Balancing Fairness and Efficiency in Tiered Storage Systems with Bottleneck-Aware Allocation

Hui Wang and Peter Varman, *Rice University*

https://www.usenix.org/conference/fast14/technical-sessions/presentation/wang

This paper is included in the Proceedings of the
12th USENIX Conference on File and Storage Technologies (FAST '14).
February 17–20, 2014 • Santa Clara, CA USA
ISBN 978-1-931971-08-9

Balancing Fairness and Efficiency in Tiered Storage Systems with Bottleneck-Aware Allocation

Hui Wang
Rice University

Peter Varman
Rice University

Abstract

Multi-tiered storage made up of heterogeneous devices are raising new challenges in allocating throughput fairly among concurrent clients. The fundamental problem is finding an appropriate balance between fairness to the clients and maximizing system utilization.

In this paper we cast the problem within the broader framework of fair allocation for multiple resources. We present a new allocation model BAA based on the notion of per-device bottleneck sets. Clients bottlenecked on the same device receive throughputs in proportion to their fair shares, while allocation ratios between clients in different bottleneck sets are chosen to maximize system utilization. We show formally that BAA satisfies fairness properties of Envy Freedom and Sharing Incentive. We evaluated the performance of our method using both simulation and implementation on a Linux platform. The experimental results show that our method can provide both high efficiency and fairness.

1 Introduction

The growing popularity of virtualized data centers hosted on shared physical resources has raised the importance of resource allocation issues in such environments. In addition, the widespread adoption of multi-tiered storage systems [2, 3], made up of solid-state drives (SSDs) and traditional hard disks (HDs), has made the already challenging problem of providing QoS and fair resource allocation considerably more difficult.

Multi-tiered storage has several advantages over traditional flat storage in the data center: improved performance for data access and potential operating cost reductions. However, this architecture also raises many challenges for providing performance isolation and QoS guarantees. The large speed gap between SSDs and HDs means that it is not viable to simply treat the storage system as a black box with a certain aggregate IOPS capacity. The system throughput is intrinsically linked to the relative frequencies with which applications access the different types of devices. In addition, the throughput depends on how the device capacities are actually divvied up among the applications. System efficiency is a major concern for data center operators since consolidation ratios are intimately connected to their competitive advantage. The operator also needs to ensure fairness, so that the increased system utilization is not obtained at the cost of treating some users unfairly.

This brings to focus a fundamental tension between fairness and resource utilization in a system with heterogeneous resources. Maintaining high overall system utilization may require favoring some clients disproportionately while starving some others, thereby compromising fairness. Conversely, allocations based on a rigid notion of fairness can result in reduced system utilization as some clients are unnecessarily throttled to maintain parity in client allocations.

The most-widely used concept of fairness is proportional sharing (PS), which provides allocations to clients in proportion to client-specific weights reflecting their priority or importance. Adaptations of the classic algorithms for network bandwidth multiplexing [49, 9, 40] have been proposed for providing proportional fairness for storage systems [48, 45, 21]. Extended proportional-share schedulers which provide reservations and limit guarantees in addition to proportional allocation have also been proposed for storage systems [46, 19, 22, 23]. However, the vast majority of resource allocation schemes have been designed to multiplex a *single resource*, and have no natural extension to divide up multiple resources.

The question of fair division of multiple resources in computer systems was raised in a fundamental paper by Ghodsi et al [17], who advocated a model called Dominant Resource Fairness (DRF) to guide the allocation (see Section 2). A number of related allocation approaches [16, 36, 11, 25, 28, 15, 14, 13, 12, 30, 38] have since been proposed; these will be discussed in Section 6. These models deal mainly with defining the meaning of fairness in a multi-resource context. For example, DRF and its extensions consider fairness in terms of a client's dominant resource: the resource most heavily used (as a fraction of its capacity) by a client. The DRF policy is to equalize the shares of each client's dominant resource. In [12], fairness is defined in terms of proportional sharing of the empirically-measured global system bottleneck. A theoretical framework called Bottleneck-based fairness [11] proves constructively the existence of an allocation giving each client its entitlement on some global system-wide bottleneck resource. While these models and algorithms make significant advances to the problem of defining multi-resource fairness, they do not deal with the dual problem of their effect on system utilization. In general these solutions tend to over constrain the system with fairness requirements, resulting in allocations with low system utilization.

In this paper we propose a model called Bottleneck Aware Allocation (BAA) based on the notion of *local bottleneck sets*. We present a new allocation policy to maximize system utilization while providing fairness in the allocations of the competing clients. The allocations of BAA enjoy all of the fairness properties of DRF [17], like Sharing Incentive, Envy Freedom [26, 7, 6], and Pareto Optimality. However, within this space of "fair" solutions that includes DRF, it searches for alternative solutions with higher system efficiency. We prove formally that BAA satisfies these fairness properties. We use BAA as part of a two-tier allocate and schedule mechanism: one module uses a standard weighted fair-scheduler to dispatch requests to the storage system; the other module monitors the workload characteristics and dynamically recomputes the weights using BAA for use by the dispatcher, based on the mix of workloads and their access characteristics. We evaluate the performance of our method using simulation and a Linux platform. The results show that our method can provide both high efficiency and fairness for heterogeneous storage and dynamic workloads.

The rest of the paper is organized as follows. In Section 2 we discuss the difficulties of achieving both fairness and efficiency in a heterogeneous storage system. In Section 3 we describe our model and approach to balance needs of fairness and system efficiency. Formal proofs are presented in Section 4. We present some empirical results in Section 5. Related work is summarized in Section 6, and we conclude in Section 7.

2 Overview

The storage system is composed of SSDs and HD arrays, as shown in Figure 1. SSDs and HDs are independent

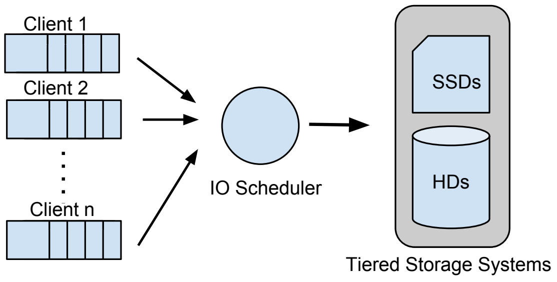


Figure 1: Tiered storage model

devices without frequent data migrations between them. A client makes a sequence of IO requests; the target of each request is either the SSD or the HD, and is known to the scheduler. An access to the SSD is referred to as a *hit* and access to the HD is a *miss*. The hit (miss) ratio of client i is the fraction of its IO requests to the SSD (HD), and is denoted by h_i (respectively m_i). The hit ratio of different applications will generally be different. It may also change in different application phases, but is assumed to be relatively stable within an application phase.

The requests of different clients are held in client-specific queues from where they are dispatched to the storage array by an IO scheduler. The storage array keeps a fixed number of outstanding requests in its internal queues to maximize its internal concurrency. The IO scheduler is aware of the target device (SSD or HD) of a request. Its central component is a module to dynamically assign *weights* to clients based on the measured miss ratios. These weights are used by the dispatcher to choose the order of requests to send to the array; the number of serviced requests of a client is in proportion to its weight. The weights are computed in accordance with the fairness policies of BAA so as to maximize system utilization based on recently measured miss ratios.

We illustrate the difficulties in achieving these goals in the next section, followed by a description of our approach in Section 3.

2.1 Motivating Examples

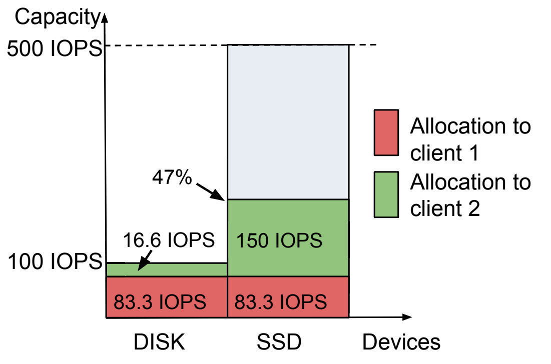
In traditional proportional fairness a single resource is divided among multiple clients in the ratio of their assigned weights. For instance, if a single disk of 100 IOPS capacity is shared among two backlogged clients of equal weight, then each client will receive 50 IOPS. A work-conserving scheduler like Weighted Fair Queuing [9] will provide fine-grained, weight-proportional bandwidth allocation to the backlogged clients; the system will be 100% utilized as long as there are requests in the system. When the IOs are from heterogeneous devices like HDs and SSDs, the situation is considerably more complicated. The device load is determined by both the allocation ratios (the relative fraction of the bandwidth assigned to clients), as well as their hit ratios. If the clients with high SSD loads have small allocation ratio, there may be insufficient requests to keep the SSD busy. Conversely, maintaining high utilization of both the HD and the SSD may require adjusting the allocations in a way that starves some clients, resulting in unfairness in the allocation.

