Performance Engineering of Distributed Software Process Architectures

Greg Hills\textsuperscript{1}, Jerome Rolia\textsuperscript{2} and Giuseppe Serazzi\textsuperscript{3}

\textsuperscript{1} School of Computer Science, Carleton University, Ottawa, ON, Canada. K1S 5B6
\textsuperscript{2} Department of Systems and Computer Engineering, Carleton University, Ottawa, ON, Canada. K1S 5B6
\textsuperscript{3} Dipartimento di Elettronica e Informazione, Politecnico di Milano, Italy

Abstract. An important goal of a system's development team is to provide a software structure that evolves gracefully with its workload's intensity and characteristics, and the technologies that support the system. We describe a computationally efficient technique that helps us recognize potential software bottlenecks in a distributed software system over a range of workload conditions. Using this technique, software changes needed to support the workload over time can be identified early. Support for these software changes can be planned in advance and built into the system's architecture. The engineering structures from the Reference Model for Open Distributed Processing (RM-ODP) are used as the basis for our software performance modelling. A case study is given that demonstrates how the proposed technique can be applied when implementing a distributed application in an environment such as the Open Software Foundation's (OSF) Distributed Computing Environment (DCE).

Keywords: bottlenecks, client-server, distributed applications, distributed systems, software architecture, software performance engineering

1 Introduction

To benefit from more cost effective and flexible technologies, organizations are moving away from centralized systems and implementing distributed systems. The need to share information across organizational boundaries leads to open distributed processing systems. Since organizations rely on distributed systems to accomplish their every day tasks and these applications can span organizational boundaries, understanding their performance behaviour is essential.

When designing a distributed application, both software structure and the underlying hardware and software technologies that support the distributed system have an impact on performance. In this paper we consider how to study the impact of software structure and hardware technology on application performance. We describe the distributed application in terms of the engineering structures of the Reference Model for Open Distributed Processing (RM-ODP). This allows us to model the application in terms of its individual components such as objects, operating system processes and nodes. In this way, our modelling environment provides enough abstraction to develop performance models for applications built using midware environments such as DCE [1] and the Object

Within a distributed application, *software technologies* can impose constraints or limits on the number of concurrently active customers that *software resources* (i.e. software components) such as objects, operating system processes and nodes are able to support. It should be noted that while hardware resources have a utilization less than or equal to one (100%), a software resource may have a utilization limit greater than one. For example, a critical section within an application would have a maximum utilization level of one: at most one customer is able to use the critical section at a time. On the other hand, a database subsystem may permit up to $n$ concurrently executing transactions at a time. In this case the utilization limit is $n$. In DCE, the number of concurrent threads that support a server’s interface is fixed when the process starts to accept calls. At a node level, there may be limits on the total number of concurrent processes or threads supported by the node.

If these software technology constraints, i.e. utilization limits, are approached then software queueing delays arise or requests for service are lost. Software queueing delays can increase a request’s response time and limit an application’s ability to fully exploit its underlying hardware. Under these scenarios, customers queue at the software resources instead of queueing at devices. A software server is a bottleneck when it reaches its utilization limit. Obviously, hardware can also limit the performance of a distributed application. The service rates of the hardware associated with the application impose limits on the throughput of the requests. Once the hardware resources become saturated, the distributed system has reached its maximum feasible throughput (based on a physical viewpoint only).

A scalable architecture of a distributed system should have the ability to support workloads that change in both intensity and characteristics, yet be able to fully reflect the performance gains offered by new technologies. In order to achieve *software performance scalability* we must recognize potential software bottlenecks under various workload scenarios and plan for the changes needed to avoid them as the system evolves. In this paper we describe an efficient approach to recognize potential software bottlenecks over a wide range of workload scenarios and show how this information can be used early in the development process.

First we consider how our approach relates to the Software Performance Engineering approach described in [3] and the Timebench toolset [4, 5]. In [3] execution graphs are created for the dominant request types and are used to relate the requests to their resource demands. The graphs are comprised of the components used by the requests and the branches and loops that connect them. The components are blocks of program statements and procedures that perform functions for the system. We borrow from this approach but deal with higher level software components. As in [3], Timebench views software components as blocks of program statements and associates them with operating system processes. Timebench is used to consider how to distribute the processes across a network.