Future Generation Computer Systems 25 (2009) 568–576

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Atomicity and provenance support for pipelined scientific workflows

Liqiang Wang[a,*], Shiyong Lu[b,*], Xubo Fei[b], Artem Chebotko[b,d], H. Victoria Bryant[a], Jeffrey L. Ram[c]

[a] Department of Computer Science, University of Wyoming, USA
[b] Department of Computer Science, Wayne State University, USA
[c] Department of Physiology, Wayne State University, USA
[d] Department of Computer Science, University of Texas - Pan American, USA

ARTICLE INFO

Article history:
Received 15 November 2007
Received in revised form
6 May 2008
Accepted 24 June 2008
Available online 5 July 2008

Keywords:
Scientific workflows
e-Science
Atomicity
Provenance
Transaction

ABSTRACT

Today many significant scientific discoveries are achieved through complex and distributed scientific computations that are structured and represented as scientific workflows. Although atomicity is a well studied topic in transaction processing and business workflows, such an important capability needs to be revisited in a scientific workflow environment. Firstly, the semantics of atomicity needs to be defined in a dataflow-oriented scientific workflow model, particularly for pipelined execution of hierarchical scientific workflows. Secondly, in a scientific workflow environment, atomic regions are specified or inferred dynamically as needed and are committed implicitly, which are in contrast to *a priori* well-defined transaction boundaries and explicit commits in transaction processing and business workflows. Finally, although atomicity and provenance are related to each other, their interactions and relationships have never been explored in the literature. In this paper, we propose: (i) an architecture for scientific workflow management systems that supports both provenance and atomicity; (ii) a dataflow-oriented atomicity model that supports the notions of commit and abort; and (iii) a dataflow-oriented provenance model that supports querying and visualizing provenance.

1. Introduction

In recent years, more and more scientists use scientific workflows [30,33,19,16,50,11,27] to integrate and structure various local and remote heterogeneous data and service resources to perform *in silico* experiments to produce significant scientific discoveries. As a result, scientific workflows have become the de facto cyberinfrastructure upper-ware for e-Science [28]. While business workflows are control-flow-oriented, scientific workflows tend to be dataflow-oriented and frequently need to access large amounts of scientific datasets [30,19].

Atomicity is an important transactional property, which requires that a transaction either runs to completion or has no partial effect (all-or-nothing). In scientific workflows, some tasks might fail during execution due to either the failure of the task itself or inappropriate input to a task. A domain scientist might require the execution of a sub-workflow to be atomic in the sense that either the execution of all the tasks of the sub-workflow runs to completion or none of them has any effect at all.

Traditional techniques for atomicity in transaction processing systems are inappropriate for complex long-running processes in distributed and heterogeneous environments. Compensation is generally considered a proper way to handle rollback in business workflows [18], as it can eliminate effects of already committed transactions. The atomicity techniques based on compensation in business workflows [26,14] are not suitable for scientific workflows. They require the explicit definitions of transaction boundaries which are often obscured in scientific workflows due to the data dependency introduced by pipelined execution (*i.e.*, next task uses input while previous task has not completed). Moreover, since scientific workflows are often computation-intensive, traditional rollback techniques are inefficient because the intermediate results of aborted transactions, which might be reusable in the future, are discarded.

Data provenance is closely related to the data lineage problem [8,13] studied in the database community, which determines the source data that are used to produce a data item. While Buneman et al. focus on the analysis of provenance of a tuple t produced by a single query Q executed over database D [13], Cui and Widom propose algorithms for lineage tracing of data warehouse data in the presence of general data warehouse transformations [13]. These approaches can be used for provenance analysis in scientific workflows when the process applied to a data product consists of database operations, such as SQL queries or data warehouse transformations. However, a scientific workflow typically

* Corresponding author.
E-mail addresses: wang@cs.uwyo.edu (L. Wang), shiyong@wayne.edu (S. Lu), xubo@wayne.edu (X. Fei), artem@cs.panam.edu (A. Chebotko), hbryant@cs.uwyo.edu (H. Victoria Bryant), jeffram@med.wayne.edu (J.L. Ram).

0167-739X/$ – see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.future.2008.06.007

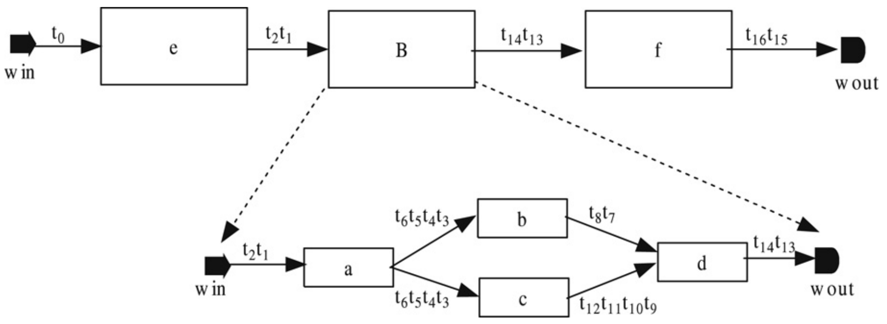


Fig. 1. An example of hierarchical scientific workflow.

consists of computational and analytical steps that are more complex than database operations. On the other hand, although several provenance models [22,37,6,12,3] have been proposed for scientific workflows, none of them supports the notion of atomicity.

This paper proposes a novel dataflow-oriented atomicity and provenance system for scientific workflows. To the best of our knowledge, our system is the first one that supports both atomicity and provenance. Our system highlights the following features: (i) an architecture that supports both provenance and atomicity; (ii) a dataflow-oriented hierarchical atomicity model that supports the notions of commit and abort; and (iii) a dataflow-oriented provenance model to support querying and visualizing provenance.

2. The atomicity management subsystem

2.1. A hierarchical scientific workflow

Our approach is based on a dataflow-oriented workflow model [5,34], in which each workflow consists of automatic tasks connected to each other by data channels. A workflow task is modeled as an *actor*. There are two kinds of actors: *atomic* and *composite*. An actor is *atomic* if it is always treated as a whole (*i.e.*, contains no sub-actors). An atomic actor can have *input ports* and *output ports* that provide the communication interfaces to other actors. A *composite actor* is a set of actors, which are either atomic actors or other composite actors. For a composite actor a and its sub-actor a_i, *i.e.*, $a_i \in a$, an input port of a_i is also an input port of a if that input port is connected with another actor not contained in a. An output port for composite actors is defined similarly. Let $InPorts(a)$ and $OutPorts(a)$ denote all input ports and all output ports of actor a, respectively.

Actors communicate by passing *data tokens* (called *token* for short) between their ports. Each token is unique in the whole workflow. For two actors a and b, a *data channel* from a to b provides the communication medium for a dataflow of tokens from an output port of a to an input port of b.

A *workflow* $w = \langle A, C\rangle$ consists of a set A of actors, and a set C of data channels between actors of A. Similar to the notion of user views introduced in [12,3], a workflow can have different views based on different observation levels on each composite actor. An atomic actor has a single view. A view for a composite actor is either a "zoom-out" view (*i.e.*, the composite actor itself) or a "zoom-in" view that consists of views for *each* sub-actor. Note that a "zoom-in" view considered more details for a composite actor, but still as a whole, instead of part of it. For a workflow $w = \langle A, C\rangle$, a *view* of w is $\langle A', C'\rangle$, where $A' \in View(A)$, and C' consists of data channels that connect the views of A'.

