Batch Invocation of Web Services in BPEL Process

Liang Bao, Ping Chen, Xiang Zhang, Sheng Chen, Shengming Hu, and Yang Yang

Software Engineering Institute, Xidian University.
Xi'an, 710071, China
{baoliangbox,zximportant,kkchensheng,yyqinghuan}@gmail.com,
{chenping,shmhu}@sei.xidian.edu.cn

Abstract. This paper presents our approach for optimizing the execution of BPEL (Business Process Execution Language) process by leveraging the features of enterprise intranet and introduces *batBPEL*, a tool for batch invocations of web services in BPEL process. The approach focuses on the decrease of connections by forming batch invocation request of web services in BPEL process. Some empirical experiments and evaluations show and prove the efficiency of our method and related algorithms.

1 Introduction

Nowadays, Web services are emerging as a prevalent paradigm for implementing the SOA[1] concept by developing and deploying business processes within and across enterprises. Since the agility and flexibility of business processes become more and more important, the Web Services Business Process Execution Language[2](BPEL for short)is in the very act of becoming the industrial standard for modeling Web services based business processes.

As an increasing number of business processes are modeled using BPEL, it is critical to guarantee the execution performance of a business process. Many optimization methods have been introduced, such as IBM Symphony project for decentralized process execution [3] and our earlier work for data-race free optimization[4].

However, all of these methods mentioned above are process-centered and fail to leverage the special features(such as operating system, networking etc.) that exist within the environment of an enterprise or organization to gain better performance. One of these features is that most applications are running on a fast and reliable intranet network. This implies that the conclusion "the time-cost to establish a new connection is far more expensive than that of transmitting a large trunk of data" holds under this circumstance. In this paper, we design and implement a framework named *batBPEL*(batch BPEL) that leverage this straightforward but important conclusion to speed up the execution of a BPEL process. The approach focuses on the decrease of connections by forming batch invocation request of web services in BPEL process. Some actual experiments prove that this solution can increase the throughput and decrease the execution time of different processes significantly.

A. Bouguettaya, I. Krueger, and T. Margaria (Eds.): ICSOC 2008, LNCS 5364, pp. 511–516, 2008.

2 Overview

In this section, we introduce our overall design of *batBPEL*. The architecture illustrated in Fig.1 describes the basic structure and interactions of *batBPEL*, which comprises the following components:

Static Analyzer: this component applies analysis techniques to a given BPEL process and forms the batch groups.

Invocation Interception: its task is to intercept the invocation of Web services in BPEL process, and to look up the batch groups and notify the client side service agent with the invocation and batch information.

Client Side Service Agent(Client Agent): after receiving the information sent by invocation interception, the client side service agent forms a batch invocation request.

Server Side Service Agent(Server Agent): when the request is received, the server side service agent un-marshals the request, identifies, separates and dispatches the request into many web service invocations.

Invocation Result Cache: once receives the response, the client agent un-marshals and separates it. The invocation result required at this time is returned to the BPEL engine directly, and other results are put into the invocation result cache for later use.

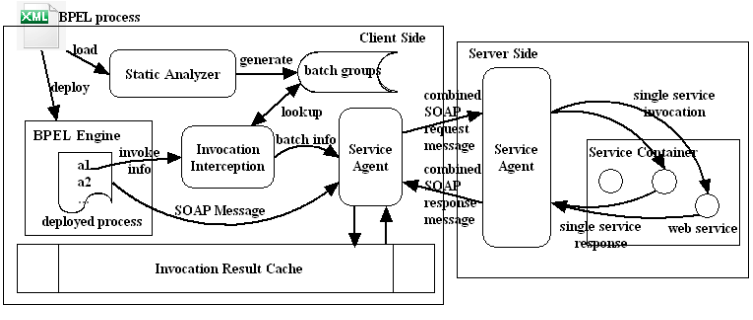

Fig. 1. The architecture of *batBPEL*

3 Static Analysis

Before a BPEL process is executed in the process engine, it is loaded into a static analyzer first. The analyzer consists of four modules, namely"BPEL Reader", "BCFG Generator", "ADG Generator" and "Batch Analyzer" respectively, which is shown in Fig.2. When a BPEL process is loaded, the BPEL Reader will read it and transform it into our memory structure of BPEL process(i.e. customized Java classes), which is implemented by JAXB(Java native XML APIs in JDK 6.0). The whole analysis procedure is totally sequential, the detailed discussion can be found in[5].

BCFG Generator: The key task of this component is to load the memory representation of BPEL process generated by BPEL reader and transform it into a corresponding BCFG(BPEL Control Flow Graph) representation.

