Demo: eQoSystem – Supporting Fluid Distributed Service-Oriented Workflows

Vinod Muthusamy, Young Yoon, Mohammad Sadoghi, and Hans-Arno Jacobsen
Middleware Systems Research Group
University of Toronto
Toronto, Canada
{vinod, yoon, mo, jacobsen}@msrg.utoronto.ca

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous

General Terms

Design, Management

1. INTRODUCTION

Many distributed applications have emerged as Web mashups [1], as well as loosely-coupled decentralized services predominant in a business ecosystem [3]. Many of these applications are implemented as service-oriented workflows and operated over cloud infrastructures. As a result, these applications demand agile development processes and low-touch maintenance life-cycles.

Furthermore, in the cloud environment, application developers must account for the complex multi-tiered ecosystem that includes the services and resources they depend on, but which none of the developers have much control over. Therefore, it is essential for application developers to have tools that proactively adapt the application to the changes of the underlying ecosystem. We meet this need through eQoSystem[1], a framework that provides distributed workflow processing, declarative modeling of service-level agreements (SLAs), event-driven resource selection, and mobility of distributed tasks.

In this demonstration, we focus on dynamically redeploying distributed workflows such that a whole or a part of the workflow is allocated to different execution engines that are possibly geographically dispersed. The redeployment of the workflow is enabled by a federated content-based publish/subscribe overlay that also serves as the basis for driving the autonomous and event-driven execution of workflows.

2. ARCHITECTURE

eQoSystem takes a novel architectural approach whereby individual process instances are executed in a distributed manner. As shown in Figure 1, a distributed execution engine decomposes a workflow, such as a BPEL process, into its individual tasks and deploys these tasks to any set of execution engines in the system. These tasks then coordinate by publishing and consuming events over the PADRES[2] distributed publish/subscribe broker overlay [2]. For example, consider the execution of a task t_i that depends on

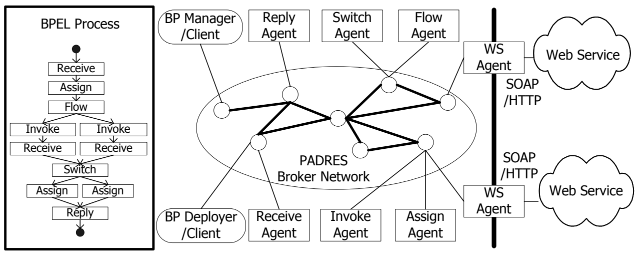

Figure 1: Distributed workflow execution platform

the completion of other tasks. To guarantee correctness, the task t_i must subscribe to the composite events that indicate the completion of all its dependent tasks [5]. The brokers in our distributed model decouple tasks in space and time as publishers and subscribers, thus making the publish/subscribe paradigm particularly well-suited for large and dynamic distributed workflows.

The execution engines in this architecture can be light-weight as they only execute fine-grained tasks, as opposed to complete processes. Another benefit of such an architecture is the ability to deploy portions of processes close to the data they operate on, thereby minimizing bandwidth and latency costs of a process. For example, for data intensive workflows, it is possible to deploy only those portions of the process that require access to large data sets close to their respective data sources.

3. FEATURES

In this section, we review the key features of eQoSystem: declarative SLA modeling, event-driven resource discovery, and transactional mobility.

3.1 Declarative SLA modeling

eQoSystem offers an approach to simplify the development of workflows governed by SLAs [6]. An SLA model, extended from the WSLA language, is used to precisely express the business goals of the process. Portions of the SLA are then mapped to a cost model that captures various relevant factors. For example, the distribution cost represents the overhead of distributing a process into fine-grained tasks, the engine cost captures the resource usage of a task on the engine it is executing at, and the service cost relates to the expense of calling external services.

The model is loosely coupled with the workflow that it is referencing. Developers can therefore evolve both the workflow and SLA model concurrently. In addition, the proposed model is highly modular, and makes it easy to construct complex SLA contracts by composing and extending elementary SLAs.

[1]Project website at http://eqosystem.msrg.org.
[2]Available for download at http://padres.msrg.org.

DEBS'11, July 11–15, 2011, New York, New York, USA.
ACM 978-1-4503-0423-8/11/07.

3.2 Event-driven resource discovery

eQoSystem supports three resource discovery models: static, dynamic, and continuous [7]. The static model is used to discover resources with fixed attributes, the dynamic model is used for resources with attributes that may be updated, and the continuous model is used to allow for real-time notifications of newly registered resources. All three models can co-exist in one system and complement one another. In addition, a similarity-based optimization algorithm takes advantage of publish/subscribe covering techniques in order to reuse the discovery results among different concurrent discovery requests.

