A Framework for Rule-Based Management of Parallel Scientific Applications *

Hua Liu and Manish Parashar
The Applied Software Systems Laboratory
Dept of Electrical and Computer Engineering, Rutgers University, Piscataway, NJ08854, USA
Email:{marialiu, parashar}@caip.rutgers.edu

1. Introduction

Large-scale parallel/distributed simulations are playing an increasingly important role in science and engineering and are rapidly becoming critical research modalities in academia and industry. However, the increasing complexity, scale and dynamism of these applications and their underlying computing environment, make their efficient and scalable formulation and runtime management a significant challenge. This is primarily because the requirements, objectives and choice of specific solutions depend on runtime state, context, and content, and are not known a priori.

In this paper we present the design of the rule-based framework for the runtime management of high-performance parallel scientific applications. The framework addresses the formulation of application management behaviors as reaction rules, the injection of rules at runtime, their correct, efficient and scalable parallel enforcement, and the detection and resolution of rule conflicts. Unlike rule-based frameworks in business management and security and resource management domains, the presented framework focuses on high-performance parallel scientific applications, which require consistent and efficient management across processors and components. The framework is part of the Accord programming system [3].

2. Framework Design and Operation

Rule Formulation: Reaction rules used by the framework capture part of the application process that can be adapted. The rules can be dynamically added, deleted and modified during application execution, and are interpreted and executed at runtime by a parallel rule engine. The rules incorporate high-level knowledge in the form of if-then expressions, i.e., *IF condition THEN action*. This simple construction of rules is deliberately used to enable efficient execution and minimize impact on the performance of the applications. The *condition* is a logical combination of sensors and events exposed by components and the system, and the *action* consists of a sequence of invocations of actuators exposed by components and the system. Two classes of rules are defined: (1) *Component rules* manage the runtime behaviors of individual components; (2) *Composition rules* manage the structure of the application and the interaction relationships among components via dynamically replacing components.

Rule Execution Model: Traditional rule-based systems directly invoke actions when rules fire [1]. However, this approach aggravates rule conflicts when multiple rules are simultaneously triggered and can lead to both uncertainty and inconsistency in rule execution. The framework presented in this paper employs a three-phase rule execution model, consisting of (1) batch condition inquiry, (2) condition evaluation and conflict resolution and reconciliation, and (3) batch action invocation.

The batch condition inquiry phase queries all the sensors S in parallel, gets their values VS, and then generates the *pre-condition*. Based on this *pre-condition*, rules whose conditions are satisfied form the active rule space $\bar{R}$. In next phase, condition evaluation for all the rules in $\bar{R}$ is performed in parallel. The overall evaluation time in this case will be determined by the longest evaluation time for an individual rule. Conflict resolution and reconciliation then takes place and the *post-condition* is generated. In the final phase, the actuators A in the *post-condition* are invoked to produce the *consequence*. This may also be done in parallel, since the actuators in the *post-condition* are independent and free of conflicts. Note that as the rule base becomes larger, the conflict resolution time will increase. However the time required for sensor queries, condition evaluations and actuator invocations will not change too much.

Conflict resolution consists of two steps. First, the rules that will change the *pre-condition* are disabled and a new set of rules $\bar{R}'$ are produced. Let SA represent the variables exposed both as sensors and actuators by the managed com-

* The research presented in this paper is supported in part by the National Science Foundation via grants numbers ACI9984357 (CAREERS), EIA 0103674 (NGS), EIA-0120934 (ITR), ANI-0335244 (NRT), CNS-0305495 (NGS).

ponents, i.e., $SA = S \bigcap A$.

if $SA \neq \phi$ then for each rule $R_i \in \bar{R}$

- *if $SA \bigcap A_i \neq \phi$ and $\exists s_i \in SA \bigcap S_i$ and $a_i \in SA \bigcap A_i$ and $a_i = s_i, Value(s_i) \neq Value(a_i)$, then disable R_i from $\bar{R}$.*

