OrgQueue: API-Based Scheduler Configuration Management

Motivation

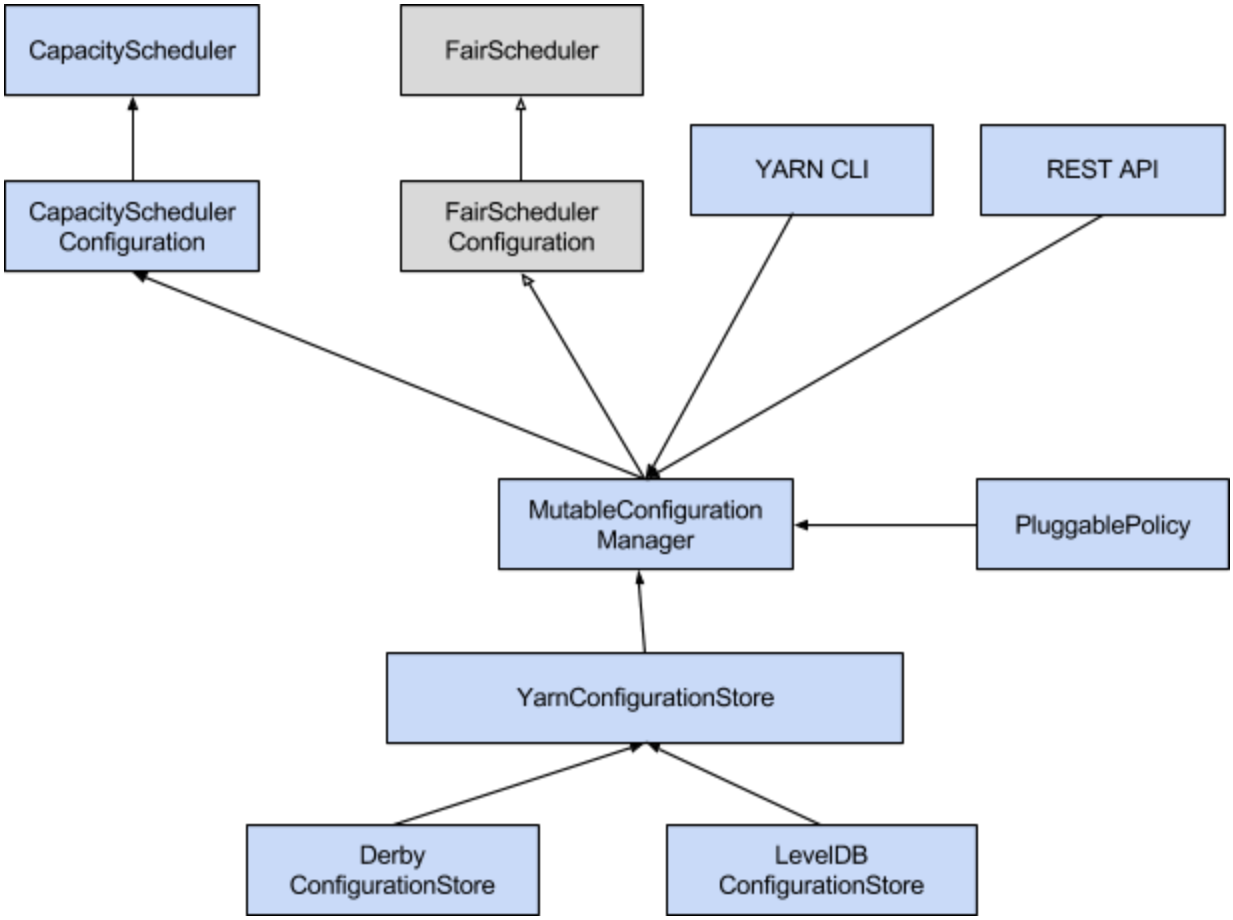
YARN's capacity and fair schedulers are currently configured using a file-based mechanism. To change the capacity scheduler's configuration, cluster administrators must manually edit this file, then refresh the queues via CLI. This is both cumbersome and error-prone.

