Splice Machine Client Server

Version 1.0

**Creation Date – January 29th 2013**

**Last Update Date –**

Modification History

|  |  |  |  |
| --- | --- | --- | --- |
| **Revision** | **Date** | **Name** | **Description** |
| 1.0 | 01.29.13 | John Leach | Initial Draft |

**Table of Contents**

1. Overview/Background 4

2. Requirements 5

3. Proposed Solution 6

4. Assumptions/Limitations 15

5. Outstanding Issues **Error! Bookmark not defined.**

6. Other Design Considerations 16

6.1 QA Considerations 16

6.2 Documentation/Help Considerations 16

6.3 Hardware/OS Considerations 16

6.4 I18N/L10N Considerations 16

6.5 Public API Considerations 16

6.6 Operational Considerations 16

6.7 Build/Release Considerations 16

6.8 Upgrade/Migration Considerations 16

6.9 Future Considerations 16

# Overview/Background

# Requirements

# Proposed Solution

## Proposed System Schema

Several new system level tables will be added to the existing Derby System Schemas.

TXN

Purpose: This table will maintain the transactional record through its state changes.

|  |  |  |
| --- | --- | --- |
| Fields | Data Type | Description |
| rowKey | long | Transaction Start Timestamp or Transaction ID |
| Attributes:Tsc | long | Transaction Commit Timestamp from Zookeeper |
| Attributes:Status | varchar | Active, Error, Commit, Abort, Complete |

PLAN

Purpose: This table maintains the progression of the statement execution

|  |  |  |
| --- | --- | --- |
| Fields | Data Type | Description |
| rowKey | complex | ActivationID and result set number |
| Attributes:activationID | varchar | Activation ID (Class Name) |
| Attributes:resultSetNumber | int | Result Set Number for the activation. |
| Attributes:status | varchar | Wait, Executing, Complete |

## Client Server Architecture

Derby’s JDBC, ODBC, and IJ clients will need to be able to connect via JDBC to a clustered set of Derby Instances running on HBase Region Servers. The Derby instances will be run via an endpoint co-processor created on the running region server. The endpoint co-processor will be extended in the future to retrieve running information (connections, queries, etc.) from the Derby Instance.

The drivers will determine the servers available by connecting to a zookeeper quorum. If a connection disconnects, zookeeper will have to be checked to determine which Region Servers are still represented as ephemeral nodes connected to zookeeper. This structure is illustrated in Figure X.



## SQL Parsing, Planning, and Execution Durability

Since each region server hosting a derby instance runs on commodity hardware that could fail at any time, the SQL parsing, planning, and execution process needs to be durable. If a failure occurs during the parse and planning phase, the SQL will need to be resent via the JDBC client and the parse and plan process retried. Once the Derby Plan transitions to the Splice Optimization Plan, each Operation Branch (the operations that can be parallelized onto a region) must be serialized into a PLAN table to record the completion of each step. Individual failures in the Branch (via coprocessor or remote lookup RPC) have retry mechanisms that will automatically occur. If the Region Server fails during the Splice Optimization Plan’s Execution, the JDBC client will resend the query to another region server Server to pickup execution at the last completed step. If the step that failed was the scan phase (Transfer to JDBC client), the client will remember the number and the record of the last result set returned. This will allow failures to be handled anywhere in the execution chain.

## Replication

<http://hbase.apache.org/replication.html>

## Backups

<https://github.com/oclc/HBase-Backup>

## Recoverability

Rolling back a database to a point in time. The schema changes in Zookeeper would need to be rethought.

# Assumptions/Limitations

## 4.1 Point in Time Recoverability

Our current implementation utilizes Zookeeper for schema definitions (Conglomerate Definitions). We would need to change this to an HBase table if we would like to be able to fully recover to a point in time.

# Other Design Considerations

[Subsections here cover additional design thoughts to the feature]

## QA Considerations

[Thoughts about the types of Unit, Functional, Performance, etc. tests that should be written. Also, will these design changes impact/break existing QA test plans?]

## Documentation/Help Considerations

[Thoughts around the impact of this feature on Documentation and Help are useful here]

## Hardware/OS Considerations

[Are there any hardware-specific or OS-specific issues?]

## I18N/L10N Considerations

[How is this design dealing with Internationalization issues?]

## Public API Considerations

[Are there public API’s? If so add as an additional section]

## Operational Considerations

[Are there any additional considerations for the Operational part of the business?]

## Build/Release Considerations

[Are there any impacts on the build or release processes?]

## Upgrade/Migration Considerations

[When an upgrade or migration occurs, what must be taken into consideration? Are there scripts to accommodate migration? Etc]

## Future Considerations

[What features or design ideas came up that we should consider for the future? List these here to help out Product Management, Development, etc]

# Outstanding Issues

[Remaining issues that need to be resolved still]

# Appendix A

@Override

**public** **void** start(CoprocessorEnvironment e) **throws** IOException {

SpliceLogUtils.*info*(*LOG*, "Starting the coprocessor CoProcessor %s", SpliceDerbyRegionObserver.**class**);

**super**.start(e);

**synchronized** (**this**) {

**if** (*server* == **null**) {

**try** {

*server* = **new** NetworkServerControl();

*server*.start(**new** DerbyOutputLoggerWriter()); // This will log to log4j

SpliceLogUtils.*info*(*LOG*, *server*.getSysinfo());

} **catch** (Exception exception) {

SpliceLogUtils.*logAndThrow*(*LOG*, "Could Not Start Derby - Catastrophic", **new** IOException(exception));

}

}

}

}

/\*\*

\* Logs the stop of the observer.

\*/

@Override

**public** **void** stop(CoprocessorEnvironment e) **throws** IOException {

SpliceLogUtils.*info*(*LOG*, "Stopping the CoProcessor %s",SpliceDerbyRegionObserver.**class**);

**super**.stop(e);

**synchronized** (**this**) {

**if** (*server* != **null**) {

**try** {

*server* = **null**;

} **catch** (Exception exception) {

SpliceLogUtils.*logAndThrow*(*LOG*, "Could Not Start Derby - Catastrophic", **new** IOException(exception));

}

}

}

}