Smart Triggers/Push Data

NAGARAJU & BRIAN

Value Proposition

- Selectively trigger events based on changes in server data
- Real time ‘push’ notifications help clients avoid polling the server
- Small data flow allows simple small clients to work with many triggered events at once

What are Smart Triggers in JDBC

- Smart Triggers are registered events on the server that you subscribe to from your JDBC client
  - Triggers are based on a SQL statement query that matches changes made to a table
  - SELECT id FROM CUSTOMER WHERE cardBalance > 20000;
- One client can listen to many events from many tables, allowing a wide range of monitoring opportunities
  - Monitor account balances
  - Take action on suspicious behaviors

What does a Smart Trigger Look Like?

- It's designed to be a simple set of classes/interfaces in Java
- Designed for both simple standalone monitor applications as well as integration into multi-threaded environments
- Leverages the Push Notification feature in the server to do the heavy lifting
- Receives JSON documents when a trigger occurs

Use case: Banking

- Bank accounts
  - I want to be alerted when an account balance drops below zero dollars
  - I don't want to write SPL or install stored procedures and triggers
  - I want to be notified in my client application
  - I don't want to poll the database for this information or re-query each time a balance changes from the client

Smart Trigger Bank Code

```
public class BankMonitor implements IfmxSmartTriggerCallback {
      public static void main(String[] args) throws SQLException {
            IfxSmartTrigger trigger = new IfxSmartTrigger(args[0]);
            trigger.timeout(5).label("bank_alert");
            trigger.addTrigger("account", "informix", "bank",
                  "SELECT * FROM account WHERE balance < 0", new BankMonitor());
            trigger.watch(); //blocking call
      }

      @Override
      public void notify(String json) {
            System.out.println("Bank Account Ping!");
            if(json.contains("ifx_isTimeout")) {
                        System.out.println("-- No balance issues");
            }
            else {
                        System.out.println("-- Bank Account Alert detected!");
                        System.out.println("   " + json);
            }
      }
}
```

Demo!

Smart Triggers in Other Languages

- Adding Smart Triggers to the JDBC driver allows other languages to have this support
- Groovy, JavaScript (NodeJS), Python, Scala and more

NodeJS Smart Trigger Example

```
var java = require("java");
java.asyncOptions = {
  syncSuffix: ""
};

java.classpath.push("ifxjdbc.jar");

var smartTrigger = java.newInstanceSync("com.informix.smartTrigger.IfxSmartTrigger",
"jdbc:informix-sqli://localhost:20290/sysadmin:USER=informix;PASSWORD=informix");

smartTrigger.timeout(10);

smartTrigger.open();
smartTrigger.addTrigger("pushtest", "informix", "ewdb", "SELECT * FROM pushtest", "smart-trigger");
smartTrigger.tableRegistration();

var foo = smartTrigger.readFromSmartBlobObject();
console.log(foo);
```

