TELKOMNIKA, Vol. 11, No. 10, October 2013, pp. 6068 ~ 6073
ISSN: 2302-4046

An Architecture to Implement Event-Driven Web Monitoring Systems

Gao Ying*[1], Mu Lei[2], Hao Zhonghu[3], Zheng Weiyang[4]
[1]School of Computer Science and Engineering, South China University of Technology Guangzhou, 510006, China
[2]Shangdong Provincial Communications Planning and Design Institute Jinan, 250031, China
[3]School of Computer Science and Engineering, South China University of Technology Guangzhou, 510006, China
[4]Information Center R&D Department, China Southern Airlines Company Limited Guangzhou, 510406, China
*Corresponding author, e-mail: gaoying@scut.edu.cn, 13953196967@163.com, haozhonghude@163.com, zhengwy@csair.com

Abstract

Traditional monitoring systems are based on C/S mode because desktop software still has inherent advantages though desktop software is replaced by web applications rapidly in many fields. Meanwhile, traditional monitoring systems use relational database as data source, however, relational database lacks the ability to process influx of queries per second. In this paper, we will do an in-depth research to design an architecture to implement event-driven web monitoring systems.

***Keywords**: event-driven, Comet, bayeux, Esper*

1. Introduction

The most important requirement of monitoring system is processing events (or messages) in real-time or near real-time. Key considerations for monitoring applications are latency, throughput and complexity of the logic required [1].

Low latency-applications that react in real-time to conditions that occur (from a few milliseconds to a few seconds)

High throughput-applications that process large volumes of messages (between 1000 to 10000 messages per second)

Complex computations-applications that detect patterns among events, filter events, aggregate time or length windows of events, join event streams, trigger based on absence of events etc.

To implement a real-time or near real-time web monitoring systems, these three requirements must be met. Consequently, two problems appear:

1.1. How to Push New Data from Servers to Browsers Instantly?

It's easy for desktop software to meet the requirement of low latency because software has an inherent advantage in pushing instant messages to clients but difficult for web applications, because traditional web applications are restricted to Request/Response pattern. This limitation of HTTP protocol leads servers not be able to push data to clients. That's why most monitoring systems are based on C/S mode though web applications are taking place of desktop software rapidly in recent years.

1.2. How to Process Large Volumes of Messages and Perform Complex Condition Checking Efficiently?

Usually, applications use relational database as data source. However, relational database and SQL are designed for applications in which most data is fairly static and complex queries are less frequent. To retrieve data from a database, an application must issue a query. Always, using relational database in monitoring systems, you need to periodically poll for state changes. This poor design puts a lot of strain on both web server and database, costing you

Received March 24, 2013; Revised July 12, 2013; Accepted July 23, 2013

CPU cycles, bandwidth, and a decrease in performance. Database triggers can be used to fire in response to database update events but they tend to be slow and often cannot easily perform complex condition checking and implement logic to react [2]. So relational database does not apply to monitoring system in which influx of data need be processed in a short period.
This paper is going to give an introduction to how to solve these problems and then design an architecture to implement event-driven web monitoring systems.

2. Solutions and Primary Architecture

2.1. COMET

To solve problem n INTRODUCTYION, two methods are widely applied [3]:

• With the help of plug-ins (such as flash, applet or ActiveX) installed in browsers, servers can push new data instantly by sockets [4] [5].

• Client browsers send Ajax [6] [7] request to server at regular intervals to check whether the server has got new data [8].

However, these two methods have their own problems. The first method has compatibility issue because not all the browsers support the plug-ins. The best condition to realize the second method is the polling intervals are equal to the server's update intervals, otherwise, it not only can't meet the real-time requirements but also results in waste of resources.

Recently, a web application model named "Comet" in which a long-lived HTTP connection allows web servers to push data to browsers without plug-ins is more popular. Specific methods of implementing Comet fall into two major categories: streaming and long polling [9].

