# Analysis of Scheduling Policies for Next-Generation Rabbit Architecture

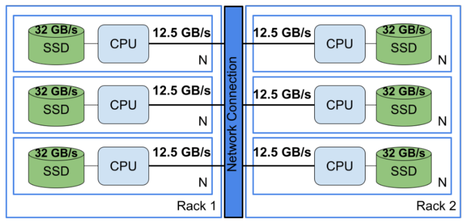
Keith Bateman [1,2] Stephen Herbein (Advisor) [1] Anthony Kougkas (Advisor) [2]
Xian-He Sun (Advisor) [2]

[1]Lawrence Livermore National Laboratory [2]Illinois Institute of Technology

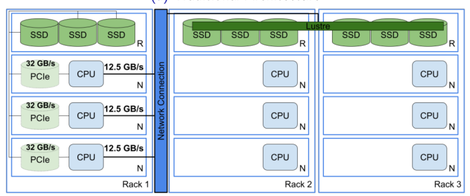
## Introduction

- Rabbit node architecture is a supercomputer architecture with SSDs pooled at the top of racks [1, 2].
  - SSDs connected to compute nodes through both PCIe connection and the network fabric simultaneously.
  - Will be integral in upcoming El Capitan supercomputer at Livermore (the primary usecase where this work is expected to be applied).
  - Complicates storage scheduling.
- With Rabbit node architecture, the scheduler has to balance **load balancing** (allocating SSDs across different racks) and **locality** (optimizing the connection speed by placing SSDs on the same rack as the job).
  - Jobs which require direct-attached (PCIe) storage need storage to be purely rack-local always.
  - Jobs which require network-attached (fabric) storage may be rack-local, but also may want to balance their load across as much of the network as possible.
- The problem arises when a single rack is full from a direct-attach request and a network-attach request wants to place one of its SSDs on that rack or vice-versa (when a network-attach request is split across the whole cluster and a direct-attach request wants a single rack). It is the responsibility of the scheduler, then, to broker the resources in order to strike a balance between locality and load balancing.

## Methodology




(a) Traditional Architecture




(b) Rabbit Node Architecture

Figure 1: Changes in Supercomputer Architecture Demonstrated

- The **objective** of this research is to discover ideal scheduling policies for Rabbit node architecture.
- Overall process
  - devise a set of metrics
  - diagnose problems with current policies according to those metrics
  - evaluate potential solutions to those problems.
- Evaluations use the Flux resource-query utility [3]
  - Simulates scheduling requests across resources with timing and allocation info
  - input: scheduling policy, system resource information set, jobspecs
- Resource sets
  - Suppose the hypothetical installation of Rabbit nodes on the Sierra supercomputer (large scale, comparable to other supercomputers).
- **Demonstration**: figure 1 shows the expected changes evident in the Rabbit architecture.
  - Rack-local PCIe accesses of Rabbit node SSDs could have up to 32 GB/s bandwidth in a Sierra-like architecture.
  - Fabric accesses of Rabbit node SSDs could have up to 12.5 GB/s bandwidth in a Sierra-like architecture.
  - Although these numbers would differ between architectures, there is a general expectation that local PCIe accesses will be faster than remote fabric accesses.

## Metrics

- **Makespan**: The time that a schedule takes, from beginning to end.
- **Average Job Wait Time**: The average time that it takes after submitting a job before it is able to run.
- **SSD Utilization**: The percentage of SSDs in the cluster which are allocated at a given time
- **Rack Imbalance**: Standard deviation of allocated SSDs contained in each rack across the cluster, a measurement of load balance.
- **Binary Locality**: Sum of SSDs which are allocated on different racks from the CPUs of their job allocation, a simple and network-agnostic measure of locality (i.e. each SSD contributes one hop or zero hops, depending on whether it's on a different rack from the allocated CPUs or the same)

## Policies

Scheduling policies traverse the graph representing the resource set to find resources that match the request. This traversal is performed in Depth-First order.

- **Default**: The "high" policy, prefers higher resource ID for a match.
- **Uniform-Storage**: A policy which prefers to match to SSDs from racks with fewer SSDs allocated.
- **First**: Just grabs the first available match. Fast and able to match unique resource requests (such as requesting direct-attach and network-attach simultaneously), but less intelligent than other policies.

## Rack Imbalance Problem

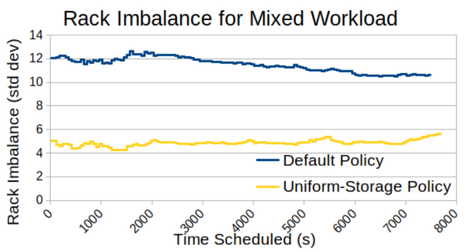
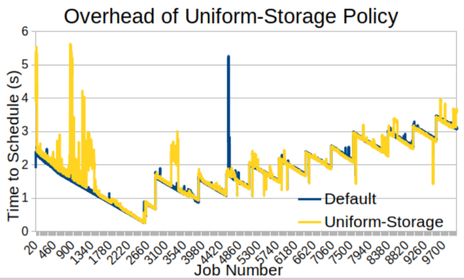



Figure 2: The rack imbalance of uniform-storage and default policies




- Rack imbalance can be ameliorated by using a uniform-storage policy.
- Figure 2 shows rack imbalance for a mixed workload of a third each of network-attached Rabbit jobs, direct-attached Rabbit jobs, and jobs that don't request Rabbits (with other characteristics of jobs taken from Lassen traces). Even though the only jobs that can be balanced are the network-attached, we end up seeing an average 57.6% decrease in imbalance across the whole workload.
- Uniform-storage policy has some (relatively small) overhead, as shown in figure 3. Also notice the pattern overall, which is that overhead decreases as the cluster shrinks, spikes as the cluster run out of space and we move to the next free time, and then repeats that pattern, increasing overall as finding the next free time for the cluster becomes more time-consuming.

## Locality Problem

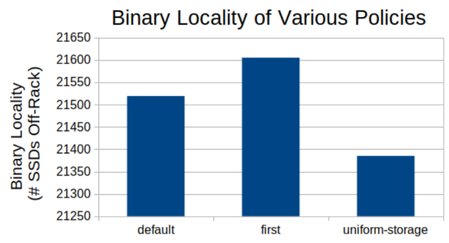



Figure 4: Comparison of Binary Locality of Scheduling Policies

- The expectation might be that uniform-storage policy would increase the binary locality metric.
- It can be seen, however, that uniform-storage shows a 1-2% decrease in the metric from other policies.
- This is most likely because the other policies do nothing to optimize locality, so that spreading out SSDs actually has a chance of putting them on the same rack as CPUs
- If this is correct, then a further policy could be devised which would optimize locality even further

## Conclusions

- Our results have focused on the uniform-storage policy, and shown that, with negligible overhead, this policy can improve both rack-imbalance (by about 60%) and locality (by 2%) from other policies.
- Since rack imbalance is anticipated to be the most common concern of network-attached job requests, this policy is expected to be very valuable as a baseline policy for those types of requests.
- Locality is typically not a concern for these types of requests because the requests usually go through the network fabric, so optimizing locality further for this type of request, while useful, is not the primary concern.
- Future work in this project will include evaluating more policies, potentially new ones to attempt to improve upon the existing policies.
  - In particular, a policy which could schedule storage evenly across racks while also handling jobspecs that request both direct-attach storage and network-attach storage would be useful, as the current uniform-storage policy can't traverse requests like that.
  - We're also considering a locality-optimized policy.
  - Lastly, evaluations on real hardware or using information about the upcoming El Capitan supercomputer would be valuable. In particular, information about the El Capitan network would lead to more detailed locality evaluations while information about the scale of El Capitan would help to ensure more parity between evaluations and reality.

## Acknowledgment

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

## References

[1] Timothy Prickett Morgan. Livermore Converges A Slew Of New Ideas For Exascale Storage. https://www.nextplatform.com/2021/03/09/livermore-converges-a-slew-of-new-ideas-for-exascale-storage/, 2021.

[2] Tiffany Trader. Livermore's El Capitan Supercomputer to Debut HPE 'Rabbit' Near Node Local Storage. https://www.hpcwire.com/2021/02/18/livermores-el-capitan-supercomputer-hpe-rabbit-storage-nodes/, 2021.

[3] Flux Team. Resource Query Utility. https://github.com/flux-framework/flux-sched/blob/master/resource/utilities/resource-query.cpp, 2021.



kbateman@hawk.iit.edu