YARN-1547

Prevent DoS of ApplicationMasterProtocol by putting in limits

The YARN RM is vulnerable for DOS attacks over the 3 RPC protocols it implements. With the introduction of AMRMProxy (YARN-2884) we can easily prevent DoS attacks on ApplicationMasterProtocol by enforcing throttling/bounding from misbehaving ApplicationMasters.

For this Jira we can implement a new class DoSInterceptor that extends the AbstractRequestInterceptor class. It provides an implementation that forwards the AM requests to the YARN RM and prevent DoS attacks.

ApplicationMasterProtocol implements 3 APIs, let's analyze them in details.

- registerApplicationMaster(RegisterApplicationMasterRequest)

Multiple requests from the same container are not blocked from AMRMProxy, they will reach the YARN RM.
AM can re-register in case of failover in the cluster. **We need to count the number of requests.**
Its request contains information about host, port, and tracking URL. **We need to validate them.** One possible solution can be adding a validate inside RMWebApp, by using Jetty DoSFilter.

- finishApplicationMaster(FinishApplicationMasterRequest)

Multiple requests from the same container are not blocked from AMRMProxy, they will reach the YARN RM.
AM can deregister multiple times to be sure the RM deallocated the AM. **We need to count the number of requests.**
Its request contains information about diagnostics and final trackingURL. **We need to validate them.** One possible solution can be adding a validate inside RMWebApp, by using Jetty DoSFilter.

- allocate(AllocateRequest)

By design the AMRMProxy receives multiple requests from the same container. Theoretically we can get unlimited number of allocate calls with unlimited number of resourceAsks, containersToBeReleased and resourceBlacklistRequest in AllocateRequest. On top of this we also can get unbounded number of priorities and/or resourceRequests in each ask.

To prevent this, we may need to:

1. **Count the number of allocate calls and determinate its frequency;**
2. **Count the size of resourceAsk, containersToBeReleased, resourceBlacklistRequest's lists, increaseRequests and decreaseRequests list;**
3. **Validate priorities and number of Container in resourceAsk.**

We need to determinate some limits for those. In case the counting passes the threshold we should back-off all the requests from the AM. Instead in case the values in resourceAsk are not valid we can **reject** or **back-off** the request.

To do this, we can implement a sliding window system that counts the number of request per minute and determinates if in a time slot we AM exceeds the number or requests.