Decentralized Execution of Event-Driven Scientific Workflows

Guoli Li[†], Vinod Muthusamy[‡], H.-Arno Jacobsen[†‡], and Serge Mankovski[*]
gli@cs.toronto.edu, {vinod, jacobsen}@eecg.toronto.edu, smankovski@cybermation.com

[†]Department of Computer Science
[‡]Department of Electrical and Computer Engineering
University of Toronto

[*]Cybermation Inc., Markham, Ontario, Canada

Abstract

Scientific workflows (SWF) are traditionally coordinated and executed in a centralized fashion. This creates a single point of failure, forms a scalability bottleneck, and often leads to too much message traffic routed back to the coordinator. We have developed PADRES, a content-based publish/subscribe platform that serves as a runtime environment for the decentralized execution, control, and monitoring of SWF. Publish/subscribe is a natural paradigm for event-driven applications such as SWF management, as the loosely-coupled nature of publishers and subscribers relieves the coordinator from maintaining client connection and capability information. PADRES has been developed with features inspired by the requirements of SWF management. Its unique features include an expressive subscription language, composite subscription processing support, a rule-based matching and routing mechanism, a query-based historic data access mechanism, and support for the decentralized execution of SWFs specified in XML.

1. Introduction

Essentially all scientific disciplines have started to develop large-scale e-science infrastructures comprised of inter-connected research instruments, computing facilities, data and storage centers, visualization resources, and networks. These infrastructures can be found today in physics (e.g., the CERN Large Hadron Collider or the Canadian MRI Facility), in geology and oceanography (e.g., the US-Canadian Neptune project), or in astronomy (e.g., the European eVLBI project). These infrastructures are characterized by the large volumes of data and events generated, by the distributed and heterogeneous nature of the involved resources, and by the need for the effective management and operation of the distributed resources. A primary and common problem in this context is how to seamlessly access the infrastructure and how to make it widely available to the research community to solve complex scientific problems. Scientists use the infrastructure for data collection, querying, analysis, and visualization to experiment, validate, test, and develop scientific theories. A typical use involves the specification and execution of a workflow, also referred to as a *scientific workflow* (SWF). The SWF ties distributed resources together that are required for experimentation. The execution of SWF is event-driven.

Workflow management solutions have existed for a number of decades [10, 14]. The traditional approach to workflow management is based on using a central server to coordinate and schedule tasks [10, 14]. This has also been the predominant solution for scientific workflow processing [16]. The centralized approach is inherently limited, not only due to the single point of control that constitutes a single point of failure and potential performance bottleneck, but, especially, due to the characteristics of SWFs. SWFs are by nature distributed requiring the integration of heterogeneous computing resources, often operate over unreliable public networks and unstable computing resources, require resources distributed across administrative domains, are long running requiring fine-grained control and status monitoring, involve large volumes of data and data movement, and are operated by scientists who prefer to focus on the experiment at hand, rather than the computer engineering issues.

The geographic distribution inherent to the SWF processing and the above listed requirements have prompted us to re-think the centralized approach to scientific workflow processing and propose a fully decentralized approach. Moreover, we propose a SWF runtime environment based

Proceedings of the IEEE Services Computing Workshops (SCW'06)
0-7695-2681-0/06 $20.00 © 2006 IEEE

on a distributed publish/subscribe model that naturally supports the event-driven nature of the workflow execution directly supporting control, execution, and monitoring functions in a decentralized manner. We have fully implemented this runtime environment and the workflow management facility in our PADRES distributed, content-based publish/subscribe system project.

A publish/subscribe system delivers relevant information from publishing data sources to subscribing data sinks. While publish/subscribe has been widely applied to data dissemination in wide-area networks, we are not aware of any existing approach for the decentralized management of workflows in general and scientific workflows in particular.

In Section 2 we provide background and related work on publish/subscribe and workflow processing. In Section 3 we provide an overview of PADRES. In Section 4 we describe our approach to decentralize the execution of SWF. Our presentation aims at providing an intuitive description of SWF deployment, SWF execution, and SWF monitoring, avoiding formal details due to the limited space. This section also reports on a few experimental results evaluating PADRES for SWF processing. Finally, Section 5 discusses the approach and outlines directions for future work.

