AUTOMATED USER INTERFACE HARMONIZATION IN A MULTI-SOURCE BIG DATA STREAMING SCENARIO

Shakthi Priyan T* and Dr.Kanchana Rajaram**

* PG Scholar, Department of Computer Science and Engineering, SSN College of Engineering, Chennai, India
** Associate Professor, Department of Computer Science and Engineering, SSN College of Engineering, Chennai, India

ABSTRACT: Ever growing data are streamed rapidly from multiple sources in any dynamic business application. These data are naturally high in velocity and in volume. Challenges in managing this data are capturing, storage, analysis, transferring and visualization. For capturing and transferring the data, AJAX technology enables asynchronous communication over web which polls for updates. Such a 'pull' mechanism of AJAX is inadequate in the scenario of Big data streaming, to achieve desired performance and scalability objectives. Hence, it is essential to have a mechanism for pushing the database updates to user interfaces of application. Towards this objective, this paper proposes to use Web-Socket technology to capture the updates in database and push to any user interface without requiring data requests from applications. This work harmonizes data in user interfaces designed in different technologies such as JSP, AngularJS, and Bootstrap. From experimental results, it is observed that the proposed approach is scalable for 10000 simultaneous updates and takes merely 33 seconds for pushing 100 updates to 5000 user interface clients of different technologies.

KEYWORDS: Big Data, AJAX, WebSocket, AngularJS, Bootstrap, Data Streaming, User Interface.

INTRODUCTION

In the present decade, large set of data are handled as technologies and devices rapidly grow in all fields in the society. Especially, in the areas of sensors, social media, and industrial data sources, the data are large in volume, transmitted in high velocity with various patterns. Challenges in managing these data are capturing, storage, analysis, transferring and visualization. For capturing and transferring the data, technology like AJAX enables asynchronous communication over web. However, AJAX Web server uses huge number of loops to stream content in real-time (Tonon.L,2011) and it is resource intensive in both memory usage and network bandwidth. As a result, AJAX does not yield expected performance and does not scale well, in the context of real-time big data streaming. Hence, it is essential to use a new technology and approach which can overcome the limitations of AJAX when working in real time streaming. Puranik et al. have proposed an open-source real-time instrumentation middle ware (Tonon.L,2011) for Distributed Real time and Embedded (DRE) system and compared the performance of using AJAX and WebSocket in terms of memory and network bandwidth consumption, throughput, and data loss / delay. The authors concluded that WebSocket is suitable for real time monitoring as compared to AJAX.

Moreover, since WebSocket operate in the application layer and uses TCP as transport layer protocol, it works well in high network traffic generating systems Skovorc et al.(2014) and it is suitable for use in big data streaming scenario. Hence, it is proposed to use WebSocket technology to capture the streamed updates in database. A push approach is proposed in this work, to reflect the streamed database updates in the respective User Interfaces (UI) without requiring data requests from applications. Push approach using WebSocket would be useful as it avoids unnecessary loops and repeated HTTP headers used in AJAX's continuous polling.

The major contributions of this paper are as follows:

1) Developing an approach for data harmonization, in order to fetch the updates from the database and push to user interface models of different technologies.
2) Verifying the scalability and performance of the pro-posed approach when high volume of data updates are streamed from multiple sources in high velocity.

The remainder of this paper is organized as follows, Section II describes existing work. Section III discusses the background of the work. Section IV discusses the proposed approach algorithm framed for pushing the real time updates in different models of user interfaces. Section V discusses the experimentation of the test bed designed and Section VI concludes.

EXISTING WORK

There are very few works that have used WebSocket technology. Real-time web applications such as instant messaging system and on-line games involve dynamic changes of data. In order to reflect the score updates in user interface clients of these applications, Lubbers et al. have compared the performance of WebSocket and HTTP (Bijin chen et al., 2011; Tonon.L,2011). They have observed that WebSocket can reduce latency and up to 500:1 in HTTP header traffic reduction. In addition, WebSocket can reduce the gap between a web browser and desktop applications because an API can be designed to be integrated in a web browser.

Skvorc et al. (2014) have evaluated the performance of the Web-Socket protocol with respect to underlying TCP protocol and compared it against the latency and amount of network traffic generated. WebSocket based communication possesses small overhead due to initial handshaking which does not consume more network traffic than plain TCP based communication.

Pimentel et al. (2012) have built a web application that measures one way transmission latency of sending real-time wind sensor data in order to compare WebSocket communication with HTTP polling. A Jetty servlet has been implemented to upgrade an HTTP connection that would be used for initial handshaking to a WebSocket connection. The authors have observed that WebSocket communication works for low-volume continuous real-time traffic occurring at a rate of 4 Hz and concluded that WebSocket protocol provides better latency for real-time Internet communication. However, the results are based on a small sample of the Internet communication paths for a very limited time.

