Deployment Time Optimization of Distributed Applications

Kang-Won Lee[1] Kyung Dong Ryu Sangjeong Lee Jong-Deok Choi Dinesh Verma Manish Gupta
{kangwon, kryu, leesang, jdchoi, dverma, mgupta}@us.ibm.com
IBM T. J. Watson Research Center
Hawthorne, NY 10532, U.S.A.

Abstract

This paper proposes a novel optimization technology called *deployment time optimization* and presents several examples where the proposed technology can improve the performance of distributed applications. Using the configuration information collected from the target operation environment of an application, the proposed optimization approach tries to deploy only necessary components and best alternatives for the target environment. Since deployment time optimization works at module level, we may call this approach a macro level compilation. To enable deployment time optimization, we have designed a novel framework called *Blue Pencil*. The Blue Pencil framework consists of policy module for storing rules for optimization, programming environment to support generating glues between distributed modules, configuration discovery module that can collect information about a target environment, and code transformer that can customize an application for a target operation environment. We conclude this paper by presenting a simple example on how to select appropriate bindings between distributed objects as a proof of concept.

1. Introduction

The performance of a software system is a product of the design and implementation of the system. Naturally substantial efforts have been put into inventing development time optimization techniques. However, such techniques are fundamentally limited in certain sense, since at development time, crucial information about the target environment (where the software will be deployed) is not available. Since software developers cannot assume the specifics of the actual operational setting of the software, they typically employ generic APIs.

The same problem exists for distributed applications, which typically involve interoperation among multiple components that may be running on different platforms. In order to handle such issues the distributed computing community has developed layers of middleware abstractions. They include standardized network socket interfaces, remote procedure call and remote method invocation models, common object request broker architecture (CORBA), and more recently service-oriented architecture (SOA) based on Web Services. In particular, CORBA and Web Services aim to provide communication amongst distributed objects regardless of their platform and language differences.

Such middleware layers provide nice abstraction to software developers. However, naïve adoption of such technologies can adversely impact the performance of distributed applications. For example, programmers may develop an application on top of a heavy-weight Web service binding, based on SOAP/XML technology, even when a more light-weight invocation mechanism such as Java RMI is sufficient. In another example, database application written against the standard JDBC APIs may not exploit vendor-proprietary features offered by the database system in the actual operating environment. In other words, the performance of a distributed application may become crippled because the developers have opted to use generic middleware layers, which are convenient to program upon, without carefully trading off its performance implication. On the other hand, in many cases, this problem cannot be easily

[1] Correspondent author: Kang-Won Lee; E-mail: kangwon@us.ibm.com; Telephone: 914-784-7228; Postal Address: 19 Skyline Drive, Hawthorne, NY 10954, U.S.A.

avoided since little is known about the target operation environment during the development time of software.

Current optimization techniques focus on individual components in an isolated manner. For example, techniques exist that attempts to improve the performance of communication layers between distributed components. Such techniques include high-performance XML parsing, efficient implementation of protocol layers (e.g., SOAP). Other techniques include data caching at network layer (e.g., SOAP result caching, and DB query caching), availing of multiple ways to communicate between the components (e.g., Web Service Invocation Framework), and manually selecting the most appropriate binding during code development and/or inspection time.

However, these existing optimization techniques have drawbacks: First they do not utilize the information about the target environment where the software will be deployed and configured. Second, in many cases, those optimization techniques are applied independently to improve the performance of a specific function without considering the interactions between the components. Third, although such techniques may improve the performance of individual components, there are inherent overheads of going through multiple layers of abstraction. Finally, manual optimization techniques do not scale and cannot be applied in various domains.

In this paper, we propose a novel approach to improve the performance of software application, called the *deployment time optimization (DTO).* In a departure from conventional techniques used in development time and run-time optimization, DTO tries to optimize the compositions of distributed applications and the bindings between the modules based on the features and the configuration settings of the target environment where the application will be deployed and executed. This objective can be achieved by discovering the configuration of the target environment, composing only the necessary modules that are best suited among multiple alternatives, and configuring them in the most optimal way. We present the basic ideas of deployment time optimization using Web service deployment as a concrete example. We then present the Blue Pencil framework that has been developed to enable deployment time optimization in Web service environment.

