Service-Oriented Architecture: Performance Issues and Approaches

M.Swientek[1, 2, 3], U.Bleimann[1] and P.S.Dowland[2]

[1]University of Applied Sciences Darmstadt, Germany
[2]Centre for Information Security and Network Research, University of Plymouth, United Kingdom
[3]sd&m, Software Design & Management AG, Berliner Str. 76, 63065 Offenbach, Germany
e-mail: martin@swientek.org

Abstract

The introduction of a Service-Oriented Architecture (SOA) can affect performance in a negative way. This exacerbates the application of SOA to systems for bulk data processing. This paper describes specific aspects of Service-Oriented Architectures that impact performance particularly. It discusses several approaches to these issues that are currently established and motivates the need for a framework to implement an SOA for bulk data processing systems.

Keywords

SOA, Service-Oriented Architecture, batch processing, performance

1. Introduction

Service-Oriented Architecture (SOA) is becoming a popular approach to integrate heterogeneous applications into an application landscape. Apart from functional requirements, an IT system has to meet the non-functional requirements of the functional and technical operations. Implementing an SOA has certain impacts on these non-functional requirements that ought to be considered beforehand.

Performance is an important non-functional requirement of an IT system and is vital to the acceptance and the operability of the system. Since performance tests are usually not performed until the system is already in place, performance issues are often revealed at a late stage in the development process. In order to improve the performance of the system extensive changes to the architecture are needed which ultimately leads to significant project risks. As the introduction of SOA can deteriorate the performance it is crucial to be aware of the performance drawbacks and how to address them properly at the stage of the system design.

Bulk data processing in particular demands a high-performance implementation. Current approaches to implement an SOA using web service technologies and infrastructures do not match very well with a batch-processing model since they are focused on a request-response communication scheme.

This paper describes the performance issues specific to SOA and discusses current approaches to address them. It motivates the need of a framework to integrate a batch processing system in a service-oriented application landscape. The paper is organized as follows: The next section introduces the concept of Service-Oriented Architecture, describes common properties of batch processing systems and the understanding of performance used in this paper. Section 3 describes the aspects of an SOA that have an impact on performance. The next Section discusses current approaches to the performance issues identified in the preceding section. Section 4 motivates the need of a framework for bulk data processing. This paper concludes with a summarization of the presented performance issues and approaches.

1.1. Service-Oriented Architecture

Service-Oriented Architecture (SOA) is an architectural pattern to build application landscapes from single business components. These business components are loosely coupled by providing their functionality in form of services. A service represents an abstract business view of the functionality and hides all implementation details of the component providing the service. The definition of a service acts as a contract between the service provider and the service consumer. Services are called using an unified mechanism which provides a plattform independent connection of the business components while hiding all the technical details of the communication. The calling mechanism also includes the discovery of the appropriate service (Richter et al., 2005).

By separating the technical from the business aspects, SOA aims for a higher level of flexibility of enterprise applications.

Building an SOA involves concrete technical decisions how to implement its concepts. This includes how to implement services, how to discover the appropriate service and how to interconnect them. This paper focusses on these technical decisions that need to be made in order to implement an SOA in a performant way.

1.2. Batch Processing Systems

A batch processing system is an application that processes bulk data without user interaction. Input and output data is usually organised in records using a file- or database-based interface. In case of a file-based interface, the application reads a record from the input file, processes it and writes the record to the output file.

A batch processing system exhibits the following key characteristics:

- **Bulk processing of data**
  A Batch processing system processes several gigabytes of data in a single run. Multiple systems are running in parallel controlled by a job scheduler to speed up processing.

- **No user interaction**
  There is no user interaction needed for the processing of data. It is impossible due to the amount of data being processed.

- **File- or database-based interfaces**
  Input data is read from the file system or a database. Output data is also written to files on the file system or a database. Files are transferred to the consuming systems through FTP by specific jobs.

