Runtime Fault-Handling for Job-Flow Management in Grid Environments

Gargi Dasgupta[1], *Onyeka Ezenwoye*[2], *Liana Fong*[3], *Selim Kalayci*[4], *S. Masoud Sadjadi*[4] *and Balaji Viswanathan*[1]
[1] IBM India Research Lab, New Delhi, India, {gdasgupt,bviswana}@in.ibm.com
[2] South Dakota State University, Brookings, SD, USA, onyeka.ezenwoye@sdstate.edu
[3] IBM Watson Research Center, Hawthorne, NY, USA, llfong@us.ibm.com
[4] Florida International University, Miami, FL, USA, {skala001,sadjadi}@cs.fiu.edu

ABSTRACT

The execution of job flow applications is a reality today in academic and industrial domains. In this paper, we propose an approach to adding self-healing behavior to the execution of job flows without the need to modify the job flow engines or redevelop the job flows themselves. We show the feasibility of our non-intrusive approach to self-healing by inserting a generic proxy to an existing two-level job-flow management system, which employs job flow based service orchestration at the upper level, and service choreography at the lower level. The generic proxy is inserted transparently between these two layers so that it can intercept all their interactions. We developed a prototype of our approach in a real Grid environment to show how the proxy facilitates runtime handling for failure recovery.

Keywords: job-flow management, meta-scheduler, generic proxy, fault-tolerance, job-flows.

1. INTRODUCTION

In a Grid computing environment, individual jobs are typically executed in the context of higher-level functional units known as job flows. Our ongoing work in the Job Flow Management project [1] addresses many specific issues related to job flow management in Grid computing environments. In this paper, we focus on adding self-healing to job flow management in a non-intrusive manner. We have developed a two-level distributed architecture that is illustrated in Figure 1. As shown in the figure, the main middleware components that are involved in this architecture are the *job flow manager*, responsible for maintaining concurrency and sequencing among jobs in the flow, and the *meta-scheduler*, responsible for resource selection and job execution control.

In this figure, there are two resource domains, namely, FIU and IBM and each are managed by their representative job flow manager along with a meta-scheduler. The job-flow manager submits individual jobs to the meta-scheduler or partial workflows (sub-flows) to the peer job flow managers. Within a domain, the meta-scheduler can route jobs for execution to multiple local schedulers. Peering relationships between job-flow managers and between meta-schedulers are established through a set of protocols that exchange dynamic resource capacity and capability information. This enables them to route sub-flows/jobs for remote execution at partner domains.

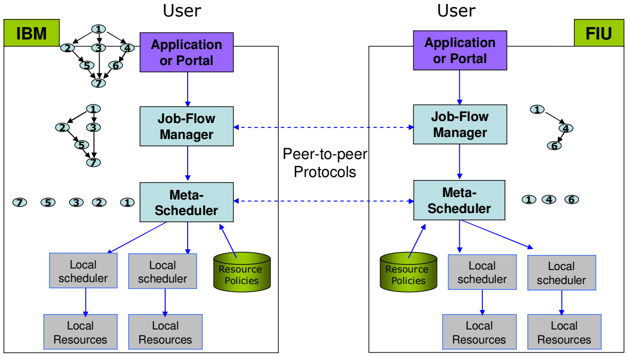

Figure 1. A distributed architecture for flow manager and meta-scheduler spanning multiple domains.

Due to the long running nature of these jobs and sub-flows, the support for fault tolerance and recovery strategy is especially important. Moreover, the interaction of Grid services with dynamic distributed resources makes fault-tolerance a critical aspect of job flow management. Run-time job flow failures need to be addressed for individual jobs as well as for sub-flows. Specification of recovery actions and handling of individual job failures can be handled at the local scheduler level. However, the failure of a single job within a job-flow often cannot be treated in isolation and recovery actions may need to be applied by taking the dependency between jobs into consideration. Thus, specifying flow level recovery mechanisms become important in such scenarios.

A prevalent way to handle flow level compensation is to include failure management logic at modeling time [2]. However, this requires modification of the original workflow to incorporate additional fault-handling logic and also assumes pre-knowledge of all different failure scenarios that can arise. An alternate approach is to handle these job failures at runtime, without explicit changes to job flow process logic. The TRAP/BPEL [3] framework employs this approach for stateless Web service orchestration. In TRAP/BPEL, an intermediate proxy intercepts calls from the flow engine, and deploys runtime failure handling on behalf of the workflow. The advantage of this technique is that no changes need to be made to the workflow at modeling time. In this paper, we utilize this approach to enable runtime job-flow failure handling in Grid environments, with dynamic selection of recovery policies.

978-0-7695-3175-5/08 $25.00 © 2008 IEEE
DOI 10.1109/ICAC.2008.16

2. PROTOTYPE IMPLEMENTATION

This section presents our fault-tolerant job flow management prototype setup across two sites at IBM and FIU. For building the IBM domain, we use IBM's Websphere Process Server (WPS) and IBM Tivoli Dynamic Workload Broker (ITDWB) [4] components. For building the FIU domain, we use the ActiveBPEL engine and the Globus Toolkit 4 (GT4) [5] and Gridway meta-scheduler [6]. The prototype implementation resulted in the overall job-flow execution environment in Figure 2. An adapted job-flow in WS-BPEL format is intercepted by a generic proxy that uses the underlying meta-scheduler functionality to submit job, monitor job or get notifications about the job. In case of a failure, the generic proxy makes use of its policies and takes the necessary action specified in the policy.