The organization of the remainder of the paper is as follows: Section 2 presents the basic idea of deployment time optimization using some examples. Section 3 presents the Blue Pencil framework, which enables the proposed optimization technique. Section 4 presents the efficacy of Blue Pencil by presenting performance results using several real world Web services. Section 5 briefly discusses the related work in this area, and finally Section 6 concludes the paper.

2. Deployment Time Optimization

The basic idea of deployment time optimization (DTO) is to pick the right components to run a distributed application in a specific target environment, and compose them at a module level during the deployment of a distributed application so that it can perform optimally in the target environment. In this section, we first provide several examples where deployment time optimization can help the performance of distributed applications.

2.1 Binding Selection Example

Deployment time optimization can be used to select the right binding between multiple distributed modules. Consider a distributed application comprising a server and a client: a server is a module that implements a service, and a client is a module that calls the server to invoke the service. When implementing remote invocation, the software developer may choose to use SOAP/HTTP binding between the client and the service. By using SOAP, the client can talk to the server even when the server is behind a firewall since SOAP is encapsulated in HTTP. In addition, Web service enables language and

platform independence. However, if both client and server are located in the same local area network, it may be better to communicate using a lighter weight binding such as RMI.[2] Furthermore, it is possible that the client is deployed on the same machine as the server in certain usage scenario. In that case it is desirable to bypass the entire networking stack and allow them to communicate directly through local calls. However, during the development process, the programmer may not know about the operation environment and thus have no other option but to use the most generic SOAP binding.

The Blue Pencil framework provides a programming environment, where the programmer can develop a code as he or she would on top of the SOAP proxy interface. Once the program is written, the programmer would execute Blue Pencil stub generator to generate a *smart* stub, which can handle multiple bindings, instead of executing an RMI stub compiler or a SOAP proxy generator. In this way, the program can talk to the server via one of multiple binding options. When the smart stub-enabled code is deployed, it is handled by a Blue Pencil deployment module. The deployment module collects the configuration information and communicates the information to the smart stub. Then the decision logic in the smart stub selects the best binding for the operation environment. In this way, Blue Pencil enables deployment time optimization without requiring software developers to change the way they would develop applications.

2.2 Database Example

Consider a scenario where a large amount of data must be loaded to a database table. Without assuming a specific database management system, the programmer can write a Java program using generic database interfaces such as JDBC APIs. In this way, they can create a database application that can talk to various different DBMSs (DB2, Oracle, Sybase, etc.). Loading a huge amount of data using the default *load* operation provided by JDBC may take a long time, however, since it checks the integrity of data for every row that it writes to the table. While most vendors provide bulk data load APIs, those commands are different from vendor to vendor. For example, DB2 supports bulk loading by load command options, whereas Oracle provides a bulk load command called *sqlldr*, and SQL Server provides *bulk insert* statement. Such differences are difficult to account for during the code development process without knowing exactly which type of DBMS that the application will be run with.

Deployment time optimization can help in this situation. By collecting the information about backend database, such as vendor information, version, metadata information, etc., during the deployment of the database application, the smart deployment module of Blue Pencil can select the right database driver that will work best for the operation environment. Furthermore, it can transform the database application so that the application can fully utilize the vendor-specific features provided by the backend DBMS. Bulk loading operator is one example of vendor-specific features that can be enabled by the deployment time optimization technique.

Another example can be found when we try to select a JDBC driver that satisfies certain functional requirements. Not all applications exercise all the main features of JDBC drivers, such as connection pooling, data source abstraction, distributed transaction, row sets. Suppose now we have an application, which only uses connection pooling and data source abstraction, and there are two types of JDBC drivers: first one implementing all four main features but slightly slower in overall performance and the second one implementing only connection pooling and data source but is much faster. If both the drivers can work with the DBMS in the target environment, then it is better to hook the application up with the second driver.