- **Operation within a limited timeframe**
  A batch processing system often has to deliver its results in a limited timeframe due to service level agreements (SLA) with consuming systems.

- **Offline handling of errors**
  Erroneous records are stored to a specific persistent memory (file or database) during operation and are processed afterwards.

Typical applications that are implemented as batch processing systems are billing systems for telecommunication companies used for mediating, rating and billing of call events.

1.3. Performance

Performance is a quality attribute of a software system and is crucial to the acceptance of a developed system both by users and IT operations.

The performance of a system can be described by multiple metrics. The following metrics are relevant to the understanding of this paper:

- **Response Time**
  Time it takes for the service consumer to receive a response from the service provider

- **Throughput**
  Number of requests a service provider is able to process in fixed timeframe

- **Latency**
  Time it takes that a services request is received by the servicer provider and vice versa

2. SOA Performance Hotspots

This section describes the different aspects of a Service-Oriented Architecture where performance issues typically occur.

A system implemented according to the principles of SOA is a distributed system. Services are hosted on different locations belonging to different departments and even organizations. Hence, the performance drawbacks of a distributed system

generally also apply to SOA. This includes the marshalling of the data that needs to be sent to the service provider by the service consumer, sending the data over the network and the unmarshalling of data by the service provider.

Apart from these general issues of a distributed system certain properties of an SOA deteriorate the performance even more.

2.1. Integration of Heterogeneous Technologies

A main goal of introducing an SOA is to integrate applications implemented with heterogeneous technologies. This is achieved by using specific middleware and intermediate protocols for the communication. These protocols are typically based on XML, like SOAP (SOAP Specification, 2007). XML, as a very verbose language, adds a lot of meta-data to the actual payload of a message. The resulting request is about 10 to 20 times larger than the equivalent binary representation (O'Brian et al., 2007), which leads to a significant higher transmission time of the message. Processing these messages is also time-consuming, as they need to get parsed by a XML parser before the actual processing can occur.

The usage of a middleware like an Enterprise Service Bus (ESB) adds further performance costs. An ESB usually processes the messages during transferring. Among other things, this includes the mapping between different protocols used by service providers and service consumers, checking the correctness of the request format, adding message-level security and routing the request to the appropriate service provider (See, for example, Josuttis, 2007 or Krafzig et al., 2005).

2.2. Loose Coupling

Another aspect of SOA that has an impact on performance is the utilisation of loose coupling. The aim of loose coupling is to increase the flexibility and maintainability of the application landscape by reducing the dependency of its components on each other. This denotes that service consumers shouldn't make any assumptions about the implementation of the services they use and vice versa. Services become interchangeable as long they implement the interface the client expects.

Engels et al. consider two components A and B loosely coupled when the following constraints are satisfied (Engels et al., 2008):

- **Knowledge**
  Component A knows only as much as it is needed to use the operations offered by component B in a proper way. This includes the syntax and semantic of the interfaces and the structure of the transferred data.

- **Dependence on availability**
  Component A provides the implemented service even when component B is not available or the connection to component B is not available.

- **Trust**
  Component B does not rely on component A to comply with pre-conditions. Component A does not rely on component B to comply with post-conditions.

Coupling between services occurs on different levels. Krafzig et al. describe the following levels of coupling that are leveraged in an SOA (Krafzig et al., 2005).

Level	Tight Coupling	Loose Coupling
Physical coupling	Direct physical link required	Physical intermediary
Communication style	Synchronous	Asynchronous
Type system	Strong type system	Weak type system
Interaction pattern	OO-style navigation of complex object trees	Data-centric, self-contained messages
Control of process logic	Central control of processing logic	Distributed logical components
Service discovery and binding	Statically bound services	Dynamically bound services
Platform dependencies	Strong OS and programming language dependencies	OS and programming languages independent

Table 1: Levels of coupling (Krafzig et al., 2005)

The gains in flexibility and maintainability of loose coupling are amongst others opposed by performance costs.

