**Design doc for expanded ask**

**1. Description**

This is a sub-JIRA to the Tetris JIRA. It goes into further details on how to expand the ask from AM to RM to include additional information about tasks profiles and how we propose to estimate the resource demands of tasks.

**2. Changes**

We propose two changes:

1. Enhance the ask capability with additional fields that describe tasks’ resource requirements for disk and network
2. Provide a mechanism to estimate tasks resource requirements

**2.1 Extend ask capability**

Current ask capability propagates the ***memory*** and the number of ***virtual cores*** as task resource requirements. [YARN-2139](https://issues.apache.org/jira/browse/YARN-2139) propose to add a new field ***vdisks*** to specify disk requirements (currently in terms of number of physical disks). As an extension, we propose the following ask. For each new field, we will (a) edit the ResourceProto structure at yarn\_protos.proto, (b) add methods to set and get the field’s value in the ***Resources.java*** and (c) corresponding changes to protobuf.

message ResourceProto {

optional int32 **memory** = 1;

optional int32 **virtual\_cores** = 2;

optional int32 **network\_read** = 3;

optional int32 **network\_write** = 4;

optional int32 **disk\_read** = 5; // same as YARN-2139

optional int32 **disk\_write** = 6;

optional int32 **remaining\_tasks\_to\_sched** = 7;

optional int32 **duration** = 8;

}

Fields explanation: field name [metric]; how it is computed.

**virtual\_cores** **[%]**: A number from 0 to 100 denoting the fraction of available cores at a server that is needed for this task. Estimated based on the CPU cycles used by the task during its lifetime.

**memory** **[MB]**: A number illustrating the memory to be set aside for this task in MB. Estimated as the peak memory usage consumed by the task at any point during its lifetime

**network\_read** **[MBps]**: As above, a number illustrating the amount of network read bandwidth to be set aside for this task. Estimated based on the rate at which the task can read input data from the network

**network\_write** **[MBps]**: Estimated as the rate at which the task can write data over the network.

**storage\_read** **[MBps]**: Estimated as the rate at which the task can read data from the storage

**storage\_write** **[MBps]**: Estimated as the rate at which the task can write data to the local storage

**remaining\_tasks\_to\_sched**: remaining number of tasks of this type to be scheduled

**duration** [sec]: average task duration

Here **GRACE\_FACTOR** is a constant factor used to ensure that inaccuracies in estimation do not under-estimate task’s real demands. This is a configuration variable.

Also note that these demands are similar across all tasks in a stage (map- or reduce-). Hence, we expect to use statistics gathered at task completion or through the RT client/ RT master to infer these numbers per stage. For recurring jobs, we expect to use cluster-wide history servers to collect these statistics.

Finally, note that the actual demands of a task depend on its placement. If placed local to its input, a task would require storage\_read but not network\_read. This adjustment is performed by the RM when using the ask.

* 1. **Estimating tasks’ resource needs**

We already hinted at how each of the asks’ fields can be estimated. Here, we add a bit more detail.

Each different type of task has a default resource profile specified by the user either through configuration parameters passed along with the job execution script or through a repository that records the expected usages of tasks in previous runs of the same or similar jobs through a tight communication with the job history entity. For example, jobs that run the same computation but on different input size, or different jobs but whose resource consumption pattern is similar.

Tasks demands are updated whenever a task has successful finished by extracting relevant counters from its history statistics. We are using the following counters to compute tasks demands:

**CPUCYCLES** = TaskCounter.**CPU\_MILLISECONDS**

**memory** = TaskCounter.**PHYSICAL\_MEMORY\_BYTES**

**input** = MAX(TaskCounter.**INPUT\_BYTES\_PROCESSED**, TaskCounter.**SHUFFLE\_BYTES**)

**output** = MAX(TaskCounter.**OUTPUT\_BYTES**, FileSystemCounter.**BYTES\_WRITTEN**)

**duration** = Task.getReport().**getFinishTime()** – Task.getReport().**getStartTime()**

Knowing the **input** and the **output** values for every task, we can compute the amount of data read/written from/to disk or network as following:

1. Modify **RMContainerAllocator.java** to keep track for every MAP task if it’s executed local or remote as well as recording how much output is produced by mappers on every machine in the cluster by exploiting the TaskCounter.**MAP\_OUTPUT\_MATERIALIZED\_BYTES** counter.
2. Whenever a **MAP** task finish, if it was executed local, we update {**INPUTSTORAGE** = **input, INPUTNETWORK** = **0**}, otherwise {**INPUTNETWORK** = **input, INPUTSTORAGE** = **0**}. The {**OUTPUTSTORAGE** = **output, OUTPUTNETWORK** = **0**} as the MAP output is always written to the local filesystem.
3. Whenever a **REDUCE** task finish, we compute how much data it has to retrieve from all the machines in the cluster and adjust the **INPUTNETWORK** and **INPUTSTORAGE** accordingly as a function of size of the input on every machine, total number of reduce tasks and of a skew factor. Additionally, {**OUTPUTSTORAGE** = **output, OUTPUTNETWORK** = **output**} as each reduce will write a copy of the replica to the local filesystem and pipeline the others replicas through the network.