---

[2] It has been reported that RMI is orders of magnitude faster than SOAP by many in the past. We have observed a modern SOAP layer, which is highly optimized, can work as efficiently as RMI under certain workloads and configurations. However, RMI is still more efficient than SOAP with most workloads.

We can draw a similar example when we have a choice between different JDBC driver types. In particular, a type 2 JDBC driver, which consists of both Java component and native code, may be more efficient but may not be portable. On the other hand, a type 4 JDBC driver, which consists of Java based layers, may be portable while its performance may be slower than that of a type 2 driver. Now depending on the applications need and system management policy, one can choose to deploy a type 2 driver for performance or deploy a type 4 driver for future portability.

2.3 Other Examples

Suppose we need to build an employee information directory application for a company. Now this application can be implemented by utilizing an LDAP directory. When deploying this application, let's assume we have a choice between two different LDAP implementations: one that provides a database-backed transaction semantic to LDAP operation (such as IBM LDAP server), and the other that provides simple directory functionality without transaction semantic. If the application does not require transaction semantic and does not exercise any features that are related to heavy duty LDAP then it may be better to connect to a light weight LDAP implementation.

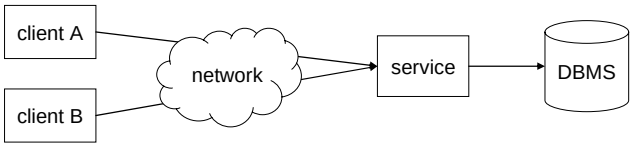
In another example, consider that we have an application that is based on a database system. Without knowing the requirement of the application, we may hook it up with a large scale enterprise database. However, the application may in fact need only a simple database support that can be provided by a simple in-memory database. If the deployment module can determine this characteristic, we can save a huge amount of space and overhead incurred by running a heavy duty database system.

In general, deployment time optimization can boost the performance of applications in various different scenarios by: inserting caching layer for query results, removing redundant protocol layer processing, reducing the burden of secure communication within a secure local domain, skipping legacy support features during the installation, etc. We discuss this proposed approach more in detail in the next section.

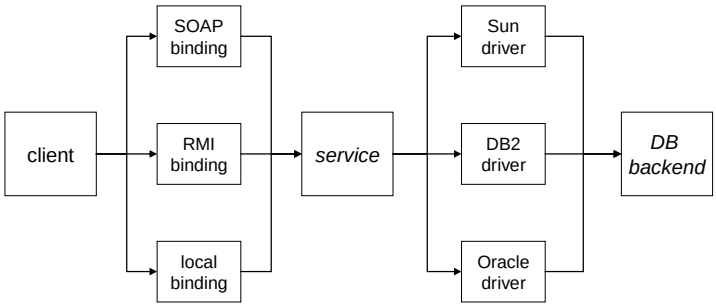
2.4 Deployment Time Optimization

The main idea of deployment time optimization is to remove redundant processing, replace slow components with faster alternatives, and add caching and other modules transparently to improve the performance, during the installation of the deployment process of an application using the configuration information about the target operation environment.

The approach taken by deployment time optimization can be considered a *macro level compilation* since it tries to improve the performance by composing the *right* components at module level. By components, we mean libraries, database drivers, network protocols, middleware components, and other files that are linked with the applications. Figure 1 shows a component-oriented view of a distributed application, which consists of clients, a service, and a backend database. In the figure, the service and the DB backend modules are assumed "abstract" and thus written in italics. By abstract, we mean their configuration details are not fixed from the viewpoint of the client developers. Therefore, there are multiple options to connect to these modules. Those options are represented with multiple paths and components on the paths between them.



