

Towards Distributed Storage Resource Management using Flow Control

Ajay Gulati, Irfan Ahmad
VMware Inc., Palo Alto, CA 94304
{agulati,irfan}@vmware.com

Abstract

Deployment of shared storage systems is increasing with rapid adoption of virtualization technologies to provide isolation, better management and high utilization of resources. Quality of service (QoS) in such environments is quite desirable for meeting IO demands of virtual machines. The lack of QoS support at typical storage arrays, simultaneous access by multiple hosts and concerns regarding under-utilization of resources makes this problem quite challenging. In this paper, we study the problem of providing fairness among hosts accessing a storage array in a distributed manner while maintaining high efficiency. Towards this goal, we investigate whether local latency estimates at each host can be used to detect overload and whether limiting host issue queue lengths can provide fairness across hosts. In principle, the approach is similar to mechanisms used by TCP at each host for flow control. Initial experiments provide encouragement to develop a complete framework.

1 Introduction

Storage arrays form the backbone of modern data centers by providing consolidated data access to multiple applications simultaneously. More and more organizations are moving towards consolidated storage using Storage Area Network (SAN) or Network-Attached Storage (NAS) boxes. Easy access from anywhere/anytime, ease of backup, flexibility in allocation and centralized administration are some of the reasons favoring the trend. This trend is further fueled by the increase in demand for virtualization technology. In most virtualized data centers, multiple servers (hosts) access a single storage array, where each host is running multiple virtual machines. The number of hosts accessing one or more logical units (LUNs) on a storage array can vary from 4 to 32 in typical configurations. This contention at the array for resources such as array controller, cache, buses and disk arms leads to unpredictable IO completion times. It is often desirable to have proportionate allocation of IO throughput (IOPS) or bandwidth (Bytes/s) and contention management among these hosts.

In this paper, we target the problem of providing coarse grained fairness among servers while maintaining high utilization of the storage array. The problem is quite challenging because of the lack of support from the array, unpredictable available bandwidth and distributed nature of access from multiple hosts. Figure 1 shows a typical setup, with multiple hosts accessing one or more LUNs on the storage array in a distributed manner.

Most existing solutions [9, 10, 13, 15, 18, 21] provide bandwidth allocation among multiple applications running at a *single host*. In that case, one centralized scheduler has complete control over requests going to the storage system. Other approaches [15, 21] try to control the queue length at the array to provide tight latency control, but they are also centralized. In a distributed case, throttling based approaches such as Triage [14] have been proposed but such a solution may lead to under-utilization of array resources. Conceptually, existing solutions based on host throttling use centralized monitoring and work at a very coarse granularity which may cause substantial loss in utilization. Running them at finer granularity may cause a prohibitive increase in communication costs.

We map this problem of distributed storage access from multiple hosts to flow control in networks. The problem of providing coarse grained fairness with high utilization is, in principle, similar to multiple hosts trying to estimate available bandwidth in the network and trying to consume it in a fair manner. The network is like a black box to the hosts, with little or no information about the current state and number of participants. In this paper, we investigate if latency observed at each host can be used as a reliable indicator for load at the array and if adjusting number of IOs issued per host (i.e. window size) can provide coarse grained fairness. We also study the effects of various storage specific issues such as degree of sequentiality, reads vs writes, IO size etc., that can lead to variable latency. Our initial experiments show that it is possible to use average latency as an indicator for overload at the array. Further, we notice that controlling queue length of issued IOs at each host is an effective way of providing coarse grained fairness.

One of the assumptions made is that all hosts are using the same protocol in order to ensure fairness. This is usu-

ally true in virtualized clusters with multiple hosts running the same hypervisor and the solution can be built into the hypervisor. A complete implementation of the flow control mechanism based on latency estimates and dynamic adaptation of queue length per host is under progress and will be presented in a full paper.

2 System Model

Our system shown in Figure 1 consists of a set of consolidated servers (hosts) accessing one or more storage volumes at the array connected over SAN. Each volume is constructed using one or more LUNs and gives the abstraction of a storage device to the host. A virtual machine disk is represented by a file on one of the volumes. Since a host runs multiple virtual machines (VMs), the IO traffic coming out of the host is the aggregated traffic of all the VMs currently doing IO. Each host maintains a set of pending IOs at the array, represented by an *issue queue*. This queue represents the IOs scheduled by the host and currently pending at the array; there may be other requests pending at the host waiting to be sent to the storage array. Typically the issue queues are per LUN and have a fixed maximum size (e.g. 16 or 32 IOs per LUN).