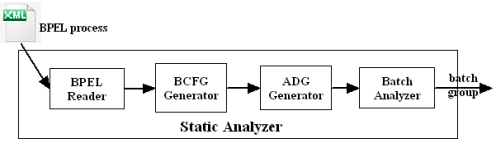

Fig. 2. Static analyzer

ADG Generator: Load the memory representation of BCFG and transform it into a corresponding ADG(Activity Dependence Graph) representation.

Batch Analyzer: Enforce static slicing algorithm[6] to the ADG formed above, generate a legal sequence of execution(a sample) and apply our batch algorithm to it[5].

4 Invocation Interception

When an *invoke* activity in a batch group is executed, *batBPEL* need to be notified with related invocation information. This is archived by introducing an invocation interception module in it. The implementation of such interception is based on a modified version of BPEL engine named ActiveBPEL [7], which uses AspectJ [8] to introduce invocation interception as cross-cutting concerns(inspired by [9]).

Since we are only interested in the Web service invocation, we have decided to intercept the process when it performs *invoke* activities. In order to do so we set pointcuts before and after the engine calls the execute method on these activities.

5 Service Agent

By definition, the main responsibilities of the service agents(both client and server agent) are two folds: (1) pack one or more invocation information up and form a single invocation request and (2) read and parse the packaged invocation request, dispatch the different invocations(for server agent) or save the different results in the invocation cache(for client agent).

5.1 Client Side Service Agent

Client side service agent(client agent for short) is located in the same host as the process engine is. The client agent is structured around four key components: (i) request interception, (ii) batch marshaller, (iii) batch un-marshaller and (vi) service invocation cache reader/writer.

The core function of request interception component is SOAP request monitoring and interception.In *batBPEL*, it is implemented by Apache Synapse, which is an open source XML and Web services management and integration broker that can form the basis of an SOA and Enterprise Service Bus (ESB). We use it here to perform SOAP message interception.

5.2 Server Side Service Agent

Server side service agent(server agent for short) is located in the same host as the service container is. Fig. 3 gives a detailed architecture of it. The batch un-marshaller and batch marshaller are both mediators implemented in Java class, just like their counterparts in client agent. When a batch invocation arrives, the batch un-marshaller and invocation dispatcher are added to separate and dispatch the batch invocation request to desired web services. The whole process is totally concurrent. Once all of these invocations are returned, the related results are collected and formed a single response by marshaller. It is then pushed to the intranet as a batch invocation result.

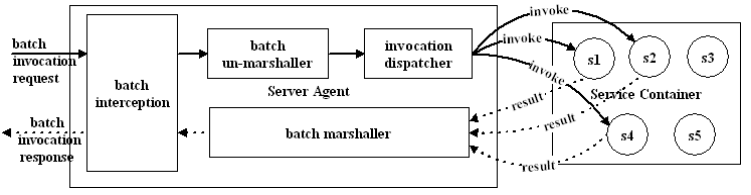

Fig. 3. Server agent

6 Experiment and Evaluation

In this section we report the comprehensive performance gain that brought by batch invocation from both the service and BPEL process level.

The typical setting for evaluation is a cluster of Intel Pentium based Windows machines(2.8G, 512MB RAM) connected by a 100 Mb/s LAN. Fig. 4 shows the comparison of the execution time duration on the service level between the batch method and the conventional invocation method. The duration presented in the figure marked *original* includes fifty invocations. Every five invocations are packed into one invocation when batch method is exploited, and accordingly, the client invokes the server for ten times. When the message size is less than 50 KBytes — the most case for message size, the batch method is about four times fast than the traditional method. The acceleration factor deteriorates slightly when the message size is bigger than 100 KBytes as data marshal and un-marshal take relatively longer time.

The duration for executing a service, which varies from many seconds to many hours or even days, and the impact of it that will be explored statically, is not taken into account in Fig. 4. It is worth nothing at this point that *batBPEL* that utilizes batch invocation can speed up the execution of BPEL process appreciably.

Table 1 displays the process level comparisons between these two invocation paradigms with the focus on invocations and execution duration. Among all the processes we have investigated, six of them are represented. These processes can be mainly categorized into three types: computation-intensive, service-invocation-intensive and the compound of them. The Office Automation(OA) and Draining System(DS) process belong to the first type; the Tool Integration(TI) and Travel Reserve(TR) fall into the second category; and the other two processes — Online Book Purchase(OBP) and Train Tickets(TT) pertain to the last type.

