Real-Time RIA's with Apache Derby and Grizzly Comet

Jeanfrancois Arcand
Francois Orsini
Senior Staff Engineers
Sun Microsystems

Agenda

- Introduction
- What is Ajax Push (aka Comet)?
- Potential Drawbacks and Pitfalls
- Mixing Apache Derby and Grizzly Comet
- Demo

What is Comet Request Processing (or Ajax Push)

Comet is a programming technique that enables web servers to send data to the client without having any need for the client to request for it. It allows creation of event-driven web applications which are hosted in the browser.

What is Ajax Push (aka Comet)?

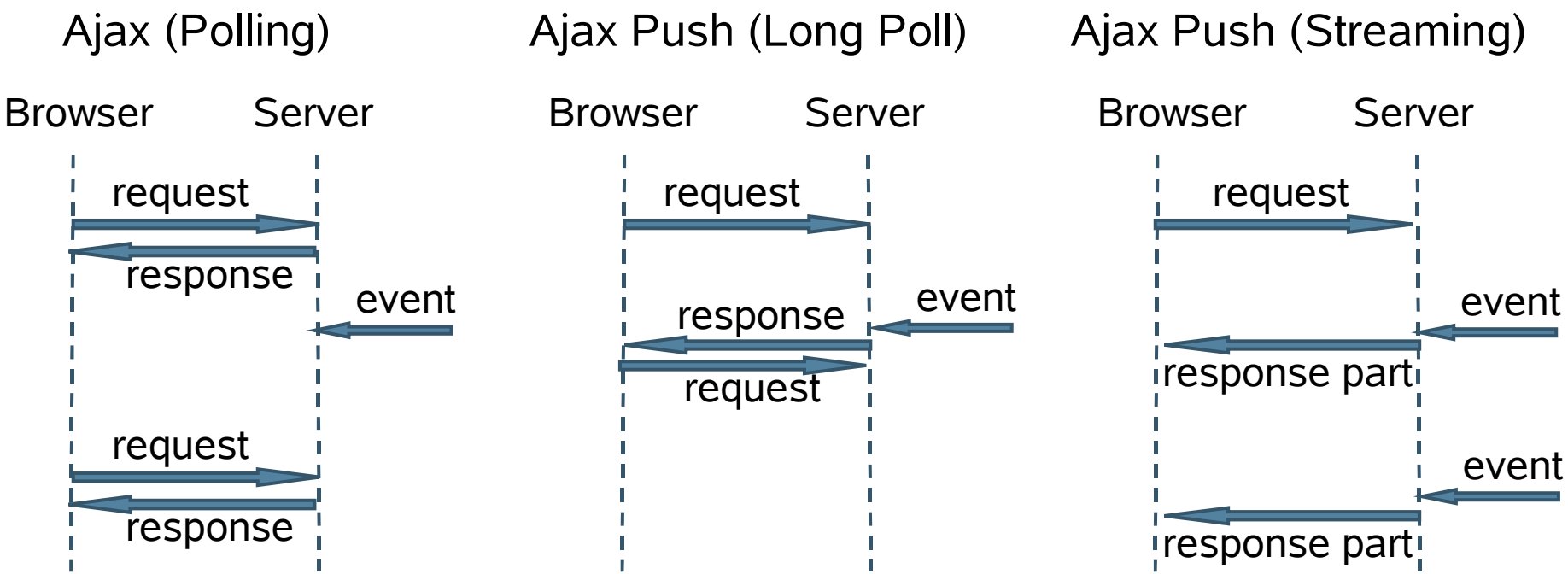
- Use it to create highly responsive, event driven applications in a browser
  - Keep clients up-to-date with data arriving or changing on the server, without frequent polling
- Pros
  - Lower latency, not dependent on polling frequency
  - Server and network do not have to deal with frequent polling requests to check for updates
- Example Applications
  - GMail and GTalk
  - Meebo
  - JotLive
  - KnowNow
  - 4homemedia.com
  - Many more …

How does the “Push” to the browser works

- Deliver data over a previously opened connection
  - Always “keep a connection open”; do not respond to the initiating request until event occurred
  - Streaming is an option by sending response in multiple parts and not closing the connection in between

How does “Push” to the browser work?

