July 26, 2017

A Framework For Large-Scale Fault-Tolerant Data Pipelines

Long Nguyen

Zachary Frazier

A FRAMEWORK FOR LARGE-SCALE FAULT-TOLERANT DATA PIPELINES

ABSTRACT

A system and a method for frequent and fault-tolerant running of large-scale data pipelines across data centers are disclosed. The system includes one or more data centers, each having multiple servers, with a framework including a mutex design pattern for multi-binary data processing. The mutex design includes a minimal locking scheme where locks are acquired on the outputs of the stages on startup from a distributed locking service, and are released upon failure or successful writing of the output. A multi-homed pipeline may be utilized to make the system fault-tolerant. The method wraps each binary in the pipeline with locking on output data tables. The locking scheme guarantees that instances of the same binary will not interfere with each other. The disclosed system and method automatically coordinates pipelines in case of a failure or data delay to produce consistent and replicated output data tables on time.

BACKGROUND

Large-scale data pipelines, such as multi-binary pipelines, read multiple input data tables and produce multiple output data tables of large size (e.g. 10-100 TB) each day. Execution of these large-scale pipelines may fail in multiple ways. One such issue is the unpredictable availability of data input tables, which may cause failure and require a manual on-call investigation. Other issues include planned or unplanned data center failures, and job evictions in the cloud environment.

DESCRIPTION

A system and a method for frequent and fault-tolerant running of large-scale data pipelines are disclosed herein. The system as depicted in FIG. 1 includes data centers or clusters, each having multiple servers connected to a production environment with a framework including

a mutex design pattern implemented through a distributed locking service. The system aims to provide automatic data recovery by retrying mechanism as illustrated in FIG. 2 and FIG. 3. Stages illustrated in FIG. 3 may correspond to computing nodes in a given workflow. The mutex design may include a minimal locking scheme where locks are acquired on the outputs of the stages on startup from a distributed locking service, and are released upon failure or successful writing of the output. Thus, during the data processing step, if locks are revoked for any reason, the running process will stop, and exit. The system may also include extendable leases as an alternative to locking. For example, processes can stay alive and process data and extend the leases until data processing is complete. If it cannot extend the lease for any reason, the system may cancel the running process. A multi-homed pipeline may be utilized to make the pipelines fault-tolerant. The system works for pipelines of (1) single input single output, (2) single input multiple outputs, (3) multiple inputs and single input, and (4) multiple inputs multiple outputs. For instance, if there are two replicated pipelines running from two data centers, the pipelines may execute with the availability of replica with a safety guarantee of consistent data. The data pipeline may be described in a programming code such as C++ or Python, etc.

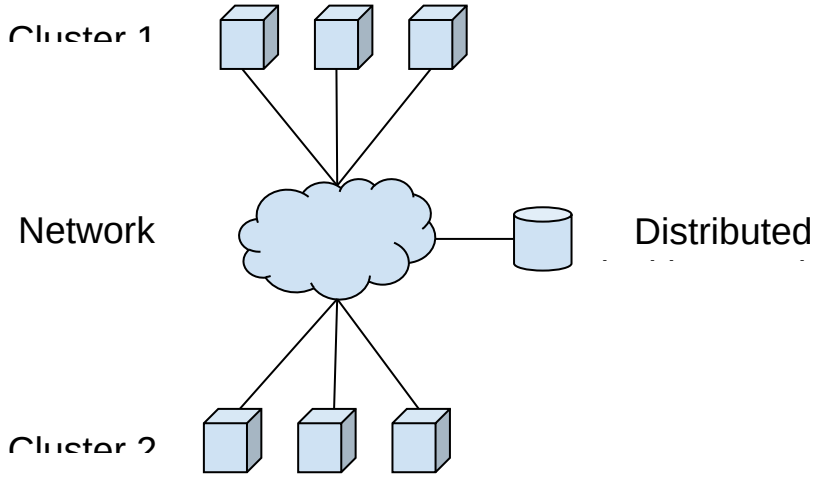

FIG. 1: A system for frequent and fault-tolerant running of large-scale data pipelines