(a) Configuration of a distributed application



(b) Component-oriented view of the same distributed application

Figure 1. Component-oriented view of a distributed application

Since DTO only works at component level, the granularity of each component can affect the effectiveness of the optimization. Also it will be helpful if the application itself is to some extent componentized. For example, if an application encapsulates the binding logic and contains only one type of binding, then it cannot readily benefit from the target environment supporting multiple types of binding. In such cases, we may need additional steps to transform the original application into a componentized form. This can be facilitated by code transformation.

Deployment time optimization can employ code transformation technique to rewrite some part of application to make it run more efficiently in the target environment. As the case discussed in the database example, if we have an application that repeatedly calls insertion operation to load a large amount of data into a table, DTO may detect such behavior and replace the related code fragments with an equivalent procedure using bulk loading statements. The code transformation technique should be used with caution, however, since we do not want to alter the semantic of the original application. We discuss the code transformation technique more in detail in Section 3.4.

To enable such capabilities, we have developed a DTO framework called Blue Pencil at IBM Research. In the next section, we present the Blue Pencil framework in more detail.

3. The Blue Pencil Framework

A dictionary definition of blue pencil is “a writing instrument used for editing,” and we feel this succinctly represents the objective of this framework. The Blue Pencil framework consists of five main components: (i) *policy repository* that stores the rules and the policies for configurations and transformations, (ii) *integrated development environment* that helps to generate code for later transformation, (iii) *code transformation module* that transforms applications so that it can be customized for the deployed environment, (iv) *component identification module* that collects the information about the application package, and (v) *application installation and configuration discovery modules* that orchestrate the deployment of application by identifying the configuration of the target environment.

Figure 2 presents the overall structure and operation of the Blue Pencil framework. In the following, we present the details of each module using the binding selection problem as a concrete example.