Standard Ajax compared to Ajax Push options

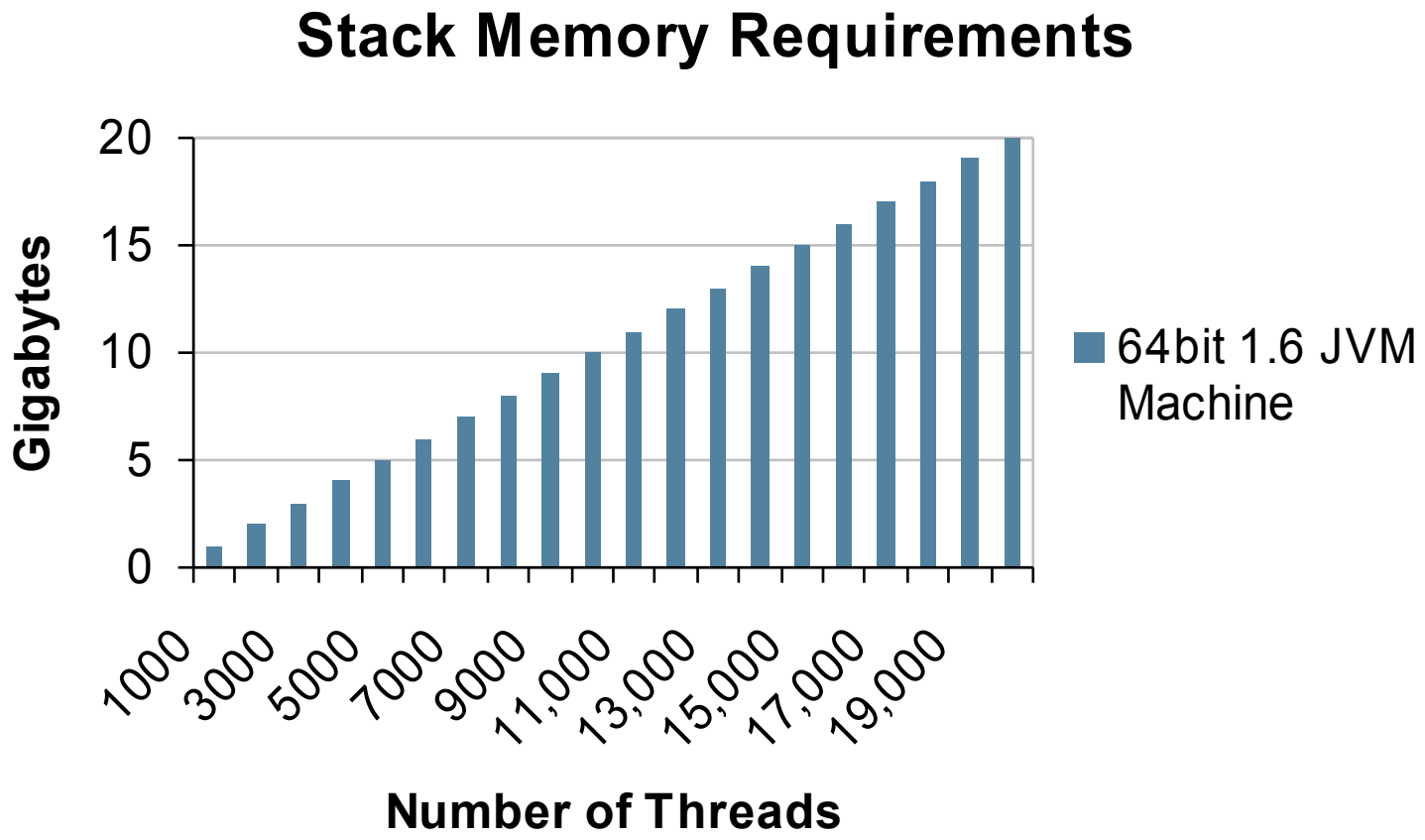


Architecture Challenge

- Using blocking, synchronous technology will result in a blocked thread for each open connection that is “waiting”
  - Every blocked thread will consume memory
  - This lowers scalability and can affect performance
  - To get the Java Virtual Machine (JVM™) to scale to 10,000 threads and up needs specific tuning and is not an efficient way of solving this
- Servlets 2.5 are an example of blocking, synchronous technology

