Final project

From source code to design models

Hadoop Yarn Architecture

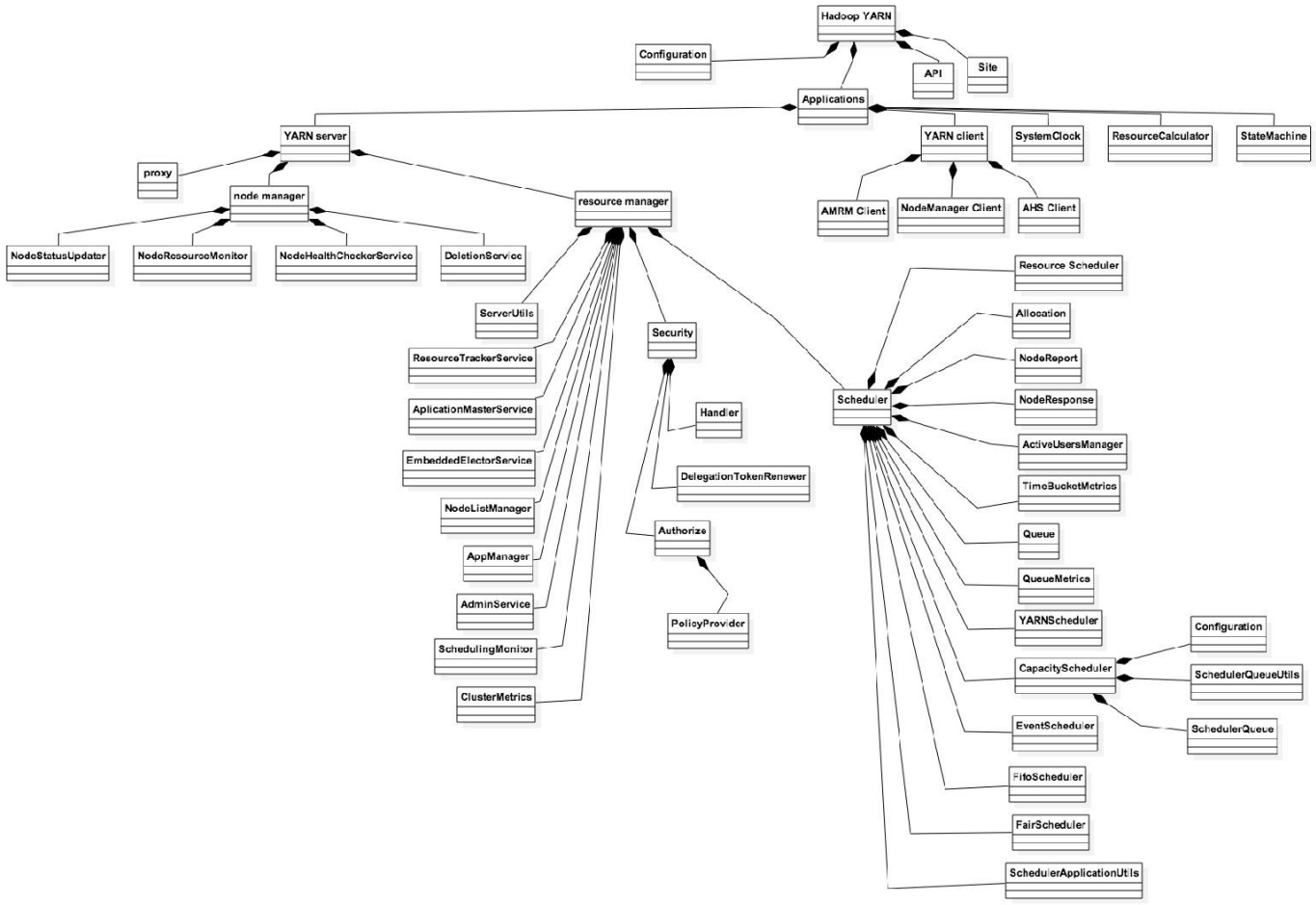
Software Modeling SWEN 705-01

Presented by:

