THE UNIVERSITY OF TEXAS AT AUSTIN

THE CENTER FOR ADVANCED RESEARCH IN SOFTWARE ENGINEERING

(UT ARISE)

WHITE PAPER

AN ASPECT-ORIENTED APPROACH FOR FINE-GRAINED CONTROL AND ALLOCATION OF RESOURCES FOR COMPUTATIONAL GRIDS

Mark Grechanik and Dewayne E. Perry

October 2003

INTRODUCTION

One ultimate purpose of computational grids is to enable various organizations to share existing computing resources. To view computer hardware and software as resources creates serious challenges with their fine-grained control and allocation in a grid environment. Specifically, users attempt to gain access to different resources whose owners should be able to exercise a fine-grained control of such access depending on various local factors while ensuring that the overall security and computational integrity of the system is not compromised. This problem is especially exacerbated in corporate enterprise environments where a slight breach in security may lead to disastrous consequences thereby holding back the much needed grid systems that can be used to solve many important problems.

Existing grid solutions are typically based on user-level programs called *agents* running under minimum-security privileges. These agents can only accomplish parallel data processing tasks, and fall short of enabling fine-grained access to selected resources. Moreover, many agents are based on a polling mechanism that wakes them up at predefined time intervals to run some tasks and then puts these agent programs back to sleep. Polling agents often miss events that occur in the midst of the polling interval, and waste computational resources when awakened at times when their services are not needed.

We introduce a novel approach called *monitoring and administering computer resources* (*MARS*) in a grid environment that allows grid participants to exercise a fine-grained control and allocation of computer resources uniformly. In our approach, we:

- treat hardware and software resources as first-class objects that can be monitored and manipulated;
- reify the states of the resource objects via operations so that monitoring can be done uniformly and with minimum complexity;
- reify the relationship topology among resource objects so that it can be manipulated; and
- use aspects to provide the monitoring and manipulation of these resource objects.

SOME MOTIVATING EXAMPLES

We demonstrate the importance of fine-grained control of grid tasks and difficulties in solving them using examples of process monitoring, fine-grained security, and the redirection of resource accesses. These are common problems encountered every day by developers and system administrators in the distributed enterprise environments and grids, yet there is no simple and uniform way to solve them.

Consider a situation when a multiple copies of some application are installed on computers in a grid environment. During the execution this application accesses some remote resources.

Suppose that an administrator decides to group these applications to use different remote resources for each group. It means that each copy of the application should be reconfigured using its GUI or configuration file. This is a manual and laborious task to which no simple and satisfactory solution exists (apart from our approach). With our MARS-based fine-grained uniform solution administrators can enable the redirection of each copy of the application to access the appropriate resources easily and uniformly.

Many grid tasks require that certain application should execute and need to ensure that other grid programs do not "steal" resources from it. Consider a situation shown in Figure 1 when process A executes starting at time A_s^1 and finishing at time A_e^1. Process A should be given the highest priority, and the task of a grid administrator is to suspend other processes that try to run simultaneously with A. Suppose that the grid administrator is an agent that polls at times t_1 and t_2 to monitor the computer state. Between the t_1–t_2 time interval the process B starts at time B_s and finishes at time B_e. Thus, the process B avoids being detected by the grid agents, and it may interfere with the execution of the process A.

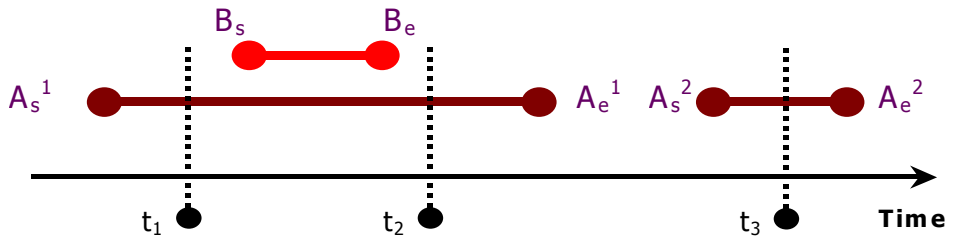

Figure 1. Process B executing concurrently with instances of the process A.

Suppose that process A terminated at time A_e^1 and its new instance started at time A_s^2. The polling agent detects this instance at time t_3, however, it is unable to tell whether it is a new instance. The process identifier may be reused by the operating system (OS) and assigned to the new instance of A. Existing grid solutions that enable real-time detection of events associated with the asynchronous start and termination of programs involve OS kernel modifications that makes them difficult and impractical.

