in the next section.

B. A Dataflow-based Scientific Workflow Composition Model

Our proposed locking scheme is based on our previous dataflow-based scientific workflow composition model [16]. Here, we consider workflow-level composition, in which workflows are the only operands for workflow composition. A set of workflow constructs including unary constructs have been proposed in [16], such as Map, Reduce, Tree, Conditional, Loop, and Curry. These constructs transform workflows to new workflows. In addition, A workflow definition includes a workflow interface and a workflow body. A workflow interface declares the workflow identifier, workflow name, description, and input/output ports. A workflow body defines the implementation of the workflow. There are currently three kinds of implementations: primitive workflow body, unary-construct based workflow body, and graph-based workflow body. Primitive workflows are the basic building blocks of our model which are constructed

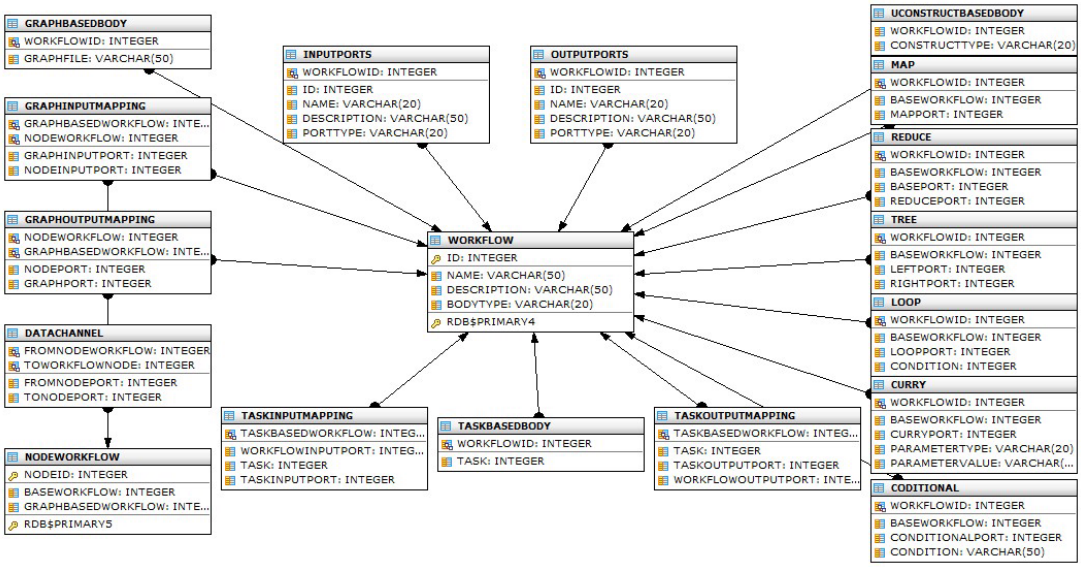


Figure 2. Relational database schema for our scientific workflow composition model.

from tasks. Unary-construct based workflows are created by applying unary constructs on existing workflows. Graph based workflows are defined from a workflow graph which are constructed from a set of workflows and a set of data link constructs.

We realize our model in a relational database schema as shown in Figure 2. The *WORKFLOW* table holds the general information including the workflow identifier, name and description. The *INPUTPORTS* and *OUTPUTPORTS* tables hold the information of the workflow interface and include a foreign key workflowID to relate ports to corresponding workflows. The *TASKBASEDBODY*, *TASKINPUTMAPPING*, and *TASKPUTMAPPING* tables defines the task based workflow bodies including the task identifiers and the input/output mapping between the task and task based workflow. The *UCONSTRUCTBASEDBODY* table defines the type of unary construct as a gateway to the *MAP*, *REDUCE*, *TREE*, *CONDITIONAL*, *LOOP*, and *CURRY* tables, which hold the definitions of the corresponding unary constructs. Because a unary construct defines a one-to-one mapping, the input/output mappings are by default and abbreviated. The *NODEWORKFOW* and *DATALINK* tables define the workflows and data link constructs that constitute the workflow graphs. The *GRAPHINPUTMAPPING* and *GRAPHOUTPUTMAPPING* tables define input/output mapping between the workflow graph and graph based workflow. The *GRAPHBASEDBODY* table manages the URLs of files that defines graphical information to visualize the workflow graph in workbench.