Example I Suppose the HD and SSD have capacities of 100 IOPS and 500 IOPS respectively. The system is shared by backlogged clients 1 and 2 with equal weights and hit ratios of $h_1 = 0.5$ and $h_2 = 0.9$ respectively. Under proportional-share allocation, both clients should get an equal number of IOPS. The *unique* allocation in this case is for each client to get 166.6 IOPS. Client 1 will get 83.3 IOPS from each device, while client 2 will get 16.6 IOPS from the HD and 150 IOPS from the SSD. The HD has 100% utilization but the SSD is only 47% utilized (Figure 2(a)). In order to increase the system utilization, the relative allocation of the clients needs to be changed (Figure 2(b)). In fact, both devices can be fully utilized if the scheduler allocates 100 IOPS to client 1 (50 IOPS from the HD and 50 from the SSD), and 500 IOPS to client 2 (50 from the HD and 450 from the SSD). This is again the *unique* allocation that maximizes the system utilization for the given set of hit ratios. Note that increasing the utilization to 100% requires a 1 : 5 allocation ratio, and reduces client 1's throughput from 167 to 100 IOPS while increasing client 2's throughput from 167 to 500 IOPS.

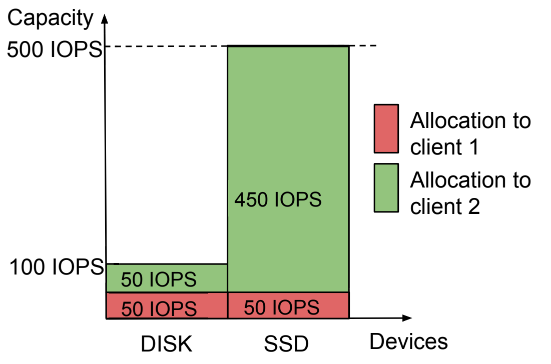
The example above illustrates a general principle. For a given set of workload hit ratios, it is not possible to precisely dictate both the relative allocations (*fairness*) and the system utilization (*efficiency*). How then should we define the allocations to both provide fairness and achieve good system utilization?

Consider next how the DRF policy will allocate the bandwidth. The dominant resource for client 1 is the HD; for client 2 it is the SSD. Suppose DRF allocates n IOPS to client 1 and m IOPS to client 2. Equalizing the dominant shares means that $0.5n/100 = 0.9m/500$ or $n : m = 9 : 25$. This results in an SSD utilization of approximately 77% (Figure 2(c)). The point to be noted is that none of these earlier approaches considers the issue of resource efficiency when deciding on an allocation. The policies deal with the question of how to set client allocation ratios to achieve some measure of fairness, but do not address how the choice affects system utilization.

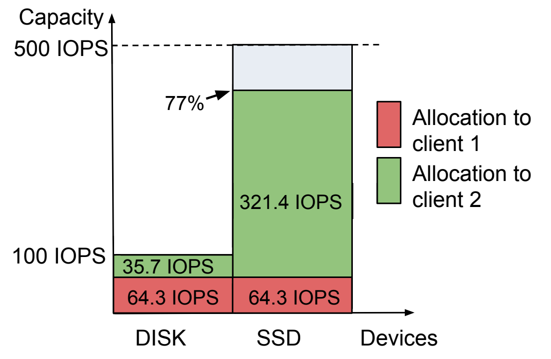
Example II In the example above suppose we add a third backlogged client, also with hit ratio 0.5. In this case, proportional sharing in a 1 : 1 : 1 ratio would result in al-



(a) Allocations in 1:1 ratio. Utilization of the SSD is only 47%



(b) Allocations with 100% utilization. Allocations are in the ratio 1 : 5.



(c) Allocations using DRF. Allocations are in the ratio 9 : 25. Utilization of the SSD is 77%.

Figure 2: Allocations in multi-tiered storage. HD capacity: 100 IOPS and SSD Capacity: 500 IOPS. The hit ratios of the clients are 0.5 and 0.9 respectively.

locations of 90.9 IOPS each; the client with hit ratio 0.9 would be severely throttled by the allocation ratio, and the SSD utilization will be only 34.5%. If the weights were changed to 1 : 1 : 10 instead, then the HD-bound clients (hit ratio 0.5) would receive 50 IOPS each, and the SSD-bound client (hit ratio 0.9) would receive 500 IOPS. The utilization of both devices is 100%. The DRF policy will result in an allocation ratio of 9 : 9 : 25, allocations of 78, 78 and 217 IOPS respectively, and SSD utilization of 55% (further reduced from the 77% of example 1). This shows that the system utilization is highly dependent on the competing workloads, and stresses how relative allocations (fairness) and system utilization (efficiency) are intertwined.

2.2 Fairness and Efficiency in Multi-tiered Storage

As discussed in the previous section, the ratio of allocation and the system utilization cannot be independently controlled. In [17], DRF allocations are shown to possess desirable fairness properties, namely:

- *Sharing Incentive*: Each client gets at least the throughput it would get from statically partitioning each resource equally among the clients[1]. This throughput will be referred to as the *fair share* of the client. In this paper we will use fair share defined by equal partition of the resources[2].
- *Envy-Freedom*: A client cannot increase its throughput by swapping its allocation with any other client. That is, clients prefer their own allocation over the allocation of any other client.
- *Pareto Efficiency*: A client's throughput cannot be increased without decreasing another's throughput.

We propose a bottleneck-aware allocation policy (BAA), to provide both fairness and high efficiency in multi-tiered storage systems. BAA will preserve the desirable fairness features listed above. However, we add a fundamentally new requirement, namely maximizing the system utilization.

The clients are partitioned into *bottleneck sets* depending on the device on which they have a higher *load*. This is a local property of a client and does not depend on the system bottleneck as in [11, 12].

In our model, clients in the same bottleneck set will receive allocations that are in the ratio of their *fair share*. However, there is *no predefined ratio* between the allocations of clients in different bottleneck sets. Instead, the system is free to set them in a way that maximizes utilization as long as Envy Freedom, Sharing Incentive and Pareto Optimality properties are preserved by the resulting allocation.

[1][36] extends the definition to weighted clients and weighted partition sizes.

[2]We can apply the framework of [36] to BAA to handle the case of unequal weights as well.

3 Bottleneck-Aware Allocation

In this section, we will discuss our resource allocation model called *Bottleneck-Aware Allocation* (BAA) and the corresponding scheduling algorithm.

3.1 Allocation Model

We begin by precisely defining several terms used in the model. The *capacity* of a device is defined as its throughput (IOPS) when continually busy. As is usually the case, this definition abstracts the fine-grained variations in access times of different types of requests (read or write, sequential or random etc), and uses a representative (e.g. random 4KB reads) or average IOPS number to characterize the device. Denote the capacity (in IOPS) of the disk as C_d and that of the SSD as C_s.

Consider a client i that is receiving a total throughput of T IOPS. This implies it is receiving $T \times h_i$ IOPS from the SSD. The fraction of the capacity of the SSD that it uses is $(T \times h_i)/C_s$. Similarly, the fraction of the capacity of the HD that it uses is $(T \times m_i)/C_d$.

Definitions

1. The *load* of a client i on the SSD is h_i/C_s and on the HD is m_i/C_d. It represents the average fraction of the device capacity that is utilized per IO of a client.

2. Define the system load-balancing point as $h_{bal} = C_s/(C_s + C_d)$. A workload with hit ratio equal to h_{bal} will have equal load on both devices. If the hit ratio of a client is less than or equal to h_{bal} it is said to be *bottlenecked* on the HD; if the hit ratio is higher than h_{bal} it is bottlenecked on the SSD.