2. Background and related work

Scientific Workflow Management: Grid-Flow [17] defines a scientific workflow as a procedure that applies specific computations (processes) on selected data according to certain rules. Most of the current SWF management systems use a directed acyclic graph (DAG) to describe the workflow process in a graphical model [17]. Tasks in the workflow are described in a node of the DAG in a specified order over a set of dispersed data and computing resources. Since it is hard to express loops in a DAG, Petri Net models are being introduced for SWF management to model workflows [14]. Two popular scientific workflow specification languages are the Abstract Grid Workflow Language (AGWL) [5] and the Grid Services Flow Language (GSFL) [8]. Both languages are XML-based. Tasks in SWFs are often distributed across a heterogeneous environment. SWFs orchestrate a large number of tasks involving diverse resources, such as computing resources, data storage resources, research instruments, and network resources. Because the data and computing are dispersed in a physically distributed environment, the SWF management system needs a mechanism for data transfer, often involving huge volumes of data. Many Grid workflow systems have been explored for scientific environments [4, 15]. SWF management systems support workflow specification, workflow execution, workflow monitoring and workflow control. Task execution events, such as the completion status of a task, determine the execution of the successor tasks. The event-driven nature of SWF processing lends itself well to a coordination model based on publish/subscribe interactions. The unique benefit of the application of the publish/subscribe model in this context is the natural decoupling of all workflow tasks and the seamless distribution inherent to the model. We are not aware of any related approaches exploring content-based publish/subscribe for SWF management. In a somewhat related approach, IFLOW [9] uses a channel-based publish/subscribe layer to coordinate information flows, but does not exploit the more expressive content-based publish/subscribe semantics nor does it support the in-network processing capabilities of composite subscriptions.

Publish/Subscribe: In the publish/subscribe paradigm, information producers submit data as publications to the system and information consumers indicate their interests by submitting subscriptions. Publications represent *events* that have occurred in the system. Possible events in SWFs are "a task has successfully finished", "a task has failed", "data are available at specified sites", etc.. In this paper, we use the terms event and publication interchangeably. The messaging substrate consists of a distributed broker overlay network. On receiving an event (publication), the broker overlay determines the subset of matching subscriptions and notifies the appropriate subscribers. The key benefit of distributed publish/subscribe is the natural decoupling of publishers and subscribers. Since the publishers are unconcerned with the locations and identities of the potential consumers of their data, and the subscribers are likewise unaware of the potential producers of interesting data, the client interface of the publish/subscribe system is simple and intuitive. This paradigm has found wide-spread use in distributed event-based applications ranging from selective information dissemination to network and distributed system management.

Content-based publish/subscribe systems typically utilize *content-based routing* in lieu of the standard address-based routing. Effectively, the content-based *address* of a subscriber is the set of subscriptions issued. Messages are

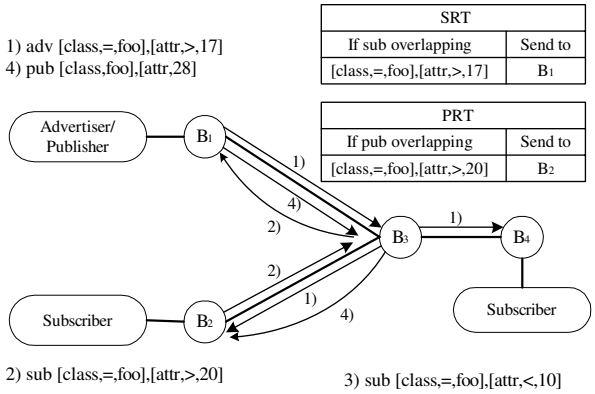

Figure 1. PADRES network architecture

then routed according to their content. SIENA [3] is a notification service based on the content-based publish/subscribe paradigm. Existing systems [13, 11] have explored optimizations for content-based routing, such as covering-based routing and merging-based routing, and we have applied both these techniques in PADRES. Advertisements, which are indications of the data that publishers would publish in the future, are used to form routing trees along which subscriptions are propagated, further reducing network traffic by avoiding subscription flooding. PADRES uses the advertisement-based routing model [3] and a rule-based approach for message matching and routing at each broker.

3. PADRES system description

PADRES [11, 12] is a distributed content-based publish/subscribe system consisting of a set of brokers connected by a peer-to-peer overlay network. Clients connect to brokers using binding interfaces such as the Java Remote Method Invocation (RMI) and the Java Messaging Service (JMS). The PADRES subscription language is based on the traditional `[attribute,operator,value]` predicates used in many existing content-based publish/subscribe systems.

