Chapter 7
Monitoring and Controlling Large Scale Systems

INTRODUCTION

The architectural shift presented in the previous chapters towards high performance computers assembled from large numbers of commodity resources raises numerous design issues and assumptions pertaining to traceability, fault tolerance and scalability. Hence, one of the key challenges faced by high performance distributed systems is scalable monitoring of system state. The aim of this chapter is to realize a survey study of existing work and trends in distributed systems monitoring by introducing the involved concepts and requirements, techniques, models and related standardization activities.

Monitoring can be defined as the process of dynamic collection, interpretation and presentation of information concerning the characteristics and status of resources of interest. It is needed for various purposes such as debugging, testing, program visualization and animation. It may also be used for general management activities, which have a more permanent and continuous nature (performance management, configuration management, fault management, security management, etc.). In this case the behavior of the system is observed and monitoring information is gathered. This information is used to make management decisions and perform the appropriate control actions on the system. Unlike monitoring which is generally a passive process, control actively changes the behavior of the managed system and it has to be considered and modeled separately. Monitoring proves to be an essential process to observe and improve the reliability and the performance of large-scale distributed systems.

There are a number of fundamental issues related to monitoring of distributed systems. In a distributed environment a large number of events are generated by the system components during its execution or interaction with external objects (e.g. users or processes). Monitoring such events is necessary for observing the run-time behavior of large scale distributed systems and providing status information required for debugging, tuning and managing such applications (Al-Shaer et al., 1997). However, error/failure events (due to software bugs or improper implementation) or events that convey the application status are more likely to appear and can be dispersed over many different

DOI: 10.4018/978-1-61520-703-9.ch007
locations and application entities. This makes the tasks of testing, debugging, monitoring and the management decisions process (e.g. fault recovery or performance tuning procedures) much harder to achieve in distributed environments. Furthermore, the correlated monitored events, either simple (local) or complex (global) events, are generated concurrently and could be distributed in various locations in the applications environment, which complicates the management decisions process and thereby makes monitoring systems an intricate task. For example, application errors related to the communication operations obviously involve observing the sender(s) and the receiver(s). Similarly, the knowledge of application performance is also distributed in the application environment. For instance, calculating the average load of the system must involve all participant machines. As monitoring data may encounter network latency, delays in transferring it from the place of production to the place of storage are inherent which means it could be out of date. Hence, it proves difficult to obtain a global, consistent view of a distributed system. Clock skewing problems and variable delays in reporting events may result in recording events in the incorrect order. Another problem is that the large volume of event traffic which flows in the system may swamp the monitoring process, therefore the monitoring systems should scale gracefully with the number of nodes of the distributed system. Moreover, as the monitoring system shares the resources with the observed system, intrusiveness becomes a key issue which may alter the accuracy of the monitoring process.

In order to overcome these problems, one should design a monitoring system with respect to a set of requirements which we detail in the next section. We further discuss in this chapter the existing architectural models and the implemented solutions.

MONITORING REQUIREMENTS

Due to the distributed nature and the large number of participants or application entities, reliability and performance of specific applications become critical issues. The wide geographical distribution and large interaction of such applications may sometimes increase the possibility of failures/errors or performance bottlenecks. Unlike centralized or isolated applications, large scale systems are likely to deal with different environments that may be dynamically changing. For these reasons, observing the run-time behavior is essential to discover reliability and performance problems and initiate the proper reactions to alleviate these problems. These actions are performed either at run-time such as fault recovery and applications steering or at development-time such as fixing bugs and design enhancements. The system developers or managers (could be human or software components) require feedback information on the system behavior for testing and debugging purposes or for fault recovery and performance tuning procedures. Monitoring distributed systems is an effective means to observe applications at run-time and provide this feedback information to the system developers and system managers in order to improve reliability, robustness and performance of large scale systems.

In general, monitoring is essential to improve the quality of any process (e.g. management process and production control process). Similarly, in software systems, the process of developing, maintaining and operating distributed applications can be complex. An efficient monitoring of these applications is an essential mechanism to produce good quality applications (e.g. reliable, robust, secure, high-performance). The necessity of monitoring significantly increases when using large-scale distributed systems since they are more susceptible to many issues such as reliability and performance and also more difficult to debug and steer. We can summarize the motivation for monitoring complex systems as follows.