YARN-1547: Prevent DoS of ApplicationMasterProtocol by putting in limits

The YARN RM is vulnerable to DOS attacks over the 3 RPC protocols it implements. Denial of Service attacks can result in significant loss of service, money and reputation for organizations.

This document explains our idea how to prevent them on ApplicationMasterProtocol by enforcing throttling/bounding from misbehaving ApplicationMasters.

For this Jira we will implement a new component “DoSInterceptor” that:

- keeps tracks of all the requests and counts the occurrences of them;
- counts the size of their list fields;
- checks the correctness and formats their numerical and string fields;
- forwards to the YARN RM the correct calls from AM.

This new component should be located in the server side or in AMRMProxy (a new service running on each node, upon start the AM is forced via tokens and configuration to direct all its requests to a new services running on the NM that provide a proxy to the central RM – YARN-2884) as extension of AbstractRequestInterceptor frameworks.
All the framework and the interceptor will be moved and implemented under the project hadoop-yarn-server-common to be accessible from all the server entity.

DoSInterceptor task: Keeps tracks of all the requests

One example of Denial of Service attacks is to attempt to “flood” the server, hereby preventing legitimate user traffic.

We need to keep tracks of all the requests and limit the number of requests over time. To do this, we can use a sliding window system that counts the number of request per time. The sliding window advancement will be a configurable value (e.g. every 60 seconds).

For this example, the sliding window size is 3 slots (1 minute each) and the limit of requests per 3 slots is 10.

Time	Sliding window Status	Current Slot	Event	Total
0	0 0 0	0	Creation of Sliding window	0
1	1 0 0	0	Request arrive	1
61	5 0 0	1	Advance to slot 1 (between time 2 and 60, the AM sent 4 requests)	5
121	5 4 0	2	Advance to slot 2 (between time 62 and 120, the AM sent 4 requests)	9
125	5 4 1	2	Request arrive. The AM reaches its limit.	10
126	5 4 1	2	Request arrive. The AM passes its limit: **Action Require**. No more request are accepted.	10
181	0 4 1	0	Advance to slot 0 (3 mod 3 = 0), Clear the slot 0	5
182	1 4 1	0	Request arrive	6

At time 126, when a request reached the DoSInterceptor, the counter was 10; an action is required as kill the Application, reject the request, raise an alert …

DoSInterceptor task: Count the size of list fields

One example of Denial of Service attacks is to attempt to “fill” the server’s data structures, hereby preventing uses for legitimate users’ calls.

The ApplicationMasterProtocol requests have list fields. We need to count the size and limit the number of them. We can use a sliding window system for summing the number of list’s size.

For this example, the sliding window size is 3 slots (1 minute each) and the size limit of requests per 3 slots is 100.

Time	Sliding window Status	Current Slot	Event	Total
0	0 0 0	0	Creation of Sliding window	0
1	5 0 0	0	Request arrive. This request has a list with 5 elements.	5
61	55 0 0	1	Advance to slot 1 (between time 2 and 60, the AM sent 10 requests, each contains a list of 5 elements)	55
100	55 45 0	1	Request arrive. This request has a list with 5 elements. The AM reaches its limit. (Between time 62 and 100, the AM sent 4 requests, each contains a list of 10 elements)	100
102	55 45 0	1	Request arrive. This request has a list with 5 elements. The AM passes its limit: **Action Require**. No more request are accepted.	100
121	55 45 0	2	Advance to slot 2	100
122	55 45 0	2	Request arrive. This request has a list with 5 elements. The AM passes its limit: **Action Require**. No more request are accepted.	100
181	0 45 0	0	Advance to slot 0 (3 mod 3 = 0), Clear the slot 0	45
182	2 40 0	2	Request arrive. This request has a list with 2 elements.	47

At time 102 and 122, when a request reached the DoSInterceptor, the counter was 100; an action is required as kill the Application, reject the request, raise an alert …

We can apply an additional control to check if the list-size of a single request overflows a specific threshold.

DoSInterceptor task: checks and formats string and numerical fields

One example of Denial of Service attacks is to attempt to "crash" it by giving wrong input parameters, hereby preventing uses for legitimate users.

The ApplicationMasterProtocol requests contain numerical and string fields. We need to check and format their inputs.
String fields need to be checked for their length and for their correctness. We will truncate the strings if they pass a specific limit, and reject if they don't pass our correctness control. For example, structure as regular expression, e.g. web address.

Numeric fields need to be checked for their correctness. We will check if their values are in particular range and reject them in case of negative result, e.g. size of container, web port.

ApplicationMasterProtocol requests

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

- registerApplicationMaster(RegisterApplicationMasterRequest)

Object	**Object type**	**Type of Limit to be put**	**Correctness check**	**Behavior on limit-overflow**
RegisterApplicationMasterRequest	RPC Request	Sliding window counting		Rejected if list-size overflow
RegisterApplicationMasterRequest.Host	String	Size of string		String truncated
RegisterApplicationMasterRequest.RPCPort	Integer		Valid port number	Rejected if wrong value
RegisterApplicationMasterRequest.TrackingUrl	String	Size of string	Valid URL expression	Rejected if wrong value

- finishApplicationMaster(FinishApplicationMasterRequest)

Object	**Object type**	**Type of Limit to be put**	**Correctness check**	**Behavior on limit-overflow**
FinishApplicationMasterRequest	RPC Request	Sliding window counting		Rejected if list-size overflow
FinishApplicationMasterRequest.FinalAppStatus	FinalApplicationStatus			
FinishApplicationMasterRequest.Diagnostics	String	Size of string		String truncated
FinishApplicationMasterRequest.TrackingUrl	String	Size of string	Valid URL expression	Rejected if wrong value

- allocate(AllocateRequest)

Object	**Object type**	**Type of Limit to be put**	**Correctness check**	**Behavior on limit-overflow**
AllocateRequest	RPC Request	Sliding window counting		Rejected if list-size overflow
AllocateRequest.responseID	Integer		Positive; greater than the previous one	Rejected if wrong value
AllocateRequest.appProgress	Float		Positive and 0 <= x <=100	Rejected if wrong value
AllocateRequest.resourceAsk	List<ResourceRequest>	Sliding window counting		Rejected if list-size overflow
AllocateRequest.containersToBeReleased	List<ContainerId>	Sliding window counting		Rejected if list-size overflow
AllocateRequest.resourceBlacklistRequest	2 List<String>	Sliding window counting		String truncated; Rejected if list-size overflow
AllocateRequest.increaseRequests	List<ContainerResourceChangeRequest>	Sliding window counting		Rejected if list-size overflow
AllocateRequest.decreaseRequests	List<ContainerResourceChangeRequest>	Sliding window counting		Rejected if list-size overflow

- ResourceRequest

Object	**Object type**	**Type of Limit to be put**	**Correctness check**	**Behavior on limit-overflow**
ResourceRequest.Priority	Priority		Valid priority value	Rejected if wrong value
ResourceRequest.HostName	String	Size of string		String truncated
ResourceRequest.Capability	2 Integer		Positive; within a range.	Rejected if wrong value
ResourceRequest.NumContainers	Integer	Sliding window counting	Positive	Rejected if list-size overflow; Rejected if wrong value
ResourceRequest.RelaxLocality	Boolean			
ResourceRequest.LabelExpression	String	Size of string		String truncated

We can analyze the 2 ContainerResourceChangeRequest lists as a single one. ContainerResourceChangeRequest contains a Resource field as ResourceRequest; additional check may require.