Pipeline Migration and Pipeline II, v1.5

Performance Priorities which involve a database change

1. Splitting the ProcessInstance table into ProcessInstance and BatchProcessInstance
    a. This drastically reduces the size of the ProcessInstance table and prevents row migration
    b. Do we remove the fullpath from the logFile column, and assume it will always be under the WorkingDirectory?
2. Partition the ProcessInstance table
    a. Partitioning requires inserting all rows into a new table.
    b. This may need to be modified to partition to Stream reference.
    c. Partitioning by month seems to be easiest way to eek out extra performance of current transactions
    d. Partitioning by month also helps to theoretically boost performance of the web interface, specifically the task.jsp page if we allow users to limit the stats of processInstances to be within the last week/month(s), which currently isn't enabled.
3. Partition the Stream
    a. Should probably be done before ProcessInstance partitioning
    b. Not sure what the best strategy is.
    c. One strategy is to partition based on Task, or a reference to a root-level task possibly. This gets a little messy.
    d. Second strategy is to partition based on RootStream. The problem with this is that it requires processing of the ancestor stream at table insertion.
        i. But maybe we could add the column to the current database, and start computing those values now. Then later, we can insert those into a new table with interval partitions
    e. Time-only partitioning requires all queries to specify a time range in order to limit search partitions
4. Add a RootStream column to the Stream table, enable RootStream locking.
    a. Enables fast Stream Tree locking
    b. When a process instance acquires a lock, it will no longer lock on it's parent node, which is a stream. It will, from now on, lock on Stream with the primary key of the RootStream of the PI's parent stream node, which will prevent any dead locks as each change to a process instance's status will preclude the modification of any other part of the tree.
    c. Enables the canceling of top-level streams. Without this, it's impossible to know if another database connection has a lock on a child node of a top level stream.
    d. To maintain full backward compatibility, the stream table will need to be modified to populate this column, which may take a while for all streams.
    e. Should speed up rolling back
5. Dead branch isLatest decrementing
    a. Currently, Stream Tree >= Stream Tree where isLatest = 1 >= Stream Latest Tree
    b. Proposal: When a branch is declared dead, all child nodes would be decremented by 1
    c. This changes to (Stream Tree where isLatest = 1 == Stream Latest Tree), which eliminates recursive queries needed to find the latest tree
    d. This requires more work when rolling back a stream, for instance, but may also be offset with the speed gained from a reduced query time.
    e. May benefit greatly status changes to a stream execution tree

Backwards Compatibility Issues

1. Active streams during transition
2. Recently active streams during transition (streams which may be rolled back within a day or so)
3. Streams that haven't been active for a week.

With weekly/monthly partitioning, we can lock an older partition for a Stream, for example, and prevent modification by another process while we are backfilling computed values, say for isLatest decrementing.

Software Migration

- Continue to move as much code from stored procedures back into the pipeline so we can support Postgres and SQLite. This was started with the last few releases, we're continuing on
    - It also enables us to actually understand what operations are slow, and potentially modify them. Right now, Stored Procedures are a black box.
- In conjunction with the Schema changes and partitioning, this should still provide with an overall performance boost
- Move all database statements to Java 7 for readability (A branch with most of these changes already exists)
- Add lots of tests!

Testing

Database Migration Testing

- Need to be able to revert changes in the current database that I will test for all Performance Priorities. This should probably be done with backup files of the current data on zglast-oracle03.
- This is more exploratory in nature. Once I'm able to decide on schema changes which are performant, I will write scripts to perform schema changes. Right now, I'm using the software Alembic and SQLAlchemy to aid in these migrations. One major benefit of Alembic is the database tables are defined in Python, and DBMS-specific SQL is automatically generated for every database system we plan to support.

Server Software Testing

First off, I Would like to expand to support three databases going forward:

- Oracle 11g, possibly 12c in the future
- PostgresSQL 9.2+ (and by proxy, commercial product EnterpriseDB)
- Sqlite 3.8.3+ (3.8.3 includes recursive with statement support)

Now, a bit about Unit Testing and Integration Testing.

Unit testing should only test the code. Certain unit tests can/should be done with a database. Currently we use Oracle, but it could be convenient if, in addition/instead of oracle, we used SQLite as it is file based and flexible. Right now, the majority of unit tests only test the creation and uploading of a new Task. Many unit tests which aren't related to Task uploading are really some sort of integration test. Overall testing coverage is pretty low, and needs to be worked on.

Integration testing should test how certain parts of the code will interact with a database, and effectively simulate how the server would operate. Integration testing is loosely related to Database migration testing as well.

Integration testing is a lot harder than unit testing. Ideally, we would have some virtual machines and have integration with jenkins which can bring up/tear down these multiple virtual machines with a well-defined environment. Another option is for Amazon, or some sort of VPS service, as they are well-targeted to create various different environments easily. Integration testing should not be performed every time new code is checked in, but possibly ran on a daily basis. There's a lot of urgency in this migration, but it's a very high priority for me to make sure that any changes I make to the server software for the migration DO NOT preclude potential support for Postgres/SQLite, and the best way I think I can ensure this is with an adequate integration testing environment, but I don't believe I will be able to set up all these testing infrastructures in a reasonable amount of time.

Caveats of these database systems:

SQLite has no row-locking or partitioning.

PostgresSQL doesn't have friendly/automatic partitioning.

Oracle works best with CONNECT BY vs recursive with statements.

SQLite is a bit more complicated to setup, as it requires a native library. The version we need to use is pretty new, and the java support currently requires building by hand (https://bitbucket.org/xerial/sqlite-jdbc), but I was able to perform this successfully.

Caveats of current systems:

Current Oracle installs do not use timestamp with time zone. Not a problem if the server is in the same time zone (PST) as the database, which is the case. There are no plans to change this.

Additional Definitions

Task: A node which includes other nodes (sub-Tasks) or leaf-nodes, which are of type Process

Process: A leaf node of a Task which defines an process which can be ran as either a Batch job, or internally to the pipeline server as a Jython scriptlet.

Stream: A stream is an execution instance node of a task.

ProcessInstance: An execution instance of a Process inside a Stream.

Stream Tree: The tree, with the root being a top level stream, of all streams include those in "dead" branches. The root of a dead branch is the first node where isLatest is not equal to 1. A node is a Stream

Stream Latest Tree: The tree, with the root being a top level stream, of all streams which are in the latest path.

Task Forest: All Stream Trees for a given root-level task.

Root Stream: The top level stream for a given child, grand child, etc... stream.