The method is as illustrated in FIG. 2. Data pipelines contain many stages of data processing. The disclosed framework may wrap each and every stage of a given pipeline with the the following pattern of locking, aka MutexRun, as illustrated in FIG. 2. In particular, the framework may check for existence of input data tables as necessary conditions for executing binaries. Once input tables are available, it may obtain locks for output data tables in order to avoid possible deadlocks. Upon locks acquired (in sufficient conditions), and in the absence of output data table, it may execute the binary to produce output tables from input tables. During the execution of the binary, if any lock or lease is revoked, it may stop the binary, and exit. Upon completion of the output tables, it may release the locks, and move to the next stage in the pipeline.

Each stage may be scheduled to run automatically and periodically to decouple from the pipeline approach. The data pipelines may be automated by running them more frequently, e.g. every hour, instead of a daily run. For instance, a job scheduler may be scheduled to run each pipeline instance once per hour. Two job requests will not both attempt the same work due

to the locking mechanism. The pipeline stages do not have to be scheduled together as coordination is accomplished through retries and the locking mechanism.

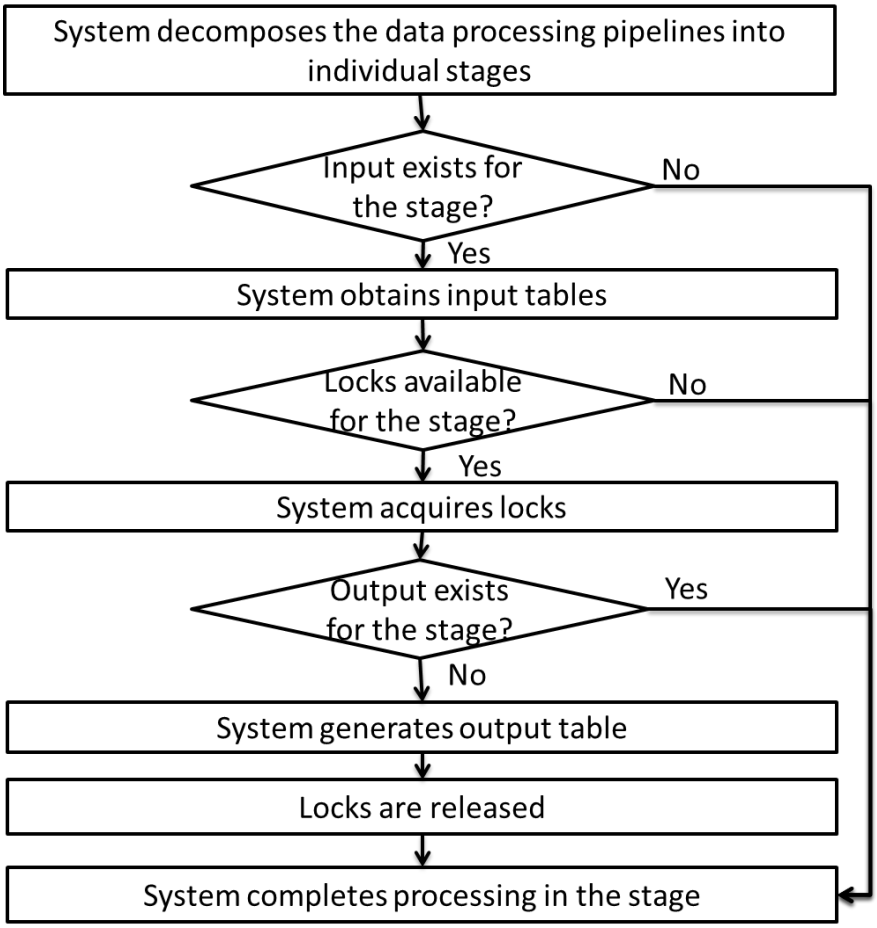

FIG. 2: Mutex design framework for frequent and fault-tolerant running of large-scale data pipelines