PADRES is distinguished from existing content-based publish/subscribe systems by the following novel features inspired from the requirements of supporting the decentralized execution and management of scientific workflows. First, unlike existing publish/subscribe systems, PADRES allows the subscriber to subscribe to data published in both the *future* and the *past*. For future events (publications), PADRES uses the standard publish/subscribe messaging paradigm. Clients can subscribe to historic events by specifying *time range* predicates in their subscriptions. Historic databases are attached to the brokers through database bindings. Upon receiving a request for the historic events, the brokers re-publish relevant publications from their databases. From the client's point of view, PADRES transparently delivers both past and future events in the same manner. The subscription language supported in PADRES allow clients to correlate past and future events through temporal joins. Second, PADRES explores sophisticated event correlations in publish/subscribe networks. This is achieved by allowing subscribers to express *composite subscriptions* [12], which consist of a set of *atomic subscriptions*. Composite subscriptions provide a higher level view of events for clients by combining information into higher level concepts through aggregation or logical operators. Furthermore, predicates in composite subscriptions can contain variables for correlating events matching the individual composite parts (i.e., the atomic subscriptions.) For example, a composite subscription {{`[appl, eq, $x] [task, eq, B]`...} *and* {`[appl, eq, $x] [task, eq, C]`...}} is satisfied when both tasks *B* and *C* have successfully executed and are part of the same application. More details on the composite subscription language in PADRES is discussed in [6].

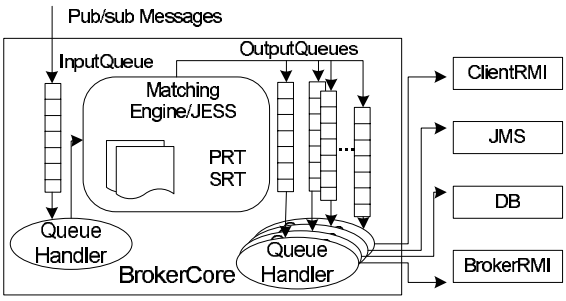

Figure 2. PADRES broker architecture

PADRES Network Architecture: The overlay network connecting the brokers is a set of connections that form the basis for message routing, and each broker maintains information about its overlay neighbors in the Overlay Routing Tables (ORT). We assume that publications are the most common messages, and advertisements the least common. A publisher issues an advertisement before it publishes, and using the ORT these advertisements are effectively flooded to all brokers along the overlay network. A subscriber may subscribe at any time, and these subscriptions are routed according to the Subscription Routing Table (SRT), which is built based on the advertisements seen by a broker. The SRT is essentially a list of `[advertisement,last hop]` tuples. If a subscription overlaps an advertisement in the SRT, it will be forwarded to the last hop of that advertisement. Subscriptions are routed hop by hop to potential publishers, who may publish information of interest to the subscriber. The propagation of subscriptions are used to construct the Publication Routing Table (PRT). Like the SRT, the PRT is logically a list of `[subscription,last hop]` tuples, and is used to route publications. If a publication matches a subscription in the PRT, it will be forwarded to the last hop broker of that subscription until it reaches the subscriber. A diagram showing the overlay network, SRT and PRT is provided in Fig. 1. In this figure, an advertisement *1)* is propagated from B_1. A matching subscription *2)* enters from B_2, and is routed along the SRT. For instance, *2)* overlaps *1)* at broker B_3, and it is sent to B_1. Subscription *3)* does not overlap advertisement *1)*, so it is not forwarded by B_4. Lastly, publication *4)* matching subscription *2)* is routed along the PRT formed by *2)* to B_2.

Broker Architecture: The PADRES brokers are modular software components built on a set of queues: one input queue and multiple output queues. Each output queue represents a unique message destination. A diagram of the broker internals is provided in Fig. 2. The matching engine in the `BrokerCore` maintains the SRT and PRT and is

built using the Java Expert System Shell (JESS) rule engine. For example, in the PRT, subscriptions are mapped to rules, and publications are mapped to facts. The matching is performed by the JESS rule engine. When a new message is received by the broker, it will be put into the input queue. The matching engine takes the message from the input queue. If the message is a publication, it is inserted into the matching engine as a fact. When a publication matches a subscription in the PRT, its next hop destination is set to the last hop of the subscription, and it is placed into the corresponding output queue and sent to the next hop broker as an event. If the message is a subscription, the matching engine first routes it according to the SRT, and, if there is an advertisement overlapping with the subscription, the subscription will be inserted into the PRT as a rule. Essentially, the rule engine performs matching and decides the next-hop destinations of the messages. This novel approach allows for powerful subscription semantics and naturally enables composites subscriptions, which are more complex rules in the matching engine. Mapping the subscription language to a rule language is relatively straightforward, and extending this subscription language does not require significant changes in the engine. Furthermore, rule engines are well-studied, allowing PADRES to take advantage of existing research. The rule engine-based matching is quite efficient, especially for composite subscriptions.