The OrgQueue extension to the YARN scheduler provides a programmatic way to change configurations by providing a REST API that users can call to modify queue configurations. There are a few benefits here:

1. This enables automation of queue configuration management. We can check that the desired queue configuration changes are sane before reinitializing the capacity scheduler. This also enables use cases such as scheduled configuration changes without the need for cluster admin action.
2. API-based instead of file-based scheduler configuration management allows queue administrators to self-manage their own queues. This frees cluster operators from being involved in every scheduler configuration change request such as sub-queue creation, sub-queue capacity management, and acl changes.
3. Deleting queues right now is not natively supported - the configuration file must be modified in multiple places to delete a queue, which is error-prone.

To solve these issues, we propose an alternative configuration management mechanism, in which scheduler configurations are stored inside a configuration store instead of the scheduler-specific XML config file. We further provide a set of REST APIs for retrieving and changing the scheduler configurations.

Overall Design



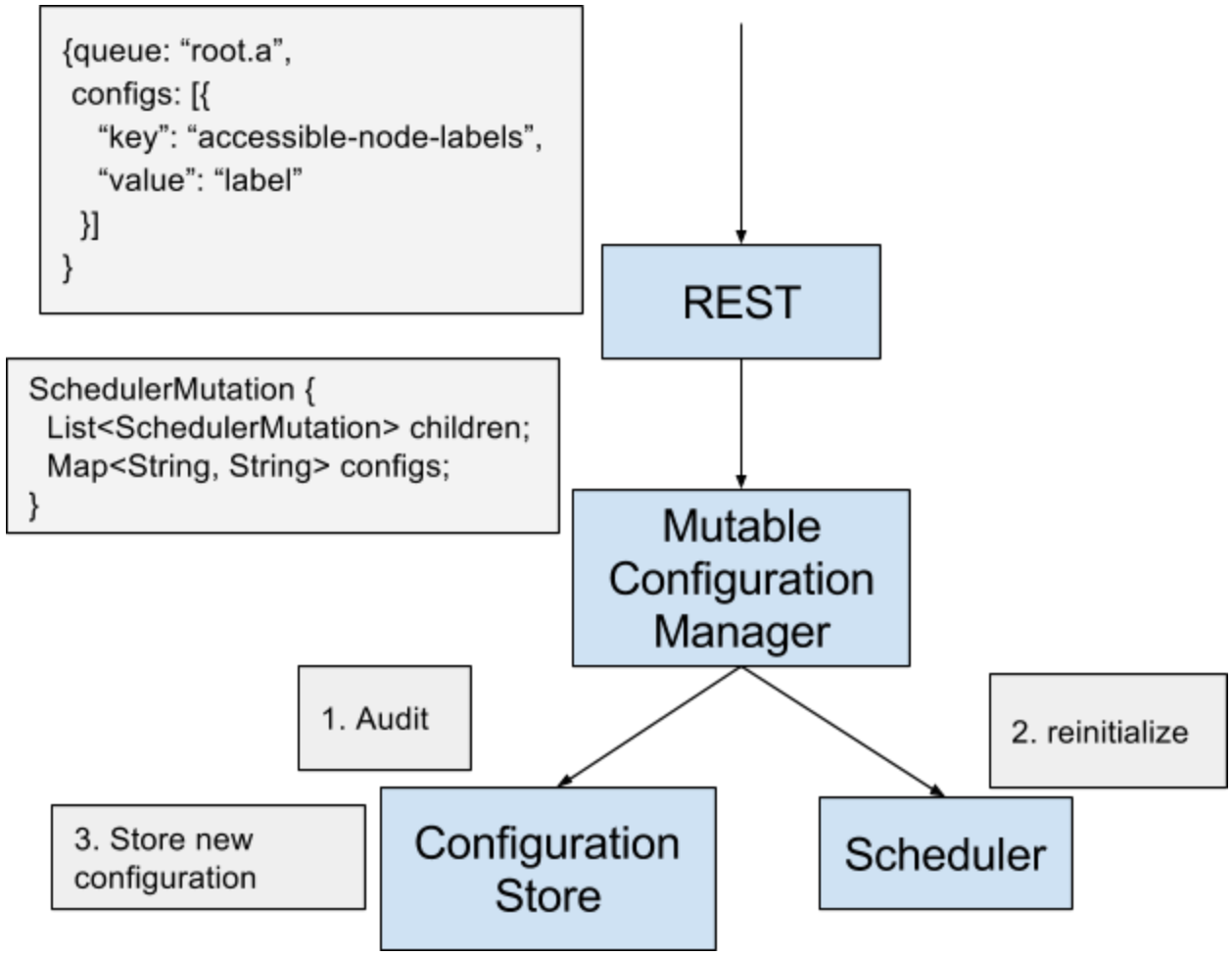
A few highlights for this design are listed below:

- The user provides a payload that describes a portion of the queue hierarchy and the included queues' configurations inside a JSON object. This allows users to specify multiple configuration changes with one REST call. The proposed configuration change is validated for authorization and correctness before committed to the config store. In addition, we allow specifying a queue's fully qualified name (root.qA.qB) so updating a queue local configuration does not require specifying the entire hierarchy of queues.
- A configuration store is proposed on the RM side to manage storing configurations in a non-file based store. This allows the configuration store and the RM state store to use separate backing implementations and keeps the application state store API and configuration store API separate.
- The MutableConfigurationManager component is proposed which takes care of the initial conversion of file based scheduler configuration to the config store based mechanism as well as handling scheduler configuration updates. This decouples backend configuration

store from the scheduler that is currently being used. This component will extend ConfigurationProvider as well, to provide Configuration object to the schedulers.

- A pluggable policy component handles queue configuration authorization. The initial policy will simply be based on queue ACLs, but it should be able to support more complicated policies (I want to give certain group the permission to kill applications in a queue but not really modifying queue configurations). We can leverage YarnAuthorizationProvider for this part.
- To delete a queue, the REST client can change the state of a queue to DELETED. The feature proposed in YARN-5724 will then take care of stopping new requests to the queue and eventually deleting the queue gracefully.

The typical flow from user issuing REST invocations to scheduler reinitializing itself to apply and store the updated configuration is demonstrated below:



- User supplied JSON is parsed into SchedulerMutation objects and given to MutableConfigurationManager
- MutableConfigurationManager handles authentication of configuration changes. It also acts as a ConfigurationProvider, providing the configuration contained in the store. When it receives a proposed configuration change, it first logs the change. Then it attempts to

reinitialize the scheduler with the updated configuration by exposing the new configuration to the scheduler.
    - If success, it returns success to the client, then it sends a flattened Map to YarnConfigurationStore containing the configuration updates, which maps queue path to a map of key/value pairs representing the list of configuration parameters to be updated for a set of queues.
    - If failure, it un-exposes the updated configuration, re-exposes the old configuration, and returns failure to client.
- YarnConfigurationStore stores queue configuration parameters as key/value pairs and handles updating the list of config changes atomically.

API

- REST
    - As referenced above, the REST API takes desired configuration changes and passes it to the mutable configuration manager. The mutable configuration manager will parse the object into a standard format to store in the YarnConfigurationStore as a key-value format.
        - Example for changing a set of configurations:

```
<add>
        <queue>root</queue>
        <add>
                <queue>a</queue>
        </add>
        </queue>
</add>
<add>
        <queue>root.b.b1</queue>
        <params>
                <entry>
                        <key>capacity</key>
                        <value>30</value>
                </entry>
        </params>
</add>
<remove>
        <queue>root</queue>
        <remove>
                <queue>c</queue>
        </remove>
</remove>
<remove>
        <queue>root.d</queue>
```

```
</remove>
<update>
        <queue>root.e</queue>
        <params>
                <entry>
                        <key>accessible-node-labels</key>
                        <value>label</value>
                </entry>
                <entry>
                        <key>user-limit-percent</key>
                        <value>5</value>
                </entry>
        </params>
        <update>
                <queue>e1</queue>
                <params>
                        <entry>
                                <key>user-limit-percent</key>
                                <value>10</value>
                        </entry>
                </params>
        </update>
</update>
```

This adds a child of root, “a”, and a child of root.b, “b1”, with capacity 30, removes the “c” child of root, removes the “d” child of root, updates root.e with accessible-node-labels “label” and user-limit-percent 5, and updates root.e.e1 with user-limit-percent 10.

Note that in the add, remove, and update cases, the queues can be passed with full path name or hierarchically.

Also note that if a child is added to a parent which does not exist, the whole hierarchy should be created.

- CLI
    - The user gives a list of operations to the CLI command grouped by type (add/remove/update).
        - Example: yarn schedconf --update queue=root.a,capacities=a3:10,a4:40;queue=root.a.a1,accessible-node-labels=test --add parent=root.a,queue=a5,capacity=30;queue=root.b.b2 --remove parent=root.b,queue=b1
        - This example updates root.a.a3’s capacity to 10, root.a.a4’s capacity to 40, as well as root.a.a1’s’ accessible node labels to “test”. Also it adds queue a5 under root.a with capacity 30, queue b2 under root.b with 0 capacity, and removes queue b1 from root.b. The configuration changes

within the same type are delimited by semicolons, and the parameters within a configuration change are delimited by commas.

Configuration Storage

- Use a YarnConfigurationStore (analogous to RMStateStore) to store configuration.
    - Extend this abstract class for different implementations of backing store. For example, we will provide a DerbyConfigurationStore which uses embedded Derby DB. This will be the default implementation (we have tested this version which has been running in production for over a year now). Other options are LevelDBConfigurationStore, etc.
- Implement a MutableConfigurationManager which is the interface between configuration clients (REST and schedulers) and the configuration store.
    - The MutableConfigurationManager should be able to initialize the YarnConfigurationStore with a user provided capacity-scheduler.xml. It parses the config file and insert the relevant content to the backend configuration store.
    - The MutableConfigurationManager should also be able to update the relevant part of parameters stored in YarnConfigurationStore according to the object handed over from the REST APIs.
    - The MutableConfigurationManager has a pluggable Policy mechanism which checks if a given configuration change by a certain user is allowed.

Queue State Management

The API-based scheduler configuration management (YARN-5734) is focused on removing the file-based configuration management system and supporting configuration changes via API for generic queue configurations. The focus of YARN-5724 is to support queue configuration and state consistency on queue addition and deletion. This also has a configuration management component, but since YARN-5734 is for general configuration changes, the objectives of YARN-5734 and YARN-5724 are orthogonal.

Audit Logging

The audit log serves two purposes; first as a write-ahead log for durability, second as a way to track what changes are made to the scheduler configurations at which time by whom. We create a new AuditLogger for scheduler configuration changes, and the audit entries are stored in the same backing store as the scheduler configuration. For bulk updates, the storage implementation takes care of logging the bulk update atomically. Then the scheduler is refreshed to attempt to apply the changes to validate before the configuration change itself is stored in the backing store.

If there is failover after the log is written and before the successful change is committed to the store, the logs that have not been applied to the store will be replayed on the configuration in

the store (e.g. by giving each log entry and the configuration store a version number, and applying the log entry iff its version number is greater than that of the configuration store). This ensures any successful in-memory configuration changes will still exist even if the RM restarts before the in-memory change is reflected in the store.

Backwards Compatibility

We still want to support file-based configuration management, as it exists now. We will have a configuration “yarn.scheduler.capacity.dynamic-queue-config.enabled” which is true if using store-based configuration management, false if using file-based. On startup, if using store-based configuration management and the scheduler configuration is not in the store, it will populate it with the file’s configuration (e.g. capacity-scheduler.xml). Else, it will load the configuration from the store.

If store-based scheduler configuration is enabled, “yarn rmadmin -refreshQueues” will be disabled since this causes the in-memory configuration to conflict with the configuration in the store. Also, if file-based scheduler configuration is enabled, the scheduler configuration REST API and CLI will be disabled.