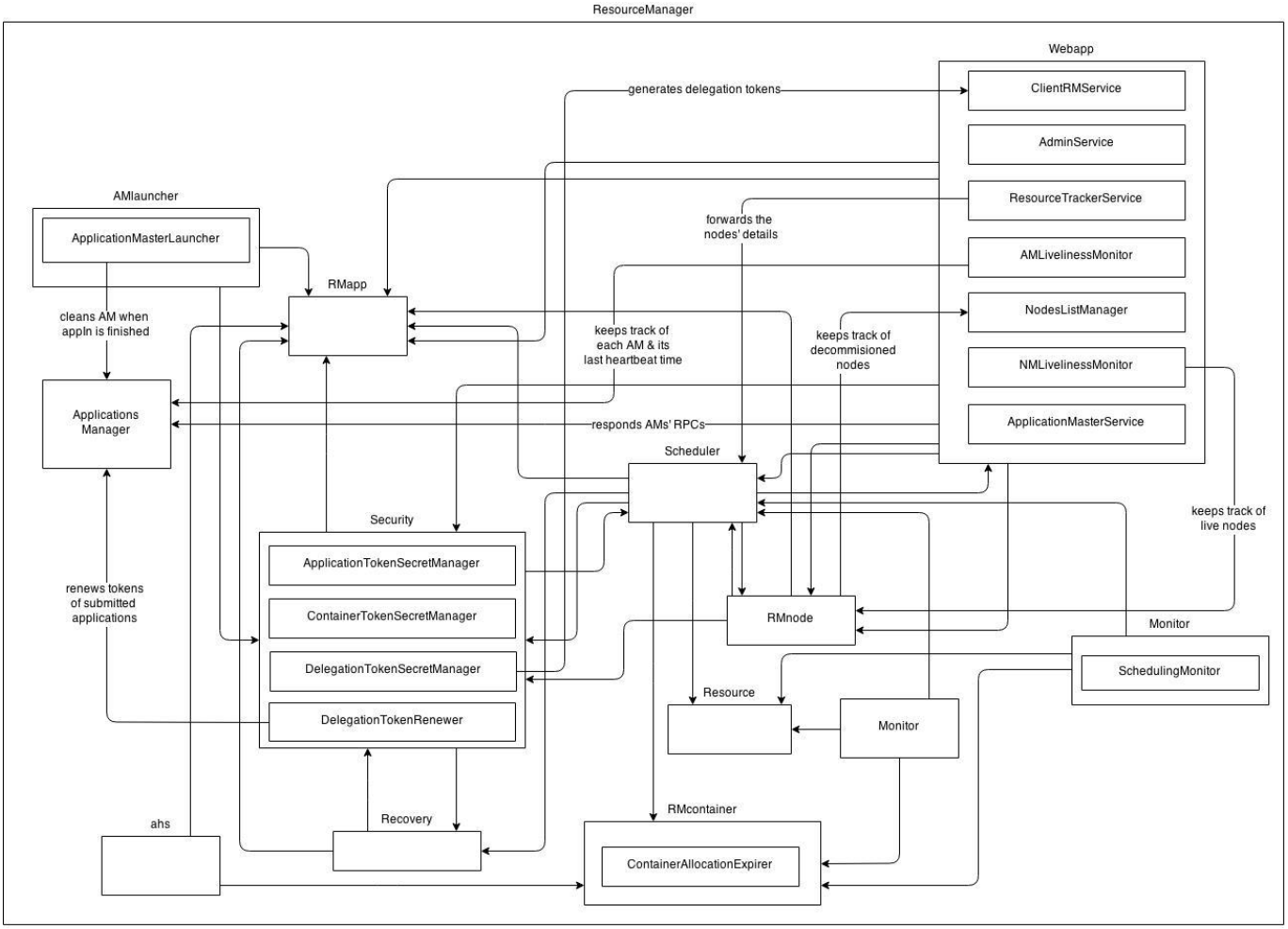
Ivan Tactuk
Omeya Jadhav
Yue Hua
Leonardo Matos

