Network as a resource design document

By Varun Vasudev and Sidharta Seethana

Introduction

As part of allowing multiple workloads on YARN, we need to add support for network as a first class resource to allow applications such as Storm to run on YARN. Storm, for example, tends to be very network I/O bound. For Storm to run more predictably on YARN, we need to be able to specify and enforce bandwidth limits. In addition, adding network as a resource will also help other applications such as MapReduce and Tez run in a more predictable manner.

Network as a resource

When it comes to network as a resource, there are two aspects that applications tend to care about - bandwidth and network ops per second. We have decided to add support for (outbound) network bandwidth as a resource for now. There are two aspects to network bandwidth - sustained rate and burst rate. The “sustained” rate is the rate at which a container will be able to send outbound traffic without being throttled due to enforcement rules applied by the NM. However, depending on overall network utilization, the container maybe be allowed to send data at a higher “burst” rate. This “burst” rate is not guaranteed in any way. Container allocation requests can only include a request for “sustained rate”. Similar to CPU,

cluster administrators will be allowed to enable or disable elasticity (“burst”) on the enforcement side.

Please note that we’ll only be supporting scheduling and enforcement of “outgoing traffic” only due to limitations of policing traffic on the incoming side. For some additional details, please see below.

Scheduling

Given the decision to model only network bandwidth, scheduling becomes straight-forward, similar to memory and CPU. All NMs will report available bandwidth(in mbit/sec) to the RM.

The Resource class will be modified to accept network bandwidth (in units of mbit/sec) as part of the constructor. We propose changing the function signature to the following

```
public static Resource newInstance(int memory, int vCores, int
outboundNetworkBandwidth)
```

Getter and setter functions will be added to the Resource class. A new repeated field `resources` will be added to the ResourceProto.

```
message ResourceProto {
  optional int32 memory = 1;
  optional int32 virtual_cores = 2;
  optional int32 outbound_network_bandwidth = 3;
}
```

The DominantResourceCalculator will be modified to consider the `outboundNetworkBandwidth` when making scheduling decisions.

The following configuration options will be available:

1. `yarn.scheduler.enable-network-scheduling -` Configuration setting to enable or disable network as a resource allowing administrators to enable it when they see fit. If this setting is set to *off*, it will result in the network bandwidth on the node being reported to the RM but ignored for the purposes of scheduling.
2. `yarn.scheduler.minimum-allocation-outbound-network-bandwidth-mbit`, `yarn.scheduler.maximum-allocation-outbound-network-bandwidth-mbit` - settings for minimum and maximum outbound network bandwidth allocations. These default to 1 mbit/sec and 1000 mbit/sec respectively.

Rack awareness

When it comes to network as a resource, there is an additional consideration to be made during allocation. Some applications will launch containers that communicate with each other. In such cases, it would be preferable to launch containers within the same rack. In cases where communication is across racks, it might not be possible to guarantee the bandwidth that the containers desire. In such cases, it would be beneficial to allocate containers on racks that have the most bandwidth available.

The scheduler could adopt the policy below;

1. If this is the first container allocated, allocate it on the rack with the highest network bandwidth available.
2. If containers for this application have already been allocated, try to allocate containers on rack(s) which already have containers for this application. If there is no capacity on these racks, find the closest rack(in terms of topology) that can fit the container.

While this might not to lead to optimal performance, we feel it is appropriate as a first step. This policy can always be revisited in the future.

Rack information is already available to YARN via the `net.topology.script.file.name` configuration field in `core-site.xml`. The proposal is to use this information and use the policy listed above.

Enforcement

Overview

With regards to enforcement, this document is restricted to : 1) linux support only 2) egress traffic only.

A note on ingress traffic

Unlike outbound traffic, ingress traffic is not partitioned or tagged in the same manner that outbound traffic can be (for details on outbound traffic shaping, see below). Ingress *qdisc* is not classful and we cannot apply fine-grained filters and rules in the same manner as we can for egress traffic. In addition, by the time traffic reaches the ingress *qdisc*, network bandwidth has already been consumed. At this stage, if some sort of ingress policing is in place, there is no queue (and no delay mechanism) available[1], and packets are simply dropped. For these reasons, ingress traffic support is not in scope for the time being.

