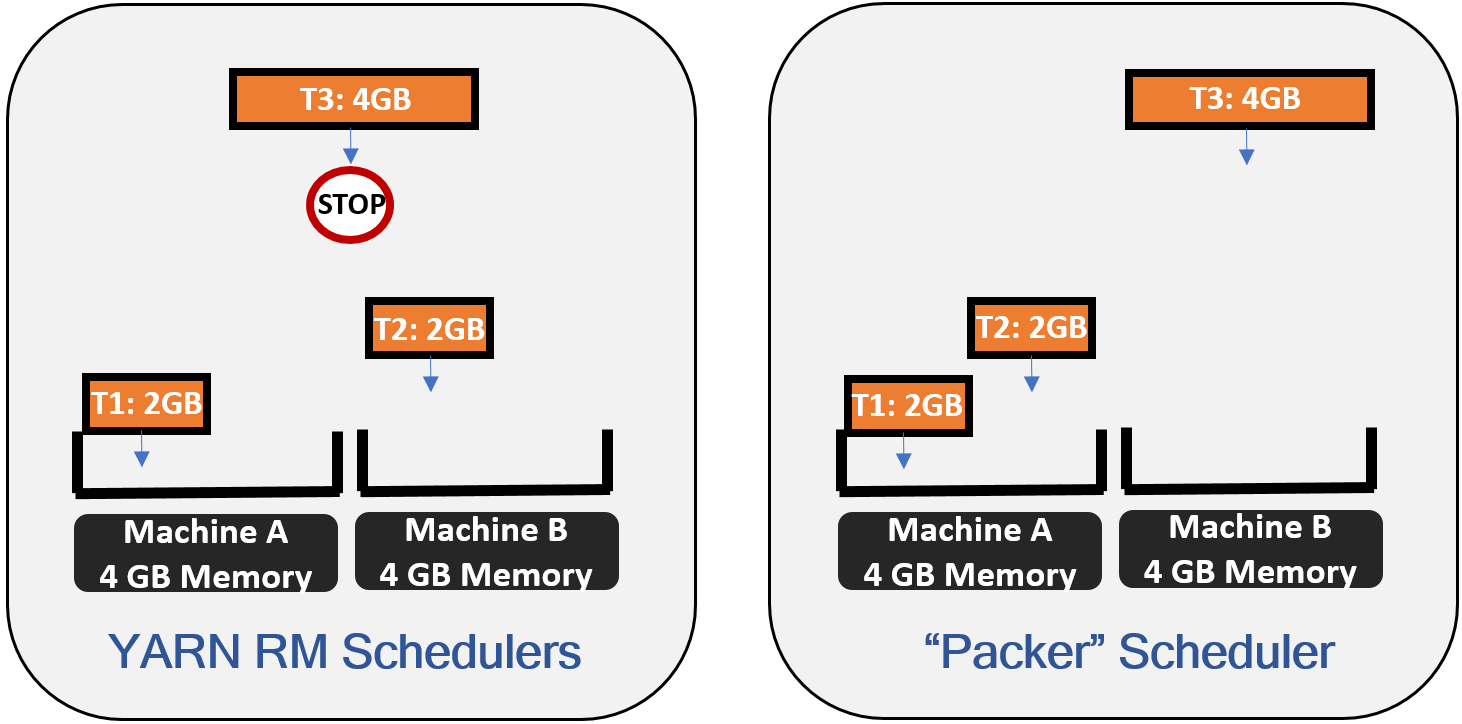
**Extend YARN to support multi-resource packing of tasks**

**Motivation**

Current YARN cluster schedulers allocate resources in slots, which correspond to some desired amount of memory and cores. They offer the slots greedily to the job that is furthest from its *fair* share [CS, FairScheduler]. Since they neither pack tasks nor consider all the relevant resource demands, they lead to ***fragmentation*** and ***over-allocation*** of resources. The result is lower cluster throughput and higher average job completion times.

Here are two simple examples to illustrate the problems.

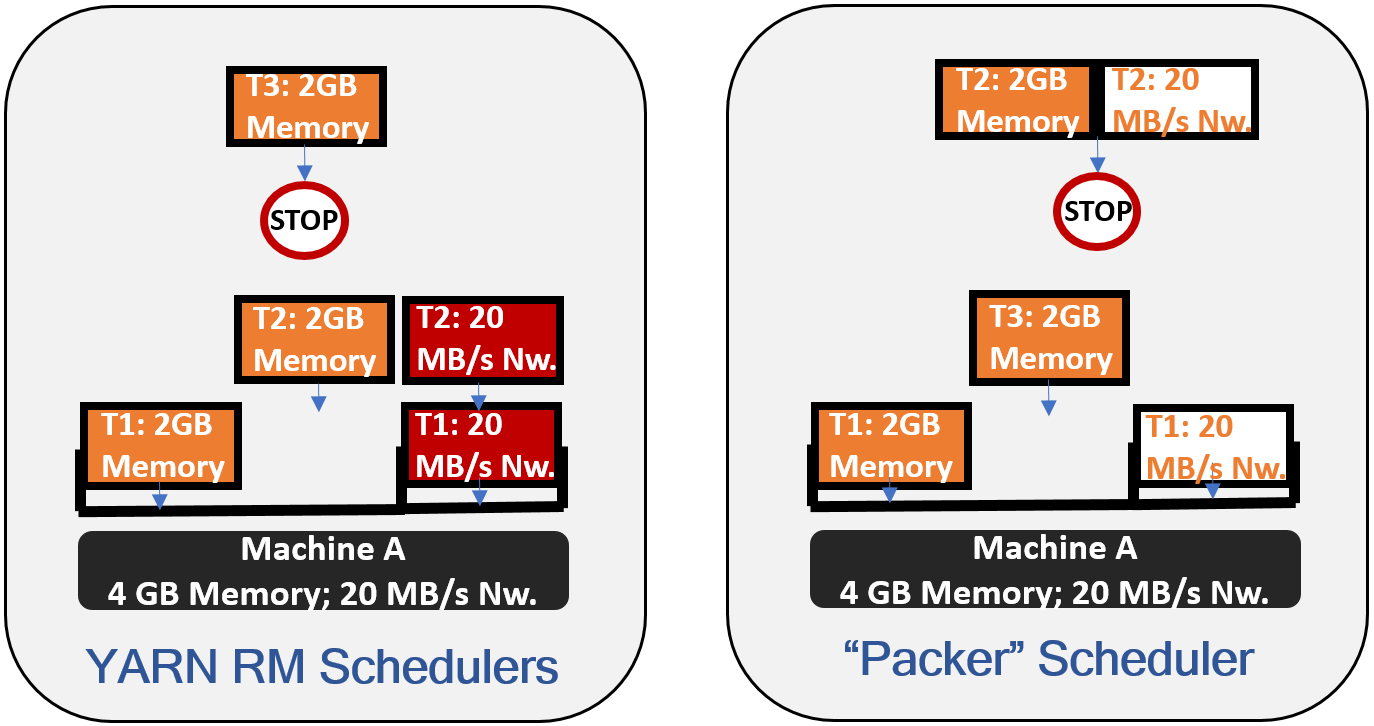
1. R**esource fragmentation**: Consider the example below. The YARN RM scheduler *can* assign T1 to machine A and T2 to machine B. There is not enough space for T3 at either machine. Instead, scheduling them per the schedule on the right leads to about 25% higher throughput.



The fundamental problem is that none of today’s schedulers explicitly consider packing resources. The order in which tasks are assigned depends largely on the order in which the heartbeats arrive and fairness considerations. There is indeed some support to not starve the tasks with big resource demands. However, that does not directly avoid fragmentation.

At an analytical level, the throughput loss due to resource fragmentation for a greedy assignment linearly increases with the number of pertinent resource dimensions. Online bin packing is a known hard problem. Worse in data-parallel clusters, task placement affects the resource demands of a task (local placement => no need for network). We contribute a new heuristic for online multi-resource packing.

1. **Over-allocation:** Consider the example below. Tasks T1 and T2 need network but T3 only needs memory. Since current schedulers do not explicitly consider network and disk demands, they could lead to the schedule on the left. Here, the network is over-allocated, which slows down both T1 and T2; perhaps by up to 2X or larger. Instead, the schedule on the right runs T1 and T3 together which prevents over-allocation and leads to higher (task) throughput.



The fundamental problem here is two-fold. First, since over-allocation increases the runtime of tasks that depend on the over-allocated resource, the other resources (CPU or memory) are wasted since these tasks hold on to them. Instead, other tasks could make progress with these resources. Second, in the extreme case, over-allocation can lead to collapse, since the extra load can cause interference on the disk or network incast.

We find that this happens often. Typically, reduce tasks use network bandwidth and today’s schedulers can put many reduce tasks on the same machine as long as memory or cores are available.

**Our contributions**

At a high-level, this JIRA offers three things.

1. An online bin packing heuristic that avoids over-allocation and tries to reduce fragmentation. This helps improve cluster throughput.
2. A shortest-job-first heuristic that preferentially steers resources towards jobs with less remaining work. This helps improve average job completion time.
3. A method to trade-off fairness with the above. In general, *perfect* fairness results in significant loss in efficiency (measured as throughput or average job latency). However, with this method, we show that we can get nearly perfect fairness with no loss in efficiency.

**What we have done**

We built a prototype atop Yarn that implements the above ideas and evaluated it in a deployment on a 200 server cluster. There is a peer reviewed publication that describes both the ideas and our initial results in more detail. The rest of this document will focus on our design to incorporate these changes into Yarn.

**Summary of main changes**

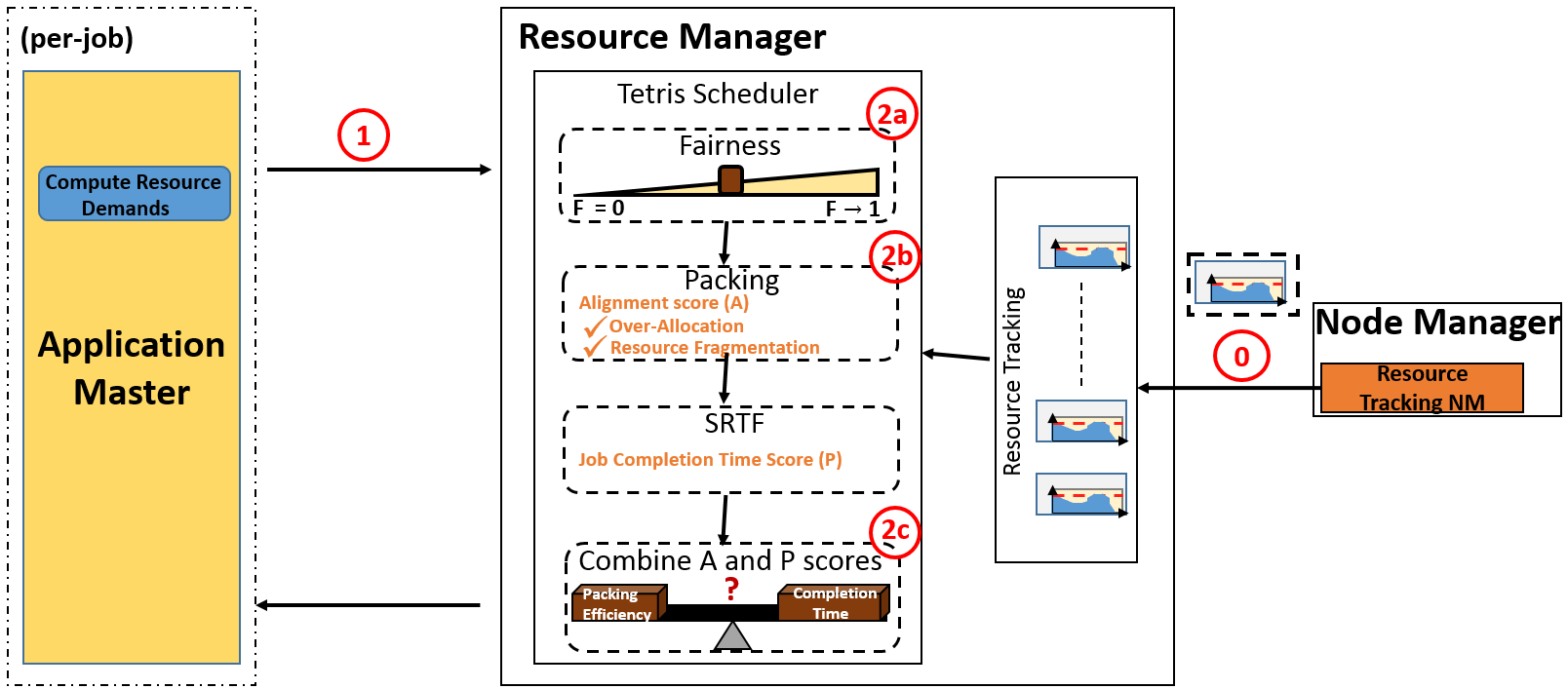
Update the container allocation logic in the RM scheduler. This change is the core. It enables “packing” tasks, preferring jobs with less remaining work and trades off fairness for efficiency.

Expand the AM->RM resource ask to expose tasks’ disk and network resource demands to the scheduler.

Support for cluster-wide resource tracking: want per-machine resource usage information available at the RM.

**End-to-End Design**

To help understand the big picture of our design, Figure 3 illustrates the steps in Tetris’s functioning along with the relevant Yarn components.



1. Out-of-band and periodically, the resource tracker at each node reports to the RM the actual resource usage at that machine. [[YARN-2965](https://issues.apache.org/jira/browse/YARN-2965)]
2. When an AM asks the RM for containers for its tasks, it also specifies the expected resource usage per task (peak estimates for CPU, memory, network and disk), expected duration per type of task, and the expected remaining number of tasks of each type. [[YARN-2966](https://issues.apache.org/jira/browse/YARN-2966)]
3. On arrival of the heartbeat from a NM, the RM invokes new logic due to Tetris to decide which tasks should be assigned in the heartbeat response as follows. [[YARN-2967](https://issues.apache.org/jira/browse/YARN-2967)]
   1. First, this logic uses the fairness knob to pick a set of “jobs” or “queues”. The *knob* fgoes from 0 to 1, *where* *f*=1 implies perfect fairness and f=0 implies best efficiency. Based on the f value, this logic ranks all jobs and queues in descending order of distance from their fair share and then picks the top ***⎡1-f⎤ fraction.*** That is, for f=0, all jobs are picked, whereas f=1 implies picking the most unfair job or queue. Deficit accounting is done similar to DRR.
   2. Second, for each job *j* picked in (a), the logic computes a score Pj that is inversely proportional to its remaining work in job j. The logic also computes an alignment score At for every pending task t belonging to the jobs or queue in (a). The alignment score is larger for the task that is best for packing.
   3. Third, from among the tasks scored in (b) above, the logic picks the task to assign to the machine based on At + ε Pj where ε trades-off between packing efficiency (throughput) and job completion time (latency). Assigning a task will update the deficit counters for the corresponding job.
   4. Repeat, above steps until all resources on the NM have been assigned.

**Implementation Sub-JIRAs**

Tetris consists of the following sub-JIRAs.

1. Resource usage tracking: This JIRA([YARN-2965](https://issues.apache.org/jira/browse/YARN-2965)) enhances the NMs to monitor the resource usage on the machine and report the unused resources to the RM in their heartbeats. The RM has a global resource tracking entity that maintains state of the available resources on each machine. It uses the available resources for its scheduling decisions to pack efficiently. The resources to track will be CPU, memory, disk and network bandwidths (or any subset thereof).

*Relevant Components:* Resource Manager, Node Manager

1. Expand ask between AM and RM to include disk/network demands: Architecturally, this sub-JIRA([YARN-2966](https://issues.apache.org/jira/browse/YARN-2966)) expands on the scope of [YARN-2139](https://issues.apache.org/jira/browse/YARN-2139). The datastructure of the ask is expanded. To fill in correct values in the ask, we add logic to estimate tasks’ resource demands? The simplest approach is to start with a “default” estimate for map and reduce tasks and continuously update the estimate based on the actual usages of tasks that are running/ have completed within the map- or reduce- stage. A slightly more comprehensive implementation also has a repository that records the expected usages of tasks in previous runs of the same or similar jobs. Datastructures at the RM are also expanded to record the information from asks. A non-trivial item is to encode the expanded asks in a manner similar to the current approach which allows consolidating datastructures at the RM.

*Relevant Components:* Resource Manager, Application Master

1. Changing the task matching logic at the RM: This sub-JIRA([YARN-2967](https://issues.apache.org/jira/browse/YARN-2967)) changes the matching logic at the RM. We expect different extensions to both the CS and the FS schedulers. These changes should work independent of the other changes. That is, with just CPU and memory in the asks, as is the case today, the matching logic should still work. Gains should increase with the expanded asks though.

*Relevant Components:* Resource Manager