Service consumers and service provider are not bound to each other statically. Thus, the service consumer needs to determine the correct end point of the service provider during runtime. This can be done by looking up the correct service provider in a service repository either by the service consumer itself before making the call or by routing the message inside the ESB.

Apart from very few basic data types, Service consumers and service providers do not share the same data model. It is therefore necessary to map data between the data model used by the service consumer and the data model used by the service provider.

3. Current Approaches

This section describes current approaches to the performance issues introduced in the previous section.

3.1. Hardware

The obvious solution to improve the processing time of a service is the utilization of faster hardware and more bandwidth. SOA performance issues are often neglected by suggesting that faster hardware or more bandwidth will solve this problem. However,

it is often not feasible to add faster hardware in a late stage of the project because it involves more costs than initially planned.

3.2. Compression

The usage of XML as an intermediate protocol for service calls has a negative impact on their transmission times over the network. The transmission time of service calls and responses can be decreased by compression. Simply compressing service calls and responses with gzip can do this. The World Wide Web Consortium (W3C) proposes a binary presentation of XML documents called binary XML (EXI Working Group, 2007) to achieve a more efficient transportation of XML over networks.

It must be pointed out that the utilisation of compression adds the additional costs of compressing and decompressing to the overall processing time of the service call.

3.3. Service Granularity

To reduce the communication overhead or the processing time of a service, the service granularity should be reconsidered.

Coarse-grained services reduce the communication overhead by achieving more with a single service call and should be the favoured service design principle (Hess, 2006). However, the processing time of a coarse grained service can pose a problem to a service consumer that only needs a fracture of the data provided by the service. To reduce the processing time it could be considered in this case to add a finer grained service that provides only the needed data (Josuttis, 2007).

It should be noted that merging multiple services to form a more coarse grained service or splitting a coarse grained service into multiple services to solve performance problems specific to a single service consumer reduces the reusability of the services for other service consumers (Josuttis, 2007).

3.4. Degree of Loose Coupling

The improvements in flexibility and maintainability gained by loose coupling are opposed by drawbacks on performance. Thus, it is crucial to find the appropriate degree of loose coupling.

Hess et al. introduce the concept of distance to determine an appropriate degree of coupling between components. The distance of components is comprised of the functional and technical distance. Components are functional distant if they share few functional similarities. Components are technical distant if they are of a different category. Categories classify different types of components like inventory components, process components, function components and interaction components.

Distant components trust each other in regard to the compliance of services levels to a lesser extent than near components do. The same applies to their common

knowledge. Distant components share a lesser extent of knowledge of each other. Therefore, Hess et al. argue that distant components should be coupled more loosely than close components (Hess et al., 2006).

The degree of loose coupling between components that have been identified to be performance bottlenecks should be reconsidered to find the appropriate trade-off between flexibility and performance. It can be acceptable in that case to decrease the flexibility in favour of a better performance.

4. Applying SOA to Batch Processing Systems

How to apply the concepts of Service-Oriented Architecture to batch processing systems considering the arguments presented in section 2? A naive approach would be the utilisation of web service technologies for these kinds of systems as well. However, because of the performance issues mentioned in this paper, this option would not scale for bulk processing of data with a batch-processing model.

Wichaiwong et al. for example propose an approach to transfer bulk data between web services per FTP. The SOAP messages transferred between the web services would only contain the necessary details how to download the corresponding data from an FTP server since this protocol is optimized for transferring huge files (Wichaiwong et al., 2007). This approach solves the technical aspect of efficiently transferring the input and output data but does not pose any solutions how to implement lose coupling and how to integrate heterogeneous technologies, the fundamental means of an SOA to improve the flexibility of an application landscape.

In order to integrate a batch processing system into a service-oriented application landscape several design decisions need to get addressed:

- How to implement lose coupling?
- What is the appropriate degree of loose coupling?
- What is the right service granularity?
- Which middleware technologies can be utilised for the integration of heterogeneous technologies?
- Who is responsible for data transformation?
- What data formats should be used?