3. Partition the clients into two sets D and S based on their *hit ratios*. $D = \{i : h_i \leq h_{bal}\}$ and $S = \{i : h_i > h_{bal}\}$ are the sets of clients that are bottlenecked on the HD and SSD respectively.

4. Define the *fair share* of a client to be the throughput (IOPS) it gets if each of the resources are partitioned equally among all the clients. Denote the fair share of client i by f_i.

5. Let A_i denote the allocation of (total IOPS done by) client i under some resource partitioning. The total throughput of the system is $\sum_i A_i$.

Example III Consider a system with $C_d = 200$ IOPS, $C_s = 1000$ IOPS and four clients p, q, r, s with hit ratios $h_p = 0.75, h_q = 0.5, h_r = 0.90, h_s = 0.95$. In this case,

$h_{bal} = 1000/1200 = 0.83$. Hence, p and q are bottlenecked on the HD, while r and s are bottlenecked on the SSD: $D = \{p,q\}$ and $S = \{r,s\}$.

Suppose the resources are divided equally among the clients, so that each client sees a virtual disk of 50 IOPS and a virtual SSD of 250 IOPS. What are the throughputs of the clients with this static resource partitioning?

Since p and q are HD-bottlenecked, they would use their entire HD allocation of 50 IOPS, and an additional amount on the SSD depending on the hit ratios. Since p's hit ratio is $3/4$, it would get 150 IOPS on the SSD for a total of 200 IOPS, while q ($h_q = 0.5$) would get 50 SSD IOPS for a total of 100 IOPS. Thus the *fair share*s of p and q are 200 and 100 IOPS respectively. In a similar manner, r and s would completely use their SSD allocation of 250 IOPS and an additional amount on the disk. The *fair share*s of r and s in this example are 277.8 and 263.2 IOPS respectively.

Our *fairness policy* is specified by the rules below. The rules (1) and (2) state that the allocations between any two clients that are bottlenecked on the same device are in *proportion to their fair share*. Condition (3) states that clients backlogged on different devices should be envy free. The condition asserts that if client A receives a higher throughput on some device than client B it must get an equal or lesser throughput on the other. We will show in Section 4 that with just rules (1) and (2), the envy-free property is satisfied between any pair of clients that belong both in D or both in S. However, envy-freedom between clients in different sets is explicitly enforced by the third constraint.

Fairness Policy

1. **Fairness between clients in D**:
$\forall i, j \in D$, $\frac{A_i}{A_j} = \frac{f_i}{f_j}$. Define $\rho_d = \frac{A_i}{f_i}$ to be the ratio of the allocation of client i to its fair share, $i \in D$.

2. **Fairness between clients in S**:
$\forall i, j \in S$, $\frac{A_i}{A_j} = \frac{f_i}{f_j}$. Define $\rho_s = \frac{A_j}{f_j}$ to be the ratio of the allocation of client j to its fair share, $j \in S$.

3. **Fairness between a client in D and a client in S**:
$\forall i \in D, j \in S$: $\frac{h_j}{h_i} \geq \frac{A_i}{A_j} \geq \frac{m_j}{m_i}$. Note that if $h_i = 0$ then only the constraint $\frac{A_i}{A_j} \geq \frac{m_j}{m_i}$ is needed.

Example IV What do the fairness policy constraints mean for the system of Example III? Rule 1 means that HD-bound clients p and q should receive allocations in the ratio 2 : 1 (ratio of their fair shares), *i.e.* $A_p/A_q = 2$. Similarly, rule 2 means that SSD-bound clients r and s should receive allocations in the ratio 277.8 : 263.2 = 1.06 : 1, *i.e.* $A_r/A_s = 1.06$. Rule 3 implies a constraint for each of the pairs of clients backlogged on different devices: (p, r), (p, s), (q, r) and (q, s):

i $h_r/h_p = 1.2 \geq A_p/A_r \geq m_r/m_p = 0.4$

ii $h_s/h_p = 1.27 \geq A_p/A_s \geq m_s/m_p = 0.2$

iii $h_r/h_q = 1.8 \geq A_q/A_r \geq m_r/m_q = 0.2$

iv $h_s/h_q = 1.9 \geq A_q/A_s \geq m_s/m_q = 0.1$

These linear constraints will be included in a linear programming optimization model in the next section.

3.2 Optimization Model Formulation

The aim of the resource allocator is to find a suitable allocation A_i for each of the clients. The allocator will maximize the system utilization while satisfying the fairness constraints described in Section 3.1, together with constraints based on the capacity of the HD and the SSD. A direct linear programming (LP) formulation will result in an optimization problem with n unknowns representing the allocations of the n clients, and $O(n^2)$ constraints specifying the rules of the fairness policy.

The search space can be drastically reduced using the auxiliary variables ρ_d and ρ_s (called *amplification factors*) defined in Section 3.1. Rules 1 and 2 require that $A_i = \rho_d f_i$ and $A_j = \rho_s f_j$, for clients $i \in D$ and $j \in S$.

We now formulate the objective function and constraints in terms of the auxiliary quantities ρ_d and ρ_s. The total allocation is:

$$\sum_{\forall k} A_k = (\sum_{i \in D} A_i + \sum_{j \in S} A_j) = (\rho_d \sum_{i \in D} f_i + \rho_s \sum_{j \in S} f_j).$$

The total number of IOPS made to the HD is:

$$\rho_d \sum_{i \in D} f_i m_i \; + \; \rho_s \sum_{j \in S} f_j m_j.$$

The total number of IOPS made to the SSD is:

$$\rho_d \sum_{i \in D} f_i h_i + \rho_s \sum_{j \in S} f_j h_j.$$

Fairness rule 3 states that: $\forall i \in D, j \in S$,

$$\frac{h_j}{h_i} \geq \frac{\rho_d f_i}{\rho_s f_j} \geq \frac{m_j}{m_i}$$

$$\frac{h_j f_j}{h_i f_i} \geq \frac{\rho_d}{\rho_s} \geq \frac{m_j f_j}{m_i f_i}.$$

$$\beta \geq \frac{\rho_d}{\rho_s} \geq \alpha.$$

where

$$\alpha = max_{i,j} \left\{ \frac{m_j f_j}{m_i f_i} \right\} \qquad \beta = min_{i,j} \left\{ \frac{h_j f_j}{h_i f_i} \right\}$$

The final problem formulation is shown below. It is expressed as a 2-variable linear program with unknowns

ρ_d and ρ_s, and four linear constraints between them. Equations 2 and 3 ensure that the total throughputs from the HD and the SSD respectively do not exceed their capacities. Equation 4 ensures that any pair of clients, which are bottlenecked on the HD and SSD respectively, are envy free. As mentioned earlier, we will show that clients which are bottlenecked on the same device will automatically be envy free.

Optimization for Allocation

$$\textbf{Maximize} \quad \rho_d \sum_{i \in D} f_i + \rho_s \sum_{j \in S} f_j \tag{1}$$

subject to:

$$\rho_d \sum_{i \in D} f_i m_i + \rho_s \sum_{j \in S} f_j m_j \leq C_d \tag{2}$$

$$\rho_d \sum_{i \in D} f_i h_i + \rho_s \sum_{j \in S} f_j h_j \leq C_s \tag{3}$$

$$\beta \geq \frac{\rho_d}{\rho_s} \geq \alpha \tag{4}$$

Example V We show the steps of the optimization for the scenario of Example III. $D = \{p,q\}$, $S = \{r,s\}$, and the fair shares $f_p = 200$, $f_q = 100$, $f_r = 277.8$ and $f_s = 263.2$. $\sum_{i \in D} f_i = 200 + 100 = 300$, $\sum_{j \in S} f_j = 277.8 + 263.2 = 541$, $\sum_{i \in D} f_i m_i = 50 + 50 = 100$, $\sum_{j \in S} f_j m_j = 27.78 + 13.2 = 41$, $\sum_{i \in D} f_i h_i = 150 + 50 = 200$, and $\sum_{j \in S} f_j h_j = 250 + 250 = 500$. Also it can be verified that $\alpha = 0.55$ and $\beta = 1.67$. Hence, we get the following optimization problem:

$$\textbf{Maximize}: \quad 300\rho_d + 541\rho_s \tag{5}$$

subject to:

$$100\rho_d + 41\rho_s \leq 200 \tag{6}$$

$$200\rho_d + 500\rho_s \leq 1000 \tag{7}$$

$$1.67 \geq \frac{\rho_d}{\rho_s} \geq 0.55 \tag{8}$$

Solving the linear program gives $\rho_d = 1.41$, $\rho_s = 1.44$, which result in allocations $A_p = 282.5$, $A_q = 141.3$, $A_r = 398.6$, $A_s = 377.6$, and HD and SSD utilizations of 100% and 100%.

