Rack Level Scheduling for Containerized Workloads

Qiumin Xu
University of Southern California
qiumin@usc.edu

Krishna T. Malladi
Samsung Semiconductor
k.tej@samsung.com

Manu Awasthi
IIT Gandhinagar
manua@iitgn.ac.in

1 INTRODUCTION

Many large scale cloud services today that target a variety of application domains are hosted in large data centers. Such services are often interactive, resulting in sensitivity to system latency. Therefore, high performance storage solutions such as NVMe SSDs and NVMe-over-Fabrics (NVMf) that can provide lower I/O access latencies and higher throughput are becoming prevalent in data centers [1]. Furthermore, NVMf allows servers in the same rack to share high performance storage with performance similar to that of the direct attached devices.

In this paper, we propose an efficient mechanism to schedule datacenter workloads within a server rack sharing storage resources using NVMf. Current large scale server systems have a datacenter level scheduler that centralizes decision-making after considering the applicationâĂŹs Quality-of-Service (QoS) requirements and the underlying server level resources. The current schedulers consider resources like CPU cores and memory but have limited support for the storage system. In general, the scheduling mechanism attempts to minimize data movement from storage by locating jobs closer to data.

While this style of scheduling has the benefit of global resource visibility and the ability to utilize complex scheduling algorithms, it suffers from three important shortcomings: 1) The scheduling does not account for the presence of high performance storage drives like NVMe SSDs that substantially lower storage latency. These also do not consider other important factors of the storage subsystem like capacity, sequential/random read/write bandwidth, storage queues, garbage collection, wear leveling, write amplification, effects of flash translation layer and overprovisioning. 2) The scheduling also introduces additional, centralized complexity to take corrective action in the case where the algorithm has incorrectly located workloads across the datacenter. 3) Even if it were to perform such correction, it cannot account for the relative ease with which NVMf allows NVMe drives to obtain the benefits of data locality even with remote execution.

We introduce a second level of scheduling intelligence at the rack level that addresses all the aforementioned problems. In particular, we apply this to high performance distributed SSD drives in the presence of NVMf networking stack. Moreover, we consider these systems running containerized applications that are known to maximize the system utilization without application level interference.

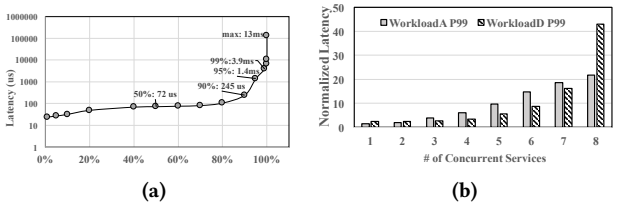

Figure 1: (a) random write-access latency distribution of an example SSD running a datacenter-level scheduler (b) normalized tail latency when running current Cassandra services (results are normalized to the p99 latency of workload A)

2 MOTIVATIONAL DATA

We first present results highlighting issues with existing datacenter level scheduling. Figure 1a shows the latency distribution of a high performance Samsung NVMe SSD running containerized applications. 80% of the storage access latency is within 100us, while 1% of the requests have latency longer than 3.9 ms. Figure 1b shows that this long tail latency increases exponentially as more services are concurrently executing on the same server and competing for resources. For each container, we loaded 100 million/100GB records in the database. The test run consisted of 100 million queries of workload A (50% reads, 50% updates, zipfian distribution) and workload D (95% reads, 5% inserts, uniform distribution) from 16 YCSB client threads. Another interesting observation is that of the eight workloads, the worst case p99 latency is much higher than the best p99 latency. This indicates that shifting the worst performing job to another node with more resources will result in decreasing the large gap between best and worst case latencies. Traditionally, this has been very difficult since migrating jobs between machines also meant migrating the data associated with those jobs. However, with techniques like NVMf becoming increasingly common within a rack, this hurdle is becoming easier to overcome. Therefore, this motivates us to design a new rack âĂŞ level scheduling algorithm that is able to reduce tail latency more efficiently.

3 RACK LEVEL SCHEDULING

As mentioned above, NVMf enables multiple server nodes in the same rack to share remote NVMe storage through high speed Ethernet. With shared storage, the job-migration

overheads are substantially reduced, since no data needs to be moved. With this insight, we describe the key components of the rack-level scheduling algorithm, namely, a tail latency anomaly monitor, a target discover unit and a service migration unit.