4. SWF management in PADRES

A distributed publish/subscribe system offers a loosely-coupled messaging paradigm, which is a natural fit for the execution of SWFs. The distributed character of a publish/subscribe system enables the decentralized SWF execution almost for "free" (i.e., it is an inherent characteristic of distributed publish/subscribe and content-based routing.) Along the same line of reasoning, a centralized publish/subscribe approach is also a natural fit for centralized workflow execution. In both cases, centralized and distributed, clients interact with the publish/subscribe system through simply publishing and subscribing. Since routing is content-based, the workflow manager does not need to maintain any address information about computational tasks of a SWF and their location in the network. The workflow manager, unlike in the centralized case, does not need to bother with the routing of data to and from different tasks, as this information is automatically delivered using content-based routing. Computational tasks interact with the publish/subscribe system through a task execution agent interface. This is a lightweight software layer with the ability to send and receive messages using the publish/subscribe protocol, essentially supporting publishing of events and subscribing to events. In this paper, for simplicity, we refer to the computational task as a publish/subscribe client or a task agent and do not distinguish between the SWF application component and the publish/subscribe interface component.

There are several advantages of using a publish/subscribe system for scientific workflow management. First, workflows are by nature event-driven. A SWF is started by a trigger event, which is a publication message issued by a workflow manager located anywhere on the network, and is driven by publication messages of finished tasks (i.e., events indicating task success or failure.) Through publish/subscribe routing, these events (publications) are automatically and transparently routed to the appropriate agents, because task agents are linked by task dependencies described in the SWF. Once execution starts, a workflow manager is not necessary for this processing, as is traditionally required with a centralized coordinator. In addition, the same kind of control, as in the centralized processing case, for monitoring, halting, or resuming a workflow is still supported (monitoring is described in more detail below.) Second, SWFs can easily be executed across multiple platforms, as the publish/subscribe paradigm seamlessly supports cross-platform applications in a distributed environment, only requiring an agreed upon message format for publish/subscribe messages. Moreover, large-volume workflow processing can be scalably supported through the distributed publish/subscribe approach. Third, the decentralized solution for executing SWFs avoids the single point of failure and potential scalability bottleneck of a centralized solution. Fourth, the monitoring and control of workflow executions are flexible. Monitoring is a natural fit for the publish/subscribe paradigm, since the workflow manager can subscribe to any task execution or status information, without having to modify the SWF it is monitoring. Furthermore, it is easy to add, modify or delete tasks from a SWF. The modification can be performed dynamically by having task agents unsubscribe and re-subscribe, for instance. Fifth, multiple SWFs can be deployed into the publish/subscribe system at the same time. Moreover, concurrent execution of several workflow instances is possible. The flexible subscribing mechanism enables the modifications of entire workflows, or of workflow instances, and seamlessly supports versioning of workflows. For example, the latter is achieved by simply including attributes referring to versions of workflows and by including predicates over these version attributes in subscriptions. Sixth, SWFs are by nature distributed and can involve vast amounts of data. Again, these characteristics are naturally accommodated by the publish/subscribe paradigm, which connects remote tasks in a distributed manner, rather than linking them through a central coordinator.

To illustrate some of these points, we present a simple working example of a scientific workflow. Fig. 3 shows the

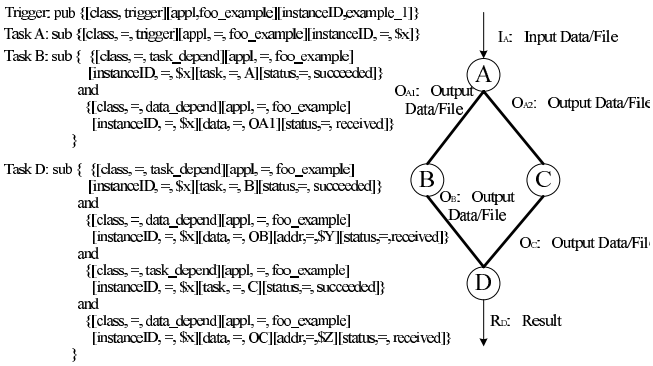

Figure 3. A SWF example

workflow specification: task *A* is applied to an input file I_A. The output files of task *A* are used as inputs to successors tasks *B* and *C*. Finally, both the output files O_B and O_C are inputs to task *D*. The workflow is completed after task *D* produces the result R_D.