Module views of the system

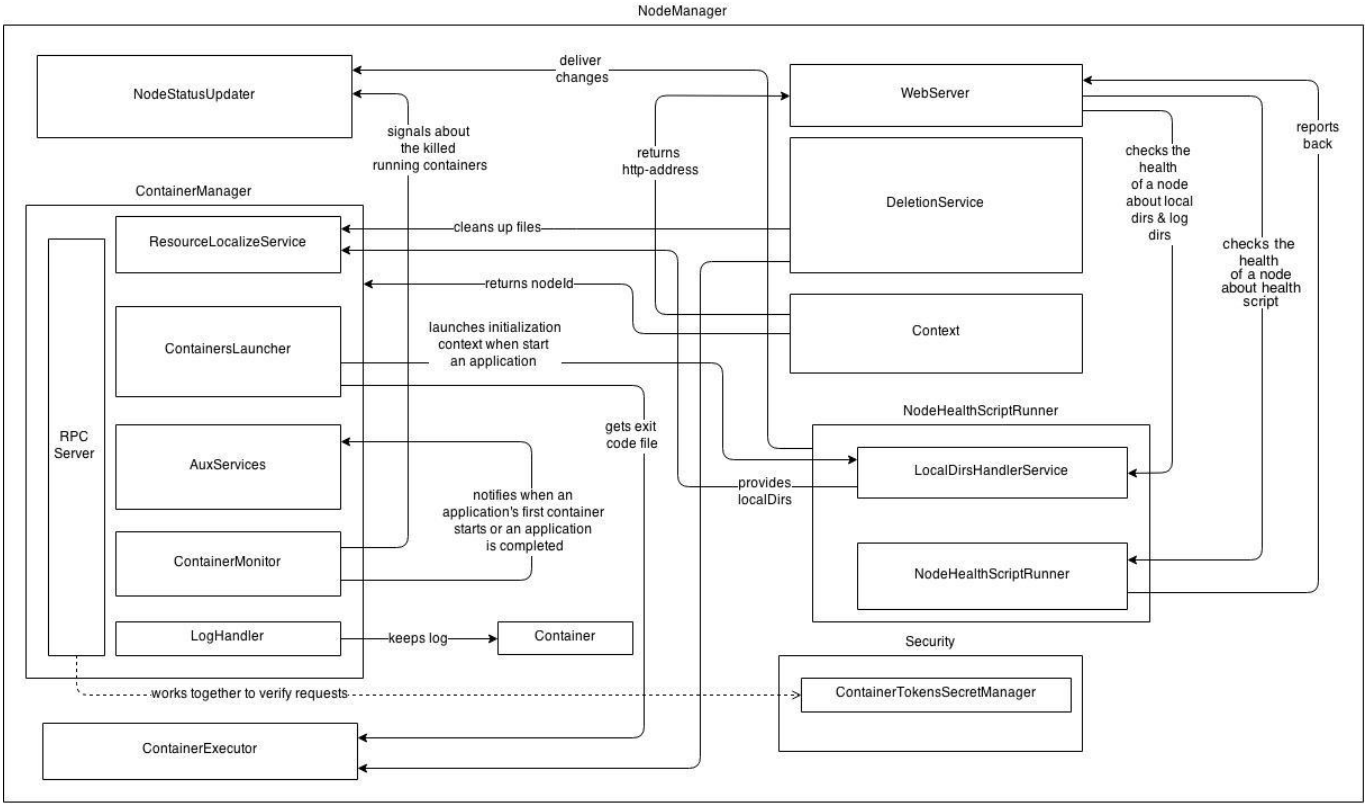
Main system model view



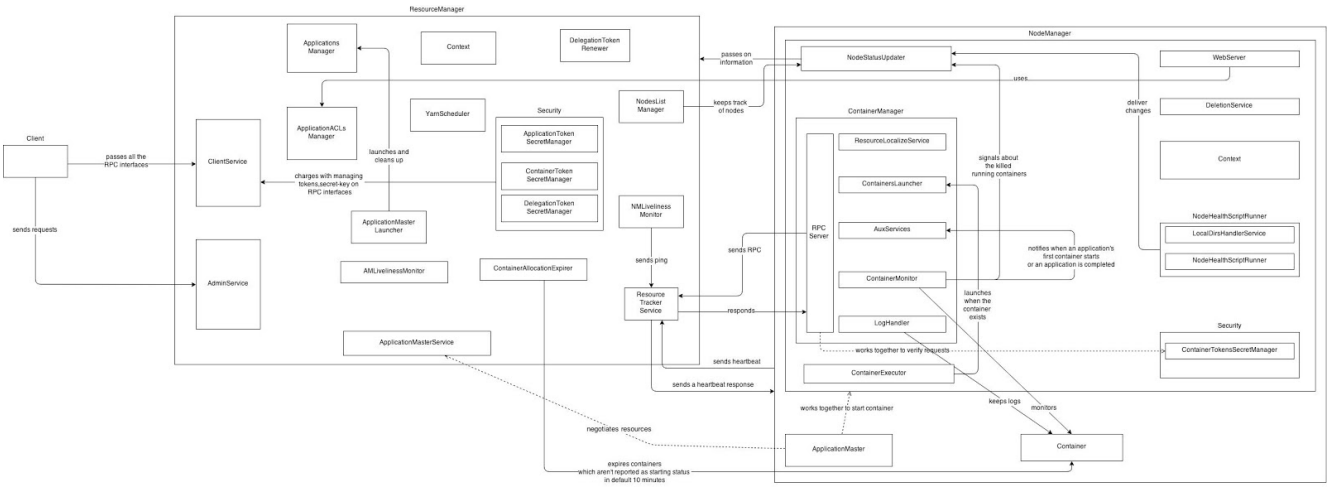
Resource Manager module view



Node Manager module view



Resource Manager and Node Manager module interaction view



Main Modules table

Component name	Belongs to package	Location of file
configuration	Hadoop yarn	/hadoop-yarn/conf
api	Hadoop yarn	/Hadoop_SourceCode/hadoop-yarn/hadoop-yarn-api
site	Hadoop yarn	Hadoop_SourceCode/hadoop-yarn/hadoop-yarn-site
applications	Hadoop yarn	/Hadoop_SourceCode/hadoop-yarn/hadoop-yarn-applications
Yarn server	applications	/Hadoop_SourceCode/hadoop-yarn/hadoop-yarn-server
proxy	Yarn server	/Hadoop_SourceCode/hadoop-yarn/hadoop-yarn-server
Node Manager	Yarn server	/Hadoop_SourceCode/hadoop-yarn/hadoop-yarn-server/hadoop-yarn-server-nodemanager
NodeStatusUpdater	Webapp	/Hadoop_SourceCode/hadoop-yarn/hadoop-yarn-server/hadoop-yarn-server-nodemanager/src/main/java/org/apache/hadoop/yarn/server/nodemanager/NodeStatusUpdater.java
NodeResourceMonitor	Webapp	/Hadoop_SourceCode/hadoop-yarn/hadoop-yarn-server/hadoop-yarn-server-nodemanager/src/main/java/org/apache/hadoop/yarn/server/nodemanager/NodeResourceMonitor.java
NodeHealthCheckerService	Webapp	/Hadoop_SourceCode/hadoop-yarn/hadoop-yarn-server/hadoop-yarn-server-nodemanager/src/main/java/org/apache/hadoop/yarn/server/nodemanager/NodeHealthCheckerService.java
DeletionService	Webapp	/hadoop-yarn/hadoop-yarn-server/hadoop-yarn-server-nodemanager/src/main/java/org/apache/hadoop/yarn/server/nodemanager/DeletionService.java
ResourceManager	Yarn server	/Hadoop_SourceCode/hadoop-yarn/hadoop-yarn-server/hadoop-yarn-server-resourcemanager/src/main/java/org/apache/hadoop/yarn/server/resourcemanager
ServerUtils	Webapp	/Hadoop_SourceCode/hadoop-yarn/hadoop-yarn-server/hadoop-yarn-server-resourcemanager/src/main/java/org/apache/hadoop/yarn/server/resourcemanager/RMServerUtils.java
ResourceTrackerService	Webapp	/Hadoop_SourceCode/hadoop-yarn/hadoop-yarn-server/hadoop-yarn-server-resourcemanager/src/main/java/org/apache/hadoop/yarn/server/resourcemanager/ResourceTrackerService.java
ApplicationMasterService	webapp	/Hadoop_SourceCode/hadoop-yarn/hadoop-yarn-server/hadoop-yarn-server-resourcemanager/src/main/java/org/apache/hadoop/yarn/server/resourcemanager/ApplicationMasterService.java
EmbeddedElectorService	webapp	/Hadoop_SourceCode/hadoop-yarn/hadoop-yarn-server/hadoop-yarn-server-resourcemanager/src/main/java/org/apache/hadoop/yarn/server/resourcemanager/EmbeddedElectorService.java
NodeListManager	webapp	/Hadoop_SourceCode/hadoop-yarn/hadoop-yarn-server/hadoop-yarn-server-resourcemanager/src/main/java/org/apache/hadoop/yarn/server/resourcemanager/NodesListManager.java
AppManager	webapp	/Hadoop_SourceCode/hadoop-yarn/hadoop-yarn-server/hadoop-yarn-server-resourcemanager/src/main/java/org/apache/hadoop/yarn/server/resourcemanager/RMAppManager.java

AdminService	webapp	/Hadoop_SourceCode/hadoop-yarn/hadoop-yarn-server/hadoop-yarn-server-resourcemanager/src/main/java/org/apache/hadoop/yarn/server/resourcemanager/AdminService.java