For example, Fig. 1 shows a hierarchical scientific workflow. Composite actor B consists of atomic actors a, b, c, d. A "zoom-in" view considers the details of all atomic actors in B. A "zoom-middle" view considers B as a whole. A "zoom-out" view considers all actors in the workflow as a whole.

This paper makes some assumptions about the scientific workflows that we analyze. Firstly, scientific workflows execute in a pipelined fashion. Secondly, each actor is "white", *i.e.*, data dependencies between input tokens and output tokens are observable. Thirdly, message-send-response relationships between actors and services are known. Fourthly, tokens are not shared by actors. Specifically, if two actors need to read the same data input, we duplicate the data into two tokens. Fifthly, each retriable Web service is modeled as a local actor, which calls the remote Web service on behalf of the user. Thus, the execution of all tasks are performed in a local machine except the execution of Web or Grid Services.

2.2. Round and data dependency

In our atomicity model, a workflow execution invokes a series of actors to run. Each actor maintains a state which stores intermediate results computed from input tokens. A state indicates some data dependencies between the output tokens and the input tokens. A state is flushed by calling `reset()`.

A *round* r on an actor a, denoted by $a.r$, contains the whole events that happen between two consecutive reset events (*i.e.*, no other reset events in the middle). Formally, a round is $\langle Id_a, I, O, D\rangle$, where Id_a is the identifier of the actor where the round occurs, I is the set of input tokens for the round, O is the set of output tokens for the round, $D \subseteq I \times O$ is the set of dependency relationships with each $(i, o) \in D$ representing that output token o depends on input token i. Let $input(a.r)$ and $output(a.r)$ denote all input tokens and output tokens of $a.r$, respectively.

The call of `reset()` is a non-blocking operation. A reset event terminates the current round of data dependencies, and starts a new round of data dependencies. Each round has a unique identifier in the workflow itself. Thus, an invocation of a workflow contains a series of invocations on actors; each invocation contains one or more rounds. A round is decided by each actor itself. When an actor calls `reset()`, it tells the workflow engine that the current round has completed. For each output token in a round, we assume that the actor can tell what input tokens that it is dependent on. Note that these dependent tokens should be some of the input tokens read so far, but may not be the all.

A round can be defined explicitly, *i.e.*, by calling `reset()`, or inferred automatically. To infer rounds, we analyze the definition of a workflow for data dependencies statically (*i.e.*, based on source code) or dynamically (*i.e.*, runtime monitoring data flows during execution). Automatic inference of rounds is in our future work.

Fig. 2(a) shows an example of rounds based on the scientific workflow in Fig. 1. Round $e.r_1$ consumes t_0 and produces tokens t_1 and t_2. There are two rounds on actor B, $B.r_1$ consumes t_1 then produces t_{13}; $B.r_2$ consumes t_2 then produces t_{14}. Similarly, actor f has two rounds $f.r_1$ and $f.r_2$.

For two tokens t_1 and t_2, if t_2 is computed from t_1, we say t_2 *depends* on t_1, denoted $t_1 \rightarrow t_2$. Token dependencies are transitive,

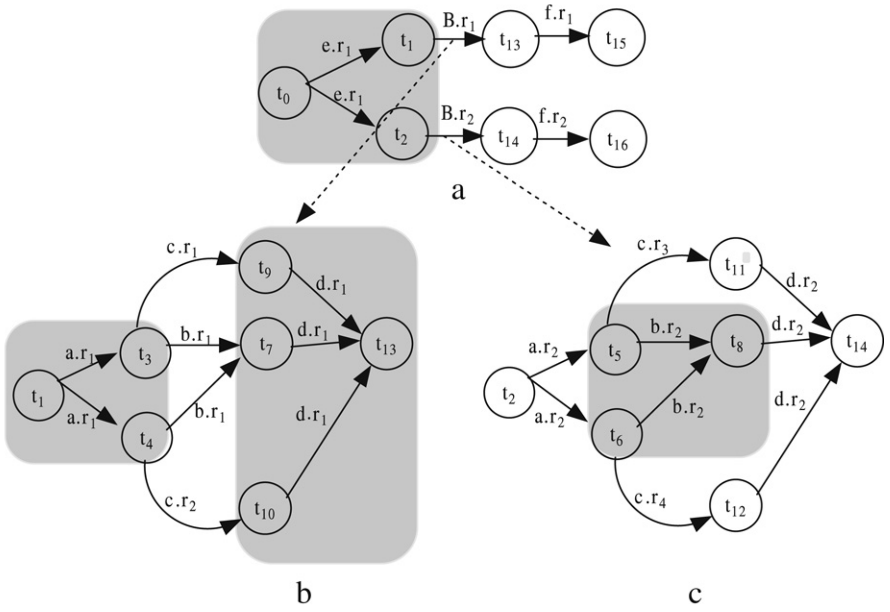


Fig. 2. Token dependency graph and hierarchical rounds. Some rounds are displayed by shaded areas.

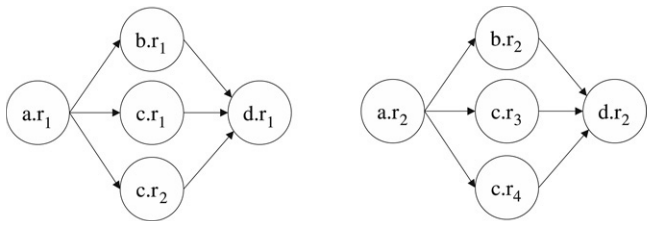


Fig. 3. Round dependency graph.

i.e., if $t_1 \rightarrow t_2$ and $t_2 \rightarrow t_3$, then we have $t_1 \rightarrow t_3$. There is no cyclic transitive dependencies on tokens. Token dependencies are not reflexive, *i.e.*, $t \rightarrow t$ is not allowed.

A round on a composite actor can be decomposed into a set of sub-rounds. Formally, let $a = \langle \{a_1, a_2, \ldots, a_n\}, C_a \rangle$ be a composite actor, and let $a.r$ be a round of a. For a round of $a_i.r_i$, where $a_i \in \{a_1, a_2, \ldots, a_n\}$, $a_i.r_i$ is a *sub-round* of $a.r$ if the following conditions are satisfied: (1) $\forall t' \in input(a_i.r_i).\exists t \in input(a.r).(t == t' \vee t \rightarrow t')$ (*i.e.*, each input token of $a_i.r_i$ is an input token of $a.r$ or depends on its input tokens), and (2) $\forall t' \in output(a_i.r_i).\exists t \in output(a.r).(t == t' \vee t' \rightarrow t)$ (each output token of $a.r$ is an output token of $a_i.r_i$ or depends on its output tokens). Let $a_i.r_i \sqsubseteq a.r$ denote that $a_i.r_i$ is a sub-round of $a.r$.