[1] http://tldp.org/HOWTO/Traffic-Control-HOWTO/elements.html#e-policing

Support for egress traffic shaping is achieved through a combination of two things : the `net_cls` cgroup subsystem and the linux traffic shaping utility `tc`

The `net_cls`[2] cgroup subsystem

The `net_cls` cgroup subsystem allows us to 'tag' outgoing packets with a 32-bit class identifier called 'classid'. This classid is specified in the file `net_cls.classid` (located under the mounted cgroup directory). The value of this classid identifies a traffic 'handle' which can be used subsequently for traffic shaping. For each YARN container that is spun up with network throttling required, a classid is generated and written to the classid file for the corresponding cgroup.

Traffic shaping using `tc`[3]

Once tagging is in place, the `tc` shell utility (short for Traffic Control) can be used to specify traffic shaping rules corresponding to the classid in use. The HTB[4] (Hierarchical Token Buffer) classful qdisc is used in order to achieve class id based traffic shaping[5].

Configuration Options

There are a few configuration options that will be made available on the NM for enforcement purposes.

1. `yarn.nodemanager.resource.network.interface` - Administrators can specify the network interface in use by YARN (and running applications). Traffic shaping rules are applied against this interface. This is determined automatically if possible, but defaults to `eth0` in case detection fails or not configured.

2. `yarn.nodemanager.resource.network.outbound-bandwidth-mbit` - Administrators can set the bandwidth available to the node in yarn-site.xml. The default bandwidth will be determined automatically if possible. If it cannot be determined, it'll be set to 1000 mbit/sec unless overriden by the administrator in

---

[2] For more information about net_cls and classids, see : https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Resource_Management_Guide/sec-net_cls.html

[3] See more information about tc here : http://www.lartc.org/manpages/tc.txt

[4] HTB and its mechanism of "borrowing" are covered in detail here : http://luxik.cdi.cz/~devik/qos/htb/manual/userg.htm and here: http://tldp.org/HOWTO/Traffic-Control-HOWTO/classful-qdiscs.html

[5] Simple example of how net_cls and tc, with a HTB classful qdisc can be used for traffic shaping is available here : https://www.kernel.org/doc/Documentation/cgroups/net_cls.txt

yarn-site.xml. This value, in conjunction with the bandwidth allocated for yarn containers (see below) is used to determine the amount of outbound network bandwidth that is available (guaranteed) to processes that don't run in YARN containers. For additional information, please refer to the NM startup section.

3. `yarn.nodemanager.resource.network.outbound-bandwidth-yarn-mbit` – Administrators can set the max total bandwidth available to YARN containers in yarn-site.xml. If this is not specified explicitly, it will default to the value of `yarn.nodemanager.resource.network.outbound-bandwidth-mbit`

4. `yarn.nodemanager.strict-resource-usage` – This is an existing configuration parameter allows administrator to specify if allowed usage is 'elastic' or 'strict'. In case strict usage is not enabled, each YARN container's network bandwidth utilization is allowed to 'burst' upto the value of `yarn.nodemanager.resource.network.outbound-bandwidth-yarn-mbit`

Validating configuration - default and non-default

Configuration is validated during startup (RM/NM). Depending on the cluster where YARN is deployed with network scheduling enabled, it is possible for the default configuration to be invalid (for example, `eth0` may not be a valid interface on the slave nodes or the configured max interface bandwidth is incorrect). In such scenarios where the defaults or the specified configuration params are invalid, the NM will log an error and fail to start.

NM startup

Once configuration is validated, the following actions are performed during NM startup :

1. `net_cls` cgroup subsystem is mounted
2. For the network interface in use, the following `tc` actions are applied on it :
    a. add a HTB queuing discipline (qdisc) attached to the root of the interface with a fixed handle. e.g :
    `tc qdisc add dev eth0 root handle 10: htb`
    b. attach a top level class to this qdisc using the max bandwidth of the interface, along with a default routing to class 3 for any 'unmatched' traffic for processes that are not YARN containers.
    `tc class add dev eth0 parent 10: classid 10:1 htb rate 1000mbit default 3`
    c. enable filter based on classid
    `tc filter add dev eth0 parent 10: protocol ip prio 10 handle 1: cgroup`
    d. now attach two classes - one for the max total allowed traffic for yarn containers and another for the the remaining bandwidth (default - class 3). The

default class has an allowed ceil corresponding to the max-bandwidth of the interface allowing it to use all the bandwidth available if not in use.

```
tc class add dev eth0 parent 10:1 classid 10:2 htb rate
700mbit
tc class add dev eth0 parent 10:1 classid 10:3 htb rate
300mbit ceil 1000mbit
```

To carry out the `tc` actions, we need root permissions. We will modify `container-executor.c` to allow this functionality.

Creating a cgroup and assigning classids to (traffic from) containers

A cgroup is created (for the net_cls subsystem) for each new container that is spun up by YARN. Each container is assigned a unique (unused) classid that is used to tag packets originating from this cgroup. For example :

```
mkdir /cgroup/net_cls/container_1
echo $PID >> /cgroup/net_cls/container_1/tasks
echo 0x100007 >  /cgroup/net_cls/container_1/net_cls.classid
```

This functionality will be implemented as part of the `CGroupsLCEResourceHandler` class.

Applying classid-based traffic shaping rules to specified interface

Once cgroup related changes are made for the container, a class with the required rate needs to be applied to the htb tree. The classid used here needs to be the same as the classid written to the `net_class.classid` file. Depending on whether strict enforcement is enabled, a 'ceil' burst rate is configured as well :

```
tc class add dev eth0 parent 10: classid 10:7 htb rate 50mbit ceil
700mbit
```

As mentioned above, `container-executor.c` will have to modified to add support for this functionality.

Resource localization, Shuffle, Log Aggregation, HDFS

There are certain shared services which are provided by the NM which could potentially consume large amounts of network bandwidth, such as resource localization, log-aggregation, and shuffle. Since these services are provided by the NM itself, we cannot apply container specific limits to them. The same issue is applicable to HDFS as well.
There are two potential solutions (on Linux) -

1. Run the NM itself under a CGroup (and an associated `tc` rule) with a new configuration parameter controlling the amount of bandwidth available to the NM. Since this would cap the amount of bandwidth available to NM (which is otherwise

uncapped), it could cause deterioration in performance. In this scenario, HDFS bandwidth utilization (e.g serving remote reads) would still be uncapped.

2. Create two sub hierarchies under the top-level HTB qdisc - one for YARN containers and one for everyone else. This would guarantee bandwidth for operations such as shuffle, log-aggregation while allowing these operations to consume more bandwidth than what is guaranteed, if available. The implication of this approach is that shuffle, log-aggregation, HDFS reads (serving) could be restricted to smaller amounts of bandwidth in the presence of high network utilization by YARN containers. The NM startup section of this document assumes that we'll use this approach.

A better longer term solution might be spawn log-aggregation and shuffle as micro-services in YARN containers to which we can then apply container specific limits as described in this document.

Miscellaneous

The RM and NM will have to be modified to enable accounting(chargeback) for network. This feature is currently enabled for memory and CPU. There are some UI changes necessary as well in order to list available/allocated network bandwidth.

MapReduce, DistributedShell

Both MapReduce and DistributedShell will be modified exposing network as a resource. In case of MapReduce, job configuration options will be added to set the desired network-bandwidth for map and reduce tasks. In the case of distributed shell, command line options will be added to allow users to specify network bandwidth.

Known issues

1. The current enforcement mechanisms are Linux only.
2. On Linux, the current enforcement mechanisms support traffic shaping for egress only. Ingress traffic cannot be shaped in the same manner. This implies that remote HDFS reads from YARN containers will not be constrained. HDFS writes however will be throttled.
3. In case of multi-home environments, administrators will have to explicitly set the network interface in the NM config. If this interface is set incorrectly, the result will be that enforcement could behave incorrectly.