For each scientific workflow application, the collaboration server maintains two in-memory tables. First is a *Dependency* table maintaining the dependency graph constructed from the scientific workflow application. The dependency table will be updated only when an insert or delete operation is performed. Second is a *Locking* table maintaining the current workflow and construct locking status based on the algorithms to be proposed in the next section.

III. A Granular Scientific Workflow Locking Scheme

Formally, a scientific workflow composition is a tuple $< \mathfrak{W}, \mathfrak{C} >$, representing a finite set $\mathfrak{W}$ of workflows and a finite set $\mathfrak{C}$ of constructs. A *construct* $c : W \rightarrow w$ is a mapping from a set of workflows $W = \{W_1, ..., W_n\}$, which is a subset of $\mathfrak{W}$, to a workflow w. A set of workflow constructs are introduced in [16].

Definition III.1. (**Scientific Workflow Dependency**) A scientific workflow w *depends* on scientific workflow w', denoted as $w' \leftarrow w$, if they satisfy one of the following conditions:

- There exists a construct $c : W \rightarrow w$ and $w' \in W$. Then w is a *parent* of w', and w' is a *child* of w.
- There exists a sequence of workflows $w_1, w_2, ..., w_k$ such that $w' \leftarrow w_1, w_1 \leftarrow w_2, ..., w_k \leftarrow w$. Then w is an *ancestor* of w'.

A scientific workflow is a *composite workflow* if it has at least one child and a workflow is a *primitive workflow* if it has no children. A scientific workflow is a *root workflow* if it has no parents. The dependency relationships of a scientific workflow application can be represented in a *scientific workflow dependency graph*, which is defined below.

Definition III.2. (**Scientific Workflow Dependency Graph**) A scientific workflow dependency graph is a finite set W of nodes and a finite set D of edges (a subset of $W \times W$), such that each node represents a workflow and each edge represents a dependency relationship. A scientific

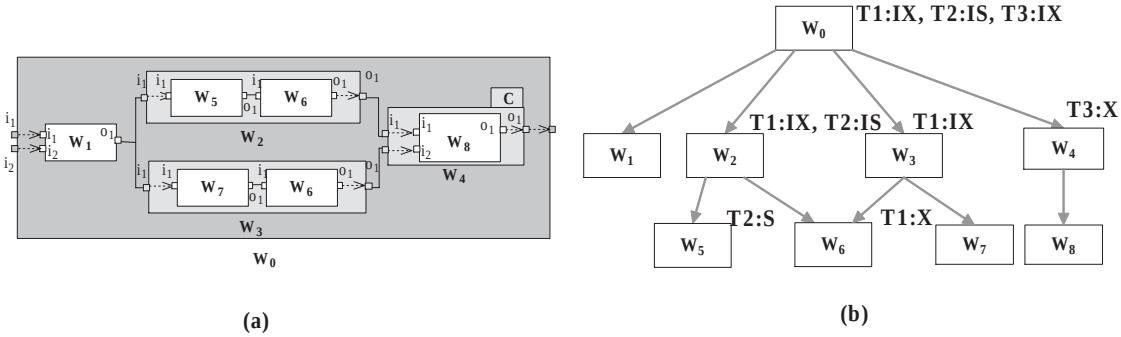


Figure 3. Example of workflow locking.

workflow dependency graph is *well-formed* if it contains exactly one root node.

A valid scientific workflow composition must contain one and only one root workflow because there must exist a main workflow as the entry of the composition. Therefore *a scientific workflow dependency graph generated by a valid scientific workflow composition is well-formed*.

