Elastic Memory Control in YARN

YARN-4599

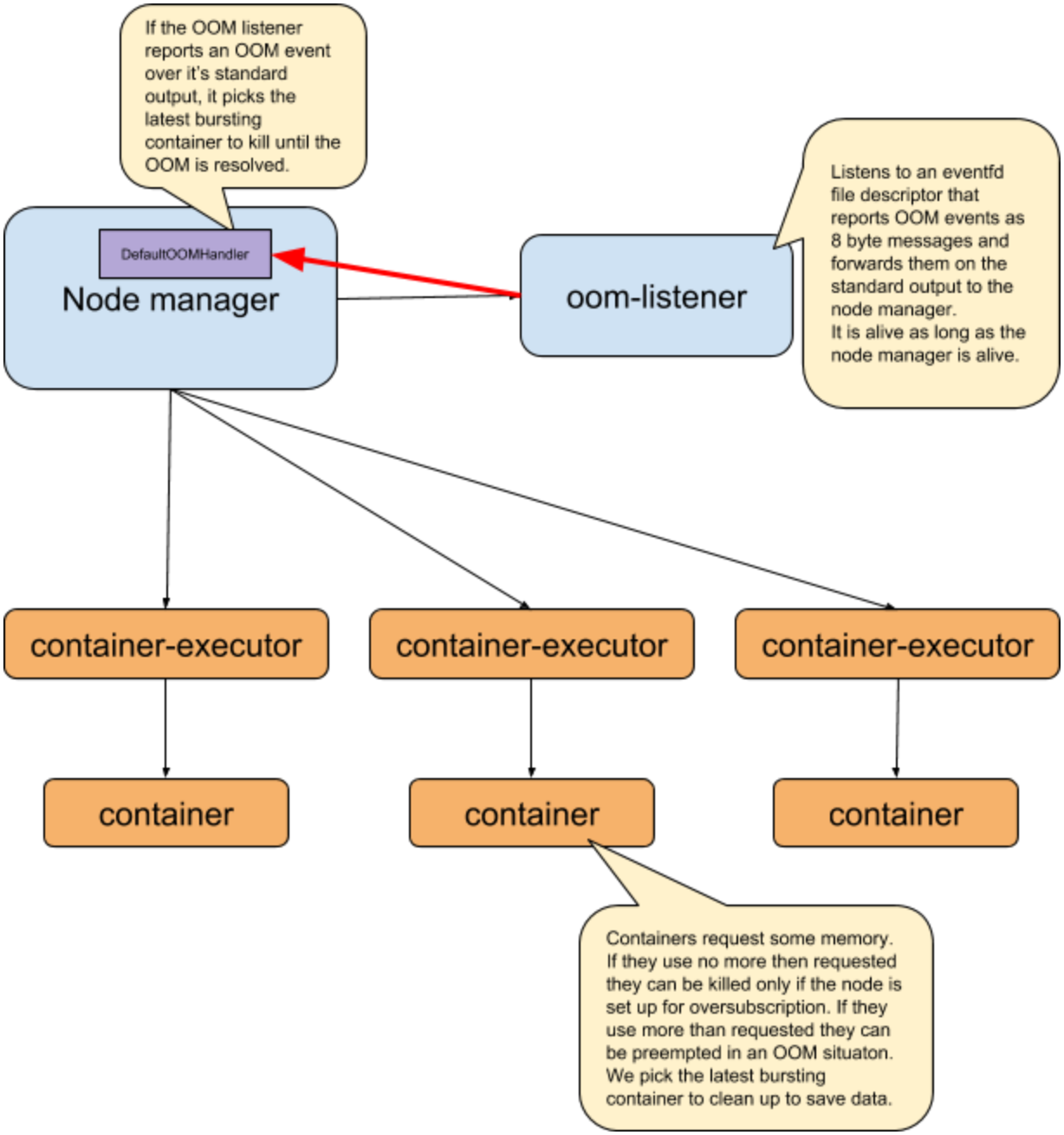
History

YARN had two memory resource management algorithms in the past. Container monitor was polling the memory usage of the containers every few seconds. If one exceeded it's virtual or physical memory limit, it was cleaned up. The drawback of this solution is that a container can escape by allocating too much too fast. If it reaches the node limits it may bring down the whole node.
The other option was to use cgroups. This is a better approach since the kernel takes care of the triggering and the cleanup, so the container cannot affect the node manager service. The drawback is that containers that would burst for a short time would need to request more memory, even if they are using it for just a short amount of time.

Solution

Elastic memory control uses cgroups. However, it sets a limit on the parent of all containers not each individual container. It triggers an OOM only when we reach the node limits. In this case it does not rely on the kernel to trigger the OOM killer but freezes all the containers for a short time. Node manager can then decide which containers to clean up to resolve the OOM situation.

Design concepts



- The basic idea is what was discussed above. It disables the OOM killer on the hadoop-yarn cgroup. This will trigger a pause on all containers when all of them exceeded the node limit.
- YARN will be notified with an executable listening to the cgroups OOM Linux event. This should be very fast. The executable is oom-listener, not container-executor. This is because it does not need to run as root. We avoided JNI to be more defensive on security and it also helps to test the executable easier.
- When YARN receives the notification, it runs a pluggable OOM handler to resolve the situation. YARN is outside the hadoop-yarn group, so it can run freely, all containers are frozen at this point. Different users may have different preferences, thus the handler is pluggable.
- The default OOM handler picks the latest container that ran out of it's request. This ensures that it kills a container that did not cost much so far but it keeps guaranteed containers running that play by the rules and use memory within their limits. It repeats the process until the OOM is resolved. Based on my experiments the kernel updates the flag almost instantaneously, so it will just kill as many containers as necessary.
- If the default OOM handler cannot pick a container with the logic above it kills the latest container until the OOM is resolved.
- If we are still in OOM without any containers, an exception is thrown and the node is brought down. This can be the case, if containers leaked processes, had processes running with another user and cannot be killed the container user or someone put a process into the parent hadoop-yarn cgroup.
- The OOM killer is not the normal container cleanup code. The standard behaviour is to send a SIGTERM to the container PGID process and if it does not respond in 250 milliseconds, it sends a SIGKILL. However, in our case all the processes are frozen by cgroups, so that they cannot respond to a SIGTERM. Because of this it uses the standard container executor code to send a SIGKILL to the PGID right away with the container user. The kernel OOM killer would do the same. This works pretty fast. It walks through all the thread/process IDs in the tasks file, so that all active PGIDs are found in the container. The current code does not delete standalone processes that are not a process group leader. If they are not part of one of the container local process groups they may be leaked. It also cannot handle processes that are running as different users other than the container user in the cgroup of the container. This should be rare.
- The code adds a watchdog to measure the time to resolve an OOM situation. The time to resolve an OOM situation takes 10-160 milliseconds based on my experiments.
- The patch contains documentation to set up and troubleshoot the feature.