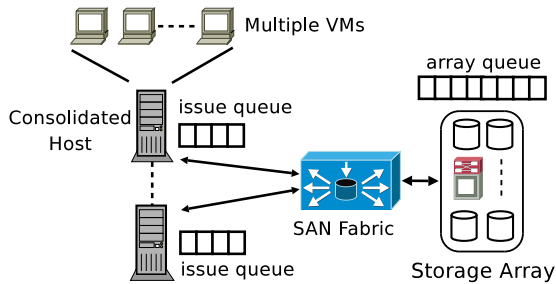


Figure 1. System Model

Each host is assigned a weight based on the sum of the IO shares allocated to each VM running on that host. Coarse grained fairness implies providing array throughput (IOPS) in proportion to the hosts' weights. The desired goals for a flow control mechanism are: (1) provide coarse grained fairness or proportionate sharing among hosts, (2) maintain high utilization, and (3) low overhead in terms of per-host work and inter-host communication.

The problem of flow control in this case is determining the maximum number of IOs that a host can keep pending at the array while meeting our goals. A small static issue queue length per host in proportion to its weight can provide good fairness but may lead to poor utilization of the array in underloaded scenarios. For better efficiency (higher utilization) of the array, we would like the issue queue lengths to be longer at each host. Hence there is a trade-off between high utilization at the array and fair-

ness among hosts. Issue queue lengths should vary dynamically depending on the amount of contention at the array. In principle, queue lengths should increase under low contention and decrease under high contention. In an equilibrium state, the queue lengths should converge to different values for hosts based on their weights. This will allow hosts to get proportional fairness in presence of contention. Note that we have been talking about IO operations (queue slots) instead of Bytes/sec, but one can map each queue slot to a fixed sized IO operation, thereby providing fairness in Bytes/s.

3 IO Resource Management

In this section we will present the analogy between flow control in networks and distributed storage access by multiple hosts. We also discuss some of the storage specific issues that make the adaptation non-trivial. Finally, we outline our adapted flow control mechanism for storage.

Our approach tries to map the problem of distributed storage management to flow control in networks. TCP running at a host tries to do flow control based on two signals from the network: round trip time (RTT) and packet loss probability. RTT is essentially the latency of an IO request as seen by the IO scheduler. Hence this signal can be used as it is. The second signal is not present for our case. *Packet loss* is neither common nor something that applications accessing data on a disk array are designed to handle. Instead we plan to use only IO latency as an indicator of congestion at the array. Latency higher than a certain threshold may trigger reduction in queue depth. The recently proposed FAST TCP [12] uses packet latency instead of packet loss probability, because loss probability is too difficult to estimate accurately in networks with high bandwidth-delay products. Similarly, earlier proposals such as RED (random early detection) [6] also proposed early detection of congestion using information/hints from routers, before a packet is lost. In networks, this has the added advantage of avoiding retransmissions. We plan to use a similar adaptive approach based on average latency to detect the congestion at the array. Other techniques that require support from routers have not been adopted even in networks due to overhead and complexity concerns, which is analogous to the current lack of any support at arrays.

TCP does more than just flow control, and is quite complex due to handling of packet loss, acks, retransmission and congestion collapse. Lack of packet loss related complexity makes flow control a little simpler for us, however we need to deal with storage-specific issues for latency estimation.

3.1 Storage-Specific Issues

Storage devices are stateful and their throughput can be quite variable, making it challenging and non-trivial to use

network-based flow control approaches. We consider three main issues to highlight the differences between storage and network systems as follows:

Request Location: It is well known that the latency of a request can vary from a fraction of a millisecond to tens of milliseconds based on its location compared to the previous request. Average seek delays for current disks range from 6 ms to 15 ms, which can cause an order of magnitude difference in service times. This makes it harder to estimate a baseline IO latency corresponding to the latency with no queuing delay. Thus a sudden change in average latency or ratio of current values to previous average may or may not be a signal for overload. Instead, we plan to look at absolute latency values in comparison to a latency threshold to predict congestion. The assumption is that latency values during congestion will have *queuing delay* as a large component outweighing an increase due to workload changes (e.g. sequential to random). The variation in latency just due to workload behavior would get subsumed by the latency threshold.