Given a well-formed scientific workflow dependency graph with a root workflow w_r, for any node workflow $w \in \mathfrak{W}$, there must exist at least one path from w_r to w, represented as $\{(w_r, w_1), (w_1, w_2), ..., (w_k, w)\}$, where each pair belongs to the set of dependencies D. Figure 3.(b) illustrates an example of a scientific workflow dependency graph generated from a hierarchical scientific workflow in Figure 3.(a).

If two scientific workflows w and w' have a dependency relationship $w' \leftarrow w$, then w' is one of the building blocks of w and any locks applied on w should be inherited by w'. For example, when locking a graph based workflow, the system must lock all the node workflows belonging to the graph as well either explicitly or implicitly, so that other users cannot modify the node workflows. Conversely, when a transaction requests to lock a node workflow, the system must first verify that the graph workflow does not contain a conflicting lock. As a result, if a lock is granted on a workflow with a large number of component workflows, other transactions will have to wait or abort. Therefore, the concurrency of the system is determined by the dependency relationships between workflows. We define the *dependency degree* of a scientific workflow in order to measure the concurrency level of a scientific composition:

Definition III.3. (**Dependency Degree**) The *dependency degree* of a scientific workflow w, denoted as $\phi(w)$, is the cardinality of the set $\{w' | w' \in \mathfrak{W} \wedge w' \leftarrow w\}$. The *dependency degree* of a scientific workflow composition is then $\Sigma_{\forall w \in \mathfrak{W}} \phi(w)$.

The *dependency degree* of a scientific workflow w defines the number of workflows that depend on w. The *dependency degree* thus affects the quantity of data locked by a particular transaction. If a lock is granted on a workflow with a high dependency degree, then a large number of component workflows will inherit this lock. Therefore, intuitively, high *dependency degree* of a scientific workflow composition will decrease its concurrency for collaborations. We will conduct some experiments to validate this hypothesis.

Following the definitions in [12], the lock modes are $M = \{NULL, IS, IX, SIX, S, U, X\}$, in which $NULL$ stands for no locks, IS stands for intent shared locks, IX stands for intent exclusive locks, SIX stands for shared with intent exclusive locks, S stands for shared locks, U stands for update locks, and X stands for exclusive locks. Lock compatibility is defined in Table I and represented by the function $LC : M \times M \rightarrow \{Yes, No\}$. For example, suppose a workflow is already locked by transaction T_1 in mode S, and now two requests are generated by transaction T_2 and T_3: T_2 requests for an S lock and T_3 requests for an X lock. Then the request from T_2 can be granted because $LC(S, S) = Yes$, but the request from T_3 will be rejected because $LC(S, X) = No$.

Table I
LOCK COMPATIBILITY MATRIX

Req \ Granted	None	IS	IX	S	SIX	U	X
IS	+	+	+	+	+	-	-
IX	+	+	+	-	-	-	-
S	+	+	-	+	-	-	-
SIX	+	+	-	-	-	-	-
U	+	-	-	+	-	-	-
X	+	-	-	-	-	-	-

A transaction can also upgrade its previously granted locks. Lock upgrading is defined in Table II and represented by the function $UC : M \times M \rightarrow M$. For example, a transaction can first request an S lock to read a workflow and later upgrade the lock to the X mode because $UC(S, X) = X$. UC is idempotent, which means that a lock can be upgraded with the same request multiple times and still leads to the same result.

Definition III.4. (**Workflow locks**) The workflow locks of a transaction T are a mapping from workflows to lock modes $WLOCK_T : \mathfrak{W} \mapsto M$ indicating the locks granted to a particular workflow.