Use Case: Blockchain

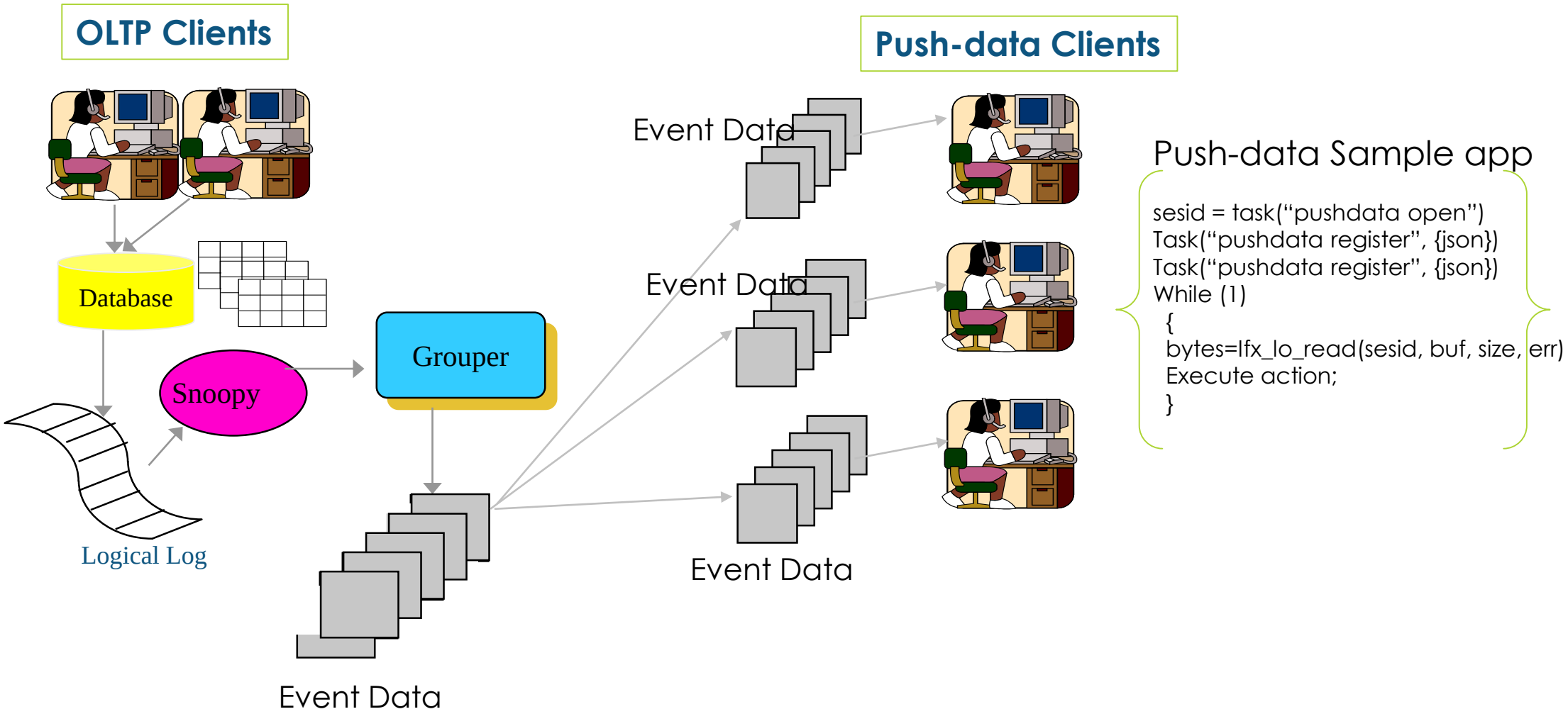
- With Smart Triggers we can integrate into Blockchain use cases
  - Changes to the data with Smart Triggers can initiate a smart contract!
  - Many blockchain examples/demos/applications use NodeJS
    - Having JDBC work with NodeJS allows us to be part of these examples and leverage our technology alongside blockchain
- Conference registration blockchain demo using Ethereum public blockchain
  - Conference registration smart-contract
    - Callback registered on conference registration blockchain smart-contract inserts data into Informix database.
  - Smart-trigger registered on Informix conference database executes Hotel Reservation smart-contract in blockchain.

Blockchain Demo!

Push data to Client (Server functionality)

- Push vs Pull architecture
- Event driven programming model

Server Architecture Diagram



Push-data Sample app

```
sesid = task("pushdata open")
Task("pushdata register", {json})
Task("pushdata register", {json})
While (1)
  {
  bytes=lfx_lo_read(sesid, buf, size, err)
  Execute action;
  }
```

API Calls

- TASK('pushdata open');
  - Register client session as a push data session
  - Returns session id, need this id to read event data.
- TASK('pushdata register', {event and session attributes});
  - Register event conditions, and session specific attributes
- Smart blob read API (ifx_lo_read() or equivalent call)to read event data
  - Pseudo smart blob interface to read event data.
  - Returns JSON document(s).
  - Can be configured as blocking or non-blocking call
- TASK('pushdata deregister', {event condition details});
  - De-register event conditions.

JSON attributes for registering new event conditions

Input attribute name	Description
table	Table name to be registered
owner	Table owner
database	Database name
query	Select statement including projection list and where clause to register for changes in a data set.
label	User defined string to be returned along with event document – useful to differentiate between events when more than one push-data event registered within the the same session
timeout	How long client gets blocked in smartblob read api for new events to be returned by server before returning timeout document.
commit_time	Return event data committed after this transaction commit time.
txnid	8 byte unique id. Higher order 4 bytes: commit work log id, lower order 4 bytes: commit work log position.
max_pending_ops	Maximum number of event records to be kept in the session pending
maxrecs	Maximum number of records to be returned by smartblob api read call.

Example Command:

```
execute function informix.task('pushdata register',
{table:"creditcardtxns",owner:"informix",database:"creditdb",query:"select uid, cardid, carddata from creditcardtxns where carddata.Amount::int >= 100",label:"card txn alert"})
```

Event Data JSON Attributes:

Attribute name	Description
operation	Operation type: Insert/Delete/Update
table	Table name
owner	Table owner
database	Database name
label	Optional user specified data for the even condition.
txnid	8 byte unique id. Higher order 4 bytes: commit work log id, lower order 4 bytes: commit work log position.
commit_time	Transaction commit time for the event data.
op_num	Increasing sequence number for the event document within a given transaction. If transaction generate 10 events, then each document returned will have incrementing op_num starting from 1 to 10.
rowdata	Row data in JSON document format. Data is returned in column name as key and column data as value.
before_rowdata	Before row data for “update” operation.
ifx_isTimeout	Document with this attribute is returned with value set to “true” if no events gets triggered within the timeout interval specified by the client.
ifx_warn_total_skipcount	Warning document with this attribute is returned with cumulative number of discarded events due to max_pending_ops attribute threshold.