The meta-scheduler accepts JSDL job and data activity submissions from the flow manager and executes them. It also optionally provides a notification on job state changes (e.g., GRAM job state changes as specified in [7]). Our meta-scheduling implementation [1] provides the necessary interface to the job-flow manager and facilitates the scheduling jobs in remote domains. Two different implementations with different internal architecture and components at FIU and IBM can basically submit and monitor JSDL jobs through a peer-to-peer protocol. Similar to the job-flow manager, but in a more detailed level, the meta-scheduler contains the intelligence for deciding which jobs to route locally and which to dispatch to remote domains. This comprises the brain of the system and particular scheduling algorithms used are dependent on the domain and its specific policies.

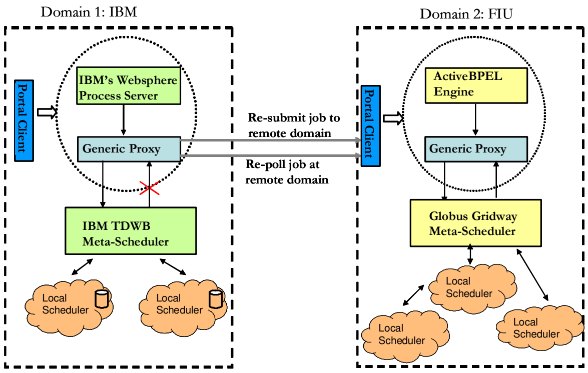

Figure 2. Prototypical architecture of our fault-tolerant job-flow management.

For runtime failure management at the level of individual jobs, we use the generic proxies. The proxy sits between the flow manager and the meta-scheduler, and intercepts calls in both directions. For all monitored invocations, the meta-scheduler interface calls are replaced with calls to the proxy interface. However, the proxy is transparent to the flow manager and to the meta-scheduler; therefore, it imposes no changes in either component. A recovery component kicks in when a failure is detected for any adapted component. In the recovery phase, the proxy applies recovery policies to the failed invocations. The policies contain rules to detect failures and a sequence of recovery actions to follow on failure detection.

An extensible repository of job flow as well as fault-tolerant patterns is maintained at the proxy. Job flow patterns comprise of common behavior that is prevalent in job-flows, represented using the combination of a flow language and a job definition language (e.g., a job submission activity is typically followed by a monitor job state activity, or a job submission activity is typically preceded by a data-staging activity etc.). The proxy by virtue of maintaining conversational state for each job is well equipped to detect and handle failures. Fault-tolerant patterns comprise common reusable recovery actions that can be specified for job flow failures (e.g. re-submit job with modified job parameters, or re-submit job to a different domain etc.). The mapping between job flow patterns and fault tolerant patterns can be manually defined at modeling time by the application developer or using pre-defined rule trees. This design of the proxy allows designers to dynamically define new failure handling.

3. CONCLUTION AND FUTURE WORK

In this paper, we present a fault-tolerant architecture for handling failures at runtime in case of long-running workflow environments, using a job-flow manager, a generic proxy and a meta-scheduler. We build a real prototype solution using standard-based components that demonstrates feasibility of the approach. In future work, we plan to extend our work to a comprehensive set of failure scenarios, explore automatic generation of mapping between job-flow patterns and fault-tolerant patterns and study their performance impacts.

Acknowledgment: This work was supported in part by IBM and the National Science Foundation (grants OISE-0730065, OCI-0636031, REU-0552555, and HRD-0317692).

REFERENCES

[1] Rosa Badia *et al*. *High Performance Computing and Grids in Action*, chapter Innovative Grid Technologies Applied to Bioinformatics and Hurricane Mitigation. IOS Press, Amsterdam, 2007. (accepted for publication)

[2] W. Tan, L. Fong, and N. Bobroff. Bpel4job: a fault-handling design for job flow management. In Proceedings of Fifth International Conference on Service Oriented Computing (ICSOC), 2007

[3] Onyeka Ezenwoye and S. Masoud Sadjadi. TRAP/BPEL: A framework for dynamic adaptation of composite services. In Proceedings of the International Conference on Web Information Systems and Technologies (WEBIST 2007), Barcelona, Spain, March 2007.

[4] V. Gucer, M. Lowry, and F. Knudsen. End-to-End Scheduling with IBM Tivoli Workload Scheduler Version. IBM Press, 2004.

[5] I. Foster and C. Kesselman, Globus: A Metacomputing Infrastructure Toolkit, Proceedings of the Workshop on Environments and Tools for Parallel Scientific Computing, SIAM, Lyon, France, August 1996.

[6] E. Huedo, R.S. Montero and I.M. Llorente. The GridWay Framework for Adaptive Scheduling and Execution on Grids, Workshop on Adaptive Grid Middleware, Intl. Conf. Parallel Architectures and Compilation Techniques (PACT 2003), New Orleans, USA, September 2003.

[7] Globus Alliance. GT 4.0 WS GRAM: Developer's Guide. Available from http://www.globus.org/toolkit/docs/4.0/execution/wsgram/develop-er-index.html