Table II
LOCK UPGRADE MATRIX

Req \ Granted	None	IS	IX	S	SIX	U	X
IS	IS	IS	IX	S	SIX	U	X
IX	IX	IX	IX	SIX	SIX	U	X
S	S	S	SIX	S	SIX	U	X
SIX	SIX	SIX	SIX	SIX	SIX	U	X
U	U	U	U	U	U	U	X
X	X	X	X	X	X	X	X

The workflow Locks satisfy the following two constraints:

- If $WLOCK_T(w) \in \{IS, S, U\}$, then either w is the root workflow, or $WLOCK_T(w_p) \neq NULL$ for all parent p of w (equivalently $WLOCKS_T(p) \in \{IS, IX, SIX, S, X\}$). By induction, $WLOCKS_T(a) \in \{IS, IX, SIX, S, X\}$ for all ancestors a of w (on all paths from w_r to w).
- If $WLOCK_T(w) \in \{IX, SIX, X\}$, then either w is the root workflow, or $WLOCK_T(w_p) \in \{IX, SIX, X\}$ for all parents p of w, By induction, $WLOCKS_T(a) \in \{IX, SIX, X\}$ for all ancestors a of w (on all paths from w_r to w).

The first constraint states that in order to get an IS, S, or U lock on a non-root workflow, the transaction must get a lock in $\{IS, IX, SIX, S, X\}$ for all parent workflows. Therefore, other transactions cannot modify any parent workflows. Similarly, the second constraint states that in order to get an IX, SIX, or X lock to a non-root workflow, the transaction must get a lock in $\{IX, SIX, X\}$ for all parent workflows so that other transactions cannot modify or read any parent workflows.

As shown in Figure 3.(b), Locks are set from the root workflow to the leaf workflows (primitive workflows). Transaction T_1 is currently working on workflow W_6 and has locked it in the X mode. Transaction T_2 is currently reading workflow W_5 and has locked it in the S mode. Suppose now a new transaction needs to read workflow W_3 and requests an S lock, then the new transaction must wait until transaction T_1 finishes the updating of W_6 and releases the lock. Transaction T_3 is currently updating workflow W_4 as a whole and has locked it in the X mode. Although there is no lock on workflow W_8, it is implicitly locked by transaction T_3. Hence, other transactions cannot access it until T_3 releases the exclusive lock on workflow W_4.

Definition III.5. (**Construct locks**) The construct locks of a transaction T are a mapping from constructs to lock modes $CLOCK_T : \mathfrak{C} \mapsto M$ indicating the locks granted to a particular construct.

The construct locks satisfy the following constraint:

- Given the construct $C : W \rightarrow w$, if $M \in \{IX, X, SIX\}$ then $WLOCKS_T(w) \in \{IX, X, SIX\}$) and for all $w' \in W$ $WLOCKS_T(w') \in \{IX, SIX, X\}$).

This constraint states that in order to get an exclusive lock on a construct, the transaction must be granted with exclusive locks on all related workflows so that other transactions cannot modify them.

Based on the proposed schemes, we designed four algorithms for workflow locking, workflow releasing, construct locking, and construct releasing.

Algorithm Lock Workflow (L(T, w, m))
Input: A transaction T requests a lock m on a workflow w.
Begin
1. **If** $m = WLOCK_T(w)$ **Then** do nothing;
2. **Else**
3. **Begin Transaction:**
4. **For** each transaction T'
5. **If** $LC(WLOCK_{T'}(w), m) = No$ **Then abort**
6. **End If**
7. **End For**
8. **If** $m \in \{IS, S, U\}$
9. **Then**
10. **For** each parent w'
11. **If** $WLOCK_T(w') \neq NULL$ **Then** do nothing
12. **Else Call** $L(T, w', IS)$
13. **End If**
14. **End For**
15. **Else If** $m \in \{IX, X, SIX\}$
16. **Then**
17. **For** each parent w'
18. **If** $WLOCK_T(w') \in \{IX, SIX, X\}$ **Then** do nothing
19. **Else Call** $L(T, w', IX)$
20. **End If**
21. **End For**
22. **End If**
23. $WLOCK_T(w) \Leftarrow UC(WLOCK_T(w), m)$
24. **End Transaction**
25. **End If**
End Algorithm

Figure 4. Workflow locking algorithm.