Traditional Web push technology like comet (Reverse AJAX) provides long way communication using two implementation modes such as long-polling based on AJAX and streaming based on iframe (Comet Programming). Zhang et al. (2013) have compared the working of WebSocket push technology with long polling and streaming approaches of comet, AJAX and iframe. They concluded that the WebSocket push technology outperforms other approaches since, it incurs less data loss, reduced server resources, and network bandwidth with more concurrent users in real-time applications.

None of the existing works that have worked with Web-Socket technology have studied and analyzed its performance in big data streaming scenario and considered pushing data to UI clients of different technologies.

BACKGROUND

WebSocket protocol is a part of HTML5, which is standardized by the W3C and IETF as RFC 6455 Anusas-amomkul.T et al.(2014). This protocol is designed for communication over web between client and server. It is an advanced technology that makes it possible to open an interactive communication session between user's web browser and a server using an API establishing a 'socket'. It provides a persistent connection and clients can receive event-driven responses without having to poll the server for a reply. The advantage of WebSocket over HTTP is that the protocol is full-duplex Skvorc et al.(2014) that allow simultaneous two-way communication. Its header is much smaller than a HTTP header more efficient communication even for small packets of data. While using a WebSocket, client uses HTML5 and JavaScript to display data from the server through TCP/IP. The server and client do not need to open a TCP connection for every transaction thus reducing the delay time and HTTP overhead. Since, WebSocket operates in application layer and uses TCP as transport layer protocol, it works well in systems that generate high network traffic.

AngularJS is a client side JavaScript framework built by Google (AngularJS tutorial). It helps to build either small components that can be plugged into the website or entire single page applications. At the core of AngularJS is a module system that allows creating providers, services, factories, and directives. They are used within controllers to create, retrieve, update, and delete data while providing features for manipulating the HTML DOM by customizing JavaScript code. AngularJS extends HTML attributes with the following ng-directives to bind data to HTML with expressions (AngularJS tutorial).

1) The ng-app directive defines an AngularJS application.
2) The ng-model directive binds the value of HTML controls such as input, select, and text area to application data.
3) The ng-bind directive binds application data to the HTML view.
4) The ng-controller directive defines AngularJS controllers that control AngularJS applications.

Bootstrap is the most popular HTML, CSS, and JavaScript framework for developing responsive UI (Bootstrap). It is a free front-end framework for faster and easier web development and includes HTML and CSS based design templates for forms, buttons, tables, navigation, models, and JavaScript plugins.

PROPOSED APPROACH

In a dynamic business application involving big data, huge volumes of data are streamed in high velocity from multiple sources. The streamed data are stored in a database. The database updates are captured by UpdateCapturer as and when updates happen and these updates are reflected in respective UIs of clients by UI Updater. The clients UIs are implemented in different technologies such as JSP, AngularJS, and Boot-strap. Data Harmonizer comprising of UpdateCapturer and UIUpdater has the ability to capture data from various data sources. The user interface is able to view the updates as and when it is persisted in the database. The proposed system architecture is shown in Figure 1.

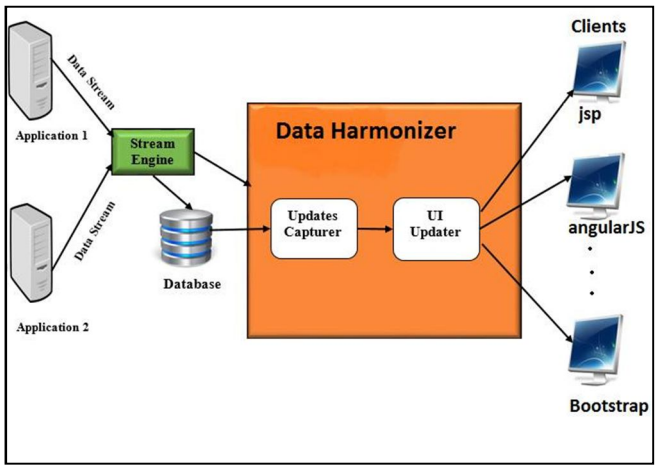

Figure 1. System architecture

Update Capturer

UpdateCapturer module identifies and captures updates in data that has been inserted into, modified in, or deleted from the database. An update in database caused by SQL statements such as INSERT, UPDATE or DELETE, triggers updated data to be captured immediately via WebSocket. Algorithm 1 depicts the procedure for implementing database trigger that communicates the updates to a WebSocket.