Computer security adds many important tasks to the grid task roster. Many OSes as well as grid solutions do not provide any fine-grained security to application deployment and resource allocation and control. Consider the following situation: a group of users with limited privileges run some application from different computers and that application requires access to a remote resource, for example, a CD-Writer. While the existing privileges do not allow these users to access this CD-Writer this restriction should be lifted when this application calls some Application Programming Interface (API) that requires access to this resource and reset right after when the call is completed. Our approach provides a solution to this problem.

OUR SOLUTION

Existing grid solutions are mostly based on agents. A grid agent is a user-level program that executes under minimal security privileges and employs primitive ad-hoc techniques to control the behavior of computer resources. A wide spectrum of these solutions, as well as the versatility of

various ad-hoc techniques that depend on specific applications and platforms on which they run, hide common properties of monitoring and administering mechanisms. We uncover these properties and present them collectively in our MARS model.

Our approach is based on converting low-level API resource calls into system-wide events that MARS programs can monitor. This conversion is accomplished by using `advice` (an aspect-oriented programming (AOP) concept) that contains event-generating code at `join points` in programs that represent grid resources. `Advice` is applied by instrumenting low-level API calls (using binary rewriting techniques) to produce the desired event notifications. We abstract and group low-level resource APIs by imposing a transactional metaphor that significantly reduces the complexity of reasoning about grid resources.

A grid resource changes its state after a client program executes some API that modifies values of some internal variables of this resource. This is a fundamental property upon which any administrating and monitoring solution is based. Suppose we have an observer who "lives" inside a CPU, "watches" internal variables of computer resources, and notifies us when their values change. If this observer can also modify the values of these variables on our behalf, then we can call him/her a MARS observer and manipulator.

A MARS observer who notifies us about changes of values of any variables and can also change these values on our behalf is the core of our MARS model. Since client programs change the values of internal variables of computer resources by calling APIs, our observer can watch for calls to certain functions that lead to changes of monitored variables and notify us about invocations of these functions. The MARS manipulator can go further by executing MARS functions that administer resources `before`, `after`, or `instead` of invoked APIs. In the MARS model the observer/manipulator uses APIs as surrogates for monitoring and manipulating states of resources.

The behavior of the MARS observer/manipulator can be easily explained using AOP concepts. The observer can be viewed as a MARS *aspect* that is applied to computer resources. Different APIs that are located in different libraries and programs that manipulate the same resource represent a *crosscut*. A MARS aspect introduces a set of standard *advice* to resource crosscuts. For example, handling notifications about changes in the state of monitored resources is accomplished by applying `before` advice to APIs that manipulate these resources.

We categorize APIs that change the state of computer resources by using a transactional metaphor. Some APIs initialize or open a resource, some APIs perform read from or write to a resource, and others close resources. By creating such categories we enable the MARS observer to notify us that some resource has just been written to by some process rather than to produce a cryptic message stating that some API has been executed with a list of its parameters.

A high-level logical view of the MARS model is shown in Figure 2. At the top level a MARS observer and manipulator detects changes in states of computer resources as well as manipulates their behavior. This observer/manipulator accomplishes this work using event and AOP models that are based on binary rewriting mechanisms. Binary rewriters are a part of low-level implementation of our MARS approach.

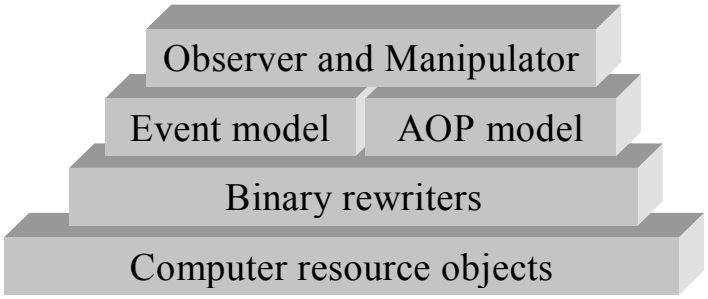

Figure 2. A logical view of MARS model.

CONCLUSIONS

The resulting contribution of our work is a powerful way to apply fine-grained monitoring and administration mechanisms to arbitrary computer resources in a grid environment. We show how to reduce the significant complexity associated with the development of MARS software by enabling a simple and powerful MARS model for monitoring and manipulating grid resources. We allow programmers to operate on resources as first-class objects thereby presenting a uniform way to write grid-enabled MARS programs. In reifying the state of grid-enabled resources, we impose a transactional metaphor on MARS systems, thus simplifying the event delivery mechanism by reducing tens of thousands of different events to only five event categories. By reifying the relationship topology of grid resources, we enable the manipulations of those relationships and thereby the use and control of those resources.

We have prototyped the basic ideas to show a proof of our basic concepts and simulated a performance model of MARS to evaluate its viability. The results of our evaluation showed the feasibility of our approach and its applicability to solve important problems in today's grid computing environments.