Thus, as a "zoom-out" view sees round $a.r$, a "zoom-in" view can see its details which contains sub-rounds $a_1.r_1, a_2.r_2, \ldots, a_n.r_n$. For example, Fig. 2(b) shows that round $B.r_1$ consists of sub-rounds $a.r_1, b.r_1, c.r_1, c.r_2$, and $d.r_1$; Fig. 2(c) shows that round $B.r_2$ consists of sub-rounds $a.r_2, b.r_2, c.r_3, c.r_4$, and $d.r_2$.

For two rounds $a.r$ and $b.r$, if $\exists t.(t \in output(a.r) \wedge t \in input(b.r) \wedge a \neq b)$, *i.e.*, $b.r$ consumes the tokens produced by $a.r$, we say $b.r$ *directly depends* on $a.r$, denoted $a.r \dot{\Rightarrow} b.r$. More generally, *round dependencies* (denoted $\Rightarrow$) are defined as follows: (1) if $a.r \dot{\Rightarrow} b.r$, then $a.r \Rightarrow b.r$; and (2) if $a.r \Rightarrow b.r$, $b.r \Rightarrow c.r$, and $a \neq c$, then $a.r \Rightarrow c.r$. Thus, round dependencies are also transitive. Similar to token dependencies, round dependencies are not reflexive. Fig. 3 shows the round dependencies based on the token dependencies in Fig. 2. Note that we do not allow cyclic transitive data dependencies on rounds in our scientific workflow model. Thus, a partial order for the executions of rounds can be generated based on a round dependency graph.

For three rounds $a_1.r$, $a_2.r$, and $a_3.r$, if $a_1.r \Rightarrow a_2.r$ and $a_3.r \sqsubseteq a_2.r$, $a_1.r \Rightarrow a_3.r$ may not exist, because $a_3.r$ may not consume any tokens that depend on $a_1.r$. Similarly, if $a_1.r \Rightarrow a_2.r$ and $a_3.r \sqsubseteq a_1.r$, $a_3.r \Rightarrow a_2.r$ may not exist.

Given a view $v = \langle A, C \rangle$ of a workflow, let $depd - parents_v(a.r) = \{a'.r | a'.r \dot{\Rightarrow} a.r \wedge a' \in A\}$ (*i.e.*, all rounds in the current view that the round $a.r$ directly depends on) and $depd - children_v(a.r) = \{a'.r | a.r \dot{\Rightarrow} a'.r \wedge a' \in A\}$ (*i.e.*, all rounds in the current view that directly depend on the round $a.r$). They can be easily computed from the log introduced in Section 3.

2.3. Commit and abort

We define the *atomicity* of a round as follows: the execution of a round $a.r$ is atomic if either it and all the rounds on which $a.r$ depends run to completion or neither it and nor all the rounds that depend on $a.r$ have any effect. Thus, users do not need to explicitly define transaction boundaries as in business workflows and database systems. Atomicity is ensured automatically by our atomicity management subsystem. Although the atomicity granularity is based on one "round" of execution of a task in this paper, the technique can be readily extended for various granularities. Our system supports atomicity through capturing data dependencies in scientific workflows.

For two rounds $a.r$ and $a'.r$, and $a.r \dot{\Rightarrow} a'.r$, if $a'.r$ consumes only some early output tokens of $a.r$, $a'.r$ might finish by calling `reset()` even when $a.r$ is still running. Thus, "reset" does not mean "commit" of the round, because we have to rollback both $a.r$ and $a'.r$ if $a.r$ fails. For a view v of a workflow, a round $a.r$ *commits* if $a.r$ has finished by calling `reset()` and every round in $depd - parents_v(a.r)$ has committed. If $depd - parents_v(a.r)$ is empty, $a.r$ commits once it is done. Intuitively, a reset event indicates the ending of the current round and the starting of the next round, and a commit event makes the results of the round be observable to the users. The left column of Fig. 4 shows how the atomicity management subsystem commits a round $a.r$. When a round $a.r$ calls `reset()`, the atomicity management subsystem writes a reset event in a log, and calls `commit(`$a.r, v$ `)` to commit round $a.r$. The log is checked to see whether all rounds that $a.r$ depends on have committed. If the commit condition is satisfied, it commits $a.r$ by writing a commit event in the log; otherwise, the

```
commit($a.r$, $v$)//commit algorithm for
a round $a.r$ w.r.t. a view $v$.
  while (true)
      boolean allCommitted = true;
      for all $a'.r \in depd\text{-}parents_v(a.r)$
          if ($a'.r$ has not committed)
              allCommitted = false;
      if (!allCommitted)
          wait$_{commit}$();
      else
          commit $a.r$;
          notifyAll$_{commit}$(
                   $depd\text{-}children_v(a.r)$)
          return;
```

```
abort($a.r$, $v$)//abort algorithm for a
round $a.r$ w.r.t. a view $v$.
  Stop the execution of $a.r$ if running;
  for all $a'.r \in depd\text{-}children_v(a.r)$
      send abort message to $a'.r$;
  while (true)
      boolean allAborted = true;
      for all $a'.r \in depd\text{-}children_v(a.r)$
          if ($a'.r$ has not aborted)
              allAborted = false;
      if (!allAborted)
          wait$_{abort}$();
      else
          for all $t \in$ output($a.r$)
              getRecoveryQueue($t$).$\neg enq(t)$;
          for all $t \in$ input($a.r$)
              getRecoveryQueue($t$).$\neg deq(t)$;
          abort $a.r$;
          notifyAll$_{abort}$($depd\text{-}parents_v(a.r)$)
          return;
```

Fig. 4. Commit algorithm and abort algorithm for a round $a.r$.

current commit process is suspended and waits for being waken up by its parent rounds. A data item is not available to the end users until the round producing it has submitted because uncommitted rounds can be rolled back.

In our system, each data channel is modeled and implemented as an extended recoverable queue adapted from [2]. An extended recoverable queue is a reliable and fault-tolerant queue which supports the following operations: *enqueue* pushes a token at the head; *dequeue* removes a token from the end and returns the token; ¬*enq* undoes the operation of enqueue; ¬*deq* undoes the operation of dequeue. For a view v of a workflow, when the atomicity management subsystem detects crashing of a round $a.r$, it will send *abort* messages to all actors that execute rounds in $depd - children_v(a.r)$ to abort the corresponding rounds, which are not necessarily the on-going rounds. Given a view v of a workflow, a round $a.r$ *aborts* if all rounds in $depd - children_v(a.r)$ have aborted. The abort of a round will eliminate all output tokens (which may still be kept by recoverable queues), then recover all input tokens. The right column of Fig. 4 shows how the atomicity management subsystem aborts a round $a.r$. The atomicity management subsystem first stops the execution of $a.r$ if it is still running and sends *abort* messages to each actor in $depd - children_v(a.r)$. Then the algorithm checks the log to see whether all rounds that depend on $a.r$ have aborted. If some rounds have not aborted, the current abort process is suspended and waits for being waken up by its children rounds. During the abort, the atomicity management subsystem looks up the log to find the corresponding recoverable queue for a given token t (*i.e.*, by calling `getRecoveryQueue`(t)); then it commands the recoverable queue to undo the previous operations. When an abort operation succeeds, an abort event is written in the log.