SchedulingMonitor	monitor	/Hadoop_SourceCode/hadoop-yarn/hadoop-yarn-server/hadoop-yarn-server-resourcemanager/src/main/java/org/apache/hadoop/yarn/server/resourcemanager/monitor/SchedulingMonitor.java
ClusterMetrics	webapp	/Hadoop_SourceCode/hadoop-yarn/hadoop-yarn-server/hadoop-yarn-server-resourcemanager/src/main/java/org/apache/hadoop/yarn/server/resourcemanager/ClusterMetrics.java
Security	resource manager	/Hadoop_SourceCode/hadoop-yarn/hadoop-yarn-server/hadoop-yarn-server-resourcemanager/src/test/java/org/apache/hadoop/yarn/server/resourcemanager/security
Handler	Security	/Hadoop_SourceCode/hadoop-yarn/hadoop-yarn-server/hadoop-yarn-server-resourcemanager/src/main/java/org/apache/hadoop/yarn/server/resourcemanager/security/RMAuthenticationHandler.java
DelegationTokenRenewer	Security	/Hadoop_SourceCode/hadoop-yarn/hadoop-yarn-server/hadoop-yarn-server-resourcemanager/src/main/java/org/apache/hadoop/yarn/server/resourcemanager/security/DelegationTokenRenewer.java
Authorize	Security	/Hadoop_SourceCode/hadoop-yarn/hadoop-yarn-server/hadoop-yarn-server-resourcemanager/src/main/java/org/apache/hadoop/yarn/server/resourcemanager/security/authorize
Policyprovider	Authorize	/Hadoop_SourceCode/hadoop-yarn/hadoop-yarn-server/hadoop-yarn-server-resourcemanager/src/main/java/org/apache/hadoop/yarn/server/resourcemanager/security/authorize/RMPolicyProvider.java
Schedular	resource manager	/Hadoop_SourceCode/hadoop-yarn/hadoop-yarn-server/hadoop-yarn-server-resourcemanager/src/main/java/org/apache/hadoop/yarn/server/resourcemanager/scheduler
ResourceScheduler	Schedular	/Hadoop_SourceCode/hadoop-yarn/hadoop-yarn-server/hadoop-yarn-server-resourcemanager/src/main/java/org/apache/hadoop/yarn/server/resourcemanager/scheduler/ResourceScheduler.java
Allocation	Schedular	/Hadoop_SourceCode/hadoop-yarn/hadoop-yarn-server/hadoop-yarn-server-resourcemanager/src/main/java/org/apache/hadoop/yarn/server/resourcemanager/scheduler/Allocation.java
NodeReport	Schedular	/Hadoop_SourceCode/hadoop-yarn/hadoop-yarn-server/hadoop-yarn-server-resourcemanager/src/main/java/org/apache/hadoop/yarn/server/resourcemanager/scheduler/NodeReport.java
NodeResponse	Schedular	/Hadoop_SourceCode/hadoop-yarn/hadoop-yarn-server/hadoop-yarn-server-resourcemanager/src/main/java/org/apache/hadoop/yarn/server/resourcemanager/scheduler/NodeResponse.java
ActiveUserManager	Schedular	/Hadoop_SourceCode/hadoop-yarn/hadoop-yarn-server/hadoop-yarn-server-resourcemanager/src/main/java/org/apache/hadoop/yarn/server/resourcemanager/scheduler/ActiveUsersManager.java
TimeBuscketMetrics	Schedular	/Hadoop_SourceCode/hadoop-yarn/hadoop-yarn-server/hadoop-yarn-server-resourcemanager/src/main/java/org/apache/hadoop/yarn/server/resourcemanager/scheduler/TimeBucketMetrics.java
Queue	Schedular	/Hadoop_SourceCode/hadoop-yarn/hadoop-yarn-server/hadoop-yarn-server-resourcemanager/src/main/java/org/apache/hadoop/yarn/server/resourcemanager/scheduler/Queue.java

