Adaptive Resource Allocation in Tiered Storage Systems

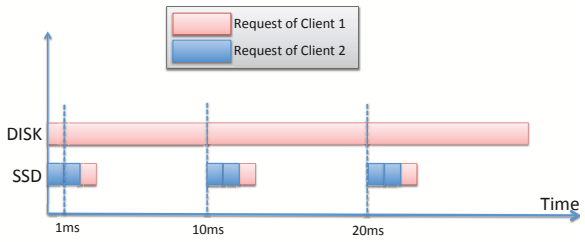
Hui Wang *Peter Varman*
hw5@rice.edu *pjv@rice.edu*
Rice University, USA

Increased consolidation in virtualized datacenters and public clouds has raised the importance of allocating shared server resources fairly among multiple tenants. In the storage domain, tiered storage made up of heterogeneous memory and storage devices are now the norm in high-end systems. In this paper we consider a two-tiered storage system made up of SSDs and hard disks (HDs), and address some of the challenges in achieving both high utilization and fair allocation. Our work is complementary to fairness versus efficiency tradeoffs studied in the context of sequential versus random IOs (e.g. [6, 5]). For tiered storage, [1, 2, 3] emphasized fairness but did not explicitly consider system utilization.

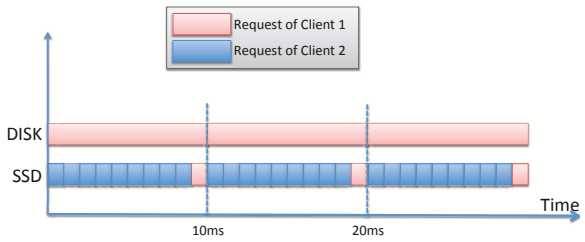
The storage system being considered is composed of SSDs and HD arrays. A client makes IO requests that may be served from either the SSD (a *hit*) or the HD (a *miss*), based on the *hit ratio*. The hit ratio will change with different application phases, but is relatively stable within a phase. The client hit ratio is monitored and used as an input to our resource allocator and scheduler.

We illustrate the tradeoff between utilization and fairness with a simple example. Suppose the HD and SSD have throughputs of 100 IOPS and 1000 IOPS respectively. A fair scheduler assigns throughputs in proportion to the assigned weights of the clients. Suppose two continuously-backlogged clients 1 and 2 have equal weights, and hit ratios of $h_1 = 0.5$ and $h_2 = 1.0$. The access pattern using a fair scheduler is shown in Figure 1(a). The throughputs for the two clients are 200 IOPS each. The utilization of the disk is 100% but the SSD is only 30%. To fully utilize the SSD, the client weights are changed to 2:9. With this ratio, the throughput of client 1 is still 200 IOPS but 2's throughput increases to 900 IOPS (see Figure 1(b)). While the relative allocations are no longer 1 : 1, the system throughput increases from 400 IOPS to 1100 IOPS. Furthermore, in this example, the throughput of client 1 is not reduced by the increased allocation to 2.

In other cases, the allocation of a client may decrease significantly when the weights are changed to increase utilization. For instance, suppose the HD throughput is 200 IOPS, the clients had hit ratios $h_1 = 0.1$ and $h_2 = 0.9$, and equal weights. In this case, their throughputs under fair scheduling would be 200 IOPS each, but the SSD utilization is only 20%. Changing the ratio of the weights to 1 : 11 results in 100% utilization of both devices, but the allocation of client 1 falls to 100 IOPS while the other increases to 1100 IOPS.



(a) Access pattern of example with weight 1 : 1



(b) Access pattern of example with weight 2 : 9

Figure 1: Access pattern of example

We propose to maximize the system utilization in a multi–tiered storage system, by dynamically computing weights for the clients, based on their measured hit ratios. However, to prevent the allocation of an individual client from falling too low (as in the second example), each client is guaranteed a minimum allocation or reservation.

Allocation Model: Each client i has a *reservation func-*

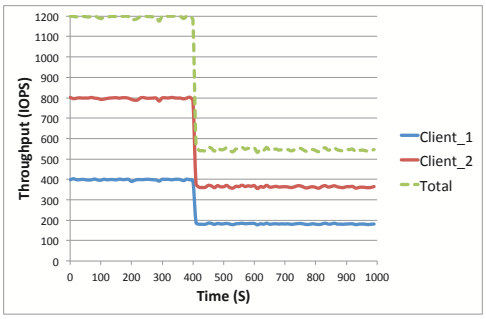
tion $R_i(h)$ and *limit function* $L_i(h)$. At periodic (or event-triggered) intervals, the system computes the allocations for each client based on its measured hit ratio. The goal is to maximize system utilization while satisfying capacity constraints and clients' reservation and limits. Admission control (see below) guarantees a feasible solution. The computed allocations are used as *dynamic weights* by a proportional weight scheduler. By allocating IOs in this ratio, all reservations will be met and excess capacity is allocated in a way that maximizes system utilization.