4.1. Architecture description

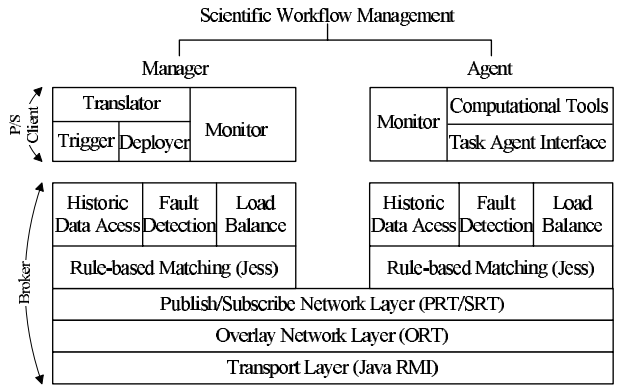

Figure 4. PADRES protocol stack

Fig. 4 shows the architecture of the decentralized SWF management system based on PADRES [1]. Workflow managers and task agents are clients of the publish/subscribe system. Workflow specifications are defined using an XML-based language, and the workflow manager converts the specification to a set of subscriptions and deploys these messages into the publish/subscribe system to set up data and control flow paths. The subscriptions encode the dependencies among tasks. For example, the agent for task *B* subscribes to messages from task *A*. After the translation, the manager uses *agent control messages* to deploy the SWF (i.e., inject it into the network.) All task agents subscribe by default to agent control messages. By using a *content-based address*, such as a name of an agent or an agent capability, the control messages can be targeted to specific agents available in the network. As a result, each task agent knows when to execute its assigned task (i.e., when its subscription condition is met.) Fig. 3 gives examples of subscriptions that encode a SWF and the corresponding publications for triggering a SWF instance. The manager issues a *trigger event* to start the execution of the deployed workflow. The first tasks, which subscribe to the trigger event, start to run after receiving the notification (i.e., after their subscriptions match the trigger publication.) Each task agent publishes a *task_status* event indicating the success of the computation carried out, which continues the execution by triggering successors (failure cases are handled analogously.) The process continues until the SWF terminates.

All interaction, message routing, data transfer, and event delivery are initiated, controlled, and performed in the publish/subscribe layer. Once a workflow is deployed, these operations are performed in a fully decentralized manner without the interference of a centralized manager. This is the fundamentally new concept for SWF management and execution realized in PADRES.

4.2. Workflow translation

SWFs are specified in XML detailing the order of task execution and the various dependencies between tasks. In order to execute a workflow, the XML documents are translated into a set of publish/subscribe messages. Three kinds of messages are created from the workflow specification for each task in the workflow: dependency subscriptions, task execution information, and advertisements.

Each task agent maintains a *dependency subscription* for its task. If the subscription is matched, a notification is delivered to the agent who issued the subscription. This signals the agent that its task is ready to run. A dependency subscription is a composite subscription, as a task depends not only on all its predecessor tasks, but also on resources, such as the availability of input data. For example, as shown in Fig. 3, task *B* subscribes to the successful completion of task *A* **and** to the availability of data O_{A1}. Both events for satisfying the composite subscription can occur at different times and in different order. After agent *B* is notified that both events have occurred, it can execute task *B*.

Several options for data transfer exist. Data can be part of a publication and transferred in the publish/subscribe layer. For instance, after task *A* is completed, the agent of task *A* publishes data O_{A1}. At the same time, the agent of task *B* subscribes to O_{A1}. If the data to be transferred is too large, it may not be efficient to include the data in the publication. PADRES handles this by including a reference to the data in the publication and requires the receiving agent (i.e., subscribing agent) to retrieve the data. Also, the data processed by a task may originate from several data sources.

1 Not all functions are discussed in this paper due to space limitations.

For example, task D processes data O_B and O_C. Assume data O_C is already available at some database or has been published in the past. In that case, the historic data access capabilities in PADRES can be used to retrieve the data. Finally, special-purpose data transfers, such as huge volume data transfers requiring the setup of dedicated network links, e.g. User Controlled LightPaths (UCLP) [2], or requiring the use of special protocols, like GridFTP can be handled by special agents interleaved in the workflow specification who expose these capabilities. These agents would be integrated in the workflow like any other agent (i.e., by subscribing and publishing.)