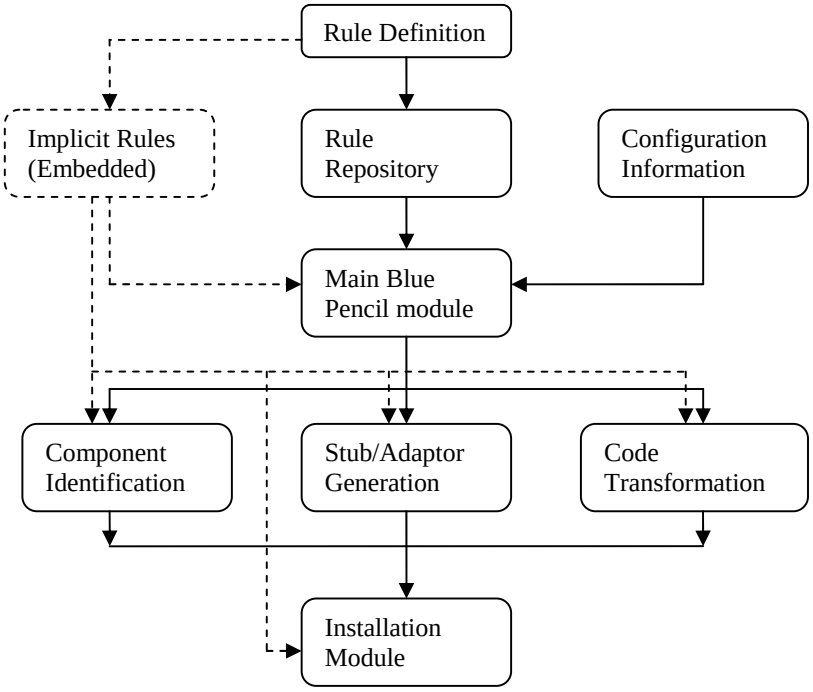


Figure 2 Overall concept of the Blue Pencil framework

3.1 Policy Repository

Deployment time optimization is a process based on rules. In the Blue Pencil framework, optimization rule and transformation rules can be expressed as policy rules. In general, a rule is defined in the (**if** *condition* **then** *action*) format, which is interpreted as "if *condition* is satisfied (or true) then apply the technique specified in the *action* field. For the purpose of deployment time optimization, the condition field may consist of target environment parameters and the *action* field may specify particular techniques to use. For example, a rule may state "if service location is local then use RMI binding (instead SOAP binding)," or "if the application is of type XYZ then do not perform transformation since it is not safe." Such rules can be defined explicitly and stored in a policy repository (e.g., policy editor storage in [1]). Alternatively, often the rules may be encoded and implemented in each of the component that needs to take such actions. In the latter, we say the rule is implicit. In Section 3.3 we present the binding selection rules that have been employed by Blue Pencil for Web service performance optimization. We note that the above if-then notation is for illustration only, and does not confine the kind of rule-based system that we can use; a rule can have any format (e.g. event-action format or subject-action-target format).

3.2 Programming Environment

The Blue Pencil framework provides an extended software development environment, such as Eclipse-based development environment, with an extension by specialized plug-ins to implement Blue Pencil features. Alternatively, such features can be provided as command line tools such as *rmic* (RMI compiler). Using the Blue Pencil programming environment, we can support developers to automatically generate stubs, containers, or interface layers so that their code can be easily restructured during deployment. For

example, to enable intelligent binding selection, we have developed a plug-in, called the *smart stub generator*. The stub generator automatically creates a client-side proxy called a *smart stub* based on the WSDL file from the service.[3] Essentially a smart stub hides the different invocation mechanisms between various types of bindings from the application developer.

A smart stub provides a simple unified interface for remote object creation and method invocation to the application programmer. Underneath it, however, it can contain several stubs for multiple bindings so that it can talk to a remote object via one of them. During the deployment of the application, the installation module calls the configuration discovery module to collect the configuration information and provides that information to the smart stub. Based on this information, the smart stub makes a decision as to which binding to create. Since it will instantiate only one of the bindings for all its communication, the smart stub does not incur any extra overhead when compared with native SOAP or RMI stub.

```
import RemoteObjectFoo;

// remote object creation
RemoteObjectFoo foo = new RemoteObjectFooProxy();

// method call
foo.bar();
```

Figure 3. Creation of a remote object using the smart stub interface

Using the smart stub interface, developers can write an application as if they were writing a standard RMI or SOAP application. Figure 3. Creation of a remote object illustrates an example of creating and accessing a remote object via the smart stub. In this example, RemoteObjectFooProxy is a proxy that has been auto-generated by the stub generator using the WSDL file of the RemoteObjectFoo. Creating a handle for that remote object is as simple as just calling the constructor of the proxy. Once the handle is created, then it can be used in the same manner as a local instance.

3.3 Installation, Component Identification, and Configuration Discovery

Application deployment can be facilitated by an installation module such as WebSphere deployer and Tivoli Solution Install. During this process, the installation module calls a Blue Pencil configuration discovery module. In the binding selection scenario, the role of a configuration discovery module is first to identify the types of bindings that the service support. This information can be found from the extended WSDL published by the service. It then determines the relative location of the service and the client – for example, they may be located in the same JVM, in the same physical machine, on the same subnet, or connected via wide area network. In addition, it also needs to determine whether there is a firewall between the client and the server. After collecting all the configuration information, it passes the information the smart stub, which will select an appropriate binding. This process is called configuration discovery and component identification, and it provides a critical piece of information for subsequent decision making process.

For example, if the client and server will be executed in the same class loader, they can communicate via EJB local interface, which is equivalent to making a native Java call. On the other hand, if they are going to share the same ORB instance, they can communicate via EJB remote interface, with local RMI, which is known to be more efficient than remote RMI. If they are on different machines, but there is no firewall in between then they can use EJB remote interface with remote RMI. Finally, if a firewall exists between the client and the service, then they must communicate through SOAP over HTTP.

[3] We can use an extension to WSDL, such as WSIF (Web Service Invocation Framework) extension, to allow specifying multiple bindings for a single service.