2.2. Event-Driven Architecture

In a world (such as monitoring system) filled with events and instant communication, traditional database is not appropriate. Consequently, event-driven architecture (EDA) was proposed [10]. In an event-driven architecture, an important thing happens inside or outside your business, which disseminates immediately to all interested parties. The interested parties evaluate the event, and optionally take action. The event-driven action may include the invocation of a service, the triggering of a business process. A classical EDA works like a database turned upside-down. Instead of storing the data and running queries, it allows applications to store queries and run the data through. With EDA, applications can trigger actions when event conditions occur among event streams, perform complex condition checking and implement logic to react quickly. So, EDA can be used to solve Problem in INTRODUCTION.

2.3. Primary Architecture

On the basis of Comet and EDA, a system architecture to implement event-driven web monitoring systems can be designed primarily as Figure 1.

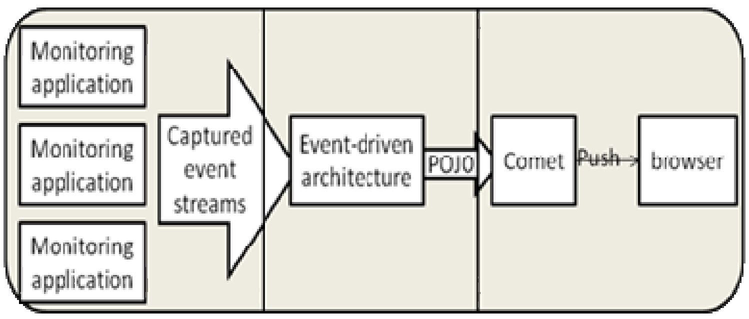

Figure 1. Primary architecture

This primary architecture can be divided into three parts:
• The monitoring applications for capturing event streams.
• The event-driven architecture for filtering, analyzing and processing the event streams from the monitoring applications
• The web applications for showing real-time information to users. This part includes Comet in server and Comet in client browser.

3. An Actual Application and Detailed Solution

We have already implemented a web monitoring system by the architecture said above. In this section, we will give a detailed solution by describing how to realize the monitoring system. Firstly, we give the list of tools, methods, platforms or protocols we used as following:

- MyEclipse6.5 with JDK1.6
- Tomcat 6 (Comet-Friendly server)
- Oracle (Yes. Relational database is still used. We will give the reason later)
- Bayeux (Protocol for Comet)
- Esper (a lightweight, embeddable open source implementation of EDA)

Some of them are familiar while others are not. Secondly, we introduce Bayeux and Esper.

3.1. Bayeux Protocol

The Bayeux Protocol defines using long polling approach [11]. HTTP requests sent from the client are hold on on the server till the requests can be fully answered. This protocol provides a flexible, saleable API based around a publish/subscribe model. In our monitoring system, we make server to be publisher and client to be subscriber. Messages encoded as JSON objects are routed via channels.

Server side implementations need to use non-blocking IO (NIO) to guarantee an efficient implementation. So we choose tomcat 6(can be set to support NIO) as server. Some Java libraries need to be imported. They are “Cometd-api.jar”, “Cometd-bayeux.jar” and “jetty-until.jar”. Then the web server can publish topics. For example, the following code pushes a JSON object named ”total” via channel “/Manage/cal” to Browsers:

```
ServletContext application=this.getServletContext();
Bayeux b=(Bayeux)application.getAttribute(Bayeux.DOJOX_COMETD_BAYEUX);
Channel c=b.getChannel(“/Manage/cal”,true);
// ……
c.publish(userId,total,”total”);
```

To implement Comet at client side, we use a JavaScript library-“dojo”. Then client browsers can subscribe interested topics from the channels. For example, the following code receives the message the server pushed:

```
<script src=”dojo-release-1.1.2\dojo\dojo.js”>
djConfig=”isDebug:false,parseOnLoad:true”>
</script>
// ……
dojo.require(“dojox.cometd”);
dojox.cometd.init(“cometd”);
dojox.cometd.subscribe(“/Manage/cal”,currentProcessor,”total”);
```

