Decoupling Container Allocation with Life-Cycle

Sunday, March 13, 2016
4:18 PM

Motivation :

- There is no way for an AM to restart a container without losing the allocation.
- Latency accrued from the AM having to re submit resource request for the same container.
- This can become prohibitive in heavily loaded clusters
- Container (and therefore Application) upgrades can get stalled

Proposal :

- Expose Allocation as a first class concept :
    - An "Allocation" is what is handed out by the Scheduling Authority (the Scheduler running on the RM) in response to a Resource Request.
    - An "Allocation" grants the requesting AM the right to request an NM to start 0 or more "Containers" on the target NM
    - The "Containers" that are started against an "Allocation" form a "Container Group"
    - An "Allocation" also establishes an upper limit to the resource consumed by a "Container Group"

New Entities and their Life Cycles:

- At the RM :
    - RMContainer will be renamed to RMAllocation.
        - Rename RUNNING to ACTIVE
        - Rename KILLED to FORCED_COMPLETED
- At the NM :
    - New Allocation class with lifecycle :
        - NEW
        - ACTIVATED : Ready to start containers (resources are available)
        - LOCALIZAING / LOCALIZED (Not sure if needed initialiiy)
        - AWAITING_CONTAINER_START : when a request to start a container has come.. Can switch to either ACTIVED (if more resources available after container start) or MAXED_OUT (no more resources after container start)
        - MAXED_OUT : No more containers can be started against this allocation
        - AWAITING_CONTAINER_STOP : when a requests to stop a container arrives
        - DEACTIVATED : AM can choose to deactivate an Allocation and kill all associated Containers.
        - COMPLETED
    - Special Event type an Allocation can Handle :
        - ACTIVATE
        - RESOURCE_CHANGE : increase or decrease a resource. A resource increase can result in a MAXED_OUT Allocation becoming ACTIVATED. A decrease can affect a state change in the opposite direction. Decrease will fail if sum of resources used by running container exceed target.
        - START_CONTAINER
        - STOP_CONTAINER
        - DEACTIVATE

Implementation Details:

- Changes at the RM :
  - RM hands out "AllocationTokens" which are similar to current "ContainerToken" and the "ContainerTokenIdentifier"
  - "ContainerId" becomes "AllocationId"
  - Node HeartBeat returns Allocation status (instead of ContainerStatus). Currently, RMNode cares only if it is RUNNING
  - ResourceTrackerService will be modified to accept AllocationStatus from Node Heartbeats.
  - RM/Scheduler Should be able to handle a Modified NodeStatusUpdate which contains Allocation Status (instead of Container Status)
    - The AllocationStatus contains the size and number of containers that are currently running against the Allocation
    - It can also perhaps contain the unused capacity of the Allocation.
    - The Scheduler/RM can considers reducing the Capacity of an Allocation rather than killing the entire Allocation when making pre-emption decisions.
    - Or it can keep track of explicit containers within an Allocation as explicitly ask the AM to to pre-empt a specific container.
- ApplicationMasterProtocol changes :
  - "AllocateRequest" sent by AM in the allocate() call will contain ResourceRequest and number of "Allocations" instead of number of "Containers". We can also have a backward compatible API which can be used by existing AMs that can convert from new to old request.
  - **Addition constraint** : An AM can receive only a single allocation on a Node, The Scheduler will "bundle" all Allocations on a Node for an app into a single Large Allocation.
  - Similarly, "AllocateResponse" will contain a list of "Allocations" instead of "Containers" which inturn holds an "AllocationToken"
- ContainerManagementProtocol changes :
  - New apis:
    - activateAllocation() : Takes a single ActivateAllocationRequest that contains an "AllocationToken". No need of supporting multiple allocations since a single Allocation can house multiple Containers.
    - deactivateAllocation() : Takes an Request containing an AllocationId and kills all containers running on the NodeManager against that allocation. The application still holds on to the allocation and is free to reactivate the Allocation and start more containers
    - completeAllocation() : Complete the allocation. Containers are killed and AM relinquishes the allocation.
    - modifyAllocation() : Used to increment / decrement resources associated with an Allocation.
  - Modified Apis:
    - startContainers() : The request should contain the "AllocationToken" (to check if the AM can start a container), and additionally a List of "ContainerLaunchContexts". Each Container is tagged with a "ContainerId" which is known only to the AM. The AllocationRequest can additionally contain a flag that states that the allocation is a **single-use** (which will be default, and which will be the case for existing AMs), in which case :
      - startContainers can activate the Allocation, if not already activated

        - assume AMs will have only 1 Container associated with an Allocation, in which case the containerId == allocationId (if not specified in the request).
    - stopContainers() : The request will contain containerIds of Containers to stop. If **single-use** allocation, then only 1 container will exist, which will be stopped and the Allocation itself will be COMPLETED.
- Changes at the NM :
    - New "AllocationManager" component which will now implement the ContainerManagementProtocol.
        - Allocations objects are created by the AllocationManager. Each Allocation is initialized with a stateMachine.
        - The AllocationManager will handle all API calls and will dispatch appropriate events to the appropriate Allocation object.
        - The transition from ACTIVATED to AWAITING_CONTAINER_START will call existing **startContainers** method on the ContainerManager.
        - The transition from ACTIVATED / MAXED_OUT to AWAITING_CONTAINER_START / DEACTIVATED will call existing **stopContainers** method on the ContainerManager.
        - Will send back "AllocationStatus" to the RM via the Node heartbeat. This will contain information pertaining to number (and size) of Containers within an Allocation.
    - Localization :
        - Activation of an allocation can also involve localizing of resources that can be shared by all the containers associated with the Allocation. This can reduce startup costs of additional containers.
        - This would require an additional "AllocationLocalizationContext", which if present in the activateAllocation() API, will result in the resources being localized.
        - Each ContainerStartRequest can also localize container specific resources but will have access to (read only copy?) of the Allocation Resources
        - (Allocation Localization will not be available initially since I forsee more extensive changes in the Localizer to handle the additional allocation scope)