Given that there are no obvious answers to these questions, there is a certain need of a framework that supports the service-oriented integration of batch processing systems by offering proven solutions for these issues. The design of such a framework for the integration of batch processing systems in a service-oriented application landscape will be carried out in a PhD Thesis.

5. Conclusion

The introduction of an SOA generally has a negative impact on performance. Among general performance drawbacks an SOA shares with other distributed technologies, two main concepts of an SOA that deteriorate performance even more are described

in this paper. The communication overhead introduced by using intermediate protocols and specific middleware like an ESB to integrate heterogeneous technologies and the utilisation of loose coupling in order to increase the flexibility and maintainability of the application landscape.

This paper discusses several approaches to improve performance in an SOA that are currently established.

The obvious approach is to utilize faster hardware and more network bandwidth. The compression of messages poses an option for reducing transmission times. Other approaches suggest reconsidering the service or architecture design. To decrease the communication overhead immanent in an SOA services should be coarse grained. Is the processing time of a coarse grained service causing problems for a specific service consumer, a finer grained service should be added.

To apply the proper approach to performance issues it is vital to know the bottleneck of the system. Unfortunately, the measuring of system performance and the investigation of bottlenecks can be done only at a late stage in the development phase. SOA Performance models are trying to anticipate the performance behaviour during the design phase but they are currently still under research.

Improving the performance of a system always impacts other quality attributes. Adjusting the degree of loose coupling affects the flexibility of the system. Merging services to more coarse grained services or splitting coarse grained services into finer grained services to solve performance issues of specific service consumers impacts the reusability of the services. Thus, it is vital to find the appropriate trade-off between performance and other quality attributes of the system like flexibility, maintainability and reusability.

Batch processing systems in particular demand a high-performance implementation. In order to integrate these kinds of systems in a service-oriented application landscape several design decisions need to get addressed. For example, what is the appropriate degree of loose coupling and the right service granularity to achieve the required performance? Given that there are no obvious answers to these questions, there is a certain need for a framework for service-oriented processing of bulk data.

The design of such a framework will be the subject of a PhD Thesis, which will be carried out at the University of Plymouth in conjunction with the University of Applied Sciences, Darmstadt.

6. References

Engels, G., Hess, A., Humm, B., Juwig, O., Lohmann, M., Richter, J.-P., Voß, M. and Willkomm, J. (2008), Quasar Enterprise, dpunkt.verlag, ISBN: 978-3-89864-506-5.

EXI Working Group (2007), http://www.w3.org/XML/EXI. (Accessed January 2008)

Hess, A., Humm B. and Voß, M. (2006), “Regeln für serviceorientierte Architekturen von hoher Qualität“, *Informatik Spektrum*, Vol. 29, No. 6, Springer Verlag, pp. 395-411.

Josuttis, N. (2007), *SOA in Practice*, O'Reilly, ISBN: 0596529554.

Krafzig, D., Banke, K. and Slama, D. (2005), *Enterprise SOA*, Prentice Hall, ISBN: 0131465759.

O'Brian, L., Merson, P. and Bass L. (2007), "Quality Attributes for Service-Oriented Architectures", *Proceedings of the international Workshop on Systems Development in SOA Environments*, International Conference on Software Engineering, IEEE Computer Society, Washington, DC.

Richter, J.-P., Haller H. and Schrey, P. (2005), "Aktuelles Schlagwort Serviceorientierte Architektur", *Informatik Spektrum*, Vol. 28, No. 5, Springer Verlag, pp. 413-416.

SOAP Specification (2007), http://www.w3.org/TR/soap (Accessed January 2008).

Wichaiwong, T. and Jaruskulchai, C. (2007), "A Simple Approach to Optimize Web Services' Performance", *Proceedings of the Third international Conference on Next Generation Web Services Practices,* IEEE Computer Society, Washington, DC