Towards Predictable Multi-Tenant Shared Cloud Storage

David Shue*, Michael J. Freedman*, and Anees Shaikh†
*Princeton University, †IBM TJ Watson Research Center

An increasing number and variety of enterprises are moving workloads to cloud platforms. Whether serving external customers or internal business units, cloud platforms typically allow multiple users, or *tenants*, to share the same physical server and network infrastructure, as well as use common platform services. Examples of these shared, multi-tenant services include key-value stores, block storage volumes, message queues, and notification services. These leverage the expertise of the cloud provider in building, managing, and improving common services, and enable the statistical multiplexing of resources between tenants for higher utilization.

Because they rely on shared infrastructure, however, these services face two key, related issues:

- ***Multi-tenant interference and unfairness:*** Tenants simultaneously accessing shared services contend for resources and degrade performance.
- ***Variable and unpredictable performance:*** Tenants often experience significant performance variations, e.g., in response time or throughput, even when they can achieve their desired mean rate [2, 8, 14, 16].

These issues limit the types of applications that can migrate to multi-tenant clouds and leverage shared services. They also inhibit cloud providers from offering differentiated service levels, in which some tenants can pay for performance isolation and predictability, while others can choose standard "best-effort" behavior.

Shared back-end storage services face different challenges than sharing server resources at the virtual machine (VM) level. These stores divide tenant workloads into disjoint partitions, which are then distributed (and replicated) across service instances. Rather than managing individual storage partitions, cloud tenants want to treat these storage systems as black boxes, in which *aggregate* storage capacity and request rates can be elastically scaled on demand. Resource contention arises when tenants' partitions are co-located, and the degree of resource sharing between tenants may be significantly higher and more fluid than with coarse VM resource allocation. Particularly, as tenants may use only a small fraction of a server's throughput and capacity, restricting nodes to a few tenants may leave them highly underutilized.

To improve predictability for shared storage systems with a high degree of resource sharing and contention, we target global *max-min fairness*. Under max-min fairness, no tenant can gain an unfair advantage over another when the system is loaded, i.e., each tenant will receive its weighted fair share. Moreover, given its work-conserving nature, when some tenants use less than their full share, unconsumed shares are divided among the rest to ensure high utilization. In all cases, each tenant enjoys performance (latency) isolation. While the mechanisms we propose may be applicable to a range of services with shared-nothing architectures [12], we focus our design and evaluation on a replicated key-value storage service, which we call *PISCES* (*P*redictable *S*hared *C*loud *S*torage).

Providing fair resource allocation and isolation at the *service* level is confounded by variable demand to different service partitions. Even if tenant objects are uniformly distributed across their partitions, per-object demand is often skewed, both in terms of request rate and request size. This imbalance in object popularity may create hotspots for particular partitions [4]. Moreover, different request workloads may stress different server resources (e.g., small requests are interrupt limited, while large requests are bandwidth limited). In short, assuming that each tenant requires the same proportion of resources per partition can lead to unfairness and inefficiency.

To address these issues, PISCES introduces a novel decomposition of the global fairness problem into four mechanisms based on the primal-allocation method of distributed convex optimization [11]. Operating on different timescales and with different levels of system-wide visibility, these mechanisms complement one another to ensure fairness under resource contention and variable demand. Figure 1 shows the high-level architecture.

(1) Partition Placement, computed at a centralized controller, ensures a feasible fair allocation by assigning (or remapping) tenant partitions (p) to nodes according to per-partition demand collected at the service nodes. Using these statistics, the controller determines each tenant's per-partition fair-share from their global weights and it moves any partitions that violate per-node resource constraints to nodes with available capacity (long timescale).