QueueMetrics	Schedular	/Hadoop_SourceCode/hadoop-yarn/hadoop-yarn-server/hadoop-yarn-server-resourcemanager/src/main/java/org/apache/hadoop/yarn/server/resourcemanager/scheduler/QueueMetrics.java
YarnSchedular	Schedular	/Hadoop_SourceCode/hadoop-yarn/hadoop-yarn-server/hadoop-yarn-server-resourcemanager/src/main/java/org/apache/hadoop/yarn/server/resourcemanager/scheduler/YarnScheduler.java
CapacitySchedular	Schedular	/Hadoop_SourceCode/hadoop-yarn/hadoop-yarn-server/hadoop-yarn-server-resourcemanager/src/main/java/org/apache/hadoop/yarn/server/resourcemanager/scheduler/capacity/CapacityScheduler.java
SchedularQueueUtils	Capacity	/Hadoop_SourceCode/hadoop-yarn/hadoop-yarn-server/hadoop-yarn-server-resourcemanager/src/main/java/org/apache/hadoop/yarn/server/resourcemanager/scheduler/capacity/CSQueueUtils.java
Configuration	Capacity	/Hadoop_SourceCode/hadoop-yarn/hadoop-yarn-server/hadoop-yarn-server-resourcemanager/src/main/java/org/apache/hadoop/yarn/server/resourcemanager/scheduler/capacity/CapacitySchedulerConfiguration.java
Queue	Capacity	/Hadoop_SourceCode/hadoop-yarn/hadoop-yarn-server/hadoop-yarn-server-resourcemanager/src/main/java/org/apache/hadoop/yarn/server/resourcemanager/scheduler/capacity/CSQueue.java
EventSchedular	Event	/Hadoop_SourceCode/hadoop-yarn/hadoop-yarn-server/hadoop-yarn-server-resourcemanager/src/main/java/org/apache/hadoop/yarn/server/resourcemanager/scheduler/event/SchedulerEvent.java
SchedularApplicationutils	fifo	/Hadoop_SourceCode/hadoop-yarn/hadoop-yarn-server/hadoop-yarn-server-resourcemanager/src/main/java/org/apache/hadoop/yarn/server/resourcemanager/scheduler/SchedulerAppUtils.java
FifoSchedular	fifo	/Hadoop_SourceCode/hadoop-yarn/hadoop-yarn-server/hadoop-yarn-server-resourcemanager/src/main/java/org/apache/hadoop/yarn/server/resourcemanager/scheduler/fifo/FifoScheduler.java
FairSchedular	Fair	/Hadoop_SourceCode/hadoop-yarn/hadoop-yarn-server/hadoop-yarn-server-resourcemanager/src/main/java/org/apache/hadoop/yarn/server/resourcemanager/scheduler/fair/FairScheduler.java

II) OCL and test cases:

1. **The next heart beat interval value should be a valid positive number.**

Context ResourceTrackerService::serviceInit()
Inv: nextHeartBeatInterval > 0

Test Case 1.1:

```
nextHeartBeatInterval=-1
try
{
    ResourceTrackerService.serviceInit()
  Assert.Fail()
}
catch(YarnRuntimeException)
{

}
```