3.2. Esper

Esper is a Java library that can be integrated into multiple applications developed by Java. In Esper, event can be map, POJO and XML. Esper offers an event pattern language (EPL) to specify expression-based event pattern matching. This method of event processing matches expected sequences of presence or absence of events or combinations of events. EPL can also be used to provide the windows, aggregation, joining and analysis functions for use with streams of events. EPL has been designed for similarity with the SQL query language. For example, an event (POJO) is defined as:

```
public class CardEvent implements Serializable{
        private String from;
        private String to;
        private Double amount;
}
```

Then, the following statement can be used to do some relevant statistics:

```
select
count(*) as noOfCard,sum(amount) as total,
avg(amount) as average
from CardEvent.win:time(15 sec);
```

With the help of EPL, Esper implements a perfect EDA. When large influx of event streams come to Esper, The EPL statements previously defined judge whether the event stream matches the EPL. This method of event processing matches expected sequences of presence or absence of events or combinations of events. Lastly, Esper outputs POJOs. In addition, Esper can also access with the historical database, a database embedded is used to ensure the consistency of the data with the external database. If necessary, Esper can combine the data stream and historical data to do complex processing (That's why we used Oracle).

3.3. Detailed Solution

On the basis of Bayeux and Esper, A detailed solution can be designed as Figure 2.

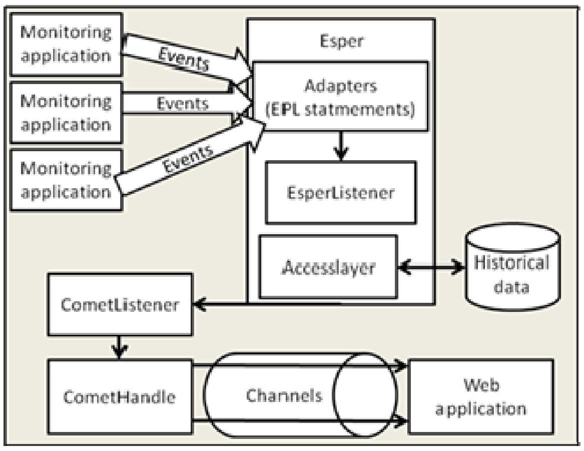

Figure 2. Detailed solution

The functions of components in the whole system:

- Monitoring applications (such as scanning equipment) are components for capturing events.
- Adapters: They are responsible for defining EPL statements to filter, analyze and process the data delivered from monitoring applications.
- Esper Listener: They are designed to be attached with the statement in "Adapters", when the statement evaluates to true, "Esper Listener" will be triggered and output POJOs.
- Accesslayer: It is for communicating with the historical database.
- Comet Listener: This component is for listening to Esper components and receiving POJOS from the Esper components.

- Comet Handle: A thread pool that contains several threads that is named “Comet Handle Thread”. These threads receive and handle asynchronous requests from the client browsers.
- Web application is for displaying data on browsers.

The most important innovative point of this architecture is combing EDA and Comet to implement event-driven web monitoring systems.

4. Architecture Analysis

4.1. Advantages

- Good performance

We did experiments to test our monitoring system. Two time gaps were recorded: the time gap between monitoring application and the events were processed over by Esper (simply recorded as “process event time gap”), the time gap between Esper and displaying data to users(recorded as “react time gap”).The experiments have been done for 1000 times. We selected randomly 10 records and show them in Table 1.