Algorithm 1 Algorithm for creating trigger

Precondition: There is an update in DB Table
Post condition: The database updated is pushed to UI clients where sessions are active

```
1: procedure UPDATETRIGGER()
2: SetTriggerEvent(AFTER UPDATE, AFTER DELETE, AFTER INSERT)
3: setTriggeronTable(<< tablename >>)
4: Call derbyobj.exec( "Create or Replace Trigger UpdateTrigger AFTERINSERT |
AFTERUPDATE | AFTERDELETE ON
    << tablename >> as BEGIN
    If UPDATE(ROW) call push(WebSocketObject, ROW)
    END")
End Procedure
```

UI Updater

After every insertion, modification or deletion, *UpdateTrigger()* would invoke a JAVA procedure that pushes the newly inserted values to clients with open session. As and when a user interface session is opened, WebSocket TCP connection would be established and the *onOpen()* method gets called in both client and server. The event handlers associated with the WebSocket connection provides information on time at which the connection was established, time at which the incoming update is received, and errors like session disconnect, etc. The UI clients that are connected through port 80 and 443 establish communication by initiating a handshake with the server. In WebSocket protocol, the handshake

consists of URL of the client and the response from the server. The communication is carried out using jetty's implementation of WebSocket API that is integrated with jetty HTTP server and jetty servlet. Jetty servlet is responsible for upgrading the HTTP connection to WebSocket connection. The *onOpen()* method of server queries the database table for updates and pushes them to clients which are connected. The design of Push approach is shown in Algorithm 2.

Algorithm 2 Algorithm for real time push

Precondition: WebSocketObject instantiated with com.websocket.Driver and runs on ws :== localhost :<< PORT >> , ROW Object initialized with <<attributelist>>,

```
1:    procedure PUSH(WEBSOCKETOBJECT, ROW)
2:    Temp Row:get < attributename > : Row:get < attributename >
          ▷Enable TCP Communication between client browser and to the WebSocket server.
3: if Successful (Connection) then WebSocketObject.send(Temp)
4:    return
```

Algorithm 3 Algorithm for intercepting message

Precondition: RowInHTML object has text/HTML content.

```
1:    procedure ONMESSAGE(WEBSOCKETOBJECT, ROWIN-HTML)
2:    Client   WebSocketObject:Initialize()
3:    Client.documentType("text/HTML").add(RowInHTML)
4:    Client.pushToDisplay()
5:    return
```

EXPERIMENTATION

Test Bed

In the test bed, streaming clients are connected to a Derby database where data are stored and captured via WebSocket protocol which is used to push database updates to different models of UI clients using different technologies such as JSP, AngularJS and Bootstrap. These streaming clients and UI clients are working in AMD A8-5500 processor with 8.00 GB RAM. The TCP connection between the client and the server is established after the occurrence of a hand-shake over the HTTP protocol. As soon as a WebSocket instance is created, server sends a HTTP header telling the client it's switching to the WebSocket protocol. Browsers of UI clients establish a connection with a server and keep sending or receiving data. Java methods such as *onOpen* and *onClose* are annotated with *@OnOpen* and @OnClose respectively that define which methods would be called for a new client connection and disconnection. Method *onMessage* is annotated with *@OnMessage* that defines which method would be called when a new message is received from the client. The WebSocket server endpoint is connected using *WebSocket()* and passing the endpoint URL:ws://localhost: 8080/byteslounge/WebSocket. The test bed designed for load testing is shown in Figure 2.
The proposed approach for data harmonization is tested with the following objectives:

1) To verify scalability of the approach in terms of data updates
2) To measure the performance in terms of response time for varied number of UI client

Experiment 1- Scalability

This experiment is carried out by load testing performed using JMeter (Dmitri Tikhanski), a multi-threading framework. Load testing is carried out by increasing the number of UI clients. For 100 UI clients, the response time is measured by varying the number of updates from 1000 to 10000. The measured response times that are tabulated in Table I indicate the time required to push the updates to respective UI clients. It is observed that, even with 10000 simultaneous updates , the proposed push approach using WebSocket incur only 36 seconds for reflecting them in the UI clients.

