Chapter 8

Architectural Practices for Improving Fault Tolerance in a Service-Driven Environment

Raja Ramanathan
Independent Researcher, USA

ABSTRACT

Enterprises that implement Service-driven applications face challenges relating to unprecedented scale, high availability, and fault-tolerance. There is exponential growth with respect to request volume in Service-driven systems, requiring the ability to provide multipoint access to shared services and data while preserving a single system image. Maintaining fault-tolerance in business services is a significant challenge due to their compositional nature, which instills dependencies among the services in the composition. This causes the dependability of the business services to be based on the reliability of the individual services in the composition. This chapter explores the architectural approaches such as service redundancy and design diversity, scaling, clustering, distributed data caching, in-memory data grid, and asynchronous messaging, for improving the dependability of services. It also explores the data scaling bottleneck in data centralization paradigms and illustrates how that presents significant scalability and fault-tolerance challenges in service-driven environments. Prevalent strategies to handle failure recovery such as backward and forward recovery mechanisms as well as the built-in mechanisms in WS-BPEL for exception handling and transactional compensation are discussed.

INTRODUCTION

Enterprise architects use service-driven architectural approaches to solve real-world computing and integration problems by leveraging Service Oriented Architecture (SOA) and REST (Representational State Transfer) Service architecture and their concepts of service abstraction to create services that are interoperable, autonomous, stateless, and language agnostic. To enable interoperability, services are designed to be loosely coupled, composable, and standards compliant. Service-driven approaches to architecture contrast with tightly coupled object-centric architectures where component interfaces are distributed to clients to enable them to talk to providers using a common protocol.

DOI: 10.4018/978-1-4666-4193-8.ch008
Enterprise systems built using service-driven approaches face a host of challenges relating to unprecedented scale. There is an exponential growth with respect to request volume in service-driven systems. Services are reusable entities that are shared among various applications in the enterprise, causing large volumes of service consumer requests and service provider responses to be generated. Furthermore, as business processes become increasingly automated, they contribute to enormous increase in processing and data load. The challenge for large-scale and data-intensive systems is to provide multipoint access to shared services and data while preserving a single system image. It is extremely important to architect for application scalability in order to handle the enormous load requirements of the service-driven environment. **Scalability** is the property of a system, application, or service that indicates how well it can handle increasing amounts of load in response to increasing processing demand. Enterprise architects are concerned with designing systems and applications that can scale dynamically to handle varying compute and data loads to satisfy the dynamic requirements of the business.

Service-driven architectural approaches introduce a set of new challenges to the continuous availability of services and systems. Service availability requires the elimination of all single points of failure within a given service and its dependents, while System (server/node) availability requires the elimination of the tight coupling between the System (server/node) and the Services that are hosted on it. **Fault-tolerance** is the indicator of how well a system or application can tolerate the failure of one or more hardware or software components and still continue to operate in an acceptable manner. Fault-tolerance ensures overall system or application availability, albeit at decreased throughput. Fault-tolerance is related to the **high availability** characteristics of the system. Highly available systems provide useful resources over a period of time and guarantee functional continuity within that time window. Enterprise architects place great emphasis on the operational reliability of systems and applications. In other words, they attempt to build highly available systems and applications that continue to operate in the event of hardware or software failures.

Service-driven approaches enable to distribute applications or services to multiple locations within an enterprise and off-premises. A Service-driven application is generally deployed in a server farm (group of servers) in a load-balanced manner. One of the goals of the Service-driven approach is to provision the application environment to handle large processing and data volumes by distributing the service entities for concurrent processing. By design, SOA and REST are intended to provide scalability; however, there are many challenges that need to be addressed before one can achieve true scalability. The challenges relate to the manner in which the application is coded and how the application performs data access and storage.

Service-driven architectural approaches decompose an application into coarse grained services and instances of each service are deployed on multiple servers. Each service is expected to be stateless, in other words, they do not retain any data with them across multiple calls. This data independence enables any instance of the service running on any server to service consumer requests, thus facilitating service scalability. However, the real world is far from the ideal situation, and services have to deal with data. Access and persistence of both session and application data causes the scalability bottleneck in service-driven environments. Service-driven applications compose complex business services by orchestration and choreography (which are service composition techniques) of finer grained application services. This causes data shuttling among services, which is affected by network latency. The latency effect is further exacerbated when the target services are distributed among servers that are spread out across the Wide Area Network (WAN), causing