MONITORING & MANGING DYNAMIC DISTRIBUTED SYSTEMS

Dileban Karunamoorthy and Neomal Devinuwara

Department of Computer Science, Informatics Institute of Technology

ABSTRACT

The growing number of middleware for building distributed systems has made it increasingly difficult to monitor and manage them. This is especially the case when different and often interacting systems based on different middleware technologies are deployed on a single site. Proper administration of such systems requires proper monitoring, which in turn requires the extraction of quality information from the underlying systems at runtime. The proxy pattern, although seemingly simple, can be put to good effect to address problems of this nature. By dynamically generating proxies capable of intercepting and logging invocations the characteristics, behavior and relationships of processes can be ascertained at runtime, providing a global view on the behavior of entire systems in operation. This paper addresses these two concerns, the need to have a comprehensive global view on distributed systems and the viability of the proxy pattern as a technique for achieving it.

1. INTRODUCTION

Distributed computing has literally revolutionized the way organizations and people work, so much that the terms *distribution* and *computing* have become inseparable. Various middleware frameworks for building distributed applications have evolved over the years, each based on different conceptual models, geared to solve different classes of problems, and based on very different technologies. CORBA for example, is a specification for building general purpose distributed object-based applications and addresses numerous classes of problems [1]. Implementations exist for different platforms and in different languages. Other popular examples include Jini, based on a general purpose service-oriented model with emphasis on the formation of dynamic or spontaneous coordination systems called *federations* [2]; JavaSpaces, a part of Jini, provides temporal and referential uncoupling of processes based on a Linda-like coordination model [3]; JXTA, a special purpose framework providing an infrastructure for building peer-to-peer applications [4]; Java RMI and .Net Remoting, simple object and event based models for remote invocation;

While these are just a few examples, the growing numbers have made it increasingly difficult for organizations to monitor and administer them. This is especially the case when organizations find themselves having to deploy multiple middleware layers to address different organizational concerns. Often these systems are also integrated via gateways to reuse existing developments and business logic. The problem is further compounded by the very nature of the technologies involved. For example, while some frameworks mandate the use of specific protocols for client-server communication, others impose no such restrictions, using any proprietary protocol. Systems based on frameworks like Jini are very dynamic in nature, allowing a collection of independent processes to announce their presence, find other processes and form coordinating federations that will soon disintegrate once the task at hand is complete. In contrast, systems based on CORBA tend to be more static in nature, with compile-time bindings to stubs. (CORBA however also possesses a dynamic aspect like Jini, with the inclusion of DII and DSI [5])

The lack of global view on the systems in operation makes it hard to ascertain the overall efficiency and usage of the systems. Such a view would require capturing key information at runtime. It is common though for enterprise level applications and some middleware implementations to include facilities for logging information at various points in the execution flow. Often such logging is enabled by the setting of a flag in a configuration file and making use of logging frameworks such as Log4J [6]. While such information is useful for tracing and debugging errors, they can prove to be too inadequate for understanding the behavior system as a whole over a period of time.

What is required is the extraction of key temporal and non-temporal information such as method level interactions between processes; data exchanged via method parameters; locations; operating platform details; hardware resource usage; network broadcasts; etc. and the presentation of such information in summarized tabular and graphical forms (topological views, graphs, etc.), that can help administrators make informed decisions on the behavior of the systems over a period.

2. MOTIVATION

Capturing such key information can provide useful insights. For example, being able to observe client-server interactions can help not only in tracing and debugging errors, but also in analyzing load distributions and performance bottlenecks across the system over a period of time. Such information, together with details on client/service locations and hardware resource usage can help administrators pin-point performance problems or