To support rerun efficiently, we reuse the results of the previous rounds if possible. For a deterministic task, when the input tokens are the same as before, it will generate exactly the same output tokens as before. Thus, when a round of a deterministic task is re-executed, we compare the input tokens with the tokens consumed in previous rounds, if they are exactly the same, we recover the corresponding output tokens and the execution of the current round can be omitted. A recoverable queue keeps as many tokens as its capacity allows. As a minimum, it should be able to keep all tokens for rounds before they commit.

In the algorithm shown by Fig. 4, if two rounds have data dependencies, they are contained in the same atomicity region. This may make commitment and rollback of atomicity computationally expensive. A feasible solution is to classify data dependencies to different levels according to their coherence. Based on different data dependencies, different granularities of atomicity can be defined. This will be in our future work.

2.4. Comparison of rounds and transactions

The rounds in pipelined scientific workflows look similar to transactions that allow dirty reads. The aborts of rounds look similar to cascaded rollbacks of transactions. But there are fundamental differences between them: (1) Round boundaries are specified or inferred dynamically as needed, which is in contrast to *a priori* well-defined transaction boundaries; (2) Conflict accesses exist between transactions, but they do not exist between rounds because rounds communicate by sending data tokens, whereas transactions communicate by accessing to the common data items; (3) Isolation is enforced for transactions using locking protocols, but serializability is automatically achieved in our model because a total order of rounds can be generated based on the partial order of the round dependency graph; (4) Transaction processing usually does not support dirty read and cascaded rollback because these operations reduce performance significantly; but pipelined execution is a fundamental technique to improve performance of scientific workflows; and (5) hierarchy of rounds are automatically inferred by the hierarchy of actors in our model; transactions do not support such a hierarchical structure, although there are extensions for nested transactions.

3. The event log

Our atomicity and provenance system records the following events for supporting atomicity: *enq*, which enqueues a token to a data channel; ¬*enq*, the compensating operation of *enq*; *deq*, which dequeues a token from a data channel; ¬*deq*, the compensating operation of *deq*; *rst*, which resets a state; *fail*, which signifies the failure of a task; *cmt*, which commits a round; *abt*, which aborts a round. These events are stored in a sequential event log, in which each row stores *event identifier*, *time stamp*, *workflow identifier*, *round identifier*, *queue identifier* for a queue event, *event type*, *token identifier* if the event is related to a token, and *dependent tokens*,

evt	rnd	type	tok	depdToks
01	$a.r_1$	deq	t_1	-
02	$a.r_1$	enq	t_3	$\{t_1\}$
03	$c.r_1$	deq	t_3	-
04	$c.r_1$	enq	t_9	$\{t_3\}$
05	$c.r_1$	rst	-	-
06	$a.r_1$	$fail$	-	-
07	$c.r_1$	$\neg enq$	t_9	-
08	$c.r_1$	$\neg deq$	t_3	-
09	$c.r_1$	abt	-	-
10	$a.r_1$	$\neg enq$	t_3	-
11	$a.r_1$	$\neg deq$	t_1	-
12	$a.r_1$	abt	-	-

evt	rnd	type	tok	depdToks
01	$a.r_1$	deq	t_1	-
02	$a.r_1$	enq	t_3	$\{t_1\}$
03	$c.r_1$	deq	t_3	-
04	$c.r_1$	enq	t_9	$\{t_3\}$
05	$c.r_1$	rst	-	-
06	$a.r_1$	enq	t_4	$\{t_1\}$
07	$a.r_1$	rst	-	-
08	$a.r_1$	cmt	-	-
09	$c.r_1$	cmt	-	-
10	$b.r_1$	deq	t_3, t_4	-
11	$b.r_1$	enq	t_7	$\{t_3, t_4\}$
12	$b.r_1$	rst	-	-
13	$b.r_1$	cmt	-	-
14	$c.r_2$	deq	t_4	-
15	$c.r_2$	enq	t_{10}	$\{t_4\}$
16	$c.r_2$	rst	-	-
17	$c.r_2$	cmt	-	-
18	$d.r_1$	deq	t_7, t_9, t_{10}	-
19	$d.r_1$	enq	t_{13}	$\{t_7, t_9, t_{10}\}$
20	$d.r_1$	rst	-	-
21	$d.r_1$	cmt	-	-

Fig. 5. A log for an execution of the workflow in Fig. 2.

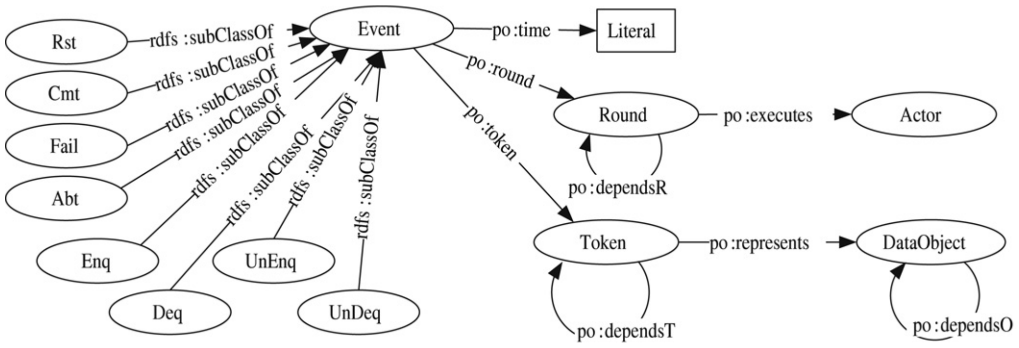


Fig. 6. An excerpt of the provenance ontology.

representing the set of tokens that contribute to the production of a token by the current event if this is the case.

Fig. 5 shows an example of a log file for the workflow run in Fig. 2, where *time stamp*, *workflow identifier*, and *queue identifier* are omitted. The left column shows an aborted workflow run. Round $a.r_1$ first dequeues t_1 and then enqueues t_3. $c.r_1$ consumes t_3, produces t_9, and then calls `reset()`. $c.r_1$ cannot commit at that time since $a.r_1$ has not committed. Finally, $a.r_1$ fails, and $c.r_1$ and $a.r_1$ have to be aborted. The right column shows a successful run.

4. The provenance subsystem

It is imperative to keep the size of the event log small, since this affects the performance of our commit and abort algorithms, and thus the performance of the workflow engine which implements them. Therefore, once the scientific workflow execution is completed, we store provenance information into our provenance system and truncate the log. We use Semantic Web technologies and an RDF repository to design the provenance system. The motivation of using Semantic Web technologies is threefold. First, large-scale e-science applications can span multiple domains and can involve global distributed workflows that consist of several heterogeneous local workflows orchestrated by different workflow engines, each of which has its own provenance manager [51]. The integration of these heterogeneous provenance systems is important for global provenance analysis. A Semantic Web approach promotes interoperability and facilitates such provenance integration. Second, RDF [43] is a property-centric, extremely flexible and dynamic data model, which captures the dynamic and heterogeneous nature of data, services, and metadata in e-science applications. Finally, we can use the inference capability of Semantic Web for deriving metadata for various provenance dependency graphs [6].