Experiment 2- Performance

This experiment is performed using JMeter to measure the performance of the proposed approach, in terms of response time. The time incurred for pushing 100 updates by varying the number of UI clients from 500 to 5000. As the number

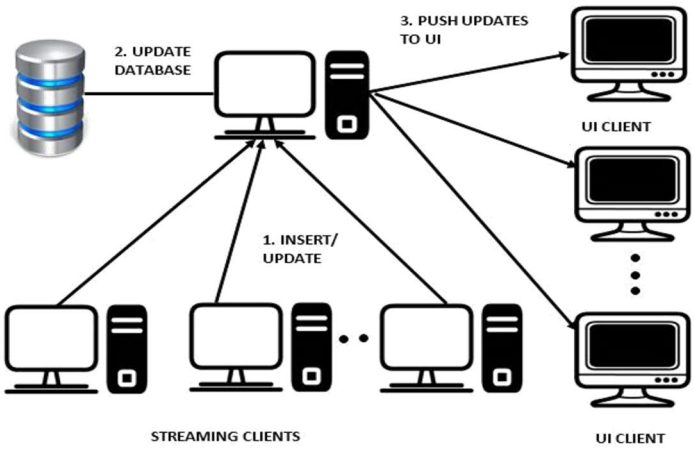

Figure 2 Test bed

Table 1. Verifying Scalability by varying number of database updates

Update in DB	Response Time(sec)
1000	3.07
2000	5.43
3000	8.33
4000	9.30
5000	11.8
6000	15.7
7000	20.2
8000	26.1
9000	29.3
10000	36.0

of UI clients increases, the time taken for pushing the updates also increases and they are tabulated in Table 2 and pictorially depicted in Figure 3. It is observed that, even with 5000 UI clients, the time taken for pushing 100 updates to respective UI clients is merely 33 seconds.

Table 2. Performance of the proposed approach by varying number of UI clients

Number Of UI Clients	Response Time(sec)
500	2.52
1000	13.6
1500	15.2
2000	19.9
2500	22.8
3000	24.2
3500	25.2
4000	27.8
4500	30.8
5000	33.2

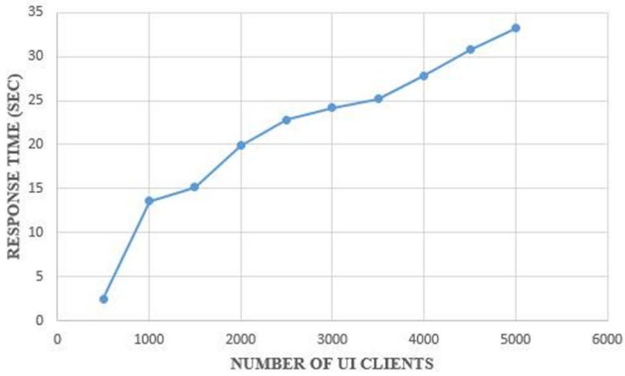

Figure 3. UI Updater response time for varying number of UI Clients

CONCLUSION

This work proposes using WebSocket technology for harmonizing user interface with data updates in contrast to clients requesting for updates. The approach works in a streaming environment where huge volume of data is streamed from multiple sources in high velocity. The user interface models are designed in different technologies such as JSP, AngularJS and Bootstrap. Our approach pushes the updates as and when they reach the database irrespective of the technologies of UI models. The approach is experimented with increased number of simultaneous updates and UI clients and it is found to be scalable for 10000 updates and incurs only 33 seconds for pushing 100 updates to 5000 UI clients of different technologies.

REFERENCES

[1] Anusas-amornkul.T and Silawong.C; *The study of compression algorithms for websocket protocol*; (2014) ECTI-CON; 1–6.
[2] Bijin Chen and Zhiqi Xu; A framework for browser-based multiplayer online games using webgl and websocket; (2011); ICMT; 471–474.
[3] Lijing Zhang and Xiaoxiao Shen; Research and development of real-time monitoring system based on websocket technology; (2013); MEC; 1955–1958.
[4] Puranik. D.G; Feiock. D.C and Hill. J H; (2013); *Real-time monitoring using ajax and websockets; ECBS*; 110–118.
[5] Pimentel.V and Nickerson.B.J; Communicating and displaying real-time data with websocket; (2012); **16;** 45–53.
[6] Skvorc.D; Horvat.M and Srbljic.S; Performance evaluation of websocket protocol for implementation of full-duplex web streams; (2014); MIPRO; 1003– 1008.
[7] AngularJS Tutorial; http://www.tutorialspoint.com/angularjs/.
[8] Bootstrap; http://www.w3schools.com/bootstrap/.
[9] Comet Programming: the Hidden IFrame Technique http://www. webreference.com/programming/javascript/rg30/2.html.
[10] Dmitri Tikhanski; WebSocket Testing With Apache JMeter; https:// blazemeter.com/blog/websocket-testing-apache-jmeter.
[11] Reverse Ajax, Part 1: Introduction to Comet. http://www.ibm.com/ developerworks/library/wa-reverseajax1/.
[12] Tonon.L; HTML5 TUTORIAL:WEBSOCKET; https://thenewcircle.com/bookshelf/html5_tutorial/web_sockets.html.