Let C_i denote the IOPS allocated to client i, and $\mathscr{C}_D$ and $\mathscr{C}_{SSD}$ denote the IOPS capacity of the HD and SSD. The following linear programming optimization problem can be formulated and solved using any standard LP solver: **Maximize** $\sum_{i=1}^{n} C_i$ **subject to**: **(i)** $\sum_{i=1}^{n} C_i \times h_i \leq \mathscr{C}_{SSD}$ **(ii)** $\sum_{i=1}^{n} C_i \times (1-h_i) \leq \mathscr{C}_D$ **(iii)** $C_i \geq R_i(h_i)$ **(iv)** $C_i \leq L_i(h_i)$, where h_i is the measured hit ratio for client i, $1 \leq i \leq n$. We must also deal with changes to the hit ratio between the resource-allocation instants. We use a scheduler similar to mClock [4] to ensure that reservations continue to be met even as the server capacity varies due to changes in the hit ratio. The computed weights are used as the shares, and reservations and limits are dynamically updated based on the changing hit ratios.

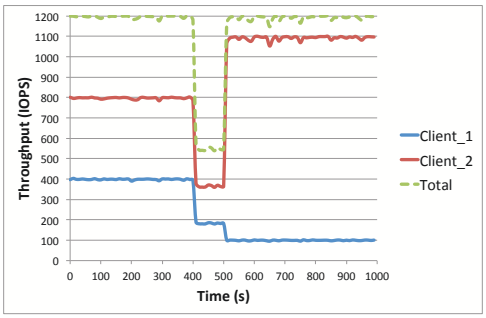
An additional issue to be addressed is *admission control*, which must guarantee enough system capacity to meet the reservations. Since throughputs vary with hit ratio, this is a challenging issue. An IOPS that is reasonable when accesses are mainly to SSD, will also require a huge HD bandwidth reservation in case the hit ratio falls. We therefore specify the reservation of a client i by a non-decreasing *function* $R_i(h)$ of the hit ratio. One can check whether all reservation can be met by calculating the maximum aggregate load on each device; this will be the maxima of the functions $\sum_i h \times R_i(h)$ (SSD) and $\sum_i (1-h) \times R_i(h)$ (HD).

Evaluation: Preliminary evaluation is done using a process-driven simulator. Figure 2 shows the simulation results with a HD of 200 IOPS and SSD of 1000 IOPS. There are two clients with hit-ratios of 0.5 and 1.0. At time 410s the hit-ratios of the clients fall to 0.1 and 0.9 respectively. We assume reservations and limits of $(0, 2000)$ for both clients. Figure 2(a) and Figure 2(b) show the results without and with adaptive allocation respectively. The adaptive approach pulls the utilization back up when the new allocations are made at 500s. If we assigned client 1 a reservation of 125 IOPS, then after 500s it gets its reservation instead of 100 IOPS. However, the system cannot achieve 100% utilization, and client 2 only receives 875 IOPS at this point.

We continue to explore the tradeoffs between fairness and allocation in tiered storage with system implementations and by extending the model to include shares.



(a) Without adaptive allocation



(b) With adaptive allocation

Figure 2: Throughput (a) without dynamic controls and (b) with adaptive weights

Acknowledgements: Support for this research under NSF Grant CNS 0917157 is gratefully acknowledged.

References

[1] ELNABLY, A., DU, K., AND VARMAN, P. Reward scheduling for QoS in cloud applications. In *12th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing* (2012).

[2] ELNABLY, A., AND VARMAN, P. Application-sensitive QoS scheduling in storage servers. In *ACM Symposium on Parallelism in Algorithms and Architecture* (2012).

[3] ELNABLY, A., WANG, H., GULATI, A., AND VARMAN, P. Efficient QoS for multi-tiered storage systems. In *4th USENIX Hot Storage Workshop* (Berkeley, CA, USA, 2012), HotStorage'12, USENIX Association, pp. 6–6.

[4] GULATI, A., MERCHANT, A., AND VARMAN, P. *m*Clock: Handling Throughput Variability for Hypervisor IO Scheduling . In *USENIX OSDI* (2010).

[5] POVZNER, A., KALDEWEY, T., BRANDT, S., GOLDING, R., WONG, T. M., AND MALTZAHN, C. Efficient guaranteed disk request scheduling with Fahrrad. *SIGOPS Oper. Syst. Rev. 42*, 4 (2008), 13–25.

[6] WACHS, M., ABD-EL-MALEK, M., THERESKA, E., AND GANGER, G. R. Argon: performance insulation for shared storage servers. In *USENIX FAST* (Berkeley, CA, USA, 2007).