The *pre-condition* is then relaxed by incrementally 'deleting' sensors in CS (a sequence of user specified sensors), until $\forall a \in \bigcap A_i, Value(a)$ has at least one value, or all the senors in CS are exhausted.

if $\bigcap A_i \neq \phi, \forall R_i \in \bar{R}'$, $\exists a \in \bigcap A_i, \bigcap Value_i(a) = \phi$ then

- *repeat*
  - *read the next cs from CS*
  - *relax cs in the pre-condition*
  - *re-evaluate rules*
- *until $\forall a \in \bigcap A_i, Value(a)$ has at least one value, or CS is exhausted.*
- *if CS is exhausted, an error is reported to users for further instructions, else, the post-condition $\{A, VA\}$ is constructed by randomly selecting a value for those actuators having multiple values.*

Reconciliation is required to generate a consistent *post-condition* for parallel SCMD applications, as each node may independently generates a different *post-condition* based on its local context. Rules are statically assigned one of two priorities. A high priority means that the execution of the rule is necessary, for example, to avoid an application crash. A low priority means that the execution of the rule is optional. During reconciliation, actions associated with the rule with high priority are propagated to all the nodes. If there are multiple high priority rules, a runtime error is generated and reported to the user. If only low priority rules involved, reconciliation uses cost functions to select the most appropriate action at all nodes.

3. Experiment Evaluation

The key concepts underlying the rule framework have been prototyped in an Accord-based Ccaffeine [2] CCA framework and evaluated using the CH_4 ignition simulation on a 64 node beowulf cluster. The overheads associated with the initialization and runtime rule execution were evaluated.

Experiment 1 (Figure 1): This experiment measures the runtime overhead introduced by the framework in a minimal rule execution mode, i.e., rules are loaded but the execution is disabled. The application execution time with and without the framework are plotted on the left and the percentage overhead is plotted on the right in Figure 1. The major overhead in this case is due to the loading and parsing of rules.

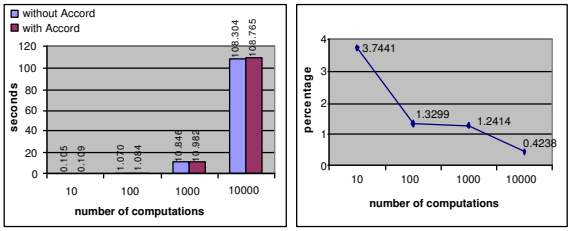

Figure 1. The runtime overhead introduced in the minimal rule mode.

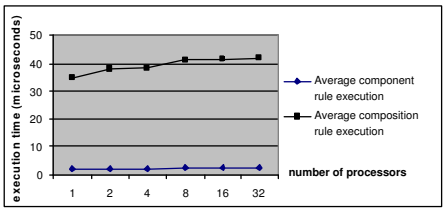

Figure 2. The overhead introduced by executing component and composition rules.

Experiment 2 (Figure 2): This experiment evaluates the average execution time of component rules and composition rules. As the number of processors increases, the average execution times of both, the component rules and the composition rules, increase slightly. This is reasonable since nodes must communicate with each other during reconciliation. The figure also shows that the average execution time of the composition rules is much larger than that of the component rules. This is because, in execution of composition rules, a new component will be instantiated, connected to other components, and loaded with new rules. However, the execution of component rules only involves invoking component actuators.

References

[1] A. Abrahams et al. An asynchronous rule-based approach for business process automation using obligations. In *Third ACM SIGPLAN Workshop on Rule-Based Programming (RULE'02)*, pages 323–345, 2002.

[2] B. Allan et al. The CCA core specification in a distributed memory SPMD framework. *Concurrency Computation*, 14(5):323–345, 2002.

[3] H. Liu, M. Parashar, and S. Hariri. A component based programming framework for autonomic applications. In *Proceedings of The 1st IEEE International Conference on Autonomic Computing (ICAC-04)*, 2004.