Architecture Challenges

Affect of Blocking threads (default thread stack size)



Technology Solutions

- Use new I/O (NIO) non-blocking sockets to avoid blocking a thread per connection
- Use technology that supports asynchronous request processing
  - Release the original request thread while waiting for an event
  - May process the event/response on another thread than the original request
- Advantages
  - Number of clients is primarily limited by the number of open sockets a platform can support
  - Could have all clients (e.g. 10’000) “waiting” without any threads processing or blocked

Do Comet enabled Servers exist?

- Yes, more and more servers that support Comet request processing are available:
  - Grizzly
  - GlassFish v2, v3
    - Support Grizzly Comet and Jetty Comet.
  - Jetty 6
  - Ligthttpd
  - Tomcat 6
  - ...
- Today we are going to focus on Grizzly's Comet support

What is Grizzly

- Grizzly is a multi protocol (HTTP, UDP, etc.) framework that uses lower level Java NIO primitives, and provides high-performance APIs for socket communications.
- In GlassFish, Grizzly is the HTTP front end.

Comet in Grizzly

- Comet support is build on top of Grizzly Asynchronous Request Processing (ARP), a scalable implementation that doesn't hold one thread per connection, and achieve as closer as possible the performance of synchronous request processing (SRP).

Goal of Grizzly's Comet

- Hide the complexity of NIO/Asynchronous Request Processing.
- Make it available to various technologies from AJAX based client, JSF, JSP, Servlet, POJO, JavaScript to “traditional” technologies such as JMS, EJB, database, etc.
- Allow complicated scenarios but also support POJO based development.
- **Main Goal: Make it simple to use!**

Potential Drawbacks and Pitfall

- Beware of flooding clients with too many events
  - > Filters (throttles) on the client or server side? Which events can be discarded, which can't?
- Firewalls may terminate connections after a certain amount of time
  - > Solution: Re-establish connection after tear-down or at certain intervals
- The HTTP 1.1 specification suggests a limit of 2 simultaneous connections from a client to the same host
  - > Some use a separate host name for the “Push” connection
- In security terms the attack surface is very similar to standard Ajax applications, for denial of service (DoS) the “wait” is a consideration
- Possible lost of data when the connection is closed
  - > Real time updates can still occurs when the application goes offline or during re-connection.

Potential Drawbacks and Pitfall (Cont')

- HTTP Streaming is more challenging
  - Portability issue to different browsers and XMLHttpRequest (XHR)
    - IE, for example, does not make data available until connection close or only if you flush 2k of white space to make it work)
    - Use IFrames instead for portability
  - With streaming data will accumulate, release memory regularly
  - Primitive proxies may buffer data in a way that interferes with streaming
- Possible lost of data when the connection is closed
  - Real time updates can still occurs when the application goes offline.

Solutions: Mixing Derby and Grizzly's Comet

- Filtering/Throttling messages can be archived locally and asynchronously via Derby and processed later, when the server is no longer overloaded.
  - Derby runs as an embedded server DB engine
- When disconnected, the server can cache any real time update and push them later.
- When disconnected, the client can cache any update and push them later.
  - Derby can be accessed via JavaScript and Java.