Further, FIG. 2 captures the MutexRun as shown in FIG. 3. In an example illustrated in FIG. 3, a linear data pipeline with 3 stages of distributed processes, which are scheduled to run one after another, is converted to a three-stage mutex chain A, B and C. Use of the disclosed method in a regularly scheduled pipeline allows handling of delays in data availability, such as logs data. It allows for any number of concurrent processes, such as a manually launched pipeline to run concurrently without interfering or corrupting the current running pipeline.

Furthermore, it allows for recovery from intermittent failures such that the next run of the pipeline will finish cleanly.

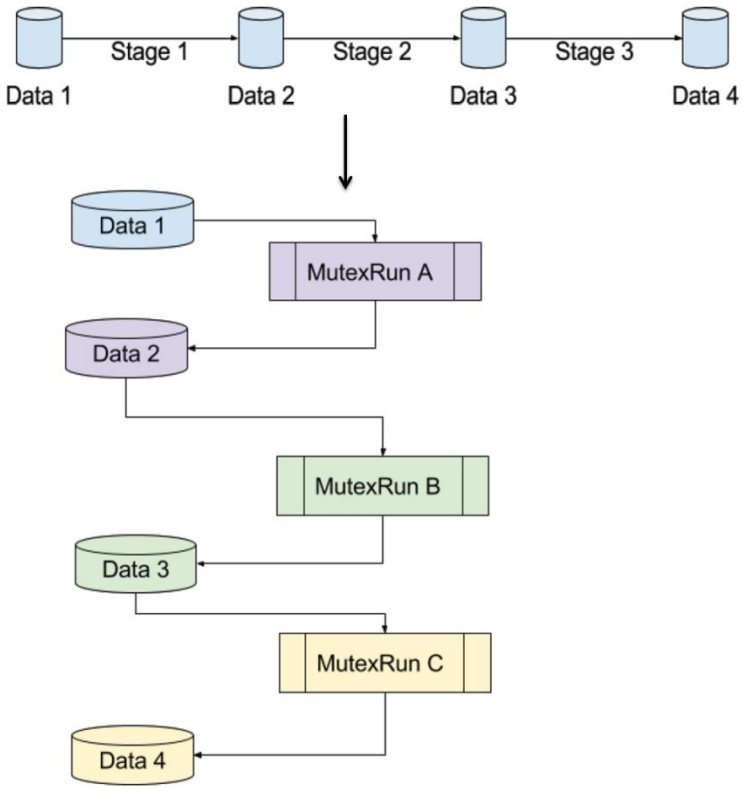

FIG. 3: A mutex run for a linear data pipeline with 3 stages of distributed process

In a second example illustrated in FIG. 4, the above mutex chain is applied for a multi-homed pipeline with 3 stages of distributed processes which are scheduled to run one after another and at each stage data may be copied from the other pipeline. Two log clusters may be utilized as data sources. Two data pipelines may be used from these sources for failure tolerance. There is no coordination between the two pipelines without a mutex chain setup and any failure or data delay requires manual investigation. Use of the disclosed system and method results in coordination between the pipelines which is automated completely. This is accomplished by treating the copy jobs as regular scheduled jobs that compete for the same output locks as the data generation pipelines. As illustrated in FIG. 4, there are two jobs competing for the

production of Data 2 with two replicas 2a and 2e, in the top pipeline: “Stage 1” in the top pipeline and “Stage 1 copy” which copies from second pipeline to first. In one example, Data 1 becomes available in second pipeline with suffix a (e.g. 1a, 2a, 3a) first allowing the copy command to provide Data 2 to the first cluster (2e). The output Data 2e in the first cluster will not produced by the pipeline even after Data 1 becomes available since the output Data 2e is already present and the lock is held by “Stage 1 copy” which is working to copy the data.