Task execution information represents the details of a computational task, such as what script and parameters the task needs for execution. The task execution information must be delivered to the task agent if it is described in the workflow specification, unless it is hard-coded in the agent and known a priori.

Advertisements are descriptions of the event templates that a publisher will publish in the future, and are used to establish subscription routing paths. These advertisements are generated for each task in the workflow so that an agent can publish events about status information of task executions. Advertisements are delivered to an agent in the deployment phase.

4.3. Workflow deployment

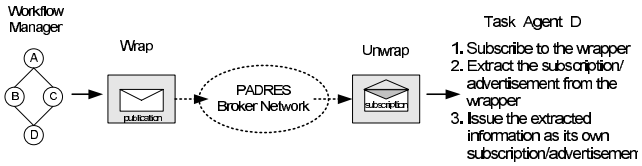

Figure 5. Workflow deployment

The goal of the workflow deployment phase is to send the subscriptions and advertisements generated from the workflow description file to the corresponding task agents. In PADRES this is performed in the publish/subscribe layer by including the subscription, for instance, as part of an *agent control message*, which is a publication compliant with the publish/subscribe messaging infrastructure. In other words, a subscription is embedded in a publication as an (attribute, value)–pair. An agent receives the publication by subscribing to *agent control messages* by default. Fig. 5 shows how an agent receives workflow information from a workflow manager. The agent extracts the subscriptions from the received agent control publication envelope, and issues the subscription as its own subscription. The same process applies for advertisements. After all the subscriptions and advertisements are delivered, the dependence information of the workflow is set up and the agents are ready to receive and publish events to drive the workflow executions. This deployment process is performed entirely in the publish/subscribe layer.

A workflow deployed in the publish/subscribe network can be modified dynamically. If we want to remove a task from a workflow, the corresponding task agent is asked to unsubscribe its task dependency subscriptions, and any successor task agent is asked to modify its subscriptions. This is done in the same message injection-based manner as the deployment of a workflow. Consequently, the modification only influences the affected task agents.

4.4. Workflow execution

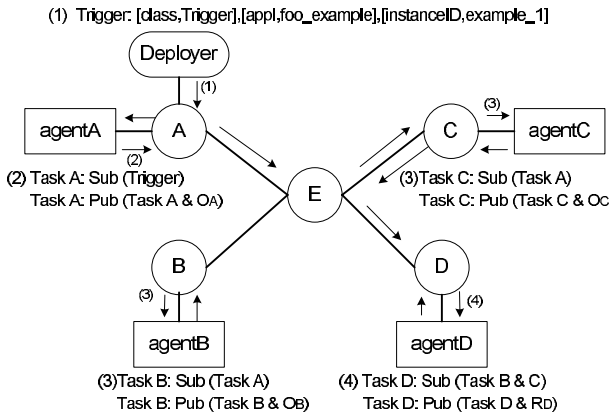

Figure 6. Workflow execution

The workflow is ready to execute after the deployment stage. Task agents, which are regular publish/subscribe clients, are responsible for executing the tasks in the workflow. Agents act as both subscribers and publishers. The dual roles enable them to exchange messages within the publish/subscribe messaging system, realizing the coordinated execution of the workflow. The manager triggers the first task in the workflow to start a new workflow instance, which is driven by events of finished tasks. Execution continues until all the tasks defined in the workflow are completed. The key to workflow execution are dependency subscriptions, which determine the execution order and data resources of the tasks. The message routing is automatic and transparent to the workflow management layer. The execution is event-driven; once the execution starts there is no centralized control. Figure 6 shows the execution of the previous SWF example.

Composite subscriptions are used to specify task dependencies, so that a task is not executed until all its predecessor tasks have completed. The use of composite subscriptions allows clients to specify complicated task dependencies without requiring significant processing or stor-

age capabilities in the agents. Without composite subscriptions, agents would have to subscribe to each predecessor task separately, and special logic would be needed to check whether a task's execution constraints are satisfied. Moreover, agents would receive a large number of publications that match only part of a composite subscription. When a workflow has thousands of tasks with complex task dependencies, an agent may be overwhelmed by individual task events. Bandwidth consumption would be high, since only a small portion of the received messages are useful. Essentially, a distributed publish/subscribe system with composite subscriptions makes the execution of SWFs efficient.