Tail Latency Anomaly Monitor (TLAM). TLAM is launched either as a background daemon or as part of storage stack driver in order to constantly monitor and tag each job with the p99 latency. If the TLAM detects p99 violation for a job, it will mark and send the job id number to the target discover unit.

Target Discover Unit (TDU). This unit samples system utilization information locally to share with other nodes in the same rack. It also makes decisions locally as to whether the job can be migrated and to which node. TDU also exchanges system utilization information with remote nodes within the same rack following a token ring like topology. The token methodology ensures that only one node can perform decision-making about job migration at a time avoiding hazards where multiple nodes could flood the same target node when offloading jobs.

The decision algorithm. Proper assignment of the new job will maximize the reward function which is a linear weighted function of number of available CPUs, size of available memory and available network and disk bandwidth:

$$Reward_{j,i}, = A_i * R_{cpu_j} + B_i * R_{mem_j} + C_i * R_{net_j} + D_i * R_{disk_j}$$

where: node i is the node making migration decision, j is a candidate target node. A, B, C, D are weights adjusted according the jobâĂŹs requirement for resources. R_{cpu}, R_{mem}, R_{net}, R_{disk} are the ratios of increased resource deltas in the candidate machine compared with available resources of the local machine to the amount of that resource ideally required by the job. If the reward is less than zero, the job will not be migrated to the remote node. Otherwise, it will identify the target node, which maximizes the rewards among all candidate nodes in the rack. The disk bandwidth is equivalent to NVMf network bandwidth when nodes are accessing storage remotely. While the current systemâĂŹs reward is focused on reducing tail latency, systems can implement power and cooling costs to increase server consolidation.

Service Migration Unit (SMU). Enabled by latest NVMf technology, nodes on the same rack are connected to a shared storage network. In this case, migrating a job to another node becomes much easier, since data on shared storage doesn't need to be migrated. The process states inside cache and memory, are flushed down to the NVMf storage before migration. In case where NVMf bandwidth is temporally unavailable, the states will first be flushed to a local disk. After a small wait period (e.g. 2 mins), the flush through NVMf will be retried.

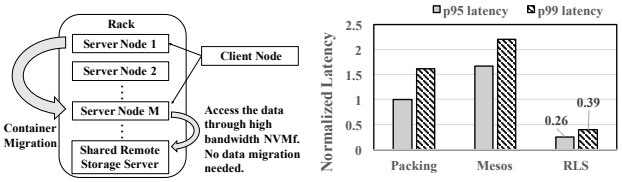

Figure 2: (a) system setup of the preliminary results (b) comparison of tail latencies of workload D (results are normalized to p95 latency of the baseline packing method)

4 RESULTS

The experimental setup consists of four server nodes (Figure 2a). Each has a dual-socket Intel Xeon E5 core, with 48 hyper-threaded CPUs. We use three nodes for Cassandra client and server configuration - one client driving YCSB traffic (over 10 Gbps ethernet) onto the other two server nodes that run multiple, containerized versions of Cassandra. The fourth node is configured as an NVMf target storage server that exports Samsung PM1725 NVMe SSDs as NVMf targets to the server nodes over a 40 Gbps, RDMA capable ethernet fabric using the RoCE protocol.

We measure and compare the tail latencies of the following scenarios in Figure 2b: (1) **Packing.** 8 Cassandra containers are packed and concurrently run on one server; (2) **Mesos.** We setup mesos + marathon + zookeeper frameworks to allocate Cassandra containers[1] (3) **Rack Level Scheduler (RLS).** We monitor various resource utilizations and determine proper assignment of containers based on the reward function. As a result, four containers are scheduled on server node 1 and other four containers on server node 2. With benefits from NVMf, the containers allocated on Node 2 can still access their data on the shared remote storage server. Overall, this allocation of the rack level scheduler leads to significant reduction of tail latencies. While the p95 latency was reduced by 3.8×, the p99 latency was reduced by 3.9× compared to baseline packing strategy.

5 CONCLUSIONS

With the advent of high performance remote storage techniques such as NVMf, workload migration between servers on the same rack becomes much cheaper. We propose rack level scheduling that exploits fast, remote storage, to provide 3.8× reduction in tail latency.

REFERENCES

[1] Q. Xu, et al., Performance analysis of containerized applications on local and remote storage, in Proc. of MSST, 2017

[2] Jeffrey Dean and Luiz André Barroso, The tail at scale, in Commun. ACM,2013

[1]Note that Mesos doesn't have disk bandwidth allocation support yet, and therefore is not very effective in scheduling containerized databases.