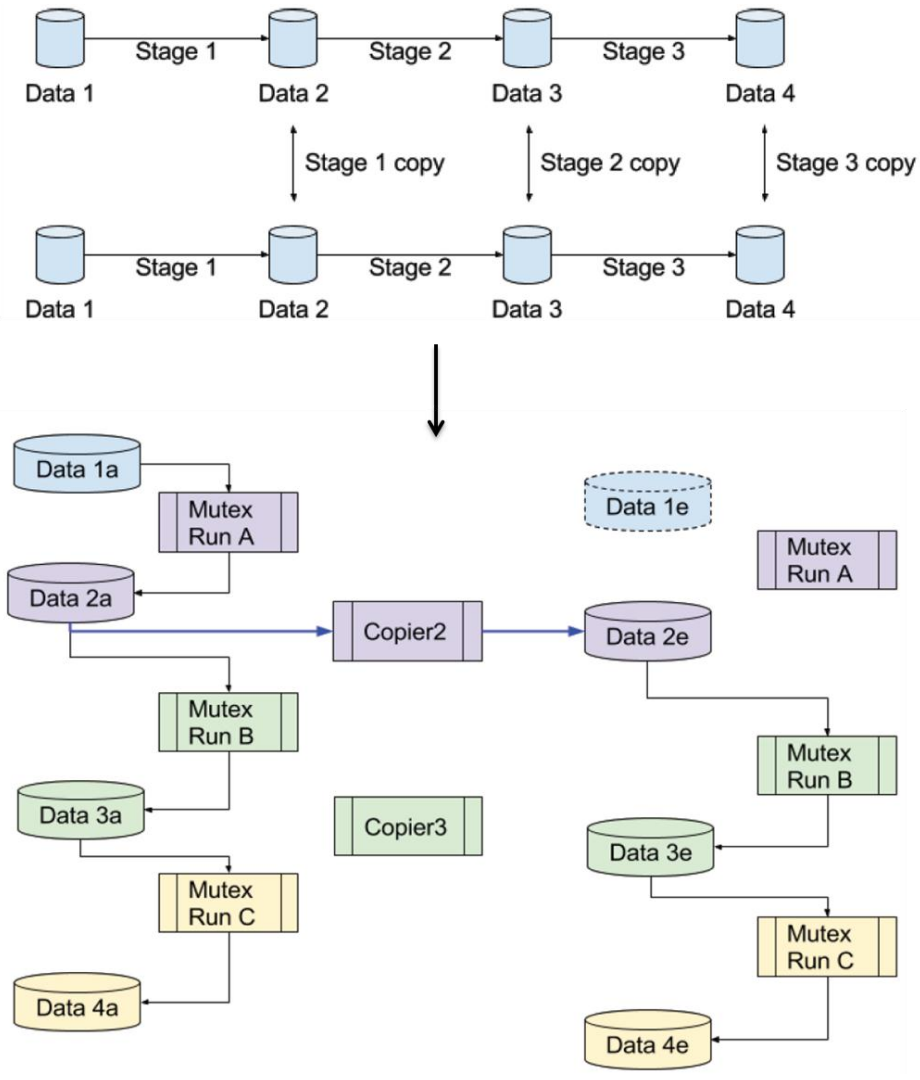

FIG. 4: A fault-tolerant mutex chain with copiers

The disclosed system and method use data pipelines to read multiple input tables

obtained from data sources, automatically coordinating the pipelines in case of a failure or data delay to produce multiple output data tables which are guaranteed to be consistent, and made available as soon as possible. The unpredictable availability of input tables is automated. The mutex design pattern works for all forms of data pipeline in single or multiple inputs and outputs. The method automates the large-data pipeline execution of multiple binaries, and optimizes for multi-home data pipelines.