Each workflow instance has a unique instance ID, allowing multiple instances of the same workflow to execute simultaneously without interfering, and also serve to keep track of workflows with long life-spans. Moreover, different workflows can be deployed in the network and multiple instances of these workflows can run concurrently.

4.5. Workflow monitoring and control

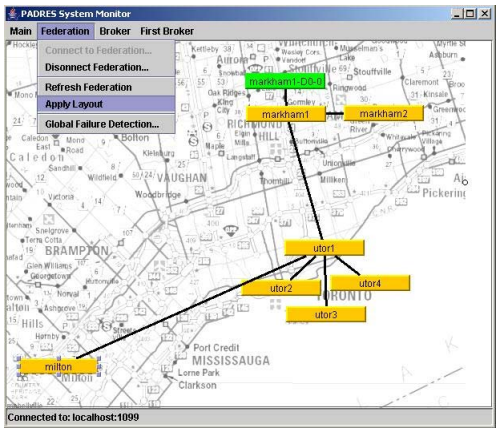

Figure 7. Workflow management monitor

PADRES provides a graphical interface which allows a user or developer to visualize the network topology, the message routing, and the SWF processing, as shown in Fig. 7. The monitor can also retrieve and display the status of a running broker. Through the monitor run-time monitoring can be performed to help debug and troubleshoot SWFs in a design environment and perform real-time execution monitoring in a production environment. The monitor can display the status of a running SWF instance by subscribing to events indicating the status of task executions. It can also visualize the execution and message exchange between tasks, and can view the details of task instances, i.e., the value of a particular variable in a task. Furthermore, the monitor can modify the value of a variable, if necessary, through *agent control messages* indicating the new value. Further run-time control can be exerted with *agent control messages*. The manager can "ask" the instance to pause, resume, or halt. The workflow control operates on all concurrent running instances or a particular instance by identifying the instance ID. With the historic data access supported by PADRES, the manager can view and manipulate previously executed and currently executing instances by issuing historic queries and correlating historic and future queries through temporal joins. The monitor can manipulate the instances by date and execution status. Moreover, through composite subscriptions, the executing and previously executed workflow instances can be monitored at the same time. Based on all this information, statistics over workflow execution instances can be generated to provide a high-level analysis of the system operation.

4.6. Evaluation

An evaluation of PADRES proves that our rule-based matching approach works efficiently. It takes about 5 *ms* to route a publication against 200,000 subscriptions, while a matching algorithm similar to the predicate counting algorithm [1] takes about 82 *ms*; a naive matching algorithm which linearly scans the routing table takes about 299 *ms*. Moreover, the SWF management system built with PADRES successfully performs the decentralized execution of workflows and workflow monitoring based on composite subscriptions and historic data access. To evaluate the network traffic overhead of the workflow execution, we design two SWFs: workflow *A* is an application with four tasks, and workflow *B* has eight tasks. When a composite subscription issued by an agent is matched, only one notification message is sent to the agent, as opposed to several individual primitive notifications. This simplifies agent processing and significantly reduces network traffic overhead by eliminating a large number of primitive events. For example, if a task depends on ten other tasks, instead of forwarding ten messages to the agent, only one notification message is forwarded to the agent in order to decide whether to execute the task or not. As a result, agents do not need extra logic to process the received messages, as composite event detection is performed in the broker network. The network traffic overhead generated by 10 instances is shown in Figure 8. The network traffic is reduced to 63% of a decentralized approach without composite subscriptions in which all primitive events are delivered to agents. It is clear that composite subscriptions can save network bandwidth.

5. Discussion

In this paper we have provided an overview of a decentralized approach to scientific workflow execution and management based on PADRES. PADRES is a distributed

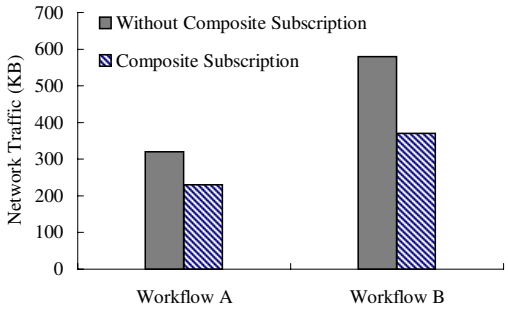

Figure 8. Workflow execution traffic

content-based publish/subscribe system. Our prototype is implemented in Java with JDK 1.4.2 using the Java Expert System Shell (JESS) [7] as a message routing engine, and RMI as the native transport protocol. To accommodate scientific workflow processing, the standard publish/subscribe paradigm had to be significantly extended. This involved the development of an expressive subscription language, supporting composite subscriptions, variable bindings in subscriptions, subscriptions to historic data, and a time-based mechanism to correlate future and past data. We discussed the advantages of using the content-based publish/subscribe paradigm for distributed scientific workflow management. As opposed to a centralized approach, the decentralized workflow management approach presented in this paper does not suffer from a single point of failure, avoids the potential performance bottleneck of a centralized approach, and reduces overall message traffic. Decentralized management, control, and monitoring of workflows is flexibly and naturally supported with the publish/subscribe mechanism.