Apache Derby

Main Characteristics

- Complete Multi-User relational database engine
- Embeddable and client/server database
- Easy to use, zero maintenance
- Small footprint (2MB)
- Standards-based [Java DataBase Connectivity (JDBC™) software, SQL92/99/2003]
- Compact, secure, mature and robust
- 100% Java technology (write once, run anywhere)
- Java DB is Sun's supported distribution of Apache Derby

Complete Relational Engine

- Multi-user, transactions, isolation levels, deadlock detection, crash recovery
- Fully ACID compliant
- Complete SQL Engine including:
  - views, triggers, stored procedures, functions
  - Foreign keys, check constraints, cost based optimizer
- Data caching, statement caching, write ahead logging, group commit
- Online backup/restore
- Database encryption, authentication, authorization

Embeddable Multi-User Database

- Database engine may run in application's virtual machine
  - No additional process
  - Database requests are method calls within the Java Virtual Machine (JVM™)
- Startup and shutdown controlled by application
- Just one Java Archive (JAR) file
- Invisible to the user
- Easy to use, zero maintenance
- Can also run as a standalone database *server*

Client-Server Database

- Database engine can run in a client-server configuration
  - Standalone server
  - Server runtime management tool
- Secure and support for various network connection mechanisms
- Easy to use, zero maintenance
- Can also run embedded in other server frameworks

The Multi-Tier DB

- Client tier
  - Embeddable storage in standalone client applications and web 2.0 RIA applications (demo)
  - Read-only DB in JAR file
  - Java DB on a memory stick
- Middle tier
  - Embeddable middle tier database engine (e.g. in GlassFish)
  - Can act as front-end cache for back-end enterprise DB's
  - Persistent cache for middleware frameworks
- Back-end tier
  - Standalone departmental server database
  - Can support large number of concurrent users

Data Synchronization

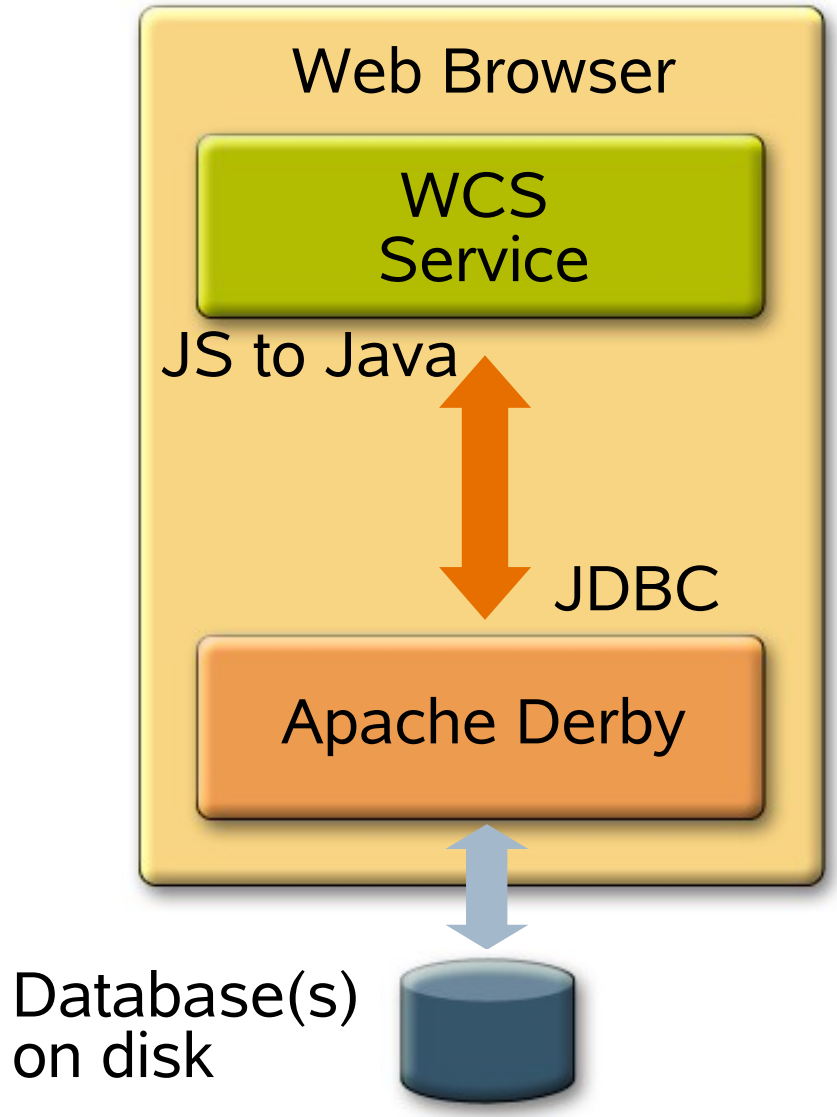
- Not always required
  - Depends on the application
- Conflicts resolution is the biggest problem
- At the application level
  - Zimbra desktop
  - Offline Derby Google Calendar (demo part of Derby)
  - Real-time synchronization with Comet
- Database level
  - Daffodil Replicator w/ Java DB
    http://sourceforge.net/projects/daffodilreplica/