We end the section by stating precisely the properties of BAA with respect to fairness and utilization. The properties are proved in Section 4.

- **P1**: *Clients in the same bottleneck set receive allocations proportional to their fair shares.*
- **P2**: *Any pair of clients bottlenecked on the same device will not envy each other. Combined with fairness policy (3) which enforces envy freedom between clients bottlenecked on different devices, we can assert that the allocations are envy free.*
- **P3**: *Every client will receive at least its fair share. In other words, no client receives less throughput than it would if the resources had been hard-partitioned equally among them. Usually, clients will receive more than their fair share by using capacity on the other device that would be otherwise unused.*
- **P4**: *The allocation maximizes the system throughput subject to these fairness criteria.*

3.3 Scheduling Framework

The LP described in Section 3.2 calculates the throughput that each client is allocated based on the mix of hit ratios and the system capacities. The ratios of these allocations make up the weights to a proportional-share scheduler like WFQ [9], which dispatches requests from the client queues.

When a new client enters or leaves the system, the allocations (*i.e.* the weights to the proportional scheduler) need to be updated. Similarly, if a change in a workload's characteristics results in a significant change in its hit ratio, the allocations should be recomputed to prevent the system utilization from falling too low. Hence, periodically (or triggered by an alarm based on device utilizations) the allocation algorithm is invoked to compute the new set of weights for the proportional scheduler. We also include a module to monitor the hit ratios of the clients over a moving window of requests. The hit ratio statistics are used by the allocation algorithm.

Algorithm 1: Bottleneck-Aware Scheduling

```
Step 1. For each client maintain statistics of its
hit ratio over a configurable request-window W.
Step 2. Periodically invoke the BAA optimizer
of Section 3.2 to compute the allocation of each
client that maximizes utilization subject to
fairness constraints.
Step 3. Use the allocations computed in Step 2
as relative weights to a proportional-share
scheduler that dispatches requests to the array in
the ratio of their weights.
```

The allocation algorithm is relatively fast since it requires solving only a small 2-variable LP problem, so it can be run quite frequently. Nonetheless, it would be desirable to have a single-level scheme in which the be desirable to have a single-level scheme in which the scheduler continually adapts to the workload characteristics rather than at discrete steps. In future work we will investigate the possibility of such a single-level scheme.

4 Formal Results

In this section we formally establish the fairness claims of BAA. The two main properties are summarized in Lemma 3 and Lemma 7, which state that the allocations made by BAA are envy free (EF) and satisfy the sharing incentive (SI) property. Table 1 summarizes the meanings of different symbols.

Symbol	Meaning
C_s, C_d	Capacity in IOPS of SSD (HD)
S, D	Set of clients bottlenecked on the SSD (HD)
ρ_s, ρ_d	Proportionality constants of *fairness policy*
f_i	Fair Share for client i
h_i, m_i	Hit (Miss) ratio for client i
h_{bal}	Load Balance Hit Ratio: $C_s/(C_s+C_d)$
n	Total number of clients

Table 1: List of Symbols

Lemma 1 finds expressions for fair shares. The fair share of a client is its throughput if it is given a virtual HD of capacity C_d/n and a virtual SSD of capacity C_s/n. A client in D will use all the capacity of the virtual HD, and hence have a fair share of $C_d/(n\times m_i)$. A client in S uses all the capacity of the virtual SSD, and its fair share is $C_s/(n\times h_i)$.

Lemma 1. *Let n be the number of clients. Then $f_i = min\{C_d/(n\times m_i), C_s/(n\times h_i)\}$. If $i\in D$, then $f_i = C_d/(n\times m_i)$; else if $i\in S$, then $f_i = C_s/(n\times h_i)$.*

Proof. The fair share is the total throughput when a client uses one of its virtual resources completely. For $i\in D$, $h_i \le h_{bal} = C_s/(C_s+C_d)$ and $m_i \ge 1-h_{bal} = C_d/(C_s+C_d)$. In this case, $C_d/(n\times m_i) \le (C_s+C_d)/n$ and $C_s/(n\times h_i) \ge (C_s+C_d)/n$. Hence, the first term is the smaller one, whence the result follows. A similar argument holds for $i\in S$. $\square$

Lemma 2 states a basic property of BAA allocations: all clients in a bottleneck set receive equal throughputs on the device on which they are bottlenecked. This is simply a consequence of *fairness policy* which requires that clients in the same bottleneck set receive throughput in the ratio of their fair shares.

Lemma 2. *All clients in a bottleneck set receive equal throughputs on the bottleneck device. Specifically, all clients in D receive $\rho_d C_d/n$ IOPS from the HD; and all clients in S receive $\rho_s C_s/n$ IOPS from the SSD.*

Proof. Let $i\in D$. From *fairness policy* (1) and lemma 1, $A_i = \rho_d f_i = \rho_d(C_d/(n\times m_i))$. The number of IOPS from the HD is therefore $A_i m_i = \rho_d C_d/n$. Similarly, for $i\in S$, $A_i = \rho_s f_i = \rho_s(C_s/(n\times h_i))$, and the number of IOPS from the SSD is $A_i h_i = \rho_s C_s/n$. $\square$

To prove EF between two clients, we need to show that no client receives more throughput on *both* the resources (HD and SSD). If the two clients are in the same bottleneck set then this follows from Lemma 2, which states that both clients will get *equal* throughputs on their bottleneck device. When the clients are in different bottleneck sets then the condition is explicitly enforced by *fairness policy* (3).

Lemma 3. *For any pair of client i,j the allocations made by BAA are envy free.*

Proof. From lemma 2, if $i,j\in D$ both clients have the same number of IOPS on the HD; hence neither can improve its throughput by getting the others allocation. Similarly, if $i,j\in S$ they do not envy each other, since nether can increases its throughput by receiving the others allocation.

Finally, we consider the case when $i\in D$ and $j\in S$. From *fairness policy* (3), $\forall i\in D, j\in S$: $\frac{h_j}{h_i} \ge \frac{A_i}{A_j} \ge \frac{m_j}{m_i}$. Hence, the allocations on the SSD for clients i and j satisfy $A_i h_i \le A_j h_j$, and the allocations on HD for clients i and j satisfy $A_i m_i \ge A_j m_j$. So any two flows in different bottleneck sets will not envy each other. Hence neither i nor j can get more than the other on both devices. $\square$

The following Lemma shows the Sharing Incentive property holds in the "simple" case. The more difficult case is shown in Lemma 6. Informally, if the HD is a *system* bottleneck (*i.e.*, it is 100% utilized) then Lemma 4 shows that the clients in D will receive at least $1/n$ of the HD bandwidth. The clients in S may get less than that amount on the HD (and usually will get less). Similarly, if the SSD is a system bottleneck, then the clients in S will receive at least $1/n$ of the SSD bandwidth.

In the remainder of this section we assume that the clients $1,2,\cdots n$, are ordered in non-decreasing order of their hit ratios, and that r of them are in D and the rest in S. Hence, $D=\{1,\cdots,r\}$ and $S=\{r+1,\cdots,n\}$.