The configuration discovery module may employ various techniques to detect the relative location and the existence of a firewall. First, it can detect whether they share the same ORB or not by using `javax.rmi.CORBA.Util.isLocal` method. Whether a client and a service are on the same subnet can be determined from the subnet mask. The existence of a firewall can be determined by sending a few probe messages to the RMI port.[4]

3.4 Code Transformation

When a developer writes an application using the Blue Pencil development environment, he or she can create a smart stub or other interface layers for the application so that later stages of deployment time optimization can be facilitated. However, when we deploy an application that has been created outside the Blue Pencil framework, it may not be in a shape that can be readily restructured or reconfigured by the installation module. For example, if an application has only a SOAP binding, then the Blue Pencil installer cannot change it to other types of binding even if it knows that SOAP is not the best option.

We apply code transformation technologies in such cases to generate codes that are easier to reconfigure during the deployment of applications. In the binding selection example, the code transformer will generate a corresponding smart stub for the client module by transforming the existing stub code. Our code transformer is based on a set of compiler technologies such as forward and backward code slicing, code pattern matching, and dead code elimination. Using these program analysis techniques, it will identify the code block to be removed and insert new code block to transform the existing client stub into a smart stub. Transforming one code to another equivalent code is a very difficult problem in general. However, we can use this technique since in most cases the stub codes have fixed pattern and frequently generated by machines, which makes the program analysis step easy.

3.5 Operation of the Blue Pencil Framework – Binding Selection Scenario

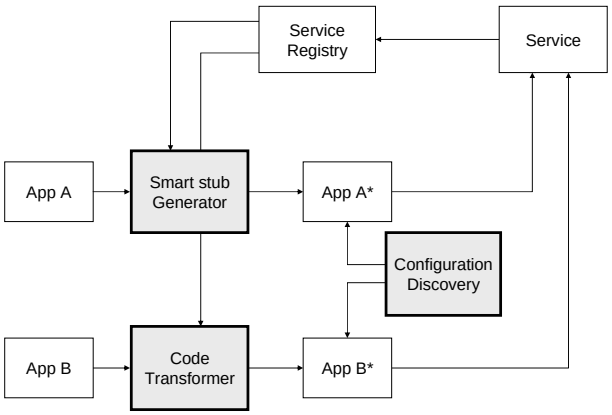


Figure 4. The Blue Pencil framework in action in the binding selection scenario

Figure 4 illustrates the main components of Blue Pencil (gray boxes) and their interaction in the binding selection scenario. In order to allow the client applications (App A, App B) to select its binding during deployment, the service must implement multiple bindings, e.g., SOAP, remote RMI, and local RMI

[4] Note that this decision logic shown is based on an assumption that an RMI binding is always better than a SOAP binding. However, optimized SOAP/XML processing techniques employed in modern J2EE application servers make SOAP binding as efficient as RMI binding for certain workload. We need to augment the decision logic for such application servers.

bindings. The service also publishes this information in a WSDL file into a service registry, such as UDDI. Since the vanilla WSDL does not allow associating multiple bindings to a single service, we need to use an extension to WSDL such as WSIF (Web service invocation framework) to do that. When developing a client, the software developer refers to the interface definition specified in the WSDL and design the client based on that interfaces. App A shows the case, when client development is done using a Blue Pencil programming environment, in which case a smart stub will be auto-generated for the application. As a result, we get an application with an intelligence to select the most appropriate binding in the target environment (App A*). When this application is deployed, the software installer invokes configuration discovery module, which will provide information so that the decision logic in the smart stub can select a binding to be used. App B shows the case when we deploy a client that has not been developed by the Blue Pencil programming environment. In this case, we can perform code transformation to replace the existing stub with a smart stub (App B*). Then it can interact with the configuration discovery module to determine the best binding to reach the service.

4. Performance Study

4.1 Test Configuration

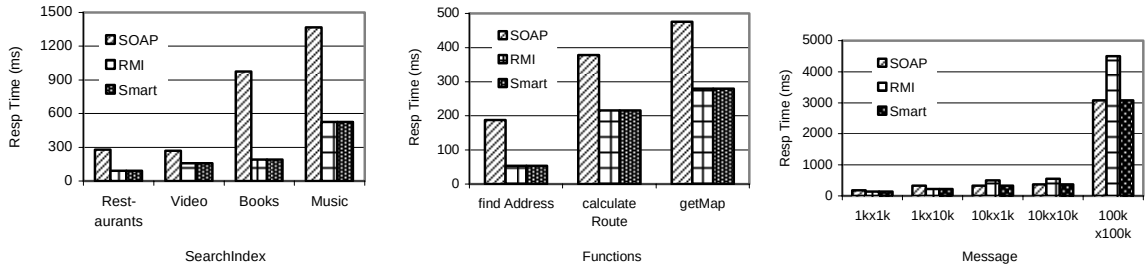
We have validated the effectiveness of the proposed deployment time optimization approach using several public several public Web Services, such as Amazon WS, Microsoft MapPoint, and Google WS. We have also used an IBM internal Web Service benchmark and Trade 6 benchmark to model more complicated interactions between a client and a service.

Amazon Web Service (AWS) [7] provides hooks to variety of Amazon search functionalities. For our experiment, we have used four most widely used search types; books, video, music, and restaurant search. Microsoft MapPoint Web Service [5] provides various geographic services related to getting maps and finding addresses. In particular, we have used the following three services; findAddress, calculateRoute, and getMap. Google WS provides Google's keyword-based Web Search functionality. In addition, we can select various options such as how many results we want to get from Google.

WSBench [8] is an internal benchmark to simulate the operations of financial transaction service. Modeling after a real-world bank transaction, it implements addition of money transfer information to a financial record. Trade 6 is a benchmark developed to measure the performance of Web application servers, and it models the stock trading operation. Since it provides different configurations to use SOAP and RMI binding, we use it to measure the performance of Web Services.

4.2 Benefit of Deployment Time Optimization

In this section, we present the performance benefit that we could achieve from the proposed deployment time optimization approach using some of the Web services presented in the previous section. In particular, we present the results with Amazon Web service, Microsoft MapPoint, and WSBench. For performance evaluation, we have created a testbed consisting of two machines, one works as a server, and the other simulates the behavior of many clients. For the server, we configured a WebSphere application server on a Pentium 4 machine with 2.0 GHz CPU, 1.5 GB main memory running the latest version of Windows XP. For the client, we used a Pentium 4 machine, with 3.0 GHz CPU, 3.0 GB main memory, also running Windows XP. To simulate the case when the server is busy, we created multiple threads of clients and sent parallel requests, and measured the response time. For all cases, the server CPU was almost fully exercised to simulate the case when the server is in full operation. Figure 5 shows the average response time results for Amazon WS, Microsoft MapPoint WS, and WSBench.



(a) Amazon – itemSearch (b) MapPoint (c) WSBench - handleAddShort

Figure 5. Average Response Times

For all three cases, we present the response time of SOAP, RMI, and smart stub bindings. In Amazon's item search service and MapPoint Web services, RMI binding provides much smaller response time than SOAP. In this case, we find that the smart stub correctly selects the more efficient bindings for Amazon and MapPoint. For WSBench Web Service, however, some times SOAP performs better than RMI (10k-in-1k-out, 10k-in-10k-out, 100k-in-100k-out). The reason why SOAP performs better is because WebSphere employs optimized SOAP processing and XML parsing techniques. In other application servers (such as the latest JBoss Application Server), SOAP is always a few orders of magnitude better than SOAP (from 50 times to up to 700 times faster). Another reason for RMI's suboptimal performance comes from the inefficient message encoding style of CORBA IIOP. We have developed an effective algorithm to estimate the performance of different bindings using only static information and implemented in the smart stub decision logic. As a result, we observe that smart stub can always select the correct binding for WSBench case too.