Request type: Write IOs are often completed to the host once the block is written in the controller’s or NVRAM. They are flushed to disk during the destage process. However, read IOs may need to go to disk more often. Similarly, two requests from a single stream may have widely varying latencies if one hits in the cache and other misses. In certain RAID systems [4] writes may take 4 times longer than reads due to parity read and update. In general, the IOs from a single stream may have widely varying response times, thereby affecting the latency estimate. We believe that an estimate over a long enough period or a moving average considering a large window should be able to absorb such variations and provide a more consistent view.

IO Size: In case of storage, the range of sizes is 512 bytes to 256K or even 1MB for certain recent devices. The estimator needs to be aware of changing IO size in the workload. This can be done by computing latency per 4KB instead of latency per IO using a linear model with certain fixed cost (e.g. $latency = m * size + C$). Size variance is less of an issue in networks since most packets are broken into MTU sized chunks (typically 1500 bytes) before transmission.

Essentially, all of these boil down to the issue of estimating highly variable latency and using that as a predictor. We need to distinguish between latency changes caused by workload and caused due to the overload at the array. We believe that some of the variation in IO latency would be absorbed by the long term averaging and considering latency per Byte instead of per IO request. Also a high enough baseline latency (the desired operating point for the control algorithm) should provide high utilization in under-loaded cases, independent of workload characteristics. Thus the algorithm would most likely operate at a point where a certain number of IOs are always kept pending.

3.2 Control Algorithm

The control algorithm at a high level will detect overload at the array based on average IO latency over a time period and adjust the issue queue length (i.e. window size) at the host. There are two main components: (1) latency estimation and (2) window size computation. For latency estimation, each host maintains a moving average of IO latency denoted as current average latency (CAL), to smooth out the short term variations. This can be done using a well known Exponentially Weighted Moving Average (EWMA). The degree of weighing past values is determined by a constant smoothing parameter $\alpha \in [0, 1]$. Let l be the current latency value; then the formula for computing a moving average CAL will be:

$$CAL(t) = (1 - \alpha) \times l + \alpha \times CAL(t - 1) \quad (1)$$

Window computation is done using an AIMD (additive increase multiplicative decrease) policy, which has been shown to be quite stable in behavior. The window estimation formula is:

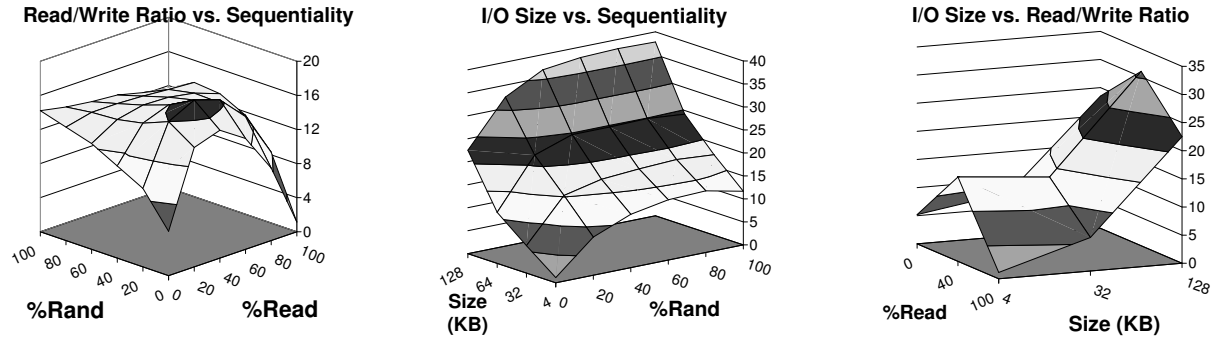
$$w(t + 1) = (1 - \gamma)w(t) + \gamma \left(\frac{LAT_{threshold}}{CAL(t)} w(t) + \beta \right) \quad (2)$$

Here $w(t)$ denotes the window size (issue queue length) at time t , $\gamma \in [0, 1]$, $LAT_{threshold}$ is the latency which corresponds to zero queuing delay at the array and β is a per-host parameter. Whenever the average latency increases above the threshold, we will decrease the window size by a multiplicative factor. When the overload subsides, and the latency gets smaller than $LAT_{threshold}$, window size will be increased based on current estimate and β per host. To avoid extreme behavior from the control algorithm, $w(t)$ is limited by w_{max} , an upper bound. This avoids very long queues at the array, bounding the latency faced by newly activated hosts.