(2) Weight Allocation distributes overall tenant fair shares across the system where most needed, i.e., skewing shares to the popular partitions, by adjusting local

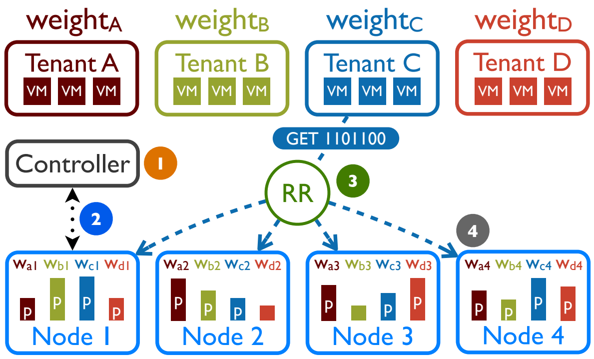


Figure 1: PISCES multi-tenant storage architecture.

per-tenant weights at each node (w_{tn}). To preserve global fairness, the controller performs *reciprocal swaps*—if tenant a takes weight from b at node x, then a must give back to b on a different node y—in an effort to minimize overall system latency (medium timescale).

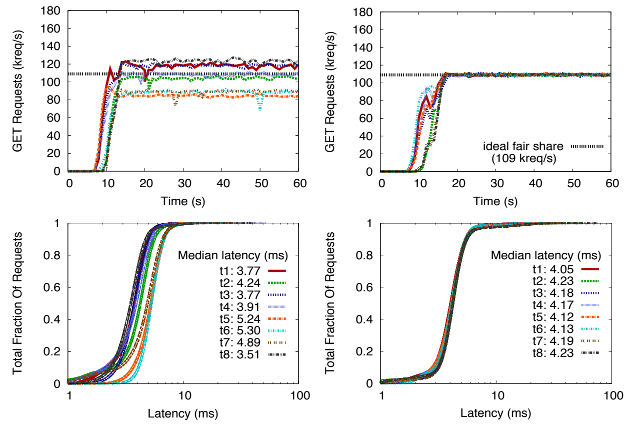
(3) Replica Selection at request routers (RR) improves both fairness and performance by directing tenant requests to service nodes in a local weight-sensitive manner. It uses a FAST-TCP [15]-like algorithm that not only selects a replica based on request latency but also throttles the request rate to enhance performance isolation (real-time).

(4) Weighted Fair Queuing at service nodes enforces performance isolation and fairness according to the local tenant weights and workload characteristics. Since workloads vary between tenants and can stress different resources, we enforce *dominant resource fairness* [3] between tenants over multiple resources by extending traditional deficit-weighted round robin, while still providing max-min fairness at the service level (real-time).

To our knowledge, PISCES is the first system to provide system-wide per-tenant fair resource sharing across all service instances. Most systems proposed in the research community either share resources on a single node among multiple clients [6, 13], share system-wide resources for a single tenant [5, 9], or allocate resources on a coarse-grained level [7, 10]. Further, since each tenant receives a weighted share of the total system capacity, PISCES can also provide minimal performance guarantees (reservations) given sufficient provisioning.[1] In comparison, recent commercial systems that offer request rate guarantees (i.e., Amazon DynamoDB) do not provide fairness, assume uniform load distributions across tenant partitions, and are not work conserving.

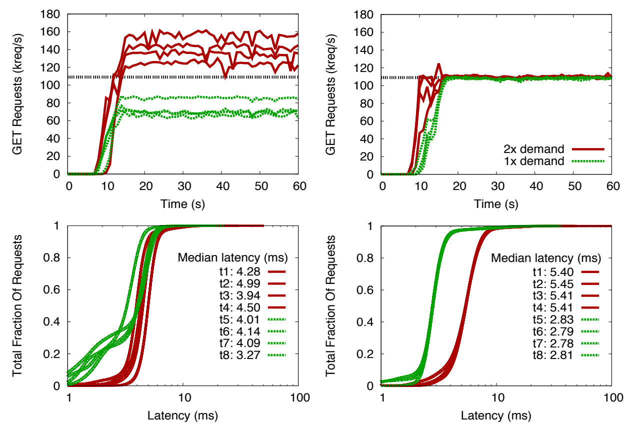
PISCES is also designed to support high server utilization, e.g., high request rates (100,000s requests per second, per server) or full bandwidth usage (Gbps per server). Through careful system design, our prototype, based on the Membase key-value storage system [1], suffers $<3\%$ overhead for 1KB requests and actually outperforms the unmodified, non-fair version for small (interrupt-bound) requests. Commercial offerings like DynamoDB seem to expect fairly low rates (e.g., rates higher than 10,000 reqs/s require special arrangements).