5. Related Work

Conventional optimization techniques focus on individual components in an isolated manner. For example, there are techniques that attempt to improve the performance of communication layers between distributed components. Such techniques include high-performance XML parsing (Screamer project), efficient implementation of SOAP protocols (Colombo project). Other techniques are data caching at network layer (e.g. SOAP result caching, and DB query caching), availing of multiple ways to communicate between the components (Web Service Invocation Framework or WSIF), and manual tuning during code inspection time.

However, existing optimization techniques have the following drawbacks. First they do not utilize the information available in the target environment where the software or service is deployed. Second, in many cases, those optimization techniques are applied independently with each other to improve the performance of a specific function or layer. Third, although such techniques may improve the performance of individual components, there are inherent overhead from multiple layers of abstraction. Finally, manual optimization techniques do not scale and cannot be applied in various domains.

Juric et al. have studied various approaches to tunnel RMI messages through firewalls, such as RMI over open ports, HTTP-to-port, HTTP-to-CGI, HTTP-to-servlet, and compared them with the SOAP-based approach in terms of functional and performance differences [1]. Although the RMI tunneling approaches provide clever mechanisms to reach services behind firewalls, such techniques are neither standardized nor recommended. Moreover, such tunneling-based approaches suffer from much poorer performance than that of native Java RMI or SOAP. Thus in our paper, we have focused the binding selection issues mainly on the RMI and SOAP.

Several works have looked at optimizing XML processing. Sundaresan et al. have proposed a compression technique that considers the structural part and the text of an XML document separately [3]. They have also developed a technique to quantify the characteristics of a document based on the schema and the document itself and determine the most appropriate compression algorithms for the document. Such compression techniques can reduce the document size significant so that they can be easily stored on small devices and easily transferred.

In [4], Mukhi et al. presented an extension to the SOA architecture so that the service provider and the requestor can negotiate and dynamically change the configuration based on the run-time characteristics of the application. Web Service Invocation Framework (WSIF) tries to enable dynamic binding selection by providing an XML schema extension and a set of Java libraries to process them. However, the intelligence to select the right binding for a certain run time environment is essentially a burden of application developers. In contrast, Blue Pencil tries to provide a programming environment where the task of the application programmer remains the same and actually selection occurs during the deployment time by the smart deployment module.

Reference

[1] Policy Management for Autonomic Computing, IBM alphaWorks, on-line document available at http://www.alphaworks.ibm.com/tech/pmac, March 2005.

[2] Matjaz B. Juric, Bostjan Kezmah, Marjan Hericko, Ivan Rozman, Ivan Vezocnik, “Java RMI, RMI Tunneling and Web Services Comparison and Performance Analysis,” ACM SIGPLAN Notices, vol. 39, no. 5, pp. 58 – 65, May 2004.

[3] Neel Sundaresan, Reshad Moussa, “Algorithms and Programming Models for Efficient Representation of XML for Internet Applications,” in Proc of International World Wide Web Conference (WWW10), May 2001.

[4] Nirmal K Mukhi, Ravi Konuru, Francisco Curbera, “Cooperative Middleware Specialization for Service Oriented Architectures,” in Proc of International World Wide Web Conference (WWW2004), May 2004

[5] Microsoft MapPoint, http://www.microsoft.com/mappoint/products/webservice/default.mspx

[6] MapPoint demo application, Java Driving Directions, http://demo.mappoint.net/

[7] Amazon Web Services, http://www.amazon.com/gp/aws/landing.html

[8] WebSphere Performance Benchmark, https://w3.opensource.ibm.com/projects/wpb

[9] Object Management Group, Inc., Common Object Request Broker Architecture: Core Specification, Version 3.0.3, on-line document at http://www.omg.org/cgi-bin/doc?formal/04-03-12, March 2004.

[10] Object Management Group, Inc., Java to IDL Language Mapping Specification, Version 1.2, on-line document at http://www.omg.org/cgi-bin/doc?formal/02-08-06, August 2002.