Test Case 1.2:

```
nextHeartBeatInterval=0
try
{
    ResourceTrackerService.serviceInit()
  Assert.Fail()
}
catch(YarnRuntimeException)
{

}
```

Test Case 1.3:

```
nextHeartBeatInterval=2
try
{
    ResourceTrackerService.serviceInit()
}
catch(YarnRuntimeException)
{
    Assert.Fail()
}
```

1. **The Minimum allocation memory should be lower or equal than the maximum**

   Context FairScheduler::validateConf()
   Inv: RM_SCHEDULER_MINIMUM_ALLOCATION_MEMORY <=
   RM_SCHEDULER_MAX_ALLOCATION_MEMORY

Test Case 2.1:

```
RM_SCHEDULER_MINIMUM_ALLOCATION_MEMORY=2
RM_SCHEDULER_MAX_ALLOCATION_MEMORY=55
try
{
    FairScheduler.validateConf()
}
catch(YarnRuntimeException)
{
    Assert.Fail()
}
```

Test Case 2.2:

```
RM_SCHEDULER_MINIMUM_ALLOCATION_MEMORY =55
RM_SCHEDULER_MAX_ALLOCATION_MEMORY=22
try
{
    FairScheduler.validateConf()
    Assert.Fail()
}
```

```
catch(YarnRuntimeException)
{
}
```

Test Case 2.3:

```
RM_SCHEDULER_MINIMUM_ALLOCATION_MEMORY =55
RM_SCHEDULER_MAX_ALLOCATION_MEMORY=55
try
{
    FairScheduler.validateConf()
}
catch(YarnRuntimeException)
{
    Assert.Fail()
}
```

1. **The Resource scheduler minimum allocation memory should be a whole number value.**

Context FairScheduler::validateConf()
Inv: RM_SCHEDULER_MINIMUM_ALLOCATION_MEMORY >= 0

Test Case 3.1:

```
RM_SCHEDULER_MINIMUM_ALLOCATION_MEMORY =-1
try
{
    FairScheduler.validateConf()
     Assert.Fail()
}
catch(YarnRuntimeException)
{
}
```

Test Case 3.2:

```
RM_SCHEDULER_MINIMUM_ALLOCATION_MEMORY =0
try
{
    FairScheduler.validateConf()
}
catch(YarnRuntimeException)
{
    Assert.Fail()
}
```

Test Case 3.3:

```
RM_SCHEDULER_MINIMUM_ALLOCATION_MEMORY =2
try
{
    FairScheduler.validateConf()
}
```

```
catch(YarnRuntimeException)
{
    Assert.Fail()
}
```

1. **The expire interval of the resource Manager should be equal or larger than the heart beat interval.**

    Context ResourceManager::validateConfigs()
    Inv: expireIntv1 >= heartbeatIntv1

Test Case 4.1:

```
heartbeatIntv1 = 2
expireIntv1 = 55
try
{
    ResourceManager.validateConfigs()
}
catch(YarnRuntimeException)
{
    Assert.Fail()
}
```

Test Case 4.2:

```
heartbeatIntv1=55
expireIntv1 =22
try
{
    ResourceManager.validateConfigs()
    Assert.Fail()
}
catch(YarnRuntimeException)
{
}
```

Test Case 4.3:

```
heartbeatIntv1=55
expireIntv1 =55
try
{
    ResourceManager.validateConfigs()
}
catch(YarnRuntimeException)
{
    Assert.Fail()
}
```

1. **The global Max attempts of the resource manager should be a positive number.**

    Context ResourceManager::validateConfigs()

Inv: globalMaxAppAttempts>0

Test Case 5.1:

```
globalMaxAppAttempts=-1
try
{
    ResourceManager.validateConfigs()
     Assert.Fail()
}
catch(YarnRuntimeException)
{
}
```

Test Case 5.2:

```
globalMaxAppAttempts=0
try
{
    ResourceManager.validateConfigs()
    Assert.Fail()
}
catch(YarnRuntimeException)
{
}
```

Test Case 5.3:

```
globalMaxAppAttempts=2
try
{
    ResourceManager.validateConfigs()
}
catch(YarnRuntimeException)
{
    Assert.Fail()
}
```

1. A valid container token should be used for correct node.

Context NMContainerTokenSecretManager::retrievePassword()
Inv: identifier.getNmHostAddress() == nodeHostAddr

Test Case 6.1:

```
identifier.setNmHostAddress("197.0.0.1");
nodeHostAddr = "197.0.0.1"
try
{
    NMContainerTokenSecretManager.retrievePassword()
}
catch(SecretManager.InvalidToken)
{
```

```
    Assert.Fail()
}
```

Test Case 6.2:

```
identifier.setNmHostAddress("197.0.0.1");
nodeHostAddr = "192.168.1.1"
try
{
    NMContainerTokenSecretManager.retrievePassword()
    Assert.Fail()
}
catch(SecretManager.InvalidToken)
{
}
```