Example event data documents

- **Sample output for Insert operation:**

{“operation”:"insert",”table”:"creditcardtxns",”owner”:"informix",”database”:"creditdb",”label”:"card txn alert",”txnid”:2250573177224,”commit_time”:1488243530,”op_num”:1,”rowdata”:{“uid”:22,”cardid”:"6666-6666-6666-6666",”carddata”:{"Merchant":"Sams Club","Amount":200,"Date":2017-05-01T10:35:10.000Z } }}

- **Sample output for Update operation:**

{“opertion”:"update",table:"creditcardtxns",”owner”:"informix",”database”:"creditdb",”label”:"card txn alert",”txnid”:2250573308360,”commit_time”:1488243832,”op_num”:1,”rowdata”:{uid:21,cardid:"7777-7777-7777-7777",”carddata”:{"Merchant":"Sams Club","Amount":200,"Date":"25-Jan-2017 16:15"} },”before_rowdata”:{“uid”:21,”cardid”:"6666-6666-6666-6666",”carddata”:{"Merchant":"Sams Club","Amount":200,"Date":2017-05-01T10:35:10.000Z } }}

- **Sample output for Delete operation:**

{“opertion”:"delete",”table”:"creditcardtxns",”owner”:"informix",”database”:"creditdb",”label”:"card txn alert",”txnid”:2250573287760,”commit_time”:1488243797,”op_num”:1,”rowdata”:{“uid”:22,”cardid”:"6666-6666-6666-6666",”carddata”:{"Merchant":"Sams Club","Amount":200,"Date":2017-05-01T13:35:06.000Z } }}

- **Sample output for multi row document when maxrecs input attribute set to greater than 1:**

{[

{“operation”:"Insert",”table”:"creditcardtxns",”owner”:"informix",”database”:"creditdb",”label”:"card txn alert",
“txnid”:2250573309999,”commit_time”:1487781325,”op_num”:1,”rowdata”:{uid:"7",”cardid”:"6666-6666-6666-6666",”carddata”:{"Merchant":"Sams Club","Amount":200,"Date":2017-05-01T15:10:10.000Z } }},

{“operation”:"insert",table:"creditcardtxns",”owner”:"informix",”database”:"creditdb",”label”:"card txn alert",”txnid”:2250573177224,”commit_time”:1488243530,”op_num”:1,”rowdata”:{“uid”:22,”cardid”:"6666-6666-6666-6666",”carddata”:{"Merchant":"Sams Club","Amount":200,"Date":2017-05-01T16:20:10.000Z } }}

]}

Onstat commands

Command to print all sessions:

Onstat –g pd

push-data session structure at 0x5950e028
push-data session id: 70 0x46
Smartblob file descriptor: 39
Number of event conditions: 1
Number of pending event operations: 51
Number of discarded event operations: 0
Total event operations returned to client: 11361

Frist tx begin work logpos: 304:e9e0f0, commit work pos: 305:b697a8
Last txn begin work logpos: 304:e9e0f0, commit work pos: 305:b697a8

Command to print all event conditions:

Onstat –g pd event

push-data subsystem structure at 0x584cc028
push-data session structure at 0x588f5028
push-data session id: 39 (0x27)
Number of event conditions: 1

Push-data event structure at 0x461ed028
Full Table Name: ycsb:informix.usertable
User data: testing...
Replicate name: pushrepl_250_1487957951_1352060721

Onstat commands

Command to print information about specific session:

Onstat –g pd 70

push-data session structure at 0x5950e028
push-data session id: 70 0x46
Smartblob file descriptor: 39
Number of event conditions: 1
Number of pending event operations: 51
Number of discarded event operations: 0
Total event operations returned to client: 11361

Frist tx begin work logpos: 304:e9e0f0, commit work pos: 305:b697a8
Last txn begin work logpos: 304:e9e0f0, commit work pos: 305:b697a8

Command to print event conditions for specific session:

Onstat –g pd 39 event

push-data subsystem structure at 0x584cc028
push-data session structure at 0x588f5028
push-data session id: 39 (0x27)
Number of event conditions: 1
Push-data event structure at 0x461ed028
Full Table Name: ycsb:informix.usertable
User data: testing...
Replicate name: pushrepl_250_1487957951_1352060721

Comparing Smart Trigger and Regular I/U/D Trigger

Smart Trigger	Regular Trigger(I/U/D)
Post Commit	Pre Commit
Register Trigger on a specific Dataset/Event	Trigger gets fired for all changes
Asynchronous and Linear Scalability	Synchronous
Data is in JSON format	SQL format
Trigger logic gets executed in the client	Trigger logic gets executed in the server
Natural fit for event driven programming model	-
No schema changes required to define new smart trigger	Require schema changes and exclusive lock on the table to modify trigger definition

Comparing Push data and CDC

Push data	CDC
Designed for Smart Triggers	Designed for Data streaming/replication
Can register where clause	No where clause support
Data in JSON format	Byte stream
Push technology	Push technology
Only committed transactions are sent to Smart Trigger analysis	All records returned to the user including rollbacked operations
*Once the client disconnect from the server, events for the client aren't captured/staged	CDC can read old log files

Questions ?