[1] Tenants that do not adopt (pay for) this higher level of service assurance form a single best-effort class with a minimum total share.



(a) Membase (no queuing) (b) PISCES (WFQ + RS)

Figure 2: System throughput fairness (8 tenants)



(a) Membase (no queuing) 2x (b) PISCES (WFQ + RS) 2x

Figure 3: System performance isolation (8 tenants)

Through experimental evaluation, we demonstrate that PISCES significantly improves the multi-tenant fairness and isolation properties of our key-value store. PISCES achieves near ideal throughput fairness for 8 tenants on an 8 node storage system (0.97 Max-Min Ratio vs. 0.68 MMR for unmodified Membase), as shown in Figure 2. PISCES also provides strong performance isolation between tenants: Figure 3 shows tenants with equal shares but unequal demand, as 4 of the tenants send requests at twice the normal rate. Despite the excess load, PISCES maintains 0.97 MMR throughput fairness for all tenants, while only penalizing the request latency of tenants with 2x demand. Additional experiments, omitted due to space constraints, show similar results across a range of object sizes (from 10 bytes to 10 kB), with different read/write workload mixes, and under highly skewed and dynamic partition distributions that require PISCES to (re)allocate local weights according to the shifting demand.

References

[1] Couchbase. Membase. `http://www.couchbase.org/`, Jan. 2012.

[2] S. L. Garfinkel. An evaluation of Amazon's grid computing services: EC2, S3 and SQS. Technical Report TR-08-07, Harvard Univ., 2007.

[3] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, and I. Stoica. Dominant resource fairness: Fair allocation of multiple resource types. In *NSDI*, Mar. 2011.

[4] P. B. Godfrey and I. Stoica. Heterogeneity and load balance in distributed hash tables. In *INFOCOM*, Mar. 2005.

[5] A. Gulati, I. Ahmad, and C. A. Waldspurger. PARDA: Proportional allocation of resources for distributed storage access. In *FAST*, Feb. 2009.

[6] A. Gulati, A. Merchant, and P. J. Varman. mClock: Handling throughput variability for hypervisor IO scheduling. In *OSDI*, Oct. 2010.

[7] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. Katz, S. Shenker, and I. Stoica. Mesos: A platform for fine-grained resource sharing in the data center. In *NSDI*, Mar. 2011.

[8] A. Iosup, N. Yigitbasi, and D. Epema. On the performance variability of production cloud services. In *CCGrid*, May 2011.

[9] J. C. McCullough, J. Dunagan, A. Wolman, and A. C. Snoeren. Stout: an adaptive interface to scalable cloud storage. In *USENIX Annual*, June 2010.

[10] P. Padala, K.-Y. Hou, K. G. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, and A. Merchant. Automated control of multiple virtualized resources. In *EuroSys*, Mar. 2009.

[11] D. Palomar and M. Chiang. A tutorial on decomposition methods for network utility maximization. *JSAC*, 24(8):1439–1451, Aug. 2006.

[12] M. Stonebraker. The case for shared nothing. *Database Engineering*, 9(1):4–9, Mar. 1986.

[13] M. Wachs, M. Abd-el-malek, E. Thereska, and G. R. Ganger. Argon: Performance insulation for shared storage servers. In *FAST*, 2007.

[14] J. Wang, P. Varman, and C. Xie. Optimizing storage performance in public cloud platforms. *J. Zhejiang Univ. – Science C*, 11(12):951–964, Dec. 2011.

[15] D. X. Wei, C. Jin, S. H. Low, and S. Hegde. Fast TCP: Motivation, architecture, algorithms, performance. *Trans. Networking*, 14(6):1246–1259, Dec. 2006.

[16] M. Zaharia, A. Konwinski, A. D. Joseph, R. Katz, and I. Stoica. Improving mapreduce performance in heterogeneous environments. In *OSDI*, Dec. 2008.