Chapter 1.27
Approaches to Building High Performance Web Applications:
A Practical Look at Availability, Reliability, and Performance

Brian Goodman
IBM Corporation, USA

Maheshwar Inampudi
IBM Corporation, USA

James Doran
IBM Corporation, USA

ABSTRACT

In this chapter, we introduce five practices to help build scalable, resilient Web applications. In 2004, IBM launched its expertise location system, bringing together two legacy systems and transforming the employee's ability to find and connect with their extensive network. This chapter reviews five of the many issues that challenge enterprise Web applications: resource contention, managing transactions, application resiliency, geographic diversity, and exception perception management. Using the IBM expertise location system as context, we will present five key methods that mitigate these risks, achieving high availability and high performance goals.

INTRODUCTION

In this chapter, we introduce five practices for building scalable, resilient Web applications. First, we briefly review the context in which IBM launched its internal expertise location system in 2004. We then introduce the challenges we faced in implementing the business requirements and present five key methods that mitigated risks to achieving our high availability and performance goals.

Specifically, we will look at:

- caching strategies for high availability Web applications: beyond storing copies of HTML (Challenger, Dantzig, & Iyengar, 1998; Iyengar & Challenger, 1997);
- asynchronous task processing within Web applications: removing non-essential linear logic from high-volume transactions (Grand, 2002);
- building self-reliant autonomous behavior: encapsulating through services achieving tight integration with *true* loose coupling (Birman, van Renesse, & Vogels, 2004);
- client-side Model View Control (MVC): moving MVC to the browser supercharging the response times and getting a "wow" user experience (Murry, 2005; Sun Microsystems, 2002);
- graceful degradation: keeping users thinking and feeling "fast," "reliable," and "always on" (Florins & Vanderdonckt, 2004).

Caching Strategies

Caching strategies are a core part of high-performing Web experiences. In many cases (Amiri, Park, & Tewari, 2002; Candan, Li, Luo, Hsiung, & Agrawal, 2001; Liebmann & Dustdar, 2004; Rodriguez, Spanner, & Biersack, 2001), the assumption is that caching occurs at the edge of the network, the closest point to the consumer and the furthest from the data or application. Another somewhat overlooked approach is object caching.

Davison (2001) provides a wonderful primer on Web caching that illustrates the principles of caching and highlights some of the issues that may arise due to its use. Edge caching, or Web caching, is focused on storing and managing Web pages (static or dynamic) to help speed up transaction times. As the complexity of Web architecture evolves, applications have become more distributed. Edge solutions exemplify this, placing caches of content, sometimes fragments of executable code, in multiple geographical locations.

In recent years, object caching has enjoyed a revival and is now seen as a more desirable component of Web application architecture. Caching objects and sharing them across an infrastructure is a compelling capability. Often, the cost associated with building an object is considered quite high. The difference with an object cache mechanism is that it often resides at the Web server (Jadav & Gupta, 1997), the Web application, or as a middleware between the data and the Web application logic. Once an object is built, it can be cached, distributed, and managed for future reuse; it delivers performance at the application layer, whereas edge caching offers performance benefits to the delivery of data.

The strategies and issues found in edge caching are very similar to those encountered in object caching, and understanding their roles can offer a more complete view of a caching strategy. If it is possible to reuse an object (whether it is HTML or an object created from multiple data sources), an opportunity exists to increase performance. Caching takes advantage of predicting what resource might be required in the near future and storing a copy for later use. Later in the chapter, we explore approaches to managing custom object caches and situations where they can provide impressive benefits.

Asynchronous Processing

Asynchronous processing is often thought of as multithreaded computing or parallel computing, both of which are far more technical than intended. A common trap in transaction processing is the preconceived notion that all processing has to be linear. There are opportunities to move non-critical or batch-oriented logic out of the critical path of handling a transaction. However, this often introduces a level of complexity that surrounds the execution of tasks that are not time-critical.