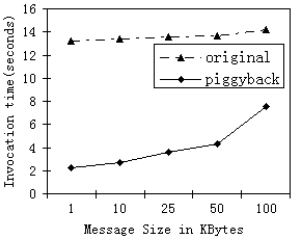

Fig. 4. Service level comparison

Table 1. Process level comparison

process	original invokes	batch invokes	original duration(s)	batch duration(s)
OA	79	63	231.23	207.5
DS	103	81	287.93	242.7
TI	183	144	436	315
TR	94	71	108.9	97.4
OBP	23	22	34.63	33.1
TT	52	39	60.83	45.2

As what can be observed in table 1, while the batch invocation opportunities for the first type is minor, it increases pronouncedly for the second type, which is what BPEL targets for. Take the TI process for example, when it is executed conventionally, which incurs one service invocation when an *invoke* activity [2] is interpreted, 183 invocations occurred. The batch invocation paradigm, however, requires only 144 invocations. The performance gain for this process is around 28 percent.

By analyzing all the data gathered between our users and *batBPEL*, we found that the performance gain, which is process dependent, ranges from 5 percent to 30 percent. Moreover, the complex and sophisticated process, such as banking and telecommunication systems that contain thousands of activities and have twisted control flow, enjoy more performance improvement.

7 Related Works

So far as we know, there are no related works on this topic. However, there are some works on execution optimization of BPEL process, such as IBM Symphony and our earlier work on *data-race free* optimization of BPEL process. In Symphony project, some techniques partition a composite web service written as a single BPEL program into an equivalent set of decentralized processes, with the goal of minimizing communication costs and maximizing the throughput of multiple concurrent instances of the input program. Our earlier approach focuses on the adapted static optimization methods of BPEL process.

Besides, there has been considerable research effort paid to BPEL. WofBPEL [10] translates BPEL processes to Petri nets and imposes existing Petri nets analysis

techniques to perform static analysis on processes. [11] modifies the CWB to support BPE-calculus by means of PAC to ensure that each *link* has one source and target activity exactly, and to guarantee that the process is free of deadlocks. Mads [12] describes some region-based memory techniques for programs that perform dynamic memory allocation and de-allocation.

8 Conclusion and Future Works

In this paper, we advocate batch invocations of Web services in BPEL process by static analysis and dynamic invocation interceptions. Furthermore, we implement our *batBPEL* that support high performance BPEL execution in the enterprise intranet environment. Our further work includes improvement of our static analysis algorithm and a careful study of the system.

References

1. Papazoglou, M.: Service-oriented computing: Concepts, characteristics and directions. In: 4th International Conference on Web Information Systems Engineering (WISE), New York, pp. 3–12. IEEE Press, Los Alamitos (2003)
2. Jordan, D.: Web services business process execution language version 2.0. OASIS Specification (2007)
3. Nanda, M.G., Karnik, N.: Synchronization analysis for decentralizing composite web services. ACM, New York (2003)
4. Sheng Chen, L.B., Chen, P.: Optbpel: A tool for performance optimization of bpel process. LNCS. Springer, Heidelberg (2008)
5. Bao, L., Zhang, X.: Batch invocation of web services in bpel process(detatiled version) (2008), `http://www.soacn.org`
6. Krinke, J.: Static slicing of threaded programs. In: ACM SIGPLAN/SIGSOFT, pp. 35–42 (1998)
7. Endpoints, A.: Activebpel engine architecture(version 4.1) (2008), `http://www.activebpel.org/docs/architecture.html`
8. Kiczales, G., Griswold, E.H.,, W.G.: An overview of aspectj. In: Knudsen, J.L. (ed.) ECOOP 2001. LNCS, vol. 2072, pp. 327–353. Springer, Heidelberg (2001)
9. Courbis, C., Finkelstein, A.: Towards aspect weaving applications. In: International Conference on Software Engineering(ICSE), pp. 69–77. ACM Press, New York (2005)
10. Ouyang, C., Verbeek, E., van der Aalst, W.M.P., Breutel, S., Dumas, M., ter Hofstede, A.H.M.: Wofbpel: A tool for automated analysis of bpel processes. In: Benatallah, B., Casati, F., Traverso, P. (eds.) ICSOC 2005. LNCS, vol. 3826, pp. 484–489. Springer, Heidelberg (2005)
11. Koshkina, M., B.F.: Modelling and verifying web service orchestration by means of the concurrency workbench. TAV-WEB Proceedings/ACM SIGSOFT 29–5(5) (2004)
12. Mads Tofte, J.P.T.: Region-based memory management. Information and Computation 132, 109–197 (1997)