Table 1. 10 Records of 1000 Times

No.	Process event time gap	React time gap	Time gap
1	25ms	119ms	144ms
2	17ms	121ms	138ms
3	20ms	110ms	130ms
4	15ms	107ms	122ms
5	17ms	119ms	136ms
6	19ms	105ms	124ms
7	17ms	104ms	121ms
8	24ms	110ms	134ms
9	25ms	103ms	128ms
10	27ms	117ms	144ms
Average	20.6ms	111.5ms	132.1ms

The average time gap of the 10 records is 132.1ms, and the average time gap of the 1000 times is 131.5ms while 50 EPL statements are included in our system and 10 of them are complex queries. It means that the architecture we designed can process events and display new data to clients instantly. Good performance is the most important advantage of this architecture.

- Good compatibility

Esper is lightweight. It’s easy to be integrated into the existing applications developed by Java. Bayeux can be used to any browsers supported Ajax, and many servlet-supporting and NIO-supporting servers.

- No firewall restrictions

The architecture is based on standard HTTP ports (80 and 443) and HTTP protocol. So, it can work behind the firewalls.

4.2. Problem and Analysis

- Scalability Problem

In this architecture, the web server needs to keep a long connection with each client to achieve server-push. When the number of concurrent clients is small, this architecture can work normally. However, when the number is very big, it’s hard to load. In this scenario, loading balance and cluster technology can be considered.

5. Summarize and Prospects

This architecture uses Comet and EDA to implement an event-driven web monitoring system. In fact, it can be applied to many web applications of high real-time demand. Consequently, the architecture introduced in this paper is valuable.

In writers' opinion, Comet is only an interim technology to realize two-way commutation between web servers and browsers. HTML5 will become standard of all browsers in the near future. At that time, Comet in the architecture we designed can be replaced by "Websocket".

Acknowledgment

The Paper is supported by National Natural Science Foundation of China No. 51204071; Science and Technology Planning Project of Guangdong Province (2012B010100019). Ministry of Education University-Industry-Research Project of Guangdong Province (2010B090400535) and The Fundamental Research Funds for the Central Universities (20112M0068).

References

[1] Kim, Yang Sok. Using knowledge base for event-driven scheduling of web monitoring systems. *Lecture Notes in Computer Science*. 2009; 5692: 169-180.
[2] Shanshan Li, Jian Zhou. Transmitter Station Remote Monitor System Based on Browser /Server Structure. *TELKOMNIKA Indonesian Journal of Electrical Engineering*. 2013; 11(3): 1594-1599.
[3] Lijing Zhang, Wei Xiong, Xuehui Xian. Research on Web-based Real-time Monitoring System on SVG and Comet. *TELKOMNIKA Indonesian Journal of Electrical Engineering*. 2012; 10(5): 1142-1146.
[4] Pohja, Mikko. Server push for web applications via instant messaging. *Journal of Web Engineering*. 2010; 9(3): 227-242.
[5] CHEN Hang, ZHAO Fang. Implementation of web instant message system based on server push technology and XMPP. *Computer Engineering and Design*. 2010; 31(5): 83-88.
[6] Zhou Xuan, Wang Li-fang, JIANG Ze-jun. Design and Implementation of AJAX-based Instant Messaging System. *Science Technology and Engineering*. 2009; 9(2): 446-449.
[7] LI Xian-jun, LIU Bo, YU Dan, MA Shi-long. Mixed C/S and B/S architecture pattern based on AJAX. Journal of Computer App lications. 2009; 19(4): 1135-1138.
[8] Yuanyuan Liao, Zhenyu Zhang, and Yanqing Yang. Web Applications Based on Ajax Technology and Its Framework. *Communications in Computer and Information Science*. 2012; 288(1): 320-326.
[9] XU Xiu-hua, WEN Bi-long, LIU Dan-Jiang, BI Shuo-ben. A realization method of instant messaging based on web. *Computer Engineering and Design*. 2003; 24(7): 40-42.
[10] Leveraging Event Driven Architectures http://onepixelahead.com/2010/08/11/leveraging-event-driven-architectures/.
[11] The Bayeux Specification http://svn.Cometd.com/trunk/bayeux/bayeux.html.