vFair: Latency-Aware Fair Storage Scheduling via Per-IO Cost-Based Differentiation

Hui Lu† , Brendan Saltaformaggio† , Ramana Kompella†‡ , Dongyan Xu†

†Department of Computer Science, Purdue University, ‡Google Inc.

Motivation

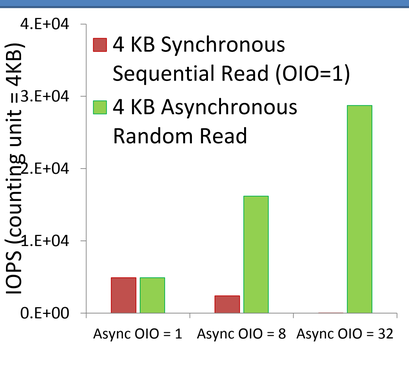


Fig. Varying IO concurrency (OIO #).

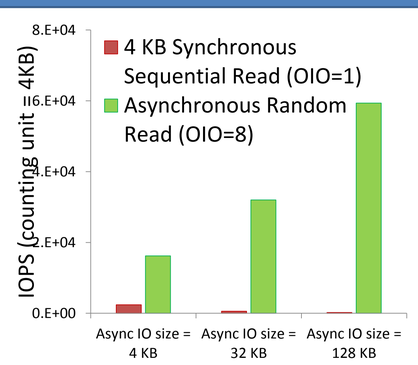


Fig. Varying IO size.

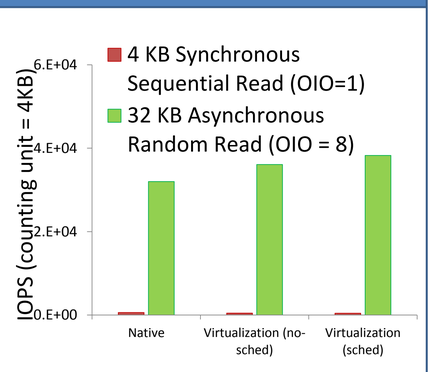


Fig. Native vs virtualization setup.

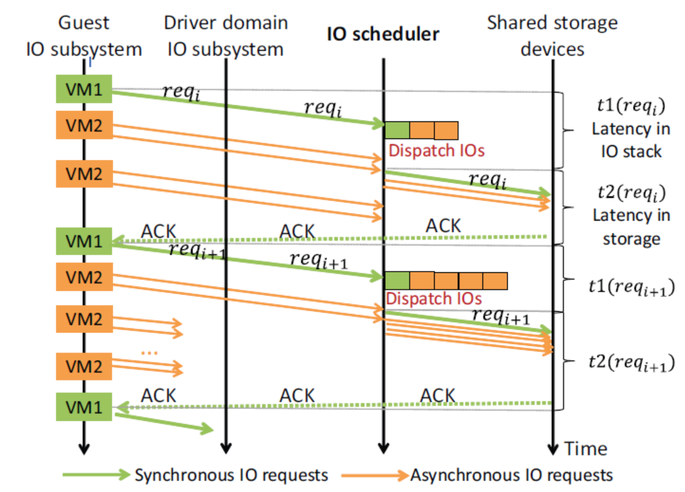


Fig. An illustration of unfairness.

Observation: Low IO-concurrency workloads receive unfair service times.

Root cause: The arrival of synchronous IO requests can be greatly delayed by asynchronous IOs.

Overview

Problem

- An IO scheduler, providing **fairness** and **isolation** while maintaining high resource **utilization,** regardless of IO patterns becomes imperative.