3.3 Transactional mobility

It is desirable for the movement of tasks in a publish/subscribe-based distributed workflow engine to be transparent to both the moving task and those it interacts with, such that an application consisting of stationary clients behaves the same as one where clients move. Among other things, this means that clients should not miss any notifications while moving, and their movement should not be visible to others. eQoSystem enables the guaranteed movement of tasks of the distributed workflow according to well-defined properties [4]. The transactional concerns of the publish/subscribe system, client, and routing layers are outlined and atomicity, consistency, and isolation properties for these three layers are specified. Unlike protocols where client mobility is handled by the

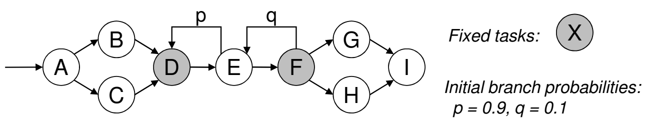

Figure 2: Evaluated sample workflow

4. USE CASES

Given the aforementioned features, eQoSystem can be utilized in many ways. For instance, workflows can be dynamically redeployed at runtime based on the automated SLA monitoring and periodic discovery of candidate resources that can better serve the tasks in the workflows.

The following experiment highlights the benefit of runtime redeployment of a simple workflow consisting of nine tasks as depicted in Figure 2. The tasks are mapped to nine execution engines, with each engine running on a machine with a 1.6 GHz Xeon processor and 4 GB of memory. The execution engines utilize an overlay of PADRES publish/subscribe brokers communicating over a 1 Gbps switch. The process is invoked every second, and each process instance traverses the process graph according to the branch probabilities p and q as indicated in Figure 2. Notice that with 90% probability task E transitions to task D (as opposed to task F) resulting in a process hotspot at tasks D and E. About halfway through the experiment, the transition probabilities of p and q are reversed, so that a hotspot now occurs at tasks E and F. Due to the initial tight loop around tasks D and E, the message overhead can be reduced if these tasks were deployed on the same engine. Hence, it is beneficial to first reconfigure the workflow deployment by moving tasks A, B, C, and E to the same engine as task D, and by redeploying tasks G, H, and

5. DEMONSTRATION

The demonstration is focused on the mobility of distributed workflows. We show how eQoSystem is developed on top of PADRES

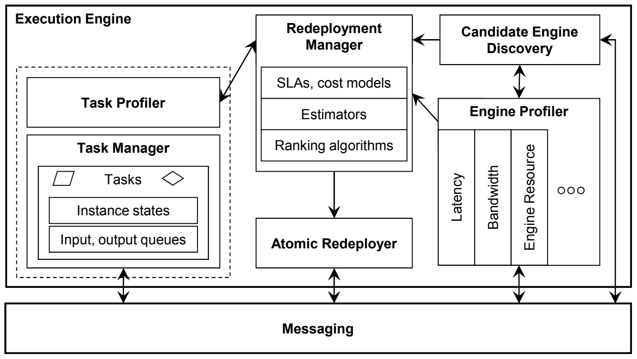

Figure 3: Architecture of autonomic execution engine

by presenting the usage of key publish/subscribe messaging APIs in-depth at a source level. Then, we illustrate the end-to-end scenario of using eQoSystem: (1) decomposition of a workflow; (2) deployment of decomposed tasks; (3) event-driven execution of the workflow; and (4) migration of tasks to different locations. This demonstration is accompanied by a workflow execution monitor that tracks messages traversing the PADRES broker overlay network.

6. CURRENT STATUS

More advanced components (as shown Figure 3) are being developed to automate the redeployment procedure. Specifically, a *Candidate Engine Discovery* component is used to find other execution engines in the system, and these candidates are periodically probed by the *Engine Profiler* to gather various statistics. The *Redeployment Manager* computes the cost function for each task executing in the engine, and determines if a more optimal placement of the task is available among the known candidate engines. Finally, tasks that are to be moved are redeployed using the *Atomic Redeployer* component which is responsible for ensuring that the movement of the task does not affect the execution of the workflow.

7. REFERENCES

[1] Yahoo Pipes. http://pipes.yahoo.com.

[2] E. Fidler, H.-A. Jacobsen, G. Li, and S. Mankovski. Distributed publish/subscribe for workflow management. In *ICFI*, 2005.

[3] S. Hu, V. Muthusamy, G. Li, and H.-A. Jacobsen. Distributed automatic service composition in large-scale systems. In *DEBS*, 2008.

[4] S. Hu, V. Muthusamy, G. Li, and H.-A. Jacobsen. Transactional mobility in distributed content-based publish/subscribe systems. In *ICDCS*, 2009.

[5] G. Li, V. Muthusamy, and H.-A. Jacobsen. A distributed service-oriented architecture for business process execution. *ACM Trans. Web*, 2010.

[6] V. Muthusamy, H.-A. Jacobsen, T. Chau, A. Chan, and P. Coulthard. SLA-driven business process management in soa. In *CASCON*, 2009.

[7] W. Yan, S. Hu, V. Muthusamy, H.-A. Jacobsen, and L. Zha. Efficient event-based resource discovery. In *DEBS*, 2009.