1. When moving an Application to a Queue leaf, NumRunnableApps cannot be greater than maxRunningApps or maxShare

Context FairScheduler::moveApplication(ApplicationId appId, String queueName)
Inv: queueMgr.getLeafQueue(queueName).getNumRunnableApps() < maxRunningApps

Test Case 7.1:

```
maxRunningApps=3
try
{
    moveApplication(new Application(id=1),queueName="RIT")
     moveApplication(new Application(id=2),queueName="RIT")
     moveApplication(new Application(id=3),queueName="RIT")
     moveApplication(new Application(id=4),queueName="RIT")
    Assert.Fail()
}
catch(YarnException)
{
}
```

Test Case 7.2:

```
maxRunningApps=3
try
{
    moveApplication(new Application(id=1),queueName="RIT")
     moveApplication(new Application(id=2),queueName="RIT")
     moveApplication(new Application(id=3),queueName="RIT")
}
catch(YarnException)
{
    Assert.Fail()
}
```

1. **CapacityScheduler cannot have two leaf queues with the same name.**

Context CapacityScheduler
Inv: Leafqueues.isUnique(Name)

Test Case 8.1:

```
Leafqueues.add(name=”A1”)
Leafqueues.add(name=”A1”)
try
{
    CapacityScheduler.parseQueue()
    Assert.Fail()
}
catch(IOException)
{
}
```

Test Case 8.2:

```
Leafqueues.add(name=”A1”)
Leafqueues.add(name=”A2”)
try
{
    CapacityScheduler.parseQueue()
}
catch(IOException)
{
    Assert.Fail()
}
```

2. Verify that the user is authorized to perform an action in the Node Manager.

Context ContainerManagerImpl::authorizeUser()
Inv: remoteUgi.getUserName() = nmTokenIdentifier.getApplicationAttemptId()

Test Case 9.1:

```
UserGroupInformation rUgi = new UserGroupInformation()
NMTokenIdentifier nmTokenId = new NMTokenIdentifier()
rUgi.setUserName(“RIT”)
nmTokenId.setApplicationAttemptId(“RIT”)
try
{
    CapacityScheduler.parseQueue()
}
catch(RPCUtil.getRemoteException)
{
    Assert.Fail()
}
```

Test Case 9.2:

```
UserGroupInformation rUgi = new UserGroupInformation()
```

```
NMTokenIdentifier nmTokenId = new NMTokenIdentifier()
rUgi.setUserName("MIT")
nmTokenId.setApplicationAttemptId("RIT")
try
{
    CapacityScheduler.parseQueue()
    Assert.Fail()
}
catch(RPCUtil.getRemoteException)
{
}
```

10. Do not support Node Manager recovery with an ephemeral server port.

Context ContainerManagerImpl::serviceStart()
Inv: initialAddress.getPort()!=0 or !context.getNMStateStore().canRecover()

Test Case 10.1:

```
initialAddress.setPort(0)
context.getNMStateStore().setCanRecover(True)
try
{
    ContainerManagerImpl.serviceStart()
    Assert.Fail()
}
catch(IllegalArgumentException)
{

}
```

Test Case 10.2:

```
initialAddress.setPort(8180)
context.getNMStateStore().setCanRecover(True)
try
{
    ContainerManagerImpl.serviceStart()
}
catch(IllegalArgumentException)
{
    Assert.Fail()
}
```

Test Case 10.3:

```
initialAddress.setPort(0)
context.getNMStateStore().setCanRecover(False)
try
{
    ContainerManagerImpl.serviceStart()
}
```

```
catch(IllegalArgumentException)
{
    Assert.Fail()
}
```

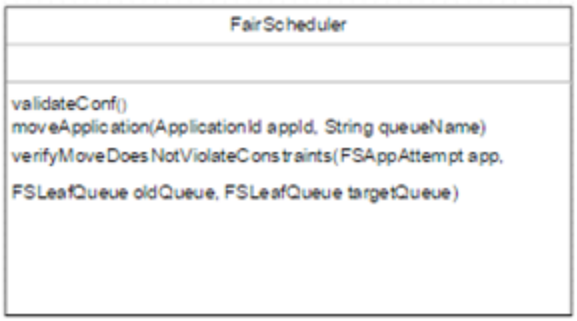
Test Case 10.4:

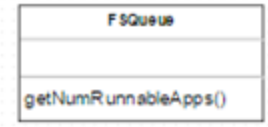
```
initialAddress.setPort(8180)
context.getNMStateStore().setCanRecover(False)
try
{
    ContainerManagerImpl.serviceStart()
}
catch(IllegalArgumentException)
{
    Assert.Fail()
}
```

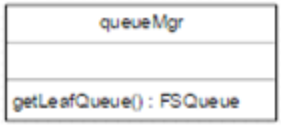
Design models that can be related to OCL

OCL 1:

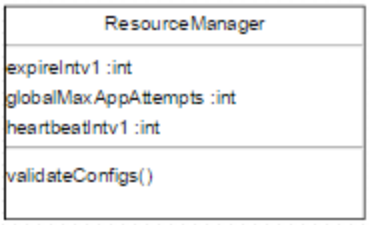
OCL 2, 3, 7:



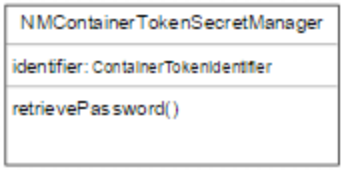




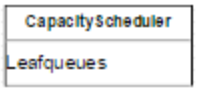
OCL 4, 5:



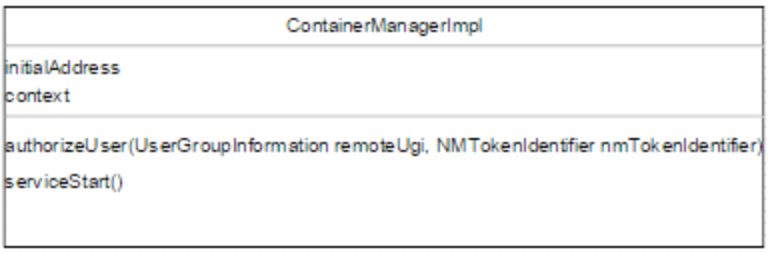
OCL 6:



OCL 8:



OCL 9, 10:



The Methodology

To generate test cases from OCL we follow the following procedure:

1. Identify the context (and the files involved) of the OCL
2. Find all the possible different combinations of the variables used in the invariance, preconditions or postconditions. For each of these combinations, one test case has to be generated.
3. For each test case, assign values to the variables between the ranges previously selected.
4. Figure out what is the expected value of the function being tested, given the specific circumstances. If it is not the expected result, raise an assert fail exception. In case that the expected result of a function is to raise an exception, this specific exception have to be caught, so the assert exception can behave as desired in the test case.

Appendix:

Hadoop YARN architecture recovery process:

For recovering this architecture we followed a top bottom approach.

At the beginning, we were not completely sure if heartbeat was implemented in Hadoop. We thought that it could be a ping echo or a heartbeat and if it was heartbeat we were not sure about the direction in which the heartbeat was sent. To know what actually was happening, we decided to search in the Hadoop issuer tracker and to look up in the code.

We got to the conclusion that they implemented an improved heartbeat in the ResourceTrackerService class. The heartbeat is sent from NM to RM and a response heartbeat message containing the next Interval heartbeat was sent from RM to NM. Even though there are exchanges of messages in both directions, we believe that this should be considered an improved heartbeat instead of a ping echo, because of the intentions. In this case, the Node Manager sends heartbeat to the Resource Manager to let him know that he is still alive and the response message is just to make more efficient the heartbeat by controlling the heartbeat interval. If it were ping echo, the Node Manager would have been using the heartbeat response just to make sure that the Resource manager received its heartbeat. Below we can see links and images that support this hypothesis.

http://grokbase.com/t/hadoop/yarn-issues/12chd3fmgm/jira-created-yarn-275-make-nodemanagers-to-not-blindly-heartbeat-irrespective-of-whether-previous-heartbeat-is-processed-or-not

Bikas Saha (JIRA) at **Dec 28, 2012 at 5:48 pm**

[https://issues.apache.org/jira/browse/YARN-275?page=com.atlassian.jira.plugin.system.issuetabpanels:comment-tabpanel&focusedCommentId=13540621#comment-13540621]

Bikas Saha commented on YARN-275:

I briefly looked at the patch. The general approach seems promising. I have some comments on how we can structure this changes
We could break this work into 2 parts
1) protocol changes in heartbeat to transfer heartbeat control frequency from NM to RM. After this, in every heartbeat the RM will tell the NM when to send the next heartbeat. That value can be hardcoded (like it is currently) but preferably we can have an RM config that defines what the minimum heartbeat interval should be and use that. For this part, I dont think we need both backoff and heartbeatinterval in the heartbeat response. We can just have only heartbeatinterval that is always respected by the NM.
2) add some logic/heuristic to the RM so that it can dynamically change the heartbeat interval based on its current processing load/rate. This way the interval can be made longer when the RM is not keeping up with heartbeats.
If you think this break-up of works makes sense then we can create 2 sub-tasks under this jira for the 2 parts.

I have some additional ideas on part 1 also.
When a heartbeat comes at time T to the RM then it can choose to
A) accept the request at time T and ask NM to heartbeat after time T+K with new information. This adds more load to the current RM load. This is what the current code does. So no change is required to do this.
B) reject the request at time T and ask NM to heartbeat after time T+K with current+new information. This does not increase load on RM but makes NM more complex because it needs to hold onto the last heartbeat data and merge in new data to it.
What do you think about these alternatives?

- Show quoted text -

reply | permalink