Lemma 4. *Suppose the HD (SSD) has a utilization of 100%. Then every $i\in D$ (respectively $i\in S$) receives a throughput of at least f_i.*

Proof. Let j denote an arbitrary client in S. From *fairness policy* (3), $A_i m_i \ge A_j m_j$. That is, the throughput on the HD of a client in D is greater than or equal to the throughput on the HD of any client in S. Now, from lemma 2 the IOPS from the HD of all $i\in D$ are equal. Since, by hypothesis, the disk is 100% utilized, the total

IOPS from the HD is C_d. Hence, for every $i \in D$, the IOPS on the disk must be at least C_d/n. A symmetrical proof holds for clients in S. □

In order to show the Sharing Incentive property for clients whose bottleneck device is not the system bottleneck (*i.e.* is less than 100% utilized), we prove the following Lemma. Informally, it states that utilization of the SSD improves if the clients in S can be given a bigger allocation. The result, while intuitive, is not self evident. An increase in the SSD allocation to a client in S increases its HD usage as well. Since the HD is 100% utilized, this reduces HD allocations of clients in D, which in turn reduces their allocation on the SSD. We need to check that the net effect is positive in terms of SSD utilization.

Lemma 5. *Consider two allocations that satisfy fairness policy (1) - (3), and for which the HD has utilization of* 100% *and the SSD has utilization less than*100%. *Let* ρ_s *and* $\hat{\rho}_s$ *be the proportionally constants of clients in S for the two allocations, and let* U *and* $\hat{U}$ *be the respective system throughputs. If* $\hat{\rho}_s > \rho_s$ *then* $\hat{U} > U$. *A symmetrical result holds if the SSD is* 100% *utilized and the HD is less than* 100% *utilized.*

Proof. We show the case for HD 100% utilized. From Lemma 2, all clients in S have the same throughput $\rho_s C_s/n$ on the SSD. Define δ_s to be the difference between the SSD throughputs of a client in S in the two allocations. Since $\hat{\rho}_s > \rho_s$, $\delta_s > 0$. Similarly, define δ_d to be difference between the HD throughput of a client in D in the two allocations.

An increase of δ_s in the throughput of client $i \in S$ on the SSD implies an increase on the HD of $\delta_s \times (m_i/h_i)$. Since the HD is 100% utilized in both allocations, the aggregate allocations of clients in D must decrease by the total amount $\sum_{i\in S} \delta_s \times (m_i/h_i)$. By Lemma 2, since all clients in D have the same allocation on the HD, $\delta_d = \sum_{i\in S} \delta_s \times (m_i/h_i)/|D|$. As a result, the decrease in the allocation of client $j \in D$ on the SSD is $\hat{\delta}_s = \delta_d \times (h_j/m_j)$.

The total change in the allocation on the SSD in the two allocations, Δ is therefore: $\Delta = \sum_{i\in S} \delta_s - \sum_{j\in D} \hat{\delta}_s$. Substituting:

$$\Delta = \sum_{i\in S} \delta_s - \sum_{j\in D} \delta_d \times (h_j/m_j) \tag{9}$$

$$\Delta = |S| \times \delta_s - \sum_{j\in D} (\sum_{i\in S} \delta_s \times (m_i/h_i)/|D|) \times (h_j/m_j) \tag{10}$$

Now for all $i \in S$, $(m_i/h_i) \le (m_{r+1}/h_{r+1})$ and for all $j \in D$, $(h_j/m_j) \le (h_r/m_r)$.

Substituting in Equation 10:

$$\Delta \ge |S| \times \delta_s - |S|\delta_s \times (m_{r+1}/h_{r+1}) \times (h_r/m_r) \tag{11}$$

$$\Delta \ge |S| \times \delta_s (1 - \frac{m_{r+1}}{m_r} \times \frac{h_r}{h_{r+1}}) \tag{12}$$

Now, $m_{r+1} < m_r$ and $h_r < h_{r+1}$ since r and $r+1$ are in D and S respectively. Hence, $\Delta > 0$.

□

Finally, we show the Sharing Incentive property for clients whose bottleneck device is not the system bottleneck. The idea is to make the allocation to the clients in S as large as we can, before the EF requirements prevent further increase.

Lemma 6. *Suppose the HD (SSD) has utilization of* 100% *and the SSD (HD) has utilization less than* 100%. *Then every* $i \in S$ (*respectively* $i \in D$) *receives a throughput of at least* f_i.

Proof. We will show it for clients in S. A symmetrical proof holds in the other case.

Since BAA maximizes utilization subject to *fairness policy* (1) - (3), it follows from Lemma 5 that ρ_s must be as large as possible. If $i \in S$, the IOPS it receives on the HD are $\rho_s C_s/n \times (m_i/h_i)$ which from the EF requirements of Lemma 3 must be no more than $\rho_d C_d/n$, the IOPS on the HD for any client in D. Hence, $\rho_s C_s/n \times (m_i/h_i) \le \rho_d C_d/n$ or $\rho_s \le \rho_d (C_d/C_s) \times (h_i/m_i)$, for all $i \in S$. Since h_i/m_i is smallest for $i = r+1$, the maximum feasible value of ρ_s is $\rho_s = \rho_d (C_d/C_s) \times (h_{r+1}/m_{r+1})$. Now, $h_{r+1} > h_{bal}$, so $h_{r+1}/m_{r+1} > h_{bal}/(1-h_{bal}) = C_s/C_d$. Hence $\rho_s > \rho_d$. Since the HD is 100% utilized we know from Lemma 4 that $\rho_d \ge 1$, and so $\rho_s > 1$. □

From Lemmas 4 to 6 we can conclude:

Lemma 7. *Allocations made by BAA satisfy the Sharing Incentive property.*

5 Performance Evaluation

We evaluate our work using both simulation and Linux system implementation. For simulation, a synthetic set of workloads was created. Each request is randomly assigned to the SSD or HD based on its hit ratio. The request service time is an exponentially distributed random variable with mean equal to the reciprocal of the device IOPS capacity.

In the Linux system, we implemented a prototype by interposing the BAA scheduler in the IO path. Raw IO is performed to eliminate the influence of OS buffer caching. The storage server includes a 1TB SCSI Western Digital hard disk (7200 RPM 64MB Cache SATA 6.0Gb/s) and 120GB SAMSUNG 840 Pro Series SSD. Various block-level workloads from UMass Trace Repository [1] and Microsoft Exchange server [31] are used for the evaluation. These traces are for a homogeneous server and do not distinguish between devices. Since we needed to emulate different proportions of HD and SSD requests we randomly partitioned the blocks between the two devices to meet the assumed hit ratio of the workload. The device utilizations are measured using Linux tool “iostat”.

5.1 Simulation Experiments

5.1.1 System Efficiency

This experiment compares the system efficiency for three different schedulers: Fair Queuing (FQ), DRF, and BAA. The capacities of the HD and SSD are 100 IOPS and 5000 IOPS respectively. The first experiment employs two clients with hit ratios 0.5 and 0.99. FQ allocates equal amounts of throughput to the two clients. The DRF implementation uses the dominant resource shares policy of [17] to determine allocation weights, and BAA is the approach proposed in this paper. All workloads are assumed to be continuously backlogged.

The throughputs of the two clients with different schedulers are shown in Figure 3(a). The figure also shows the fair share allocation, *i.e.* the throughput the workload would get by partitioning the SSD and HD capacities equally between the two workloads. As can be seen, the throughput of client 2 under FQ is the lowest of the three schedulers. In fact, sharing is a disincentive for client 2 under FQ scheduling, since it would have been better off with a static partitioning of both devices. The problem is that the fair scheduler severely throttles the SSD-bound workload to force the 1 : 1 fairness ratio. DRF performs much better than FQ. Both clients get a little more than their fair shares. BAA does extremely well in this setup and client 2 is able to almost double the throughput it would have received with a static partition. We also show the system utilization for the three schedulers in Figure 3(b). BAA is able to fully utilize both devices, while DRF reaches system utilization of only around 65%.

Next we add another client with hit ratio of 0.8 to the workload mix. The throughputs of the clients are shown in Figure 4(a). Now the throughput of the DRF scheduler is also degraded, because it does not adjust the relative allocations to account for load imbalance. The BAA scheduler gets higher throughput (but less than 100%) because it adjusts the weights to balance the system load. The envy-free requirements put an upper-bound on the SSD-bound client’s throughput, preventing the utilization from going any higher, but still maintaining fairness.

5.1.2 Adaptivity to Hit Ratio Changes

In this experiment, we show how the two-level scheduling framework restores system utilization following a change in an application’s hit ratio. The capacities of the HD and SSD are 200 IOPS and 3000 IOPS respectively. In this simulation, allocations are recomputed every 100s and the hit ratio is monitored in a moving window of 60s. There are two clients with initial hit ratios of 0.45 and 0.95. At time 510s, the hit ratio of client 1 falls to 0.2.

Figure 5 shows a time plot of the throughputs of the clients. The throughputs of both clients falls significantly at time 510 as shown in Figure 5. The scheduler needs to be cognizant of changes in the application characteristics and recalibrate the allocations to increase the efficiency. At time 600s (the rescheduling interval boundary) the allocations are recomputed using the hit ratios that reflect the current application behavior, and the system throughput rises again.

In practice the frequency of calibration and the rate at which the workload hit ratios change can affect system performance and stability. As is the case in most adaptive situations, the techniques work best when significant changes in workload characteristics do not occur at a very fine time scale. We leave the detailed evaluation of robustness to future work.

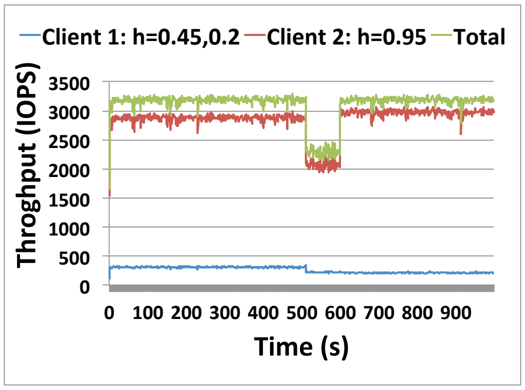
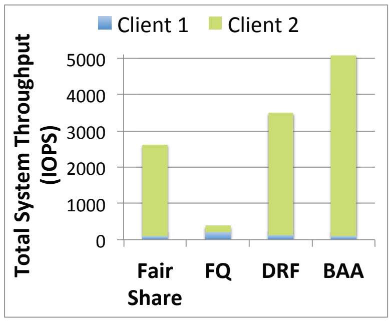


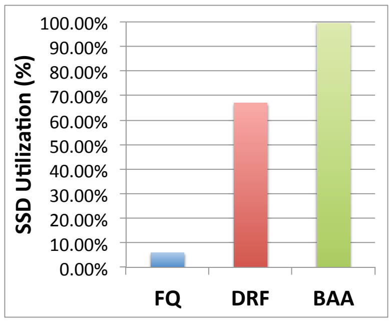
Figure 5: Scheduling with dynamic weights when hit ratio changes

5.2 Linux Experiments

We now evaluate BAA in a Linux system, and compare its behavior with allocations computed using the DRF policy [17] and the Linux CFQ [39] scheduler. The first set of experiments deals with evaluating the throughputs (or system utilization) of the three scheduling approaches. The second set compares the fairness properties.

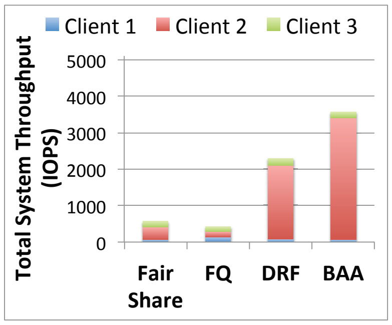


(a) Throughputs

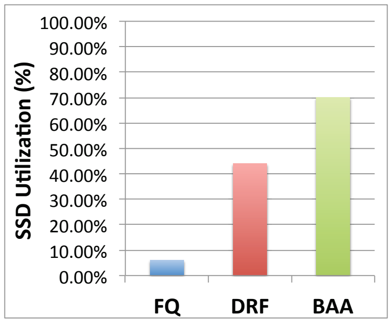


(b) Utilizations

Figure 3: Throughputs and utilizations for 2 flows



(a) Throughputs



(b) Utilizations

Figure 4: Throughputs and utilizations for 3 flows

5.2.1 Throughputs and Device Utilizations

Throughputs	BAA	CFQ	DRF
Client 1	100	101	95
Client 2	139	134	133
Total	239	235	228

Table 2: Throughputs: all clients in one bottleneck set

Clients in the same bottleneck set. Two workloads from Web Search [1] are used in this experiment. The requests include reads and writes and the request sizes range from 8KB to 32KB.

We first evaluate the performance when all the clients fall into the same bottleneck set; that is, all the clients are bottlenecked on the same device. We use hit ratios of 0.3 and 0.5 for the two workloads which makes them both HD bound. As shown in Table 2 all three schedulers get similar allocation. In this situation there is just one local bottleneck set in BAA, which (naturally) coincides with the system bottleneck device for CFQ as well as being the dominant resource for DRF. The device utilizations are the same for all schedulers, as can be expected.

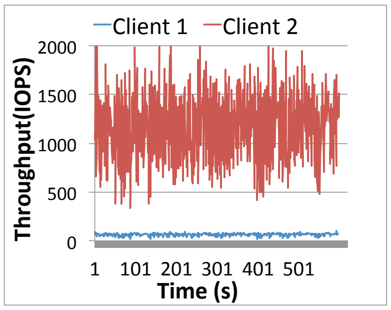
Clients in different bottleneck sets. In this experiment, we evaluate the performance when the clients fall into different bottleneck sets; that is, some of the clients are bottlenecked on the HD and some on the SSD. Two clients, one running a Financial workload [1] (client 1) and the second running an Exchange workload [31] (client 2) with hit ratios of 0.3 and 0.95 respectively, are used in the experiment. The request sizes range from 512 bytes to 8MB, and are a mix of read and write requests. The total experiment time is 10 minutes.

Figure 6 shows the throughput of each client achieved by the three schedulers. As shown in the figure, BAA has better total system throughput than the others. CFQ performs better than DRF but not as good as BAA.

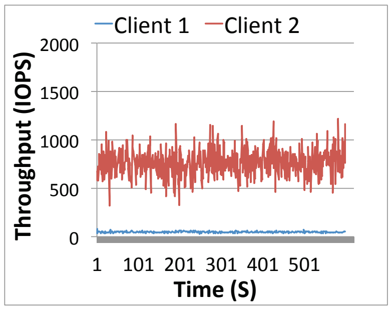
Figure 7 shows the measured utilizations for HD and SSD using the three schedulers. Figure 7(a) shows that BAA achieves high system utilization for both HD and SSD; DRF and CFQ have low SSD utilizations compared with BAA, as shown in Figure 7(b) and (c). HD utilizations are good for both DRF and CFQ (almost 100%), because the system has more disk-bound clients that saturate the disk.

5.2.2 Allocation Properties Evaluation

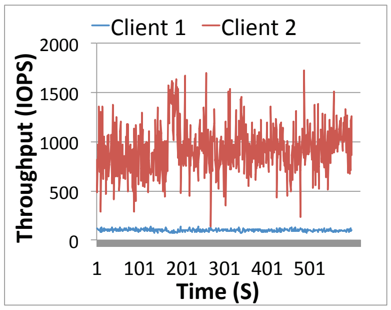
In this experiment, we evaluate the fairness properties of allocations (**P1** to **P4**). Four Financial workloads [1] with hit ratios of 0.2, 0.4, 0.98 and 1.0 are used as the input. The workloads have a mix of read and write requests and request sizes range from 512 bytes to 8MB.



(a) BAA

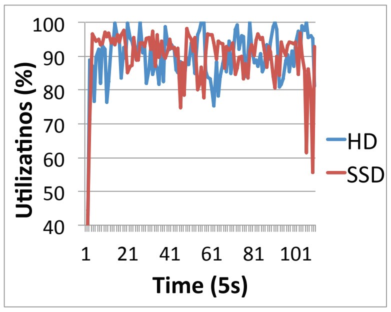


(b) DRF

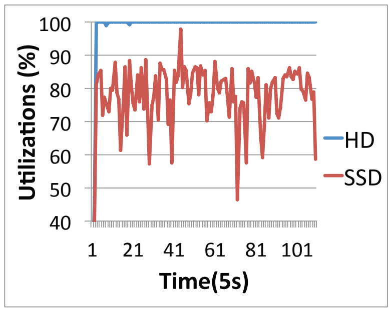


(c) CFQ

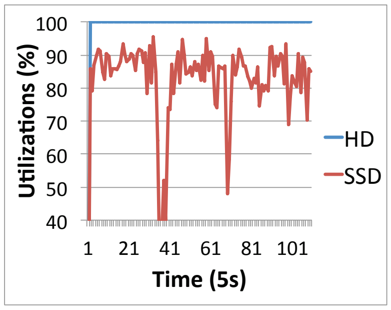
Figure 6: Throughputs using three schedulers. BAA achieves higher system throughput (1396 IOPS) than both DRF-based Allocation (810 IOPS) and Linux CFQ (1011 IOPS).



(a) BAA



(b) DRF



(c) CFQ

Figure 7: System utilizations using three schedulers. The average utilization are: BAA (HD 94% and SSD 92%), DRF (HD 99% and SSD 78%), CFQ (HD 99.8% and SSD 83%)

Clients	Fair Share (IOPS)	Total IOPS	HD IOPS	SSD IOPS
Financial 1	50	76	60.8	15.2
Financial 2	67	101	60.8	40.4
Financial 3	561	1068	21.4	1047
Financial 4	550	1047	0	1047

Table 3: Allocations for Financial workloads using BAA

Table 3 shows the allocations of BAA-based scheduling. The second column shows the Fair Share for each workload. The third column shows the IOPS achieved by each client, and the portions from the HD and SSD are shown in the next two columns.

The average capacity of the HD for the workload is around 140-160 IOPS and the SSD is 2000-2200 IOPS. We use the upper-bound of the capacity to compute the fair shares shown in the second column. In this setup, Financial 1 and Financial 2 are bottlenecked on the HD and belong to *D*, while Financial 3 and Financial 4 are bottlenecked on the SSD and belong to *S*.

First we verify that clients in the same bottleneck set receive allocations in proportion to their fair share (**P1**). As shown in the Table 3, Financial 1 and 2 get throughputs of 76 and 101, which are in the same ratio as their fair share (50 : 67). Similarly, Financial 3 and 4 get throughputs 1068 and 1047, which are in the ratio of their fair share of (561 : 550).

HD-bottlenecked workloads Financial 1 and Financial 2 receive more HD allocation (60.8 IOPS) than both workloads Financial 3 (21.4 IOPS) and 4 (0 IOPS). Similarly, SSD-bottlenecked workloads Financial 3 and Financial 4 receive more SSD allocation (1047 and 1047 IOPS) than both workload 1 (15.2 IOPS) and 2 (40.4 IOPS).

It can be verified from columns 2 and 3 that every client receives at least its fair share. Finally, the system shows that both HD and SSD are almost fully utilized, indicating the allocation maximizes the system throughput subject to these fairness criteria. Similar experiments were also conducted with other workloads, including those from Web Search and Exchange Servers. The results show that properties P1 to P4 are always guaranteed.

6 Related Work

There has been substantial work dealing with proportional share schedulers for networks and CPU [9, 18, 44]. These schemes have since been extended to handle the constraints and requirements of storage and IO scheduling [21, 19, 20, 45, 32, 27, 33]. Extensions of WFQ to provide reservations for constant capacity servers were presented in [41]. Reservation and limit controls for storage servers were studied in [29, 46, 22, 24]. All these models provide strict proportional allocation for a single resource based on static shares possibly subject to reservation and limit constraints.

As discussed earlier, Ghodsi et al [17] proposed the DRF policy, which provides fair allocation of multiple resources on the basis of dominant shares. Ghodsi et al. [16] extended DRF to packet networks and compared it to the global bottleneck allocation scheme of [12]. Dolev et al [11] proposed an alternative to DRF based on fairly dividing a global system bottleneck resource. Gutman and Nisan [25] considered generalizations of DRF in a more general utility model, and also gave a polynomial time algorithm for the construction in Dolev et al [11]. Parkes et al. [36] extended DRF in several ways, and in particular studied the case of indivisible tasks. Envy-freedom has been studied in the areas of economics [26] and in game theory [10].

Techniques for isolating random and sequential IOs using time-quanta based IO allocation were presented in [37, 34, 42, 43, 39, 8]. IO scheduling for SSDs is examined in [34, 35]. Placement and scheduling tradeoffs for hybrid storage were a studied in [47]. For a multi-tiered storage system, Reward scheduling [13, 14, 15] proposed making allocations in the ratio of the throughputs a client would receive when executed in isolation. Interestingly, both Reward and DRF perform identical allocations for the storage model of this paper [14] (concurrent operation of the SSD and the HD), although they start from very different fairness criteria. Hence, Reward also inherits the fairness properties proved for DRF [17]. For a sequential IO model where only 1 IO is served at a time, Reward will equalize the IO time allocated to each client. Note that neither DRF nor Reward explicitly address the problem of system utilization.

In the system area, Mesos [5] proposes a two-level approach to allocate resources to frameworks like Hadoop and MPI that may share an underlying cluster of servers. Mesos (and related solutions) rely on OS-level abstractions like resource containers [4].

7 Conclusions and Future Work

Multi-tiered storage made up of heterogeneous devices are raising new challenges in providing fair throughput allocation among clients sharing the system. The fundamental problem is finding an appropriate balance between fairness to the clients and increasing system utilization. In this paper we cast the problem within the broader framework of fair allocation for multiple resources, which has been drawing considerable amount of recent research attention. We find that existing methods almost exclusively emphasize the fairness aspect to the possible detriment of system utilization.

We presented a new allocation model BAA based on the notion of per-device bottleneck sets. The model provides clients that are bottlenecked on the same device with allocations that are proportional to their fair shares, while allowing allocation ratios between clients in different bottleneck sets to be set by the allocator to maximize utilization. We show formally that BAA satisfies the properties of Envy Freedom and Sharing Incentive that are well accepted fairness requirements in microeconomics and game theory. Within these fairness constraints BAA finds the best system utilization. We formulated the optimization as a compact 2-variable LP problem. We evaluated the performance of our method using both simulation and implementation on a Linux platform. The experimental results show that our method can provide both high efficiency and fairness.

One avenue of further research is to better understand the theoretical properties of the Linux CFQ scheduler. It performs remarkably well in a wide variety of situations; we feel it is important to better understand its fairness and efficiency tradeoffs within a suitable theoretical framework. We are also investigating single-level scheduling algorithms to implement the BAA policy, and plan to conduct empirical evaluations at larger scale beyond our modest experimental setup.

Our approach also applies, with suitable definitions and interpretation of quantities, to broader multi-resource allocation settings as in [17, 11, 36], including CPU, memory, and network allocations. It can also be generalized to handle client weights; in this case clients in the same bottleneck set receive allocations in proportion to their weighted fair shares. We are also investigating settings in which the SSD is used as a cache; this will involve active data migration between the devices, making the resource allocation problem considerably more complex.

Acknowledgments

We thank the reviewers of the paper for their insightful comments which helped shape the revision. We are grateful to our shepherd Arif Merchant whose advice and guidance helped improve the paper immensely. The support of NSF under Grant CNS 0917157 is greatly appreciated.

References

[1] Storage performance council (umass trace repository), 2007. http://traces.cs.umass.edu/index.php/Storage.

[2] EMC: Fully automate storage tiering. http://www.emc.com/about/glossary/fast.htm, 2012.

[3] Tintri: VM aware storage. http://www.tintri.com, 2012.

[4] BANGA, G., DRUSCHEL, P., AND MOGUL, J. C. Resource containers: a new facility for resource management in server systems. In *OSDI '99*.

[5] BENJAMIN, H., AND ET. AL. Mesos: a platform for fine-grained resource sharing in the data center. In *NSDI'11*.

[6] BERTSIMAS, D., FARIAS, V. F., AND TRICHAKIS, N. On the efficiency-fairness trade-off. *Manage. Sci. 58*, 12 (Dec. 2012), 2234–2250.

[7] BERTSIMAS, D., FARIAS, V. F., AND TRICHAKIS, V. F. The price of fairness. *Operations Research 59*, 1 (Jan. 2011), 17–31.

[8] BRUNO, J., BRUSTOLONI, J., GABBER, E., OZDEN, B., AND SILBERSCHATZ, A. Disk scheduling with Quality of Service guarantees. In *Proceedings of the IEEE International Conference on Multimedia Computing and Systems, Volume 2* (1999), IEEE Computer Society.

[9] DEMERS, A., KESHAV, S., AND SHENKER, S. Analysis and simulation of a fair queuing algorithm. *Journal of Internetworking Research and Experience 1*, 1 (September 1990), 3–26.

[10] DEVANUR, N. R., HARTLINE, J. D., AND YAN, Q. Envy freedom and prior-free mechanism design. *CoRR abs/1212.3741* (2012).

[11] DOLEV, D., FEITELSON, D. G., HALPERN, J. Y., KUPFERMAN, R., AND LINIAL, N. No justified complaints: On fair sharing of multiple resources. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference* (New York, NY, USA, 2012), ITCS '12, ACM, pp. 68–75.

[12] EGI, N., IANNACCONE, G., MANESH, M., MATHY, L., AND RATNASAMY, S. Improved parallelism and scheduling in multicore software routers. *The Journal of Supercomputing 63*, 1 (2013), 294–322.

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

[14] ELNABLY, A., AND VARMAN, P. Application specific QoS scheduling in storage servers. In *24th ACM Symposium on Parallel Algorithms and Architectures* (SPAA'12, June 2012).

[15] ELNABLY, A., WANG, H., GULATI, A., AND VARMAN, P. Efficient QoS for multi-tiered storage systems. In *4th USENIX Workshop on Hot Topics in Storage and File Systems* (June 2012).

[16] GHODSI, A., SEKAR, V., ZAHARIA, M., AND STOICA, I. Multi-resource fair queueing for packet processing. In *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication* (New York, NY, USA, 2012), SIGCOMM '12, ACM, pp. 1–12.

[17] GHODSI, A., ZAHARIA, M., HINDMAN, B., KONWINSKI, A., SHENKER, S., AND STOICA, I. Dominant resource fairness: fair allocation of multiple resource types. In *Proceedings of the 8th USENIX conference on Networked systems design and implementation* (Berkeley, CA, USA, 2011), NSDI'11, USENIX Association, pp. 24–24.

[18] GOYAL, P., VIN, H. M., AND CHENG, H. Start-time fair queueing: a scheduling algorithm for integrated services packet switching networks. *IEEE/ACM Trans. Netw. 5*, 5 (1997), 690–704.

[19] GULATI, A., AHMAD, I., AND WALDSPURGER, C. PARDA: Proportional Allocation of Resources in Distributed Storage Access. In *(FAST '09)Proceedings of the Seventh Usenix Conference on File and Storage Technologies* (Feb 2009).

[20] GULATI, A., KUMAR, C., AHMAD, I., AND KUMAR, K. Basil: Automated io load balancing across storage devices. In *Usenix FAST* (2010), pp. 169–182.

[21] GULATI, A., MERCHANT, A., AND VARMAN, P. *p*Clock: An arrival curve based approach for QoS in shared storage systems. In *ACM SIGMETRICS* (2007).

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

[23] GULATI, A., SHANMUGANATHAN, G., ZHANG, X., AND VARMAN, P. Demand based hierarchical qos using storage resource pools. In *Proceedings of the 2012 USENIX conference on Annual Technical Conference* (Berkeley, CA, USA, 2012), USENIX ATC'12, USENIX Association, pp. 1–1.

[24] GULATI, A., SHANMUGANATHAN, G., ZHANG, X., AND VARMAN, P. Demand based hierarchical qos using storage resource pools. In *Proceedings of the 2012 USENIX Conference on Annual Technical Conference* (Berkeley, CA, USA, 2012), USENIX ATC'12, USENIX Association, pp. 1–1.

[25] GUTMAN, A., AND NISAN, N. Fair allocation without trade. *CoRR abs/1204.4286* (2012).

[26] JACKSON, M. O., AND KREMER, I. Envy-freeness and implementation in large economies. *Review of Economic Design 11*, 3 (2007), 185–198.

[27] JIN, W., CHASE, J. S., AND KAUR, J. Interposed proportional sharing for a storage service utility. In *ACM SIGMETRICS '04* (2004).

[28] JOE-WONG, C., SEN, S., LAN, T., AND CHIANG, M. In *INFOCOM* (2012), A. G. Greenberg and K. Sohraby, Eds., IEEE, pp. 1206–1214.

[29] KARLSSON, M., KARAMANOLIS, C., AND ZHU, X. Triage: Performance differentiation for storage systems using adaptive control. *Trans. Storage 1*, 4 (2005), 457–480.

[30] KASH, I., PROCACCIA, A. D., AND SHAH, N. No agent left behind: Dynamic fair division of multiple resources. In *Proceedings of the 2013 International Conference on Autonomous Agents and Multi-agent Systems* (Richland, SC, 2013), AAMAS '13, International Foundation for Autonomous Agents and Multiagent Systems, pp. 351–358.

[31] KAVALANEKAR, S., WORTHINGTON, B., ZHANG, Q., AND SHARDA, V. Characterization of storage workload traces from production windows servers. In *Workload Characterization, 2008. IISWC 2008. IEEE International Symposium on* (2008), pp. 119–128.

[32] LUMB, C., MERCHANT, A., AND ALVAREZ, G. Façade: Virtual storage devices with performance guarantees. *File and Storage technologies (FAST'03)* (March 2003), 131–144.

[33] LUMB, C. R., SCHINDLER, J., GANGER, G. R., NAGLE, D. F., AND RIEDEL, E. Towards higher disk head utilization: extracting free bandwidth from busy disk drives. In *Usenix OSDI* (2000).

[34] PARK, S., AND SHEN, K. Fios: A fair, efficient flash i/o scheduler. In *FAST* (2012).

[35] PARK, S., AND SHEN, K. Flashfq: A fair queueing i/o scheduler for flash-based ssds. In *Usenix ATC* (2013).

[36] PARKES, D. C., PROCACCIA, A. D., AND SHAH, N. Beyond dominant resource fairness: Extensions, limitations, and indivisibilities. In *Proceedings of the 13th ACM Conference on Electronic Commerce* (New York, NY, USA, 2012), EC '12, ACM, pp. 808–825.

[37] 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.

[38] PROCACCIA, A. D. Cake cutting: Not just child's play. *Communications of the ACM 56*, 7 (2013), 78–87.

[39] SHAKSHOBER, D. J. Choosing an I/O Scheduler for Red Hat Enterprise Linux 4 and the 2.6 Kernel. In *In Red Hat magazine* (June 2005).

[40] SHREEDHAR, M., AND VARGHESE, G. Efficient fair queueing using deficit round robin. In *Proc. of SIGCOMM '95* (August 1995).

[41] STOICA, I., ABDEL-WAHAB, H., AND JEFFAY, K. On the duality between resource reservation and proportional-share resource allocation. *SPIE* (February 1997).

[42] VALENTE, P., AND CHECCONI, F. High Throughput Disk Scheduling with Fair Bandwidth Distribution. In *IEEE Transactions on Computers* (2010), no. 9, pp. 1172–1186.

[43] 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).

[44] WALDSPURGER, C. A., AND WEIHL, W. E. Lottery scheduling: flexible proportional-share resource management. In *Usenix OSDI* (1994).

[45] WANG, Y., AND MERCHANT, A. Proportional-share scheduling for distributed storage systems. In *Usenix FAST* (Feb 2007).

[46] WONG, T. M., GOLDING, R. A., LIN, C., AND BECKER-SZENDY, R. A. Zygaria: Storage performance as a managed resource. In *Proceedings of the 12th IEEE Real-Time and Embedded Technology and Applications Symposium* (Washington, DC, USA, 2006), RTAS '06, IEEE Computer Society, pp. 125–134.

[47] WU, X., AND REDDY, A. L. N. Exploiting concurrency to improve latency and throughput in a hybrid storage system. In *MASCOTS* (2010), pp. 14–23.

[48] ZHANG, J., SIVASUBRAMANIAM, A., WANG, Q., RISKA, A., AND RIEDEL, E. Storage performance virtualization via throughput and latency control. In *MASCOTS* (2005), pp. 135–142.

[49] ZHANG, L. VirtualClock: A new traffic control algorithm for packet-switched networks. *ACM Trans. Comput. Syst. 9*, 2, 101–124.