**Design doc for resource tracking**

This is a sub-JIRA to the Tetris JIRA. It goes into further details on how to augment the NM and the NM heartbeats in order to obtain more detailed resource accounting.

**1. Description**

This JIRA enhances the Node Managers (NMs) to monitor the resource usage on the machine, aggregates these reports and exposes them to the RM.

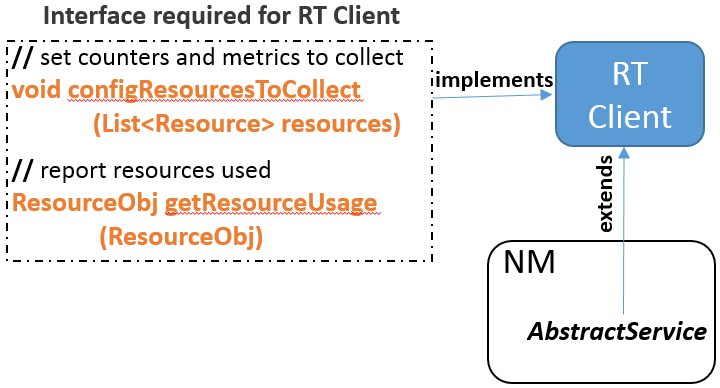
[JIRA-2745](https://issues.apache.org/jira/browse/YARN-2745) will directly benefit from this enhancement to do efficient packing scheduling, but we think its purpose is more general and will lead to further improvements in the YARN ecosystem.

**2. Changes**

In order to implement the Resource Tracking (RT) mechanism, we propose the addition of three aspects:

1. An RT Client service at the NM to collect various resource usage counters.
2. An RT Master service at the RM which aggregates resource usages from NMs and expose them to the RM scheduling logic.
3. Communication between RT Clients and RT Master.

**RT Client**



RT Client is implemented atop NM as a service by extending the ***AbstractService*** class and any of its implementations should implement***configResourcesToCollect()***and ***getResourceUsage()*** methods. It can be implemented as a standalone entity or can be built by a shim that wraps onto existing monitoring systems such as [Ganglia](file:///G:\vstf_catac\catac\p1_cr\jira\ganglia.sourceforge.net).

The ***configResourcesToCollect()*** method specifies which resources the RT Client should monitor. We propose to collect the following metrics as a first step:

|  |  |  |  |
| --- | --- | --- | --- |
| **Resource** | **Values** | **Metrics** | **E.g. commands** |
| CPU | Usage [%] | Peak, Avg., Min | uptime, dstat |
| Memory | Physical/Virtual [MB] | Peak, Avg., Min | Free |
| Network | R/W [MB] | Peak, Avg., Min | Ifconfig |
| Storage | R/W [MB] | Peak, Avg., Min | df, iostat |

NM interrogates the RT Client for used resources over a time interval through ***getResourceUsage()***. This method takes as input a *ResourceObj* object which describes the time interval for which used resources are required, and the corresponding metrics of interests(should be a subset of the ones exposed through ***configResourcesToCollect()***).

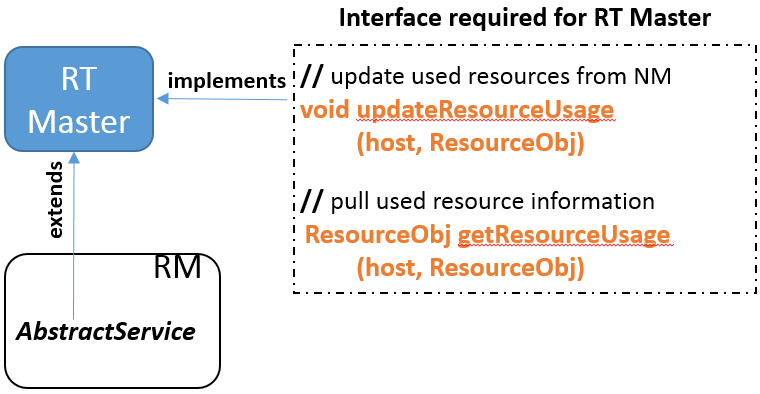
**Class Resource**:

* **type** // describes resource type (e.g. CPU, Mem., Network, Storage)
* **metrics** // describes metrics to collect (e.g. peak, average, min)
* **values** // resource counters for every metric

**Class ResourceObj:**

* **hostname**  // describes the name of the host where resources are monitored
* **time\_start, time\_end** // describe the time interval for which an action over resource information is executed
* **List<Resource> resources** // contains information about counters and metrics for used resources

**RT Master**



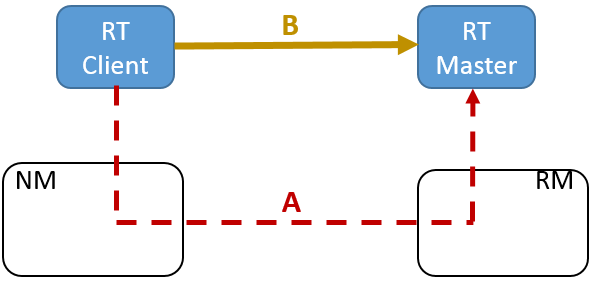
RT Master is implemented by extending the ***AbstractService*** class atop RM and any of its implementations should implement ***updateResourceUsage()*** and ***getResourceUsage()*** methods. It can be implemented as a standalone entity or can be built as a shim around an existing monitoring system client such as [Ganglia](file:///G:\vstf_catac\catac\p1_cr\jira\ganglia.sourceforge.net).

The RT Master has two main responsibilities:

1. Collects information about used resources from every RT Client. RT Client entities periodically trigger the invocation of ***updateResourceUsage()*** method to provide updates about resources used on a specific host over a time interval. RT Master can aggregate this information and do further analytics on this data.
2. Reports resource status on a specific host to RM. For example, RM can request such information, whenever the scheduling logic is invoked by invoking ***getResourceUsage()*** method which takes as input a ResourceObj object which describes the time interval and the metrics for the needed resources.

**Communication challenges**

Should the communication between RT Client and RT Master go out-of-band (option B) or should we expand the NMs heartbeat to the RM (option A)?



1. Enhance *Yarn API* to propagate a ResourceObj object through NM heartbeat by changing the yarn\_protos.proto and Resource\*.java files.

Advantages:

* Tight integration with the NM heartbeat so minimal communication disruption.
* Same reliability as the communication between NM and RM.

Disadvantages:

* potential timing overhead in waiting for RT Client to provide a ResourceObj on every heartbeat invocation, which might be expensive if the RT Client implementation is slow.
* does not allow RT Master to be deployed on a separate machine other than RM, and the potential problems which may occur at RT Master can slow down or kill the RM process.
* changes in ResourceObj structure requires changes in YARN API.

**B)** Provides out-of-band heartbeats between RT Client and RT Master.

Advantages:

* RT Master can be deployed on any machine.
* Configure the frequency of resources updates independently of NM heartbeat.

Disadvantages:

* YARN agnostic channel of communication which may lead to security breaches if not implemented well.
* Additional packet/ service overhead. However, ResourceObj size should not carry more than ~100 bytes which is a reasonable small amount of traffic.

**3. Proposed implementation**

We propose to build option (A) above. That is, the RT client and RT master run as separate threads within the NM and the RM respectively. Interface invocation translates to IPC calls between the NM and RT client and between the RM and the RT master. NM heartbeat structure is extended accordingly. We will add datastructure support at RM to store the information gathered by the RT master.

We will also add configuration variables so as to keep the RT client and RT master optional parts of the architecture.

If the RT client (or master) are unavailable, the NM (or RM) will restart them.