[YARN-1187] Add discrete event-based simulation to yarn scheduler simulator

Andrew Chung, Subru Krishnan, Konstantinos Karanasos, Carlo Curino

Motivation

YARN-1021 introduced a scheduler load simulator (SLS) for YARN to facilitate evaluation of scheduler performance through simulations. The implementation of SLS uses a real-world clock to track progression of time, and while this solves the problem of having to deploy real, physical clusters in order to evaluate and predict the performance of schedulers, it limits users who would like to evaluate traces that are longer in duration (e.g., a month). This work item (YARN-1187) seeks to address the above limitation by proposing a simulator that is tied instead to a discrete, event-driven clock, enabling simulation of scheduling decisions on long traces in a short amount of time. A modified version of this discrete-event based simulator was used in evaluating the performance of the Wing scheduler.

Proposal

The high level proposal of this document is to create a wrapper around the core scheduler interface that allows the scheduler to feed its decisions to a mocked environment tied to a discrete event clock. This enables preservation of scheduling decisions without re-implementation of the scheduler logic solely for simulation purposes. Interfaces for trace-reading and application-specific AM logic will be extensible, with provided reference implementations.

Implementation details

The core of the discrete-event based simulator is the discrete event-driven clock. The event-based clock will need to be incorporated throughout the entire mocked environment, and minor changes to the scheduler API will need to be made to tie scheduler decisions to the event-based clock.

1. *The event-based clock*: The event-based clock is, naturally, tied to events that occur at various timestamps. The event-based clock is implemented as a priority queue that always dequeues the next event in absolute time. Events (in the form of lambdas) can be scheduled on the clock either as relative or as absolute time.
2. *Trace-reading*: A TraceReader interface reads in traces and schedules trace events on the event-based clock. Users will be able to implement their own TraceReader and their own application logic as needed. A reference TraceReader implementation that reads SLS formatted traces will be provided by default.
3. *The Clock and EventClock interface*: Hadoop components access the real world clock using calls to org.apache.hadoop.yarn.util.Clock.getTime(). To allow scheduling of events on the Clock, we introduce the EventClock interface, which when running in real-time uses a thread pool executor, but when running in discrete-event based simulation

is tied to the event-based clock. The implementation to use is determined through dependency injection at runtime.

4. *The mock environment*: The mock environment consists of the event-based clock, the Mock RM, Mock AMs, and Mock NMs. All components have access to the event-based clock. AM and NM heartbeats are also scheduled on the event-based clock. Introducing delays to AM and NM heartbeats allows simulation of network delays. Mock RM, AM, and NM implementations will borrow code and ideas from resourcemanager tests and the SLS package.
5. *Changes to the scheduler abstraction/interface*: Changes made to the scheduler abstraction/interface include substituting usage of calls to Thread.sleep() and calls to the Clock object with calls to the Clock and EventClock interfaces.

Limitations and future work

1. The implemented prototype currently does not support multi-threaded scheduling decisions.
2. Certain features, such as the reservation service and the Hadoop web dashboard, are not supported in the discrete event-based simulator.
3. Debug points and interactivity are not supported.

Note

The current implementation borrows code from both tests in the resourcemanager package and the SLS package to support the simulation of RM, AM, and NMs, and to support tracing in the schedulers.