To serialize provenance metadata in RDF format, we design a provenance ontology encoded in OWL [42]. In Fig. 6, we present an excerpt of our ontology which sketches main classes and properties that are used to represent information in the event log. Note that the class names are shortened only for the conciseness of our presentation in this paper. The ontology models different events (*Event*) that can occur during workflow execution, such as reset (*Rst*), commit (*Cmt*), fail (*Fail*), abort (*Abt*), enqueue (*Enq*), ¬enqueue (*UnEnq*), dequeue (*Deq*), and ¬dequeue (*UnDeq*). Each event occurs in specific time (see *po:time*) and is related to a round (*Round*) and one or many tokens (*Token*) via properties *po:round* and *po:token*, respectively. A round refers to an actor (*Actor*) via *po:executes* and a token represents a data object (*DataObject*) via *po:represents*.

Based on this provenance representation and Semantic Web inference capability, we can build a *token dependency graph*, an *object dependency graph*, and a *round dependency graph* as in [6]. In our ontology, these dependency graphs are captured via properties *po:dependsT*, *po:dependsO*, and *po:dependsR*, respectively. The properties are defined as transitive (*owl:TransitiveProperty*), such that an inference engine can dynamically derive transitive relations. While token dependencies are directly available from the event log, we infer data object and round dependencies using our defined inference rules as explained in the following.

We use a simple language to define inference rules, such that an antecedent and a consequent of a rule are specified as SPARQL [44] basic graph patterns. If the antecedent matches triples in an RDF graph, then bound variables are used in the consequent to infer new RDF triples that are appended to the RDF graph. The rule for deriving a data object dependency graph is as follows:

$$\frac{?t1\ po:dependsT\ ?t2\ .\ ?t1\ po:represents\ ?d1\ .\ ?t2\ po:represents\ ?d2\ .}{?d1\ po:dependsO\ ?d2\ .}$$

This rule states that if token $?t1$ depends on token $?t2$, and $?t1$ and $?t2$ are tokens for data objects $?d1$ and $?d2$, respectively, then $?d1$ depends on $?d2$. Finally, the rule for deriving round dependencies is:

$$\frac{\begin{array}{c}?e1\ rdf:typepo:Enq\ .\ ?e2\ rdf:typepo:Deq\ .\\ ?e1\ po:token\ ?t\ .\ ?e2\ po:token\ ?t\ .\ ?e1\ po:round\ ?r1\ .\ ?e2\ po:round\ ?r2\end{array}}{?r1\ po:dependsR\ ?r2\ .}$$

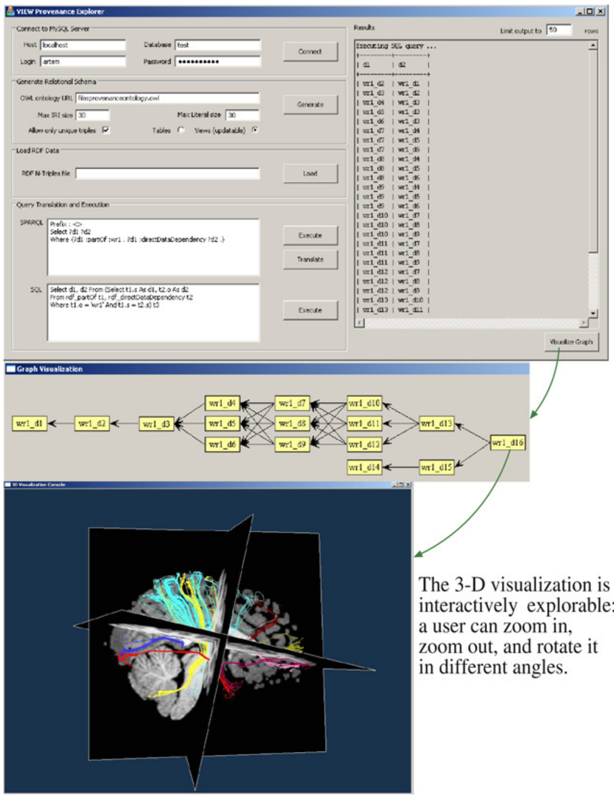
This rule states that if the same token $?t$ is dequeued and enqueued during events $?e1$ and $?e2$, respectively, and these events are initiated by rounds $?r1$ and $?r2$, respectively, then $?r1$ depends on $?r2$.

To support efficient querying of provenance metadata, we store the enhanced RDF dataset into an RDMBS-based RDF repository, which provides both SPARQL and SQL query interfaces. Since our provenance information embraces the event log and contains all information of the event log in [6], we can support all provenance queries listed in [6]. In addition, we can support the atomicity and failure related queries, which are illustrated in the following examples using SPARQL.

- *What actors have aborted rounds?*

```
Select Distinct ?a Where { ?e rdf:type po:Abt .
        ?e po:round ?r .
        ?r po:executes ?a .
      }
```

In this query, the first triple pattern matches an *abort* event $?e$; the second triple pattern matches a round $?r$ that has initiated $?e$; and the last triple pattern matches an actor $?a$ that has been executed by $?r$. The query returns all bindings of $?a$.


VIEW Provenance Explorer

Connect to MySQL Server
Host: localhost
Database: test
Login: artem
Password: ●●●●●●●●●●●

Generate Relational Schema
OWL ontology URL: file:provenanceontology.owl
Max IRI size: 30
Max Literal size: 30
Allow only unique triples [✓]
Tables ○ [✓ Views (updatable)]

Load RDF Data
RDF N-Triples file:

Query Translation and Execution
SPARQL:
```
Prefix : <>
Select ?d1 ?d2
Where {?d1 :partOf :wr1 . ?d1 :directDataDependency ?d2 .}
```
SQL:
```
Select d1, d2 From (Select t1.s As d1, t2.o As d2
From rdf_partOf t1, rdf_directDataDependency t2
Where t1.o = 'wr1' And t1.s = t2.s) t3
```

Results
Limit output to 50 rows
```
Executing SQL query ...
+---------+---------+
| d1      | d2      |
+---------+---------+
| wr1_d2  | wr1_d1  |
| wr1_d3  | wr1_d2  |
| wr1_d4  | wr1_d3  |
| wr1_d5  | wr1_d3  |
| wr1_d6  | wr1_d3  |
| wr1_d7  | wr1_d4  |
| wr1_d7  | wr1_d5  |
| wr1_d7  | wr1_d6  |
| wr1_d8  | wr1_d4  |
| wr1_d8  | wr1_d5  |
| wr1_d8  | wr1_d6  |
| wr1_d9  | wr1_d4  |
| wr1_d9  | wr1_d5  |
| wr1_d9  | wr1_d6  |
| wr1_d10 | wr1_d7  |
| wr1_d10 | wr1_d8  |
| wr1_d10 | wr1_d9  |
| wr1_d11 | wr1_d7  |
| wr1_d11 | wr1_d8  |
| wr1_d11 | wr1_d9  |
| wr1_d12 | wr1_d7  |
| wr1_d12 | wr1_d8  |
| wr1_d12 | wr1_d9  |
| wr1_d13 | wr1_d10 |
| wr1_d13 | wr1_d11 |
```

Fig. 7. A screenshot of provenance querying and visualization.

