Analysis of Scheduling Policies for Next-Generation Rabbit Architecture

Keith Bateman, Stephen Herbein (Advisor) *, Anthony Kougkas (Advisor),
Xian-He Sun (Advisor)
kbateman@hawk.iit.edu,stephen@herbein.net,{akougkas,sun}@iit.edu
Illinois Institute of Technology, *Lawrence Livermore National Laboratory

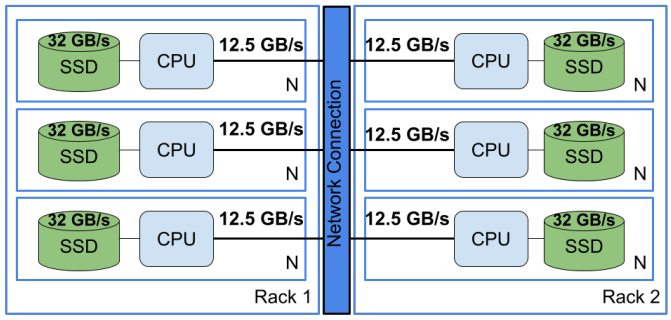
ACM Reference Format:
Keith Bateman, Stephen Herbein (Advisor) *, Anthony Kougkas (Advisor),, Xian-He Sun (Advisor). 2021. Analysis of Scheduling Policies for Next-Generation Rabbit Architecture. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

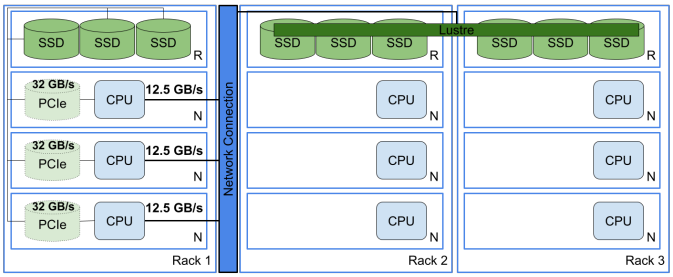
The Rabbit node architecture is a unique supercomputer architecture wherein SSDs are placed at the top of racks and connected to compute nodes through both PCIe connection and network fabric simultaneously, as in figure 1b. This architecture will be integral in the upcoming El Capitan supercomputer at Livermore, which is a valuable usecase for this work. The scheduler will have to take into account a delicate balance between *load balancing*, allocating SSDs evenly across racks, and *locality*, optimizing the connection speed by placing SSDs on the same rack as the job. These two goals represent a broader conflict between jobs which require direct-attached (PCIe) storage and jobs which require network-attached (fabric) storage, which prefer locality and load balancing, respectively. It is the responsibility of the scheduler to broker resources in order to strike a balance between locality and load balancing.

2 METHODOLOGY

The *objective* of this research is to discover ideal scheduling policies for Rabbit node architecture. The process for this involves devising a set of metrics, diagnosing problems with the current policies according to those metrics, and finally evaluating potential solutions to those problems. In order to perform evaluations in steps 2 and 3, we used the Flux resource-query utility, which simulates scheduling job requests across a set of available resources given a scheduling policy (as in section 2.2), outputting all the necessary information about timings (e.g. when a job is scheduled and how long it took to schedule) and allocations. In order to use this utility,



(a) Traditional Architecture



(b) Rabbit Node Architecture

Figure 1: Changes in Supercomputer Architecture Demonstrated

we needed a simulated resource set, and the resource set we chose supposes the hypothetical installation of Rabbit nodes on the Sierra supercomputer. This is useful because the large scale of Sierra makes it comparable to other supercomputers where this architecture is expected to be used such as the upcoming El Capitan.