Data Replication

- Not always required
  - Depends on the application
- Several open source initiatives
  - Fault-tolerant, fail-over & load-balancing
  - Support Java DB (Apache Derby)
  - Sequoia (formerly C-JDBC)
    - Apache 2.0 license
  - HA-JDBC
    - LGPL

Web Client Store Service

- WCS service access via JavaScript technology
- Asynchronous capability
- Embedded Apache Derby
- ACID compliant
- Fast
- Zero administration



Local Client Service via JavaScript Technology

- Interact with local service directly via JavaScript technology
  - No new syntax or else–All JavaScript technology
- Local service installed as a Java Plug-in software extension
  - Trusted, runs in Java platform Sandbox
  - Automatically installed on client host
  - Service versioning management handling
- http://java.sun.com/j2se/1.4.2/docs/guide/plugin/developer_guide/extensions.html
- LiveConnect to interact transparently with core service implementation in Java technology
  - JavaScript technology to Java technology and vice-versa
  - Browser agnostic

Demo—Things to Remember

- Ease of deployment over a large user base (e.g. consumer desktops)
- Transparent–embeddable and zero–administration
  - invisible to the end user
- ACID RDBMS–high levels of durability and consistency to prevent data loss
- Ease of upgrade (using Firefox or Java Web Start software)
- Small footprint
- Highly secure to ensure desktop data is safe

Derby Integration

- Apache ActiveMQ
- Apache JPA
- Apache Roller
- Apache Cocoon
- Apache Geronimo
- Apache JDO
- Apache Xalan
- Daffodil Replicator
- Data Direct SequeLink
- DB Visual Architect
- Eclipse
- Project Glassfish
- Hibernate
- IBM DB2 Everyplace
- IBM DB2 JDBC Universal Driver
- IBM WebSphere App Server
- ISQL-Viewer
- Java DB
- JBoss
- JPOX
- Jython
- Kodo 3.3.3
- Maven
- NetBeans Software
- Zimbra
- Red Hat Application Server
- AntHill Pro
- Sequoia (C-JDBC)
- SQuirreL SQL
- Sun Java Enterprise System
- Sun Java System Portal Server
- Sun Java Studio software
- Sun Java Platform, Enterprise Edition
- Sun Java System Service Registry
- SUSE Linux 9.3
- Zend core for IBM
- Tomcat

Next Release Features (10.4)

- Additional security improvements
- SQL Roles
- SQL OLAP functionality
  - e.g. LIMIT()
- More Performance Improvements
- Basic replication
- Table Functions (VTI)
- JMX management interface
- More info at:
  - http://wiki.apache.org/db-derby/DerbyTenFourRelease

Demo & Use Cases: More Information

- Additional Demo's & Information publicly available at http://developers.sun.com/javadb/
- Working with Apache Derby http://wiki.apache.org/db-derby/WorkingWithDerby
- Uses of Apache Derby http://wiki.apache.org/db-derby/UsesOfDerby
- Grizzly http://grizzly.dev.java.net
- Comet and Grizzlet

  http://weblogs.java.net/blog/jfarcand/

Conclusion

- Pushing data to the web client can form a crucial tool in the developer's arsenal when latency is important
- Loosing data when disconnected is a show stopper for any applications that wants to implement Comet based application.

Solution:

Grizzly + Apache Derby!!!

ApacheCon

Real-Time RIAs with Apache Derby and Grizzly Comet

Jeanfrancois Arcand
Francois Orsini
Senior Staff Engineers
Sun Microsystems