- *When round r1 runs, what actors simultaneously execute the rounds that depend on r1?*

```
Select Distinct ?a Where { ?e1 rdf:type po:Rst .
        ?e1 po:round r1 .
        ?e1 po:time ?t1
        ?r2 po:dependsR r1 .
        ?e2 po:round ?r2
        ?e2 po:time ?t2
        ?r2 po:executes ?a .
       FILTER ( xsd:dateTime(?t2) < xsd:dateTime(?t1) )
      }
```

In this query, the first three triple patterns match a *reset* event $?e1$ that has been initiated by round $r1$ at time $?t1$. The rest triple patterns match a round $?r2$ that depends on $r1$, an actor $?a$ that has been executed by $?r2$, and time $?t2$ for an event $?e2$ that has been initiated by $?r2$. The value constraint ensures that $?t2$ is less than $?t1$ or, in other words, that $?r2$ runs before $r1$ resets.

Finally, the provenance system can visualize various provenance graphs retrieved from the provenance repository with SPARQL or SQL queries. In addition, our system can visualize static medical images or interactive 3-D graphical models of intermediate or final data products of a workflow run. In Fig. 7, we execute a SPARQL query to retrieve all data object dependencies of a particular workflow run, then visualize the data object dependency graph and one of the data products as a 3-D brain model.

5. Related work

This paper extends our previous work [45] by supporting hierarchical workflows and rounds, and providing descriptions of our provenance ontology, inference rules for various provenance graph derivation, and visualization of provenance.

In recent years, scientific workflows have gained great momentum due to their roles in e-Science and cyberinfrastructure applications [31]. There are a plethora of scientific workflows covering a wide range of scientific disciplines. A survey of various approaches for building and executing workflows on the Grid has been presented by Yu and Buyya [47].

One important line of research for scientific workflows is data provenance, which focuses on the support of data derivation and usage trails to support operations, such as restarts, partial runs, and run-diagnosis of workflows. Although several provenance models [22,38,6,12,3] have been proposed for scientific workflows, there has been no work on the provenance system that supports the notion of atomicity. Surveys on provenance systems include a metamodel for a system architecture for lineage retrieval [4], various approaches to the development of provenance systems [36], and use cases for a provenance system in e-science [32].

Although atomicity is a well studied topic in the context of databases in transaction processing and business workflows, there has been no work on atomicity in the context of "dataflows" and "pipelined execution" in scientific workflows. The read committed assumption that existing atomicity techniques are based on does not hold in pipelined scientific workflows, where both task parallelism and pipelined parallelism are present.

In the rest of this section, we review some of the provenance management systems proposed in the literature.

The Kepler system [1,29] implements a *provenance recorder* to record information about a workflow run, including the context, data derivation history, workflow definition, and workflow evolution. The provenance recorder is parametric and customizable, allowing the user to choose different levels of granularity of provenance data for recording. Based on the provenance information, Kepler supports efficient workflow rerun for a slightly modified workflow. Bowers et al. [6] propose the Read-Write-State-Reset (RWS) provenance model for pipelined scientific workflows within the Kepler framework [30]. The execution of a pipelined workflow exhibits both task parallelism and pipeline parallelism, allowing different actors to execute concurrently. The RWS model records read, write, and state-reset events for each actor in a workflow run and stores them in a relational event log. Compared with our model, the RWS model assumes that each output token depends on all tokens input so far in the current round, whereas our model refines this by assuming actors can tell what input tokens that each output token depends on.

The myGrid/Taverna system [49,48] uses Semantic Web technologies for representing provenance metadata at four levels: process, data, organization, and knowledge. Two levels of ontologies are used. A domain-independent schema ontology is used to describe the classes of resources and the properties between them that are needed to represent the four levels of provenance. A domain ontology is used to classify various types of resources such as data types, service types, and topic of interest for a particular domain. Taverna uses out of the shelf RDF stores, such as Jena [46] and Sesame [7], to manage and query provenance.

The CombeChem [41,17] and Mindswap [20,21] systems also use a Semantic Web approach for provenance collection and representation. While CombeChem, similarly to Taverna, uses a general purpose RDF store, in particular 3store [23], to manage provenance, Mindswap publishes workflow provenance on the Semantic Web.

The Chimera [15] and Swift [50] systems introduce a Virtual Data System (VDS) consisting of a set of relations to store the description of executable programs as transformations, their actual invocations as derivations, and input/outputs as data objects. These systems use provenance for tracking the data derivation history, on-demand data generation and re-generation, and data product validation.

The Wings-Pegasus system [24] uses an OWL ontology for semantic representation [25] of provenance generated during workflow instantiation and the Virtual Data System (VDS) provenance tracking catalog for provenance generated during workflow execution. As a result, workflow instantiation provenance can be queried using SPARQL and workflow execution provenance can be queried using SQL.

The VisTrails system [16,9] is the first system that supports provenance tracking of workflow evolution. In VisTrails, workflow evolution provenance is represented as a rooted tree, in which each node corresponds to a version of a workflow, and each edge corresponds to an update action that was applied to the parent workflow to create the child workflow. Therefore, a workflow evolution tree concisely represents all the workflow versions that a scientist has explored to produce the visualization products. In this way, VisTrails can support scientists to navigate through the space of workflows and parameter settings for an exploration process. VisTrails uses XML and relational database technologies to store and query provenance.

Our VIEW system [27,10] uses OWL and RDF to represent provenance and an RDBMS-based RDF store to manage and query provenance with SPARQL and SQL. Unlike Taverna and CombeChem, which employ general purpose RDF stores, VIEW develops an in-house RDF store which is specifically optimized for provenance management.

The above provenance systems are tightly coupled with their scientific workflow environments. A couple of stand-alone provenance systems have also been developed, including the PReServ (Provenance Recording for Services) system developed under the PASOA (Provenance Aware Service Oriented Architecture) project [22] and the Karma system [37,39]. PReServe supports the recording of interaction provenance, actor provenance, and input provenance with the Provenance Recording Protocol, which specifies the messages that actors can asynchronously exchange with a provenance store to support provenance submission. PReServ uses a provenance management service which provides a common interface to enable different storage systems, such as file system, relational databases, XML databases, and RDF stores, as a provenance store. The Karma system records provenance at four dimensions: execution, location, time, and dataflow, and uses a Publish-Subscribe notification protocol for provenance collection. Karma uses XML and relational database technologies to store and query provenance. Both PReServ and Karma support web service interfaces.

Finally, Stevens et al. distinguish four types of provenance in the context of bioinformatics workflows: process provenance, data provenance, organization provenance, and knowledge provenance [40]. Under such a classification, our system supports both process provenance and data provenance.

6. Conclusions and future work

This paper proposes a hierarchical architecture for scientific workflow management systems that supports both provenance and atomicity. We have shown that, while our atomicity system can support the notion of atomicity, currently at the round level that does not contain cyclic transitive data dependencies, our provenance system has added value to existing provenance systems as we support atomicity and failure related queries.

Currently, we are applying the atomicity and provenance techniques proposed in this paper to a biological simulation workflow for the study of mate-finding behavior of the swarming polychaete, *Nereis succinea* [35]. In this workflow, we have already incorporated provenance technique for the replay of a simulation run. We will incorporate our atomicity technique to support failure-tolerant pipelined execution of simulations. Moreover, we will extend current atomicity and provenance model to various granularities of atomicity and for different models of computations.

