2010 IEEE International Conference on Web Services

Confucius: A Scientific Collaboration System Using Collaborative Scientific Workflows

Jia Zhang, Daniel Kuc
Department of Computer Science
Northern Illinois University
DeKalb, IL USA
jiazhang@cs.niu.edu

Shiyong Lu
Department of Computer Science
Wayne State University
Detroit, MI, USA
shiyong@cs.wayne.edu

Abstract—**Large-scale scientific data management and analysis usually relies on many distributed scientists with diverse expertise. In recent years, such a collaborative effort is often composed and automated into a dataflow-oriented process, a so-called scientific workflow. However, existing scientific workflow tools are single user-oriented and do not support collaborative scientific workflow composition, execution, and management among multiple distributed scientists. In this paper, we report our study of collaboration protocols towards building a tool supporting collaborative scientific workflow composition. Based on a scientific collaboration ontology, we propose a collaboration model supported by a set of collaboration primitives and patterns. The collaboration protocols are then applied to support effective concurrency control in the process of collaborative workflow composition.**

Keywords-collaborative scientific workflows; collaboration protocols; Taverna.

I. INTRODUCTION

The advancement of modern science has created sheer volume of data with increasing complexity. A phenomenon of data deluge is witnessed in each science domain, in which extreme scale of scientific data not only poses a grand challenge to data storage and access, but also to high-throughput data analysis and computation. Processing and managing such extreme-scale scientific data sets is usually beyond the realms of a single scientist to solve [1]; instead, it has to rely on many domain scientists with diverse expertise and from distributed locations. For example, the Large Synoptic Survey Telescope (LSST) experiment [2], which aims to repeatedly image half of the sky over a planned 10-year survey, will produce data at a rate of 300 MB/s, resulting in catalogs of about 130 TB for roughly 3×10^9 sources times 10 years worth of data. Analyzing and managing such extreme-scale data sets demand collaboration of a number of national labs and organizations with hundreds to thousands of scientists and engineers engaged [2, 3].

Meanwhile, such extreme-scale scientific data analysis and processing is usually composed and automated into a dataflow-oriented process, the so-called *scientific workflow*. In contrast to business workflows, which are control-flow oriented and orchestrate a collection of well-defined business tasks to achieve a business goal, scientific workflows are often dataflow-oriented and streamline a collection of scientific tasks to enable and accelerate scientific discovery [4, 5]. Scientists use scientific workflows to integrate and structure local and remote heterogeneous computational and data resources to perform *in silico* experiments [1, 6-8]. The increasingly important role of scientific workflows in modern science was recently reemphasized in an article that is published in *Science* and titled "Beyond the Data Deluge" [9]. The article concludes that, "in the future, the rapidity with which any given discipline advances is likely to depend on how well the community acquires the necessary expertise in database, *workflow management*, visualization, and Cloud computing technologies."

In short, scientific workflow and scientific collaboration are two key techniques to facilitate extreme-scale scientific data analysis and management. However, existing scientific workflow management systems (SWFMSs) [10-15] are single user oriented, focusing on helping individual scientists construct scientific workflows from available applications and services. Individual work artifacts (scientific workflows) are manually sent to collaborators (e.g., via emails) or uploaded into some shared social space (e.g., MyExperiment [16]) to enable scientific collaboration. For example, a collaborator can download a published workflow (e.g., in the format of Taverna [11], a popular scientific workflow tool) from MyExperiment, load it into her local Taverna workbench, update, and send the updated workflow back to the original collaborator for further changes.

To facilitate more interactivity between collaborators to better support exploratory collaborative data analysis and enable effective steering of the computational process in the context of scientific workflows, we have been developing a collaborative scientific workflow tool. Without reinventing the wheel, we examined a widely used single-user scientific workflow tool, Taverna [11], and extended it into a multi-user version.

In this paper, we present the preliminary results of our study of collaboration protocols supporting effective and efficient collaborative scientific workflow composition. Here we focus on multiple scientists cooperatively design and compose a common scientific workflow. We propose a scientific collaboration provenance ontology, and base on it, a collaboration model supported by a set of collaboration

978-0-7695-4128-0/10 $26.00 © 2010 IEEE
DOI 10.1109/ICWS.2010.103

primitives, and patterns. The collaboration protocols are then applied to support effective concurrency control in the process of collaborative workflow composition.

The remainder of the paper is organized as follows. In Section 2, we present related work. In Section 3, we introduce our scientific collaboration provenance ontology. In Section 4, we present our framework of collaboration protocols. In Section 5, we discuss composition concurrency control. In Section 6, we discuss system design and implementation, as well as our preliminary experiments. In Section 7, we conclude the paper.

II. Related Work

To date, several scientific workflow management systems (SWFMSs) have been developed as single-user environments, which run on local desktop computers or on Grids to help individual scientists construct scientific workflows from available resources. Representative SWFMSs include Kepler [17], Taverna [11], Triana [12], VisTrails [13], Pegasus [5], Swift [14], and VIEW [15, 18].

Kepler [17] is a Java-based open-source SWFMS, where a scientific workflow is composed of uniform components called *actors* and its execution is controlled by a dedicated computational model controller called *director*. Taverna [11] is an open-source SWFMS targeted for life science. Taverna adopts an XML-based workflow language called *SCUFL* to support workflow representation, each component being either a Web service or a Java Beanshell script-based processor supporting various bioinformatics data analysis and transformation. Triana [12] provides a sophisticated graphical user interface supporting workflow composition and modification activities, including grouping, editing, and zooming functions. VisTrails [13] focuses on workflow visualizations supporting provenance tracking of workflow evolution in addition to data product derivation history. Pegasus [5] provides a framework that maps complex scientific workflows onto distributed Grid resources. Artificial intelligence planning techniques are used for guiding workflow composition. Swift [14] combines a scripting language called *SwiftScript* with a powerful runtime system to support workflow specification and execution of large loosely coupled computations over the Grid environments. VIEW [15, 18] provides a tool that allows domain scientists to compose a scientific workflow from available resources and services. The system is featured with efficient provenance management by utilizing the power of relational databases [30].

Each of these SWFMSs provides a platform to support individual scientists in composing scientific workflows from various resources. Their foundations center on scientific workflow models and provenance models.

Some systems show some collaboration features, in the sense that they allow a scientist to compose a scientific workflow from shared resources and services, e.g., published Grid services. However, they provide limited support for multiple scientists to collaboratively compose and manipulate a shared scientific workflow. They do not address and support user interaction and cooperation that are required and essential for an effective and efficient scientific collaboration [19].

Some SWFMSs, such as Taverna [11], provide limited off-line collaborative scientific workflow composition. In such systems, researchers can publish their composed scientific workflows in a dedicated social workflow space (e.g., MyExperiment [16]); others using the same SWFMS can download the workflows, make changes, and upload the new version into MyExperiment to initiate further interactions. However, such SWFMSs do not support real-time shared scientific workflow composition.

The business community recently recognized the need of involving humans into business workflows and has developed a preliminary model [20]. However, the model is inapplicable to collaborative scientific workflows due to the fundamental differences between business workflows and scientific workflows. While business workflows are control flow oriented, scientific workflows are dataflow oriented. Furthermore, provenance data management for the reproducibility of scientific results is essential for scientific workflows but not for business workflows. Hence, scientific workflows pose a different set of requirements [6].

Sayah and Zhang [21] present their annotated business hyperchain technology enabling on-demand business collaboration with the Web services technology. They propose a set of business collaboration primitives serving business scenarios. In contrast, our collaboration primitives serve scientific collaboration scenarios. In addition, design-time collaboration provenance is automatically captured for credit acknowledgement as well as guiding future collaborative workflow composition.

We studied the state of the art of the field of scientific workflows toward the support of collaborative scientific workflows and reported our observations in [19]. We also have surveyed the literature of workflow control mechanisms in a collaborative environment in [22] and observed that current workflow control configurations have to be predefined and remain immutable throughout the execution of a workflow. With the rapid emergence of Services Computing technology [23], a workflow may select optimal available services (e.g., a specific data processing and analysis service) at runtime based on some QoS measurements. We conclude that workflow control should be driven by demand: it should be customizable and adaptive during runtime.

III. Scientific Collaboration Provenance Ontology

We develop a scientific collaboration provenance ontology to support the modeling of various traditional provenance information about scientific workflow and user interaction and collaboration patterns. The ontology is shown in Fig. 1.

Establishing a knowledge base, our collaboration provenance ontology is centered upon the concept of "workflow." Each scientific workflow comprises organized processors (tasks) and data links (aka. data channels), as well as predefined requirements and annotations (comments) dynamically generated. Each workflow maintains one or more floors that are tokens to ensure concurrency control.

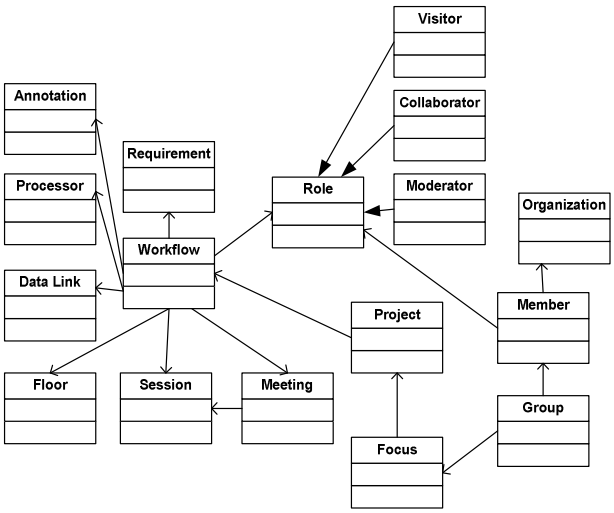

Figure 1. A scientific collaboration provenance ontology.

Long-term collaboration on a scientific workflow forms a *meeting*. A short-term synchronous collaboration is called a *session*.

Each scientific workflow belongs to a project. Each project belongs to a scientific group (could be a virtual group). Each group may comprise multiple focuses, each involving multiple projects. A group contains a set of members, each may belong to different organizations.

Collaboration on a scientific workflow is conducted by members serving in different roles. The initiator (creator) of a scientific workflow is called a *moderator*. Scientists who cooperate on the lifecycle of a workflow are called *collaborators*. They have read and write privileges. A collaboration may also involve *visitors*, who are granted with read privilege only.

Our scientific collaboration provenance ontology, which is extensible, serves as a foundation for managing collaboration provenance. Each scientific collaboration project may define customized ontology and add additional annotations into the basic ontology for special purposes. For example, a research project may introduce project-wise particular roles in their collaboration.

IV. COLLABORATION PROTOCOLS

A. Collaboration Patterns

1) Collaboration Model

Establishing a collaboration model is important to support effective human interaction and collaboration throughout the life cycle of scientific workflow composition. Such a collaboration model will be independent and can be dynamically plugged into other models to favor configurability and re-configurability. This requirement is critical for our tool to become generally applicable to various scientific collaboration projects. Different scientific research projects may adopt different collaboration rules and patterns. A fundamental collaboration model must be able to be configured to support these diverse collaboration rules and patterns, and then be plugged into a generic scientific workflow management system to support the corresponding research projects.

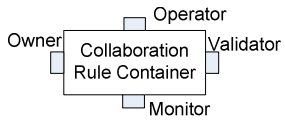

Figure 2. Collaboration rule container.

To regulate human interaction in collaborative scientific workflows, we propose a collaboration model as a 4-tuple container shown in Fig. 2:

$$C\_Rule =< Owner, Operator, Monitor, Validator >$$

The collaboration container comprises four basic plug-in roles: owner, operator, monitor, and validator. An owner role represents a group of scientists who have ownerships over a dataset or a task. An operator role represents a group of scientists who have the privilege to operate on a dataset or a task. A monitor role represents a group of scientists who have the privilege to monitor the operation process of a task or over a dataset. A validator role represents a group of scientists who have the privilege to validate an operation over a dataset or a task and claim the success of such an operation. One scientist may act in multiple roles simultaneously.

2) Collaboration Patterns

Significantly different from business collaboration, scientific collaboration is typically exploratory and unpredictable, thus requiring constant human interaction and intervention in the process. For example, a scientific workflow may not be able to be fully composed at the beginning; participating scientists may discuss and creatively decide subsequent actions in the middle of the process based on intermediate experimental results; new collaborators may join in the middle of an exploration if the need for specific expertise or domain knowledge arises; participating scientists may have different schedules and hence an asynchronous collaboration mode has to be supported in addition to a synchronous one; a scientific workflow may not have a clear time boundary and may last a long time, and so on.

Therefore, we have been studying scientific collaboration scenarios to identify data-centric collaboration patterns. We understand that in scientific collaboration, besides data to be computed, other artifacts are also important such as, *references* (for triggering discussions), *ideas* and *initiatives* (to move work forward), and *designs* (blueprint before experiments). In this paper, we only consider data-centric collaboration. As a starting point, we focus on *two-way collaboration patterns*, where only two scientists are involved in a collaborative activity. Our preliminary set includes six scientific collaboration patterns: (1) dataset request, (2) analysis request, (3) validation request, (4) discussion request, (5) co-run, and (6) co-approve.

Dataset request pattern reflects a scenario when some specific data is required, during the execution of a scientific workflow, to continue the exploration, while the dataset belongs to an external scientist group. Given a scientific

workflow *W*, scientist *A* asks for dataset *D* from scientist *B* before continuing.

Analysis request pattern reflects a scenario when some particular data obtained has to be analyzed by a specific tool or process that is owned by an external scientist group. Given a scientific workflow *W*, scientist *A* asks scientist *B* to analyze dataset *D*. Upon receipt of the request, *B* may conduct the analysis manually, using a tool or instrument, and return the analysis result to *A*. *A* can then continue with the original workflow execution.

Validation request pattern reflects a scenario when an interesting discovery is reached that requires verification and validation by a group of scientists with specific expertise to make a decision. In the context of a scientific workflow *W*, scientist *A* asks scientist *B* to validate a specific task *T* or dataset *D*. The result will be either positive or negative.

Discussion request pattern reflects a scenario when discussion is needed over some specific topics, and the results of the discussion will decide the direction (or steps) of the following action. In the context of a scientific workflow *W*, scientist *A* asks scientist *B* to discuss over a task *T* or a dataset *D*. At the end of the discussion, they will reach an agreement to decide the following steps.

Co-run pattern reflects a scenario when two scientists individually run some data analysis processes over the same dataset simultaneously. Given one dataset *D*, scientists *A* and *B* perform sub-workflows w_1 and w_2 concurrently and respectively, and then compare the results obtained from the two workflows corresponding to two alternative methods.

Co-approve pattern reflects a scenario when both scientists have to reach an agreement on an experimental result before its release. Scientists *A* and *B* need to approve each other to perform a workflow *W*.

These collaboration patterns can be represented using our proposed simple yet powerful collaboration model. For example, if applied to a dataset, this collaboration container can determine that certain scientists have the ownership over the dataset. For another example, if applied to a task representing a data analysis process, this collaboration container can determine that only certain scientists have the expertise and ownership to run the corresponding data analysis tool. These examples show that our proposed collaboration model can be applied to any workflow component (data products or tasks) to realize a fine-grained collaboration control. Note that our proposed collaboration model shows great flexibility: if a scientist reconfigures some parameters of a particular collaboration container at runtime, the collaboration policy affecting the scientific workflow may be changed accordingly.

B. Services-Oriented Collaboration Realization

Our goal is two-fold: one to facilitate communication between collaborators; the other is to enable collaboration provenance collection, meaning that the collaboration process is recorded and can be replayed later on. Thus, we construct a uniform collaboration message-based communication protocol.

1) Collaboration Primitives

TABLE I. COLLABORATION PRIMITIVES

Type	**Primitive Name**
Collaboration preparation primitives	Request for Dataset (RFD)
	Request for Data Analysis (RFA)
	Request for Validation (RFV)
	Request for Discussion (RFC)
	Request for Co-run (RFCR)
	Request for Co-approval (RFCA)
Collaboration conduction primitives	Accept or Reject Request (A/R)
	Command Submission (CS)
	Data Submission (DS)
	Update Submission (US)

Based on the collaboration patterns, we identified a set of semi-structured collaboration primitives, as summarized in Table 1. The primitives are divided into two categories: collaboration preparation primitives and collaboration conduction primitives. Since scientific collaboration may last a long period of time, we adopt an asynchronous communication mode, meaning that each collaboration primitive is associated with an instant acknowledgement.

Six collaboration preparation primitives are identified: (1) Request for Dataset (RFD), when a dataset is needed in the middle of a workflow; (2) Request for Data Analysis (RFA), when a data analysis process is needed in the middle of a workflow; (3) Request for Validation (RFV), when a data validation process is required; (4) Request for Discussion (RFC), when a discussion is required on a merged phenomenon; (5) Request for Co-run (RFCR), when concurrent sub-workflows are required; and (6) Request for Co-approval (RFCA), when an approval has to be made by multiple parties.

Four collaboration conduction primitives are identified: (1) Accept or Reject Request (A/R), when a request (e.g., RFD) is accepted or rejected by a collaborator; (2) Command Submission (CS), when a specific computational command is provided; (3) Data Submission (DS), when a specific data set is transferred; and (4) Update Submission (US), when a collaborator updates collaboration status in response to a request.

In addition to be used individually, these collaboration primitives serve can be used as building blocks for collaborators to model comprehensive collaboration patterns.

2) Collaboration Mini-Workflow

Based on the identified collaboration primitives, we apply the concept of Service Oriented Architecture to implement the collaboration patterns. Each collaboration pattern is accomplished by a mini-workflow comprising a set of configured collaboration primitives. Through different combinations over the set of collaboration primitives, different collaboration patterns can be realized.

We have constructed six example mini-workflows to realize the six collaboration patterns described in Section 4.1.2. (1) dataset request: comprising the RFD primitive,

```
<process name="RFDmicroflow"
   targetNamespace="urn:CollaborationConstructs"
   xmlns:tns="urn:samples:CollaborationConstructs"
   xmlns="http://confucius.org/constructs/">

   <sequence>
      <invoke name="invokeRFD"
         partner="CollaboratorA" portType="tns:RFDoriginatorPT"
         operation="sendRFD" outputVariable="RFD">
      </invoke>

      <invoke name="ackRFD"
         partner="CollaboratorB" portType="tns:RFDreceiverPT"
         operation="ackRFD" outputVariable="RFD_Receipt_Ack">
      </invoke>

      <invoke name="acceptRFD"
         partner="CollaboratorB" portType="tns:RFDreceiverPT"
         operation="acceptRFD " outputVariable="A">
      </invoke>

      <invoke name="ackAcceptRFD"
         partner="CollaboratorA" portType="tns:RFDoriginatorPT"
         operation="ackAcceptRFD" outputVariable="A_Receipt_Ack">
      </invoke>

      <invoke name="invokeDS"
         partner="CollaboratorB" portType="tns:RFDreceiverPT"
         operation="submitDS" outputVariable="DS">
      </invoke>

      <invoke name="ackDS"
         partner="CollaboratorA" portType="tns:RFDoriginatorPT"
         operation="receiveDS" outputVariable="DS_Receipt_Ack">
      </invoke>
   </sequence>
</process>
```

Figure 3. Service-oriented mini-workflow.

A/R primitive, and DS primitive; (2) analysis request: comprising the RFA primitive, A/R primitive, and CS primitive; (3) validation request: comprising the RFV, A/R, and CS primitive; (4) discussion request: comprising the RFC primitive and a collection of US primitives; (5) co-run: comprising the RFCR primitive, A/R primitive, and US primitive; and (6) co-approve: comprising the RFCA primitive, A/R primitive, and US primitive.

Such a mini-workflow can be formalized using the Business Process Execution Language (BPEL). Since BPEL is based on Pi-calculus, modeling mini-workflows in BPEL will allow us to formally reason about the construction of a new mini-workflow. Taking the first collaboration pattern (dataset request) as an example, Fig. 3 shows a section of its BPEL definition. For simplicity, we skipped the section defining messages (collaboration messages), partners (collaborators A and B), and variables (messages), and links (expressing synchronization dependencies).

As shown in Fig. 3, each collaboration primitive is wrapped as a Web service. Two parties (Collaborators A and B) act as service providers and service requestors alternatively. Each collaboration primitive is realized by a service call, associated with the corresponding messages. Once represented by BPEL, multiple collaboration

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            targetNamespace="http://confucius.org/class"
            xmlns:tns="http://confucius.org/class">
   <xsd:element name="Name" type="xsd:string"/>
   <xsd:element name="task" type="xsd:anyURI"
default="http://confucius.org/class#Task"/>
   <xsd:element name="workflow" type="xsd:anyURI"
default="http://confucius.org/class#Workflow"/>
   <xsd:element name="project" type="xsd:anyURI"
default="http://confucius.org/class#Project"/>
   <xsd:element name="Construct">
      <xsd:annotation>
         <xsd:documentation> A construct is the atomic unit of
collaborative work in a scientific workflow. </xsd:documentation>
      </xsd:annotation>
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element ref="tns:Name"/>
            <xsd:element ref="tns:task"/>
            <xsd:element ref="tns:workflow"/>
            <xsd:element ref="tns:project"/>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
</xsd:schema>
```

Figure 4. Schema for a transaction.

constructs may combine to form a comprehensive collaboration scenario. Such a service-oriented model enables platform-neutral and language-neutral collaboration.

3) Service-Oriented Collaboration Provenance

Provenance has been widely considered critical to the reproducibility of scientific workflows [24, 25]. Compared to existing significant amount of work focusing on provenance for scientific workflow execution, our work focuses on collaboration provenance tracking human interactions in the process of scientific workflow composition. Our method is to record all collaborative activities leading to a composed workflow.

We decided to adopt the Web services technology [23] to realize collaboration mini-workflows. As the best enabling technology of Service Oriented Architecture (SOA) to date, the Web services technology allows us to enable universal communications among participating scientists with platform independence and language independence. Collaboration primitives are encapsulated in Simple Object Access Protocol (SOAP) messages and communicated between collaborators. To enable validation and analysis, we adopt the XML Schema to uniform the format of collaboration messages. Fig. 4 shows a section of the specification of a collaboration message.

Messages are divided into request messages and response messages. Each message contains one or more primitives that form a transaction, meaning that they form an atomic unit of work in a scientific workflow. Each transaction aims to serve for a task in a workflow, which belongs to a scientific project. A message may also contain optional data such as annotations.

V. COMPOSITION CONCURRENCY CONTROL

The lifetime of a collaborative scientific workflow may last for a long period of time, thus the concurrency control over its different phases deserves consideration. In this paper, we discuss concurrency control at workflow composition time.

A. *Locking Granularity*

At composition time, multiple scientists collaborate to develop a scientific workflow. Without reinventing the wheel, our previous work extended Taverna, a popular scientific workflow tool, into a collaborative version [26]. The reason why we chose Taverna is mainly based on its popularity and big user base [11]. Another reason is that Taverna is an open-source tool developed in Java. Thus we can explore its code and turns it into a collaborative version. Adopting the instrument from an extensively tested and well proved human communication protocol, Robert's Rules of Order (RRO) [27], we establish a floor control mechanism. Each scientific workflow maintains a single floor (token), which can be assigned to one collaborator at a time. Each collaborator requests and competes for the floor. Only the collaborator holding the floor can propagate their changes on the shared workflow. After done with the update, the collaborator can release the floor and other collaborators may get it.

Such a workflow-level floor control may not be efficient to support large-scale scientific workflow composition. Since scientific research is an exploratory process, the development of a scientific workflow may undergo many discussions and changes and may last for a long period of time. Meanwhile, a collaboration group nowadays usually comprises scientists from different organizations at distributed locations. They may possess different schedules and may even reside in different time zones; thus, their collaboration may adopt both synchronous and asynchronous modes. Furthermore, a large-scale scientific workflow may involve many comprising components. It is neither efficient nor practical, if one scientist working on one component locks the entire workflow and other scientists cannot work on unrelated components.

To increase composition concurrency, we investigate the option of locking the smallest building blocks. A scientific workflow allows multiple un-overlapped locks, so that multiple scientists may work on the locked components simultaneously.

According to the existing scientific workflow management tools, the smallest building blocks in a scientific workflow are tasks and data channels. In Taverna, a task is called a *processor*; the data channels linking between processors are called data links. Fig. 5 is a highly simplified scientific workflow drawn in Taverna, which illustrates a word count example using the MapReduce programming model [28]. Two processors *Mappers* repeatedly process a list of word lines, by breaking each line into individual words and generating a list of <word, 1> pairs over all the words found. All the intermediate <word, 1> pairs are transferred, through the data links, to the processor *Reducer* that aggregates the pairs according to the words. The results are a list of <word, value> pairs that show the number of appearances of each word.

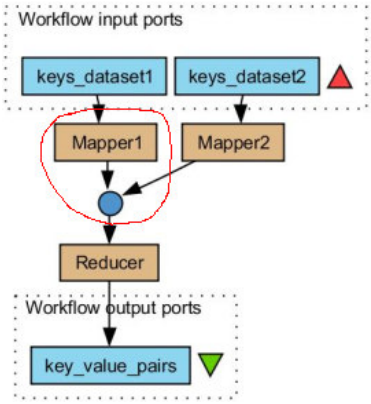

Figure 5. Word count workflow.

If we set up the locks on individual processors and data links only, two collaborators may concurrently update one processor *Mapper1* and its output data links, respectively. This situation may not be desirable, because the data link directly depends on the processor. In other words, connected processors and data links may have close semantic relationships, which need to be preserved by requiring that neighboring entities cannot be updated by different collaborators at the same time.

Furthermore, adjacent processors in a workflow may also possess semantic relationships between them. For example, as shown in Fig. 5, in which triangles represent workflow inputs and outputs, the *Mapper1* processor and the *Reducer* processor are neighbors in the workflow, and a data link connects them together. The *Reducer* processor stays at the downstream of the workflow; meaning that the output of the *Mapper1* processor servers as the input of the *Reducer* processor. Assume that two collaborators are working on the two processors simultaneously, and collaborator *A* changes some business logic at the *Mapper1* processor. Even if these changes may not change the input of the *Reducer* processor, the collaborator working on the *Reducer* processor should be aware that someone is working on the upstream processor.

Therefore, we propose a concept of *synchronization area* that represents a conceptual area in a shared scientific workflow, which allows only one collaborator to work on it at a given time. Such an area represents a dynamic semantic area. In the context of a Taverna workflow, if a user tries to lock a data link, the synchronization area is the data link. If a user tries to lock a processor, the synchronization area is dynamically delimited and includes all of the fan-out data links of the processor. In Fig. 5, the manually drawn red circle around the *Mapper1* processor and its fan-out data link represents such a synchronization area.

B. *Locking Algorithms*

Based on the concept of synchronization area, we built four algorithms, on locking/releasing a processor and locking/releasing a data link. The algorithms are shown as below.

Algorithm 1: Lock Processor
Input: A user selects a processor and presses "lock"
Requirements: Lock a processor.
1: **if** processor ∈ locked processor list **then** do nothing;
2: **else**
3: **begin transaction**
4: processor_owner ← self; lock_flag ← 1
5: **for** each outgoing data link
6: **call** lock_data_link
7: **if** return = false **then abort**
8: **end transaction**
9: **endif**

Algorithm 2: Release Processor
Input: A user selects a processor, presses "release"
Requirements: Unlock a processor.
1: **if** processor owner ≠ self **then** do nothing;
2: **else**
3: **begin transaction**
4: set processor.lock_flag ← 0
5: **for** each outgoing data link
6: **call** release_data_link
7: **end transaction**
8: **endif**

Algorithm 3: Lock Data Link
Input: A user selects a data link and presses "lock"
Requirements: Lock a data link.
1: **if** data link ∉ database **then** insert
2: **if** data link.lock_flag = 1 **then return** false
3: **else** data_link.owner ← self; lockFlag ← 1
4: **endif**

Algorithm 2: Release Data Link
Input: A user selects a data link, presses "release"
Requirements: Unlock a data link.
1: **if** data link owner ≠ self **then** do nothing;
2: **else** set data_link.lock_flag ← 0
3: **endif**

If a user selects a processor and clicks to lock the processor, we first check whether it has been locked by another collaborator. If nobody locks it, then an uninterruptable transaction starts. First, we set the lock flag of the processor, and fill the name of the owner of the processor. For each outgoing data link of the processor, we check whether there is an active lock on it. If any outgoing data link is currently locked by other collaborators, the entire locking attempt is aborted. Otherwise, we call the corresponding algorithm to lock the data link. After all outgoing data links are locked, the transaction succeeds. In summary, the lock action will automatically lock all downstream data links, in addition to the processor.

To release a processor, we will first check whether the user has the privilege, i.e., whether she is the owner of the processor. If the answer is positive, in addition to the processor itself, the action will call the corresponding algorithm to release all of the downstream data links.

To lock a data link, we first check whether the data link has been uploaded into the database (here we adopt a lazy instantiation pattern for a higher performance). After ensuring that the data link is in the database, we check whether it has already been locked. If not, the data link will be marked as being locked. Otherwise, a notification will be sent.

To release a data link, we first check whether the user has the privilege, i.e., whether she is the owner of the data link. If the answer is positive, the flag of the data link will be set as unlocked.

C. Collaboration Transactions

Our locking algorithms facilitate concurrent workflow composition. Actions by each user can be modeled as transactions to ensure concurrency control. We define four basic actions (in Taverna context): 1) insert a data link, 2) delete a data link, 3) insert a processor, and 4) delete a processor. An update action can be modeled as a delete followed by an insert. Thus, all collaborative composition actions can be mapped to database update operations. As a result, we can exploit the concurrency control facility of database management systems to ensure the serializability of all executions. Bad transactions will be automatically aborted. We are working on an exception handling facility here; which is out of the scope of this paper. After a user update is successfully committed, all collaborators will be notified, so that each collaborator can have the most-up-to date workflow.

At the database level, to support concurrent updates of scientific workflow tasks by a distributed group of scientists, we implemented *a READ COMMITTED with first-committer-win (RC-fcw)* scheme [29], which is an extension of READ COMMITTED with the first-committer-win feature from the SNAPSHOT isolation level. In RC-fcw, transactions obtain long-term write locks on items and short-term read locks. In addition, if T_1 commits to writing a data item between the time period when T_2 has read and attempted to write the same item, T_2 will be aborted (first-committer-wins). We implemented RC-fcw using the following strategy. Each task stored in a database is associated with a version number. When a transaction reads a task and intends to update the same task at a later time, a version comparison is triggered to check whether any other transaction has updated the task in between. The check and the update together are performed atomically.

VI. System Design and Experiments

A. System Implementation

We built a collaboration pattern template library. The basic building blocks are collaboration primitives. Users can build new collaboration patterns using existing collaboration primitives. Identified collaboration patterns are stored as provenance data to support the tracking, storing, and querying of interactions and coordination among scientists.

We built a central server supporting all workflow collaborations. Workflow evolution provenance and collaboration provenance are stored in a shared database on the server. Each collaborator may store an intermediate version of the workflow on the local machine, but all committed activities are stored at the server, in order to support asynchronous collaboration where collaborators may decide to work on the shared workflow at preferable time. We consider four options for selecting database systems: native XML, relational, XML-relational, and RDF. Currently we use a relational database because it is the preferred choice

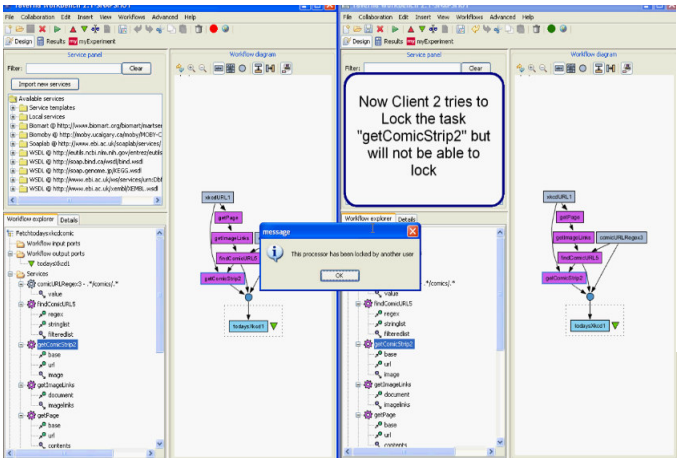

Figure 6. Screen shots of concurrent workflow updates.

for Taverna.

Fig. 6 shows a snapshot of our Confucius system supporting concurrent workflow composition. To ease illustration, we show two screens (left and right) representing two scientists running two client versions of Confucius on two distributed machines. Here we use remote desktop feature of Windows to show the two screens together. When a scientist write locks a task on the shared workflow, the other scientist cannot update the task due to our concurrency control.

B. *Concurrency Control Experiments*

We have designed and conducted a series of experiments to evaluate our *READ COMMITTED with first-committer-win (RC-fcw)* scheme for supporting concurrent updates of scientific workflow tasks by a group of scientists. Here we report our preliminary experimental study.

For experimental settings, without losing generality, we adopt a randomly generated scientific workflow comprising 20 tasks. Each collaborator is simulated by an independent Java thread, which iteratively reads a random task of the workflow, waits for a randomly generated time interval, and then performs an update on the task. While each collaborator indefinitely performs such iterative random updates, we record the total numbers of both successful task updates and unsuccessful task updates (due to abort), respectively, within a predefined time window. All experiments were conducted on a PC with Intel Core 2 Duo CPU P8800, @2.66 GHz & 2.76GHz and 3 GB main memory, running the Windows 7 Home Premium operating system. The database system used is Apache Derby 10.5.3.0. The database is installed in an embedded fashion for this experiment, so that no data transportation time is considered.

Our experiments focus on testing the throughput of the Confucius system by varying the number of collaborators. The throughput is defined as the number of successful task updates per minute. The average throughput is calculated for each collaboration group size of N (10, 20, …, 100). For each group size, the experiment is repeatedly performed 10 times with the average calculated. We also monitor the number of failed task updates performed per minute to show

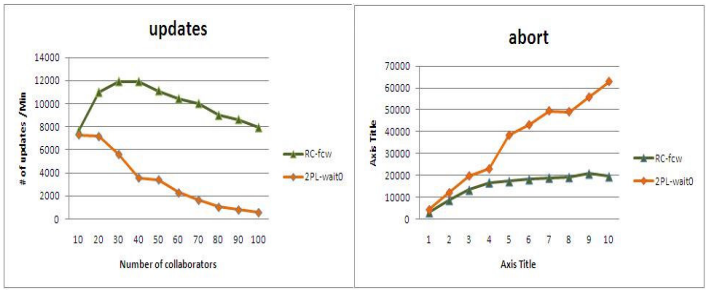

Figure 7. Test results of throughputs and failed task updates.

the trend of update conflicts as the number of collaborators increases.

Fig. 7 shows how the number of successful task updates per minute for varying number of collaborators, from 76,242 task updates per minute for 10 collaborators, to 79,015 task updates per minute for 100 collaborators. We can see that the collaboration productivity (represented by the throughout) is steadily increased as the number of concurrent scientist collaboration increases, reaching a maximum of 119,207 task updates per minute at a group size of 30. Afterwards, the group productivity starts to decline due to the increase of conflicts that leads to abortion.

Fig. 7 also releases that as the number of collaborators increases, the number of conflicts and hence the number of failed task updates per minute also increases monotonically. When the size of a group is smaller than 30, such an increasing number of unsuccessful task updates is more than compensated by the increased number of successful task updates as a result of increased collaborators, assuming a constant productivity for each collaborator. However, when the size of the group goes beyond 30, the conflicts start to dominate - an additional collaborator only decreases productivity as she introduces less successful task updates than the number of failed task updates that she causes due to increased conflicts.

We also compare our RC-fcw scheme with the Two Phase Locking No Wait (2PL-wait0) scheme, which aborts a transaction right away if the requested lock is not available. In Fig. 7, the experimental results using our scheme are shown in green color; and those using the 2PL-wait0 are shown in brown color. We can see that our RC-fcw approach surpasses the 2PL-wait approach supporting scientific collaboration. While the 2PL-wait approach supports up to 15 concurrent scientists, our approach supports up to 30 concurrent scientists. Fig. 7 also shows that our TC-fcw approach bears much lower abort rate, when the number of concurrent scientists increases.

In summary, from a concurrency control point of view, there exits an optimal number for the group size that optimizes the productivity of the system. How to increase such a number, which is the ideal speedup of productivity, is an interesting and challenging open research problem. We plan to further study this problem in our future research.

VII. CONCLUSIONS AND FUTURE WORK

In this paper, we presented our ongoing work on establishing collaboration protocols to support collaborative

scientific workflow composition. Our framework includes a collaboration ontology associated with a set of collaboration patterns, primitives, and constructs, and a number of concurrent control mechanisms to support concurrent collaborative workflow composition.

Based on the ontology, we plan to enhance collaboration provenance management performance by extending our previous work on provenance [30] to support efficient collection, storage, and querying of collaboration provenance, leveraging existing relational, RDF, and XML database techniques. Furthermore, we plan to conduct more experiments to study the effects of tuning various parameters on concurrent productivity. For example, we plan to tune the number of concurrent collaborators, the productivity of individual members, the number of tasks comprised in the shared scientific workflow, and so on.

VIII. ACKNOWLEDGEMENT

This work is supported by National Science Foundation, under grants NSF IIS-0959215 and IIS-0960014. The authors also like to thank Sha Liu for the assistance in the experimental study presented here.

IX. REFERENCES

[1] G.M. Olson, A. Zimmerman, and N. Bos, *eds.,* Scientific Collaboration on the Internet. MIT Press, Cambridge, MA, USA, 2008.

[2] LSST, "Large synoptic survey telescope," 2009, Available from: http://www.lsst.org/lsst/science.

[3] LHC, "Large Hadron Collider," 2010, Available from: http://public.web.cern.ch/Public/en/LHC/Computing-en.html.

[4] B. Ludäscher, "Scientific workflows: cyberinfrastructure for e-Science," Proc. of PNC, Oct. 19, 2007, Berkeley, CA, USA, pp.

[5] Y. Gil, E. Deelman, J. Blythe, C. Kesselman, and H. Tangmunarunkit,"Artificial intelligence and grids: workflow planning and beyond", IEEE Intelligent Systems, Jan.-Feb., 2004, 19(1): pp. 26–33.

[6] E. Deelman and Y. Gil, "NSF workshop on the challenges of scientific workflows," May 1-2, 2006.

[7] S. Wuchty, B. Jones, and B. Uzzi, "The increasing dominance of teams in production of knowledge," *Science*, 2007, 316: pp. 1036-1039.

[8] N.R. Council, Facilitating Interdisciplinary Research, 2004, National Academies Press, Washington DC, USA.

[9] "Beyond the Data Deluge", *Science*, 2009, 323(5919): pp. 1297-1298.

[10] 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," Concurrency and Computation: Practice & Experience, 2006, 18(10): pp. 1039-1065.

[11] T. Oinn, M. Greenwood, M. Addis, M.N. Alpdemir, J. Ferris, K. Glover, C. Goble, A. Goderis, D. Hull, D. Marvin, P. Li, P. Lord, M.R. Pocock, M. Senger, R. Stevens, A. Wipat, and C. Wroe, "Taverna: lessons in creating a workflow environment for the life sciences," Concurrency and Computation: Practice & Experience, 2006, 18(10): pp. 1067–1100.

[12] 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," Concurrency and Computation: Practice & Experience, 2006, 18(10): pp. 1021–1037.

[13] J. Freire, C.T. Silva, S.P. Callahan, E. Santos, and C.E. Scheidegger, "Managing rapidly-evolving scientific workflows," LNCS, May, 2006, 4145/2006: pp. 10–18.

[14] Y. Zhao, M. Hategan, B. Clifford, I. Foster, G. Laszewski, V. Nefedova, I. Raicu, T. Stef-Praun, and M. Wilde, "Swift: fast, reliable, loosely coupled parallel computation," Proc. of SWF, Jul. 9-13, 2007, Salt Lake City, UT, USA, pp. 199–206.

[15] A. Chebotko, C. Lin, X. Fei, Z. Lai, S. Lu, J. Hua, and F. Fotouhi, "VIEW: a visual scientific workflow management system," Proc. of SWF, Jul., 2007, Salt Lake City, UT, USA, pp. 207–208.

[16] D.D. Roure, C. Goble, and R. Stevens, "The design and realisation of the myExperiment virtual research Environment for social sharing of workflows," FGCS, 2009, 25: pp. 561-567.

[17] 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," Concurrency and Computation: Practice and Experience, 2006, 18(10): pp. 1039-1065.

[18] C. Lin, S. Lu, Z. Lai, A. Chebotko, X. Fei, J. Hua, and F. Fotouhi, "Service-oriented architecture for VIEW: a visual scientific workflow management system," Proc. of SCC, Jul. 9-11, 2008, Honolulu, HI, USA, pp. 335-342.

[19] S. Lu and J. Zhang, "Collaborative scientific workflows," Proc. of ICWS, Jul. 6-10, 2009, Los Angeles, CA, USA, pp. 527-534.

[20] A. Agrawal, M. Amend, M. Das, M. Ford, C. Keller, M. Kloppmann, D. König, F. Leymann, R. Müller, K. Plösser, R. Rangaswamy, A. Rickayzen, M. Rowley, P. Schmidt, I. Trickovic, A. Yiu, and M. Zeller, "WS-BPEL extension for people (BPEL4People), version 1.0," Jun., 2007.

[21] J.Y. Sayah and L.-J. Zhang, "On-demand business collaboration enablement with services computing," Decision Support Systems, Jul., 2005, 40(1): pp. 107-127.

[22] C.K. Chang, J. Zhang, and K.H. Chang, "Survey of computer supported business collaboration in support of business processes," International Journal of Business Process Integration and Management (IJBPIM), 2006, 1(2): pp. 76-100.

[23] L.-J. Zhang, J. Zhang, and H. Cai, Services Computing. Springer, 2007.

[24] A. Chapman, H.V. Jagadish, and P. Ramanan, "Efficient provenance storage", Proc. of SIGMOD, Jun. 9-12, 2008, Vancouver, Canada, pp. 993-1006.

[25] M.K. Anand, S. Bowers, T.M. McPhillips, and B. Ludäscher, "Efficient provenance storage over nested data collections", Proc. EDBT, 2009, pp. 958-969.

[26] J. Zhang, "Co-Taverna: a tool supporting collaborative scientific workflows", Proc. of SCC, Jul. 5-10, 2010, Miami, FL, USA.

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

[28] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," Proc. of OSDI, 2004, pp. 137–150.

[29] A.J. Bernstein, P.M. Lewis, and S. Lu, "Semantic conditions for correctness at different isolation levels," Proc. ICDE, Feb. 28-Mar. 3, 2000, San Diego, CA, USA, pp. 57-66.

[30] A. Chebotko, X. Fei, C. Lin, S. Lu, and F. Fotouhi, "Storing and querying scientific workflow provenance metadata using an RDBMS," Proc. e-Science, Dec. 10-13, 2007, Bangalore, India, pp. 611–618.