Figure 4 presents a workflow locking algorithm. To lock a workflow, the system will first check whether the requested lock already exists on the workflow, if so then nothing needs to be done. Then the system will check the compatibility between the requested lock and the current granted locks from other transactions. If the requested lock is compatible with existing locks, the system will request to recursively lock each parent workflow based on the workflow locking constraints. Finally, the requested lock will be granted to this workflow.

Figure 5 presents a workflow lock releasing algorithm. To release a workflow lock, the system will first check whether the lock to be released is the one currently on the workflow, if not then nothing needs to be done. Then the system will release the requested lock and also release all corresponding intent locks on parent workflows.

Figure 6 presents a construct locking algorithm. To lock a construct, the system will first check whether the required lock already exists on the construct, if so then nothing needs to be done. Then, the system will check the compatibility between the requested lock and the granted locks from other

Algorithm Release Workflow (R(T, w, m))
Input: A transaction T releases a lock m from a workflow w.
Begin
1. **If** $WLOCK_T(w) \neq m$ Then do nothing;
2. **Else**
3. **Begin Transaction:**
4. $WLOCK_T(w) \Leftarrow NULL$
5. **If** $m \in \{IS, S, U\}$
6. **Then**
7. **For** each parent w'
8. **Call** $R(T, w', IS)$
9. **End For**
10. **Else If** $m \in \{IX, X, SIX\}$
11. **Then**
12. **For** each parent w'
13. **Call** $L(T, w', IX)$
14. **End For**
15. **End If**
16. **End Transaction**
17. **End If**
End Algorithm

Figure 5. Workflow lock releasing algorithm.

Algorithm Lock Construct (L(T, c, m))
Input: A transaction T requests a lock m on a construct c, which is a mapping from a set of workflows W to a workflow w.
Begin
1. **If** $m = CLOCK_T(c)$ Then do nothing;
2. **Else**
3. **Begin Transaction:**
4. **For** each transaction T'
5. **If** $LC(CLOCK_{T'}(c), m) = No$ **Then abort**
6. **End If**
7. **End For**
8. **If** $m \in \{IS, S, U\}$
9. **Then**
10. **If** $CLOCK_T(w) \in \{IX, SIX, X\}$ **Then** do nothing
11. **Else Call** $L(T, w, IX)$
12. **End If**
13. **For** each workflow $w' \in W$
14. **If** $CLOCK_T(w') \in \{IX, SIX, X\}$ **Then** do nothing
15. **Else Call** $L(T, w', IX)$
16. **End If**
17. **End For**
18. **End If**
19. $CLOCK_T(c) \Leftarrow UC(CLOCK_T(c), m)$
20. **End Transaction**
21. **End If**
End Algorithm

Figure 6. Construct locking algorithm.

transactions. If the requested lock is compatible with existing locks, the system will lock on related workflows based on the construct locking constraints. Finally, the requested lock will be granted to this construct.

Figure 7 presents a construct lock releasing algorithm. To release a construct lock, the system will first check whether the lock to be released is the one currently on the construct, if not then nothing needs to be done. Then, the system will

Algorithm Release Construct (R(T, c, m))
Input: A transaction T releases a lock m from a construct c, c is a mapping from a set of workflows W to a workflow w.
Begin
1. **If** $m \neq CLOCK_T(c)$ Then do nothing;
2. **Else**
3. **Begin Transaction:**
4. $CLOCK_T(c) \Leftarrow NULL$
17. **If** $m \in \{IX, X, SIX\}$
10. **Then**
20. **Call** $R(T, w, IX)$
18. **For** each $w' \in W$
20. **Call** $R(T, w', IX)$
22. **End For**
23. **End If**
26. **End Transaction**
27. **End If**
End Algorithm

Figure 7. Construct lock releasing algorithm.

release the requested lock and also release all corresponding intent locks on related workflows.

IV. Experimental Study

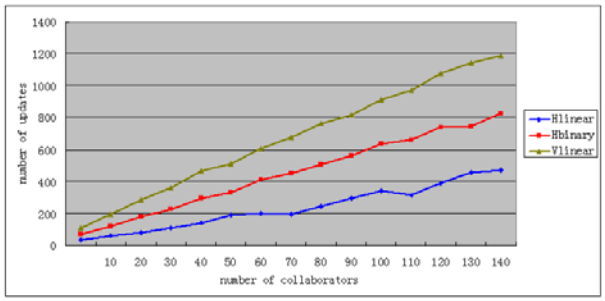
In Section III, we proposed a hypothesis that the performance of our proposed locking scheme depends on the structure of a workflow dependency graph. More specifically, the increase of the *dependency degree* will reduce the concurrency of transaction processing. We designed and conducted several experiments to test our hypothesis.

We created three synthetic scientific workflow compositions. Each composition consists of 20 workflows but the workflows are constructed with different structures as follows:

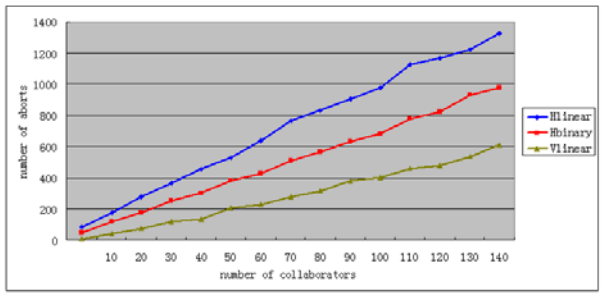
- A VLinear workflow composition, in which workflows are dependent one on another and each workflow has exactly one parent workflow except for the root workflow. The dependency graph is then a connected line.
- An HLinear workflow composition, in which there are no dependencies between workflows. The dependency graph is empty.
- An HBinary workflow composition, in which each non-leaf workflow contains exact two child workflows. The dependency graph is a balanced binary tree.

The experiment is to test the performance of the proposed locking scheme by varying the number of collaborators for the three scientific workflow compositions. Each collaborator was simulated by an independent (Java) thread, which iteratively reads a randomly selected workflow, waits a time period (5 seconds) for thinking and modification, and then updates the workflow. The experiments are repeated with different number of collaborators from 10 to 150 to validate the scalability of the locking scheme with regard to the number of collaborators.

The experimental results are reported in Figure 8. Figure 8.(a) shows the comparison of throughput for three



(a)



(b)

Figure 8. (a) Comparison of the throughput on three workflow compositions; (b) comparison of the failure rate on three workflow compositions.

scientific workflow compositions, measured as the number of successful task updates by all collaborators per minute. Figure 8.(b) shows the comparison of failure rate for three scientific workflow compositions, measured as the number of aborts by all collaborators per minute. Our locking scheme is scalable on all three compositions as the number of updates increases steadily with the increase of the number of collaborators. However, the performance between different workflow compositions are significantly different, which validated our hypothesis. The VLinear composition has the lowest throughput and the highest failure rate because it has the highest dependency degree of $19!$, while the HLinear composition has the highest throughput and the lowest failure rate because it has the lowest dependency degree of 0.

V. Related Work

SWFMS is a system that supports the specification, modification, execution, and monitoring of a scientific workflow, as well as the management of resource, data products, and provenance [10]. SWFMSs have become fundamental instruments for current and future scientific research and collaboration. Several SWFMSs have been proposed and many scientific workflows have been created. Taverna [4] provides a rich repository of services for bioinformatics data analysis and processing. The Taverna system supports hierarchical compositions by encapsulating a subworkflow in a composite task. Kepler [5] inherited various models of computation from the Ptolemy system. In Kepler, each workflow is assigned with a *director*, which is a computation model that controls the execution of a workflow. The Kepler system supports hierarchy in workflows and subworkflows encapsulated in different *composite actors* can have different directors and thus be executed under different models. Triana [6] is based on the Grid Application Prototype Interface (GAP Interface) which can bind with various services, such as Web services, Grid services, and peer-to-peer services. Triana also supports hierarchical composition by grouping tasks into a *GroupTask*, which can be used to compose with other tasks. VisTrails [7] is featured with workflow provenance, which maintains a complete history of workflow composition. Pegasus [8] provides a graphical interface to compose abstract workflows as a DAG and then maps them to concrete workflows that are executable on the grid. Swift [9] implements a scripting language called *SwiftScript*, which is designed to support large-scale computations over a Grid environment. The VIEW system [10] is designed and implemented using service oriented architecture (SOA) consisting of six loosely coupled subsystems. The system features a purely dataflow based workflow composition model [16] and provides a set of workflow constructs to facilitate various data processing patterns. Most of these existing systems support some forms of task-level collaboration features. For example, scientists can publish their workflows or programs as services and their collaborators can reuse those services to compose more advanced workflows. However, these scientific workflow management systems cannot allow multiple scientists to concurrently design and modify a common scientific workflow application.

For the first time, our previous work reported Confucius [13], a tool capable of supporting multiple scientists in designing and composing scientific workflows collaboratively, either synchronously or asynchronously. Without reinventing the wheel and as a proof of concept, our Confucius system extends the single-user oriented Taverna [4] into a multi-user version. This paper reported our continuous efforts on Confucius, aiming to propose a granular concurrency control scheme and its associated algorithms that can handle hierarchical workflow composition.

Concurrency control is a well-known problem in transaction processing [11] and locking is a general solution to this problem. Gray et al. [14] introduced intent locks to solve the hierarchical locking problem in trees or directed acyclic graph (DAG), and formalized a granular locking theorem and protocol. Chen et al. [17] proposed an instant locking scheme for a realtime collaborative graphics editing system called GRACE, in which users coordinate their activities, and conflict is very rare. Therefore, the traditional locking scheme is relaxed and independent operations are allowed to operate on the same object concurrently. Bächle et al. [18] proposed a tailor-made lock protocols for a fine-grained transaction isolation on XML document trees. Although many current scientific workflow systems realize their languages in the XML format, some workflow dependencies

are implicit and cannot be captured by the XML model. For example, a unary construct based workflow is defined by a unary workflow construct and a base workflow. However, the base workflow is previously defined and only the workflow identifier is referenced in the definition of the unary construct based workflow. As a result, those two workflows are recorded as sibling nodes in an XML specification although they have a logical dependency. Therefore, an XML locking scheme is not suitable for concurrency control in collaborative scientific workflow compositions.

In [13], we proposed a *READ COMMITTED with first-committer-win (RC_fcw)* scheme, as an extension of *READ COMMITTED* with the first-committer-win feature from the SNAPSHOT isolation level, to support currency control for simple workflow compositions, in which workflows are considered independent. In this paper, we take a step forward to propose a granular locking scheme, which captures dependency relationships between workflows and constructs, to support general hierarchical workflow compositions.

VI. Conclusions and Future Work

In this paper, we extended the reference architecture of SWFMS to support collaborative scientific workflow composition. We then proposed a formal granular scientific workflow locking scheme and proposed four algorithms for locking and releasing workflows and constructs. We conducted experiments to study its performance. While our proposed scheme guarantees the correctness of concurrent workflow update, in the future, we plan to propose techniques to further validate the consistency of the resulted workflow compositions. We also plan to study runtime issues of collaborative workflow orchestration and coordination.

Acknowledgment

This work is supported by National Science Foundation under grants NSF IIS-0959215 and IIS-0960014. The authors thank Sha Liu for her participation in the implementation of the experiments.

References

[1] G. Bell, T. Hey, and A. Szalay, "Beyond the data deluge," *Science*, vol. 323, no. 5919, pp. 1297–1298, 2009.

[2] S. Lu and J. Zhang, "Collaborative scientific workflows," in *Proc. of ICWS*, 2009, pp. 527–534.

[3] J. Zhang, "Co-Taverna: A tool supporting collaborative scientific workflows," in *Proc. of SCC*, 2010, pp. 41–48.

[4] T. Oinn, M. J. Addis, J. Ferris, D. Marvin, M. Senger, T. Carver, M. Greenwood, K. Glover, M. Pocock, A. Wipat, and P. Li, "Taverna: a tool for the composition and enactment of bioinformatics workows," *Bioinformatics*, vol. 20, no. 17, pp. 3045–3054, 2004.

[5] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. A. Lee, J. Tao, and Y. Zhao, "Scientific workflow management and the Kepler system," *Concurr. Comput. : Pract. Exper.*, vol. 18, no. 10, pp. 1039–1065, 2006.

[6] D. Churches, G. Gombas, A. Harrison, J. Maassen, C. Robinson, M. Shields, I. Taylor, and I. Wang, "Programming scientific and distributed workflow with Triana services," *Concurr. Comput. : Pract. Exper.*, vol. 18, no. 10, pp. 1021–1037, 2006.

[7] S. Callahan, J. Freire, E. Santos, C. Scheidegger, C. Silva, and H. Vo, "VisTrails: visualization meets data management." in *Proc. of SIGMOD*, 2006, pp. 745–747.

[8] E. Deelman, G. Singh, M. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, B. Berriman, J. Good, A. Laity, J. Jacob, and D. Katz, "Pegasus: a framework for mapping complex scientific workflows onto distributed systems," *Scientific Programming Journal*, vol. 13, no. 3, pp. 219–237, 2005.

[9] Y. Zhao, J. Dobson, I. Foster, L. Moreau, and M. Wilde, "A notion and system for expressing and executing cleanly typed workflows on messy scientific data," *SIGMOD Record*, vol. 34, no. 3, pp. 37–43, 2005.

[10] C. Lin, S. Lu, X. Fei, A. Chebotko, Z. Lai, D. Pai, F. Fotouhi, and J. Hua, "A reference architecture for scientific workflow management systems and the VIEW SOA solution," *TSC*, vol. 2, no. 1, pp. 79–92, 2009.

[11] K. Eswaran, J. Gray, R. Lorie, and I. Traiger, "The notions of consistency and predicate locks in a database system," *Commun. ACM*, vol. 19, no. 11, pp. 624–633, 1976.

[12] J. Gray and A. Reuter, *Transaction Processing - Concepts and Techniques*. Morgan Kaufmann, 1993.

[13] J. Zhang, D. Kuc, and S. Lu, "Confucius: A scientific collaboration system using collaborative scientific workflows," in *Proc. of ICWS*, 2010, pp. 575–583.

[14] J. Gray, R. Lorie, G. Putzolu, and I. Traiger, "Granularity of locks and degrees of consistency in a shared data base," in *IFIP Working Conference on Modelling in Data Base Management Systems*, 1976, pp. 365–394.

[15] S. Lu, A. Bernstein, and P. Lewis, "Correct execution of transactions at different isolation levels," *TKDE*, vol. 16, no. 9, pp. 1070–1081, 2004.

[16] X. Fei and S. Lu, "A dataflow-based scientific workflow composition framework," *TSC*, in press, 2011. [Online]. Available: http://www.cs.wayne.edu/~shiyong/papers/tsc11.pdf

[17] D. Chen and C. Sun, "Optional instant locking in distributed collaborative graphics editing systems," in *Proc. of ICPADS*, 2001, pp. 01–09.

[18] S. Bächle, T. Härder, and M. Haustein, "Implementing and optimizing fine-granular lock management for XML document trees," in *Proc. of DASFAA*, vol. 5463, 2009, pp. 631–645.