References

[1] I. Altintas, O. Barney, E. Jaeger-Frank, Provenance collection support in the Kepler scientific workflow system, in: Proc. of the International Provenance and Annotation Workshop, IPAW, 2006, pp. 118–132.

[2] P. Bernstein, M. Hsu, B. Mann, Implementing recoverable requests using queues, in: Proc. of the 1990 ACM SIGMOD International Conference on Management of data, 1990, pp. 112–122.

[3] O. Biton, S.C. Boulakia, S.B. Davidson, C.S. Hara, Querying and managing provenance through user views in scientific workflows, in: Proc. of the International Conference on Data Engineering, ICDE, 2008, pp. 1072–1081.

[4] R. Bose, J. Frew, Lineage retrieval for scientific data processing: A survey, ACM Comput. Surv. 37 (1) (2005) 1–28.

[5] S. Bowers, B. Ludäscher, Actor-oriented design of scientific workflows, in: 24th Intl. Conf. on Conceptual Modeling, ER'05, in: LNCS, vol. 3716, Springer-Verlag, 2005.

[6] S. Bowers, T. McPhillips, B. Ludäscher, S. Cohen, S.B. Davidson, A model for user-oriented data provenance in pipelined scientific workflows, in: Proc. of the International Provenance and Annotation Workshop, IPAW'06, Chicago, Illinois, USA, May 2006.

[7] J. Broekstra, A. Kampman, F. van Harmelen, Sesame: A generic architecture for storing and querying RDF and RDF Schema, in: Proc. of the International Semantic Web Conference, ISWC, 2002, pp. 54–68.

[8] P. Buneman, S. Khanna, W.-C. Tan, Why and where: A characterization of data provenance. Proc. of the International Conference on Database Theory (ICDT), 1973, 2001, pp. 316–330.

[9] S.P. Callahan, J. Freire, E. Santos, C.E. Scheidegger, C.T. Silva, H.T. Vo, Vistrails: Visualization meets data management, in: Proc. of the SIGMOD International Conference on Management of Data, 2006, pp. 745–747.

[10] A. Chebotko, C. Lin, X. Fei, Z. Lai, S. Lu, J. Hua, F. Fotouhi, VIEW: A visual sciEntific workflow management system, in: Proc. of the International Workshop on Scientific Workflows, SWF, 2007.

[11] D. Churches, G. Gombas, A. Harrison, J. Maassen, C. Robinson, M. Shields, I. Taylor, I. Wang, Programming scientific and distributed workflow with triana services: Research articles, Concurr. Comput. : Pract. Exper. 18 (10) (2006) 1021–1037.

[12] S. Cohen, S.C. Boulakia, S.B. Davidson, Towards a model of provenance and user views in scientific workflows, in: Data Integration in the Life Sciences, 2006, pp. 264–279.

[13] Y. Cui, J. Widom, Lineage tracing for general data warehouse transformations, The VLDB Journal (2001) 471–480.

[14] W. Derks, J. Dehnert, P. Grefen, W. Jonker, Customized atomicity specification for transactional workflows, in: Proceedings of the Third International Symposium on Cooperative Database Systems for Advanced Applications, CODAS'01, 2001, pp. 140–147.

[15] I. Foster, J. Vöckler, M. Wilde, Y. Zhao, Chimera: A virtual data system for representing, querying, and automating data derivation, in: Proc. of the International Conference on Scientific and Statistical Database Management, SSDBM, 2002, pp. 37–46.

[16] J. Freire, C.T. Silva, S.P. Callahan, E. Santos, C.E. Scheidegger, H.T. Vo, Managing rapidly-evolving scientific workflows, in: International Provenance and Annotation Workshop, Chicago, Illinois, U.S.A., May 2006, pp. 10–18.

[17] J.G. Frey, D.D. Roure, K.R. Taylor, J.W. Essex, H.R. Mills, E. Zaluska, CombeChem: A case study in provenance and annotation using the Semantic Web, in: Proc. of the International Provenance and Annotation Workshop, IPAW, 2006, pp. 270–277.

[18] H. Garcia-Molina, K. Salem, Sagas, in: SIGMOD '87: Proceedings of the 1987 ACM SIGMOD international conference on Management of data, 1987, pp. 249–259.

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

[20] J. Golbeck, Combining provenance with trust in social networks for Semantic Web content filtering, in: Proc. of the International Provenance and Annotation Workshop, IPAW, 2006, pp. 101–108.

[21] J. Golbeck, J. Hendler, A Semantic Web approach to the provenance challenge, Concurr. Comput. : Pract. Exper. 20 (5) (2008) 431–439.

[22] P. Groth, S. Miles, W. Fang, S. Wong, K. Zauner, L. Moreau, Recording and using provenance in a protein compressibility experiment, in: Proceedings of the 14th IEEE International Symposium on High Performance Distributed Computing, HPDC'05, Research Triangle Park, North Carolina, U.S.A., July 2005.

[23] S. Harris, N. Gibbins, 3store: Efficient bulk RDF storage, in: Proc. of the International Workshop on Practical and Scalable Semantic Systems, PSSS, 2003, pp. 1–15.

[24] J. Kim, E. Deelman, Y. Gil, G. Mehta, V. Ratnakar, Provenance trails in the Wings-Pegasus system, Concurr. Comput. : Pract. Exper. 20 (5) (2008) 587–597.

[25] J. Kim, Y. Gil, V. Ratnakar, Semantic metadata generation for large scientific workflows. In Proc. of the International Semantic Web Conference, ISWC, 2006, pp. 357–370.

[26] F. Leymann, D. Roller, Production Workflow: Concepts and Techniques, Prentice-Hall, 2000.

[27] Cui Lin, Shiyong Lu, Zhaoqiang Lai, Artem Chebotko, Xubo Fei, Jing Hua, Farshad Fotouhi, Service-oriented architecture for VIEW: A visual scientific workflow management system, in: Proc. of the IEEE 2008 International Conference on Services Computing, SCC 2008, Honolulu, Hawaii, USA, July 2008, pp. 335–342.

[28] B. Ludäscher, Scientific workflows: Cyberinfrastructure for e-science. Pacific Neighborhood Consortium, PNC, Berkeley, October 2007.

[29] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E.A. Lee, J. Tao, Y. Zhao, Scientific workflow management and the Kepler system, Concurr. Comput.: Pract. Exper. 18 (10) (2006) 1039–1065.

[30] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E.A. Lee, J. Tao, Y. Zhao, Scientific workflow management and the kepler system: Research articles, Concurr. Comput. : Pract. Exper. 18 (10) (2006) 1039–1065.

[31] B. Ludäscher, C. Goble, Guest editor's introduction to the special section on scientific workflows, SIGMOD Record 34 (3) (2005) 3–4.

[32] S. Miles, P. Groth, M. Branco, L. Moreau, The requirements of recording and using provenance in e-science experiments, J. Grid Computing (2006).

[33] 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, C. Wroe, Taverna: Lessons in creating a workflow environment for the life sciences: Research articles, Concurr. Comput. : Pract. Exper. 18 (10) (2006) 1067–1100.

[34] T.M. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, R.M. Greenwood, T. Carver, K. Glover, M.R. Pocock, A. Wipat, P. Li, Taverna: A tool for the composition and enactment of bioinformatics workflows, Bioinformatics 20 (17) (2004) 3045–3054.

[35] J. Ram, X. Fei, S. Danaher, S. Lu, T. Breithaupt, J. Hardege, Finding females: Pheromone-guided reproductive tracking behavior by male *Nereis succinea* in the marine environment, J. Exp. Biol. 211 (5) (2008) 757–765.

[36] Y. Simmhan, B. Plale, D. Gannon, A survey of data provenance in e-science, SIGMOD Record 34 (3) (Sept.2005) 31–36.

[37] Y. Simmhan, B. Plale, D. Gannon, A framework for collecting provenance in data-centric scientific workflows, in: Proc. of the IEEE International Conference on Web Services, ICWS'06, Washington, DC, USA, 2006, pp. 427–436.

[38] Y.L. Simmhan, B. Plale, D. Gannon, A framework for collecting provenance in data-centric scientific workflows, in: Proc. of the IEEE International Conference on Web Services, ICWS'06, Washington, DC, USA, 2006, pp. 427–436.

[39] Y.L. Simmhan, B. Plale, D. Gannon, Query capabilities of the Karma provenance framework, Concurr. Comput. : Pract. Exper. 20 (5) (2008) 441–451.

[40] R. Stevens, J. Zhao, C. Goble, Using provenance to manage knowledge of *n silico* experiments, Briefings in Bioinformatics 8 (3) (2007) 183–194.

[41] K.R. Taylor, R.J. Gledhill, J.W. Essex, J.G. Frey, S.W. Harris, D.D. Roure, Bringing chemical data onto the Semantic Web, J. Chem. Inf. Modeling 46 (3) (2006) 939–952.

[42] W3C. OWL Web Ontology Language Reference. W3C Recommendation, 10 February 2004. M. Dean and G. Schreiber (Eds.). Available from: http://www.w3.org/TR/2004/REC-owl-ref-20040210/.

[43] W3C. Resource Description Framework (RDF): Concepts and Abstract Syntax. W3C Recommendation, 10 February 2004. G. Klyne, J. J. Carroll, and B. McBride (Eds.). 2004. Available from: http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/.

[44] W3C. SPARQL Query language for RDF. W3C Candidate Recommendation, 15 January 2008. E. Prud'hommeaux and A. Seaborne (Eds.). 2008. Available from: http://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/.

[45] L. Wang, S. Lu, X. Fei, J. Ram, A dataflow-oriented atomicity and provenance system for pipelined scientific workflows, in: Proc. 2nd International Workshop on Workflow Systems in e-Science (WSES 07), in conjunction with International Conference on Computational Science (ICCS) 2007, in: LNCS, Springer, 2007.

[46] K. Wilkinson, C. Sayers, H. Kuno, D. Reynolds, Efficient RDF storage and retrieval in Jena2, in: Proc. of the International Workshop on Semantic Web and Databases, SWDB, 2003, pp. 131–150.

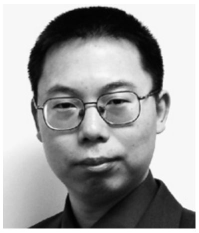
[47] J. Yu, R. Buyya, A taxonomy of scientific workflow systems for grid computing, SIGMOD Record 34 (3) (2005) 44–49.

[48] J. Zhao, C. Goble, R. Stevens, D. Turi, Mining Taverna's semantic web of provenance, Concurr. Comput. : Pract. Exper. 20 (5) (2008) 463–472.

[49] J. Zhao, C. Wroe, C.A. Goble, R. Stevens, D. Quan, R.M. Greenwood, Using Semantic Web technologies for representing e-science provenance, in: Proc. of the International Semantic Web Conference, ISWC, 2004.

[50] Y. Zhao, M. Hategan, B. Clifford, I. Foster, G. Laszewski, V. Nefedova, I. Raicu, T. Stef-Praun, M. Wilde, Swift: Fast, reliable, loosely coupled parallel computation. In IEEE International Workshop on Scientific Workflows, SWF 2007, 2007, pp. 199–206.

[51] Z. Zhao, A. Belloum, C. de Laat, P. Adriaans, B. Hertzberger, Distributed execution of aggregated multi domain workflows using an agent framework, in: Proc. of the International Workshop on Scientific Workflows, SWF, 2007, pp. 183–190.



Liqiang Wang is an assistant professor in the Department of Computer Science at the University of Wyoming. His research focuses on designing methods and tools for concurrent and distributed systems including scientific workflows. He also serves as a program committee member for several IEEE and ACM conferences. He received his Ph.D. degree in Computer Science from Stony Brook University in 2006, M.E. in Computer Science from Sichuan University at China in 1998, and B.S. in Mathematics from Hebei Normal University at China in 1995.

Dr. Shiyong Lu is currently an assistant professor in the Department of Computer Science at Wayne State University and the director of the Scientific Workflow Research Laboratory (the SWR Lab). Dr. Lu received his Ph.D. from Stony Brook University in 2002, M.E. from the Institute of Computing Technology at Chinese Academy of Sciences at Beijing in 1996, and B.E. from the University of Science and Technology of China at Hefei in 1993. His research interest focuses on scientific workflows with applications to bioinformatics, medical informatics, and biological simulations. He has published over 60 refereed international journal and conference papers in the above areas. Dr. Lu is the founder and currently the chair of the IEEE International Workshop on Scientific Workflows, an editorial board member for International Journal of Medical Information Systems and Informatics and International Journal of Semantic Web and Information Systems. He also serves as a program committee member for several IEEE and ACM conferences. He is a member of both ACM and IEEE.

Xubo Fei is currently a Ph.D. student in the Department of Computer Science at Wayne State University. His current research interests include scientific workflows and their applications in bioinformatics and biology simulation.

Artem Chebotko is an assistant professor in the Department of Computer Science at University of Texas - Pan American. He received his Ph.D. in Computer Science from Wayne State University in 2008, M.A. in Computer Science from Wayne State University in 2005, M.S. in Management Information Systems in 2003 and B.S. in Computer Science from Ukraine State Maritime Technical University. His research interests include scientific workflow provenance metadata management and semantic web data management. He has published a number of papers in refereed journals and conference proceedings and currently serves as a program committee member of two international workshops on scientific workflows. He is a member of ACM and IEEE.

H. Victoria Bryant is currently a graduate student at Florida State University. Her research interests include artificial intelligence, distributed systems and robotics. She received her Master's in Computer Science in 2007 from the University of Wyoming.

Jeffrey L. Ram received his B.A. in Physics from the University of Pennsylvania and his Ph.D. in Biochemistry and Neurophysiology from the California Institute of Technology. He is currently a Professor of Physiology at Wayne State University USA, an Honorary Professor at Hull University, UK, and a researcher at the Marine Biological Laboratory, Woods Hole, MA USA. In addition to having an active biological research program on behavioral responses to chemical attractants in organisms as widely different as bacteria and worms, Dr. Ram uses computational methods to model genetic and behavioral mechanisms involved in these organisms. Author of more than 100 research papers, his research has been supported by grants at both the federal and state level, from NIH, NSF, NOAA, and the State of Michigan.