The description includes three main parameters: an upper bound w_{max} , the system-wide $LAT_{threshold}$ and β . The upper bound can be based on typical values that are used for queue length (32 or 64) and the array configuration such as number of hosts accessing a volume, number of physical disks in the volume, etc. For latency threshold, we plan to set it empirically based on the relationship between latency and throughput. Our tests show that making the array queue longer than a certain value doesn’t lead to increase in throughput. User input can also be used to set the threshold, based on application-specific requirements. Finally, β is set based on the weight of the host. It has been shown theoretically that the equilibrium value of window size for different hosts will be proportional to the β parameter [12].

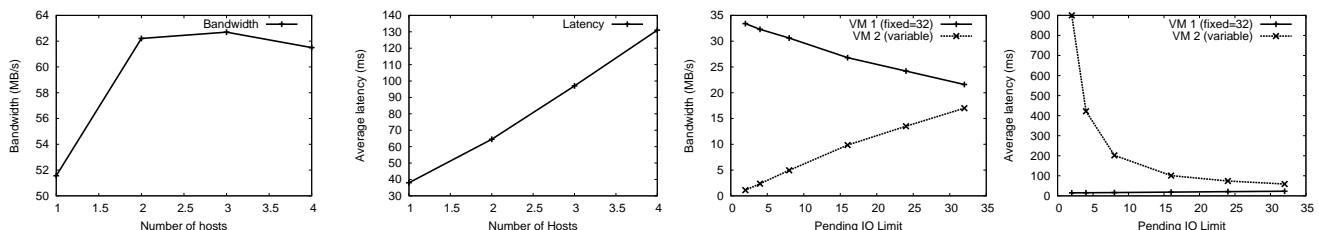
4 Experimental Evaluation

In this section, we want to answer three major questions: (1) How does average latency vary with changes in work-



(a) Variable Read% and Sequentiality (b) Variable IO size and Sequentiality (c) Variable IO size and Read %

Figure 2. Latency observed by a single host with workload changes in IO size, Read % and degree of sequentiality. First plot has fixed 4 KB IO size, second plot has fixed Read% of 100%, and third plot has fixed sequentiality of 100%. Remaining two parameters are varied for each plot.



(a) Overall Bandwidth (MB/s) (b) Overall Avg latency (ms) (c) Bandwidth per VM (d) Latency per VM

Figure 3. (a) and (b) show the overall bandwidth and latency observed by multiple hosts as we increase the number of hosts from 1 to 4. (c) and (d) show the bandwidth and latency observed by VM on each host when window size of one of the hosts is varied from 32 to 2.

load? (2) How does average latency vary with overall load on the array? (3) Can different issue queue lengths per host provide differentiated service to them? The first two questions help determine if average latency can be used to detect congestion/overload at the storage array. The third question determines whether adjusting queue lengths per host is sufficient to provide coarse grained fairness.

We experimented with multiple hosts accessing a storage array. Each host is running VMware ESX Server 3.5 [17] and has 2 Intel Xeon 3.0 GHz dual-core cpus, 8 GB of RAM and is attached to a 124 GB storage volume. The volume is hosted on a 10-disk RAID-0 striped disk group on an EMC CLARiiON CX3-40 storage array.

4.1 Workload Variations

First we run a single VM on a host with different workloads to see the variance in average latency as reported by our workload generator and measurement tool - *Iometer* [1]. We varied three parameters of the workload:

Reads: 0 to 100% in increments of 20%

IO size: 4, 32, 64 and 128 KB

Sequentiality: 0 to 100% in increments of 20%

For each of these cases we measured throughput (IOPS), bandwidth (MB/s), average, min and max latency (in ms).

The volume under test is only accessed by our workload.