2.1 Metrics

When considering which metrics were most relevant to this study, we wanted to pick basic metrics and metrics which would specifically address the locality problem inherent in the Rabbit architecture. For simple metrics, we consider makespan, average job wait time, and SSD utilization, as well as the time that it takes to schedule each job. For special-case metrics, we have a measure of rack imbalance as the standard deviation of allocated SSDs contained in each rack across the cluster, and a measure of binary locality as the sum of SSDs which are allocated on different racks from the CPUs of their job allocation (i.e. each SSD contributes one

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

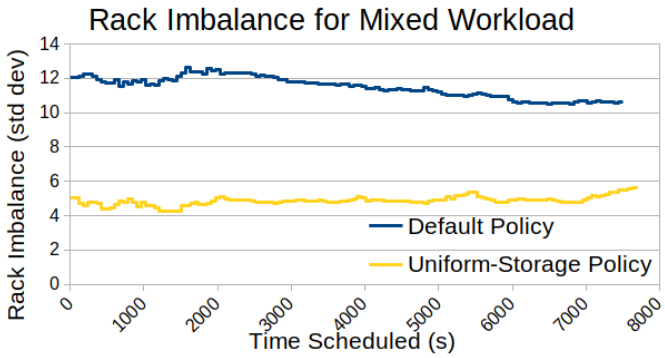


Figure 2: The rack imbalance of uniform-storage and default policies

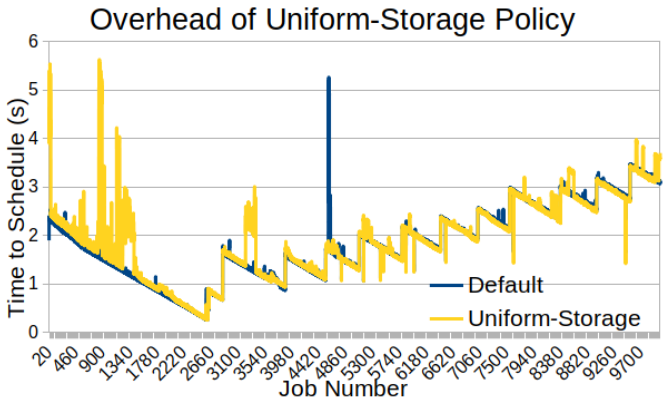


Figure 3: The overhead of the uniform-storage policy

hop or zero hops, depending on whether it's on a different rack from the allocated CPUs or the same).

2.2 Policies

Scheduling policies traverse the graph representing the resource set in depth-first order to find resources that match the request. The three major policies that we evaluate are default (or high) policy, which is a fairly straightforward matching policy which prefers higher resource IDs for match, first policy, which just grabs the first available match with minimal checking performed, and uniform-storage policy, which prefers to match to SSDs from racks with fewer SSDs allocated. Of these policies, evaluating uniform-storage policy as a potential baseline for scheduling on Rabbit architecture was considered a priority of this research.

3 EVALUATIONS

3.1 Rack Imbalance Problem

The problem of rack imbalance can be ameliorated by using the uniform-storage policy. Figure 2 shows rack imbalance for a mixed workload (one third each of network-attached

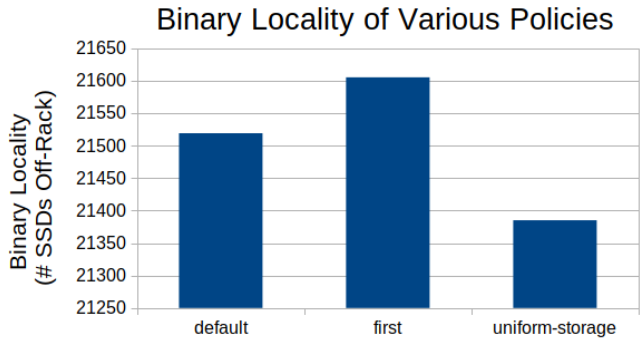


Figure 4: Comparison of Binary Locality of Scheduling Policies

Rabbit, direct-attached Rabbit, and non-Rabbit jobs, with other characteristics taken from Lassen traces). This shows an average 57.6% decrease in imbalance across the whole workload. Uniform-storage policy has some (relatively small) overhead, as shown in figure 3.

3.2 Locality Problem

It's reasonable to expect 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.

4 CONCLUSIONS AND FUTURE WORK

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 valuable as a baseline policy for those types of requests.

Future work in this project will include evaluating more policies. For example, exploration of a locality-optimized policy. Also, further network optimizations are expected, such as optimizing for the El Capitan slingshot network.

5 ACKNOWLEDGMENT

Prepared by LLNL under Contract DE-AC52-07NA27344.