PADRES is an on-going project. Directions currently under investigation include the discovery of resources in the distributed environment, the static and dynamic planning and scheduling of these resources for a given SWF, and the incorporation of network services as first-class resources for the execution of scientific workflows involving the movement of huge data volumes.

6. Acknowledgments

This research has been supported by Communications and Information Technology Ontario (CITO) a division of the Ontario Centres of Excellence (OCE) Inc., Cybermation Inc., and Sun Microsystems of Canada Inc. We would also like to thank the members of the PADRES team including Alex Cheung, Alex Wun, Eli Fidler, Pengcheng Wan, Shuang Hou, Gerald Chan, David Matheson, and Matt Medland.

References

[1] G. Ashayer, H. Leung, and H.-A. Jacobsen. "Predicate matching and subscription matching in publish/subscribe systems". In *International Workshop on Distributed Event-Based Systems*, 2002.

[2] CANARIE. `http://www.uclp.ca`.

[3] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. "Design and evaluation of a wide-area event notification service". *ACM Transactions on Computer Systems*, 19(3):332–383, 2001.

[4] E. Deelman and J. Blythe. "Mapping abstract complex workflows onto grid environments". *Journal of Grid Computing*, 1(1):25–39, 2003.

[5] T. Fahringer, J. Qin, and S. Hainzer. Specification of grid workflow applications with AGWL: an abstract grid workflow language. In *CCGRID*, pages 676–685. IEEE Computer Society, 2005.

[6] E. Fidler, H.-A. Jacobsen, G. Li, and S. Mankovski. "The PADRES distributed publish/subscribe system". *8th International Conference on Feature Interactions in Telecommunications and Software Systems*, Leisester, UK, 2005.

[7] E. J. Friedman-Hill. "Jess, the rule engine for the java platform". `http://herzberg.ca.sandia.gov/jess/`.

[8] S. Kirishnan, P. Wagstrom, and G. Laszewski. "GSFL: A workflow framework for grid services". `http://users.sdsc.edu/~sriram/talks/`, 2002.

[9] V. Kumar, Z. Cai, B. F. Cooper, G. Eisenhauer, K. Schwan, M. Mansour, B. Seshasayee, and P. Widener. Implementing diverse messaging models with self-managing properties using IFLOW. In *3rd International Conference on Autonomic Computing, Dublin, Ireland*. IEEE Computer Society, 2006.

[10] F. Leymann and D. Roller. *"Production workflow: concepts and techniques"*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2000.

[11] G. Li, S. Hou, and H.-A. Jacobsen. "A unified approach to routing, covering and merging in publish/subscribe systems based on modified binary decision diagrams". *25th International Conference on Distributed Computing System*, Columbus, Ohio, USA, 2005.

[12] G. Li and H.-A. Jacobsen. "Composite subscriptions in content-based publish/subscribe systems". In *ACM/IFIP/USENIX 6th International Middleware Conference*, Grenoble, France, 2005.

[13] G. Mühl. "Generic constraints for content-based publish/subscribe systems". In *Proceedings of the 6th International Conference on Cooperative Information Systems (CoopIS)*, Trento, Italy, 2001.

[14] W. van der Aalst and K. van Hee. *"Workflow management: models, methods, and systems"*. MIT Press, Cambridge, MA, USA, 2002.

[15] G. von Laszewski *et al.* "GridAnt: A client-controllable grid workflow system". *In Proceedings of 37th Hawaii International Conference on System Sciences HICSS*, Hawaii, US, 2004.

[16] J. Yu and R. Buyya. "A taxonomy of workflow management systems for grid computing". Technical report, Grid Computing and Distributed Systems Laboratory, University of Melbourne, Australia, 2005.

[17] Z. Guan *et al.* "Grid-Flow: A grid-enabled scientific workflow system with a petri net-based interface". Technical Report. `http://www.cis.uab.edu`, 2004.