vFair Overview

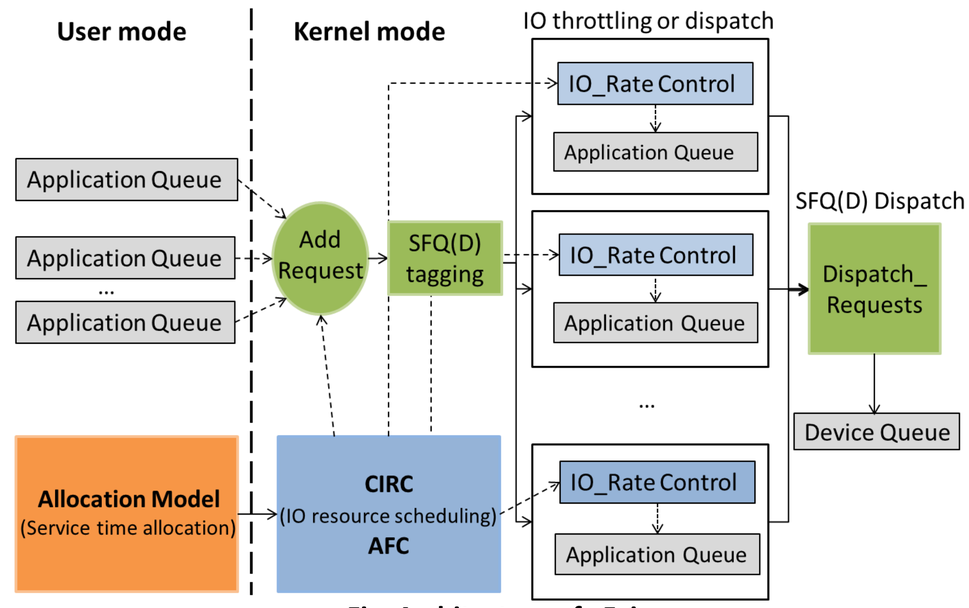


Fig. Architecture of vFair.

- A fine-grained service-time based allocation model.
- Three scheduling strategies balancing fairness and efficiency.
- A two-level scheduling architecture for realizing the model and scheduling strategies.

Per-IO Cost Allocation Model

Proportional Service Time Allocation

$$T_i = \frac{w_i}{\sum_k w_k} T$$

In practice, it is **NOT** possible to accurately measure the service time.

Key Idea – to use saturation throughput P_i

$$T_i = f(\frac{w_i}{\sum_k w_k} \cdot P_i)$$

- Each VM's workload can be modeled as an IO pattern.
- Given a particular IO pattern, it corresponds a saturation throughput P_i.
- P_i refers to the theoretical IO throughput that a VM could receive *in isolation*.
- P_i is directly related to T -- P_i refers to full utilization which is T.

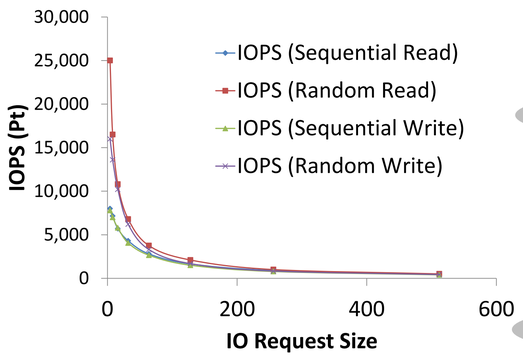


Fig. Saturated IOPS performance vs. IO request sizes for a single SSD

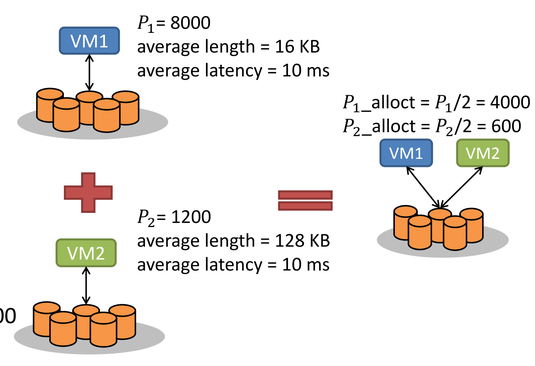


Fig. VM1 and VM2 should each obtain 50% of the shared storage throughput.

Model Approximation – to obtain P_i

$$1/P_i = \alpha \cdot \beta / P_{seq\_rd} + \alpha \cdot (1- \beta)/P_{rand\_rd} + (1- \alpha) \cdot \gamma / P_{seq\_wr} + (1- \alpha) \cdot (1 - \gamma)/P_{rand\_wr}$$

- **Posterior Knowledge**: VM's runtime IO access pattern -- α, β and γ.
- **Prior knowledge**: Basic IO pattern's P_i, constructed offline -- P_{seq_rd}, P_{rand_rd}, P_{seq_wr} and P_{rand_wr}.

Scheduling Strategy & Two-level Architecture

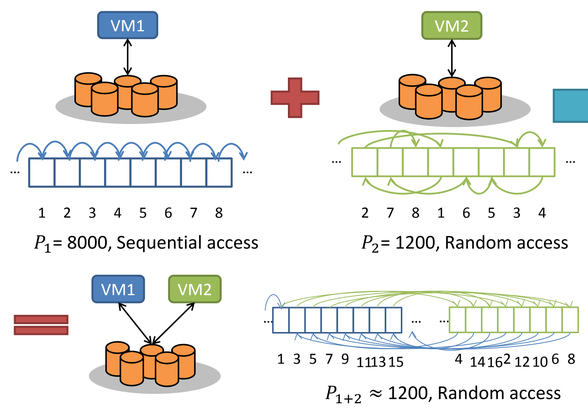


Fig. The total saturation performance becomes P1+2 = 1200 (the worst case).

Due to **IO-blender effect**, using static P_i to estimate storage capacity would become **imprecise** over time.

- **BOPS** does not take blender effect into account.
- **BAPS** ensures high efficiency with best-effort fairness guarantees.
- **BSPS** ensures the best fairness guarantees.

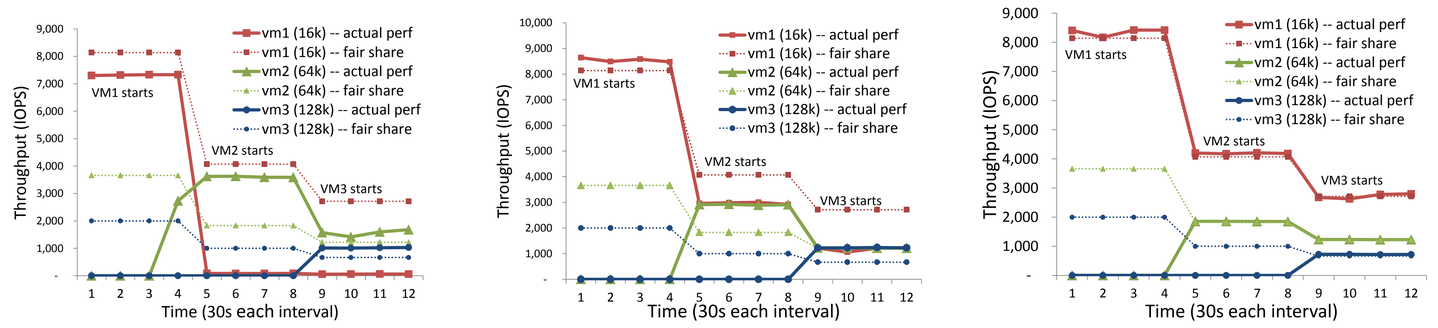
Scheduling Strategy	Fairness Guarantee	Work-conserving?	High Utilization?
BOPS(vFair)	Good	Yes	Yes
BAPS(vFair)	Better	Yes	Yes
BSPS(vFair)	Best	Yes	No
PS(mClock)	Good	Yes	Yes
CFQ(Linux)	Bad	No	No
Time-Quanta based scheduler	Best	No	No

Realization?

- First level: Credit-based IO Rate Controller
- Second level: Faire queuing scheduling SFQ(D)

Evaluation

Micro-Benchmark



CFQ(Linux) SFQ(mClock) BAPS(vFair)

Fig. Throughput (IOPS) distribution among three VMs running workloads with random IOs with various Isizes.

- vFair provides very good fairness for low IO-concurrency workloads.

Application Workloads

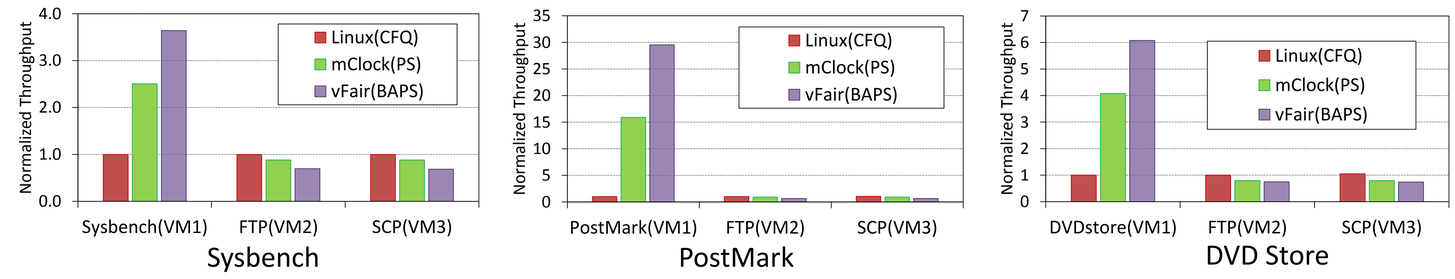


Fig. Application-level throughput (trans/second) comparisons of three schedulers using

- vFair improves fairness of storage IO scheduling by an order of magnitude.
- vFair isolates throughput of different VMs and achieves the specified proportional fair shares.