For clarity, we show latency results for two parameters at a time. Figure 2 shows the average latency (in ms) measured over a 1 second period for a VM with varying (a) Read % and degree of sequentiality, (b) IO size and degree of sequentiality and (c) Read% and IO size. These experiments indicate that with such extreme changes in workload characteristics, the latency varies between 1.14 ms to 42 ms. The minimum happens for 4 KB, 100% read and completely sequential workload. The maximum occurred for 128 KB, 100% write and 100% random workload. The variance in bandwidth was from 8 MB/s to 177 MB/s. These results show that both bandwidth and latency can vary 20-30 times just with the variation in workload. Previous studies [11] on workload characterization have also presented similar results. Also, Figure 2(c) shows that latencies are typically higher for workloads with a mix of reads/writes compared to ones dominated by either type of requests. There are two main points: (1) Actual value of latency is not very high for any configuration and (2) Latency usually increases with an increase in IO size. The average latency per Byte is almost flat or decreasing. This gives us hope that one can detect overload at the array by using high latency threshold value, that would most likely be caused by longer array queues rather than just the change in workload.

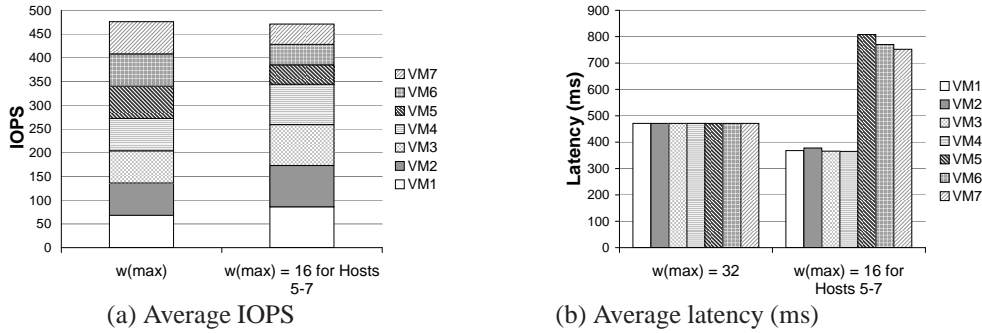


Figure 4. Bandwidth/Latency observed VMs when $w_{max} = 32$ for all hosts and when $w_{max} = 16$ for some.

4.2 Host Contention

Next we experimented with multiple hosts going to the same storage array. We increased the number of hosts from 1 to 4, where each is sending a workload of 32 KB IOs, 67% reads and 70% random. Each host is keeping 64 IOs pending at all times. Figure 3(a), (b) show the throughput and latency observed by each host with increasing contention at the array. This shows that latency increases almost linearly with an increase in the number of hosts and throughput stays almost constant after 2 hosts. This again shows that one can set a latency threshold which the controller will try to maintain and beyond a certain queue length the bandwidth is unaffected. We are currently doing experiments with more hosts and variable workloads to see the effect on latency and throughput.

4.3 Effect of Queue Limit

Finally, we experimented with fixed queue lengths per host to see the granularity of fairness provided. Queue lengths are a very coarse grained way to provide differentiated service. For this we varied the parameter w_{max} (maximum window size) at hosts to see the effect of throughput observed by hosts and overall fairness among different hosts. We experimented with two hosts, where each host is running one VM with typical load of 16 KB IOs, 67% reads and with 70% random IOs and 32 pending IOs at all times. Figure 3(c), (d) shows the bandwidths and latency observed by VMs at two hosts when the first host has a queue length of 32 and the queue length for the other host is varied from 2 to 32. We computed the degree of fairness based on variance from ideal weights and observed that fairness is quite acceptable and almost constant for different values of queue length.

We also experimented with a larger number of hosts accessing a 400 GB volume on a 5-disk RAID-5 disk group on the same array. The seven hosts in this experiment are each running one VM. The VMs are running a workload with 32 KB IOs, 67% reads, 70% random with 32 IOs pending

against 8GB virtual disks. Although each VM is keeping 32 IOs pending at all times, we experiment with two different host-level w_{max} settings: (a) $w_{max} = 32$ for all hosts and (b) $w_{max} = 16$ set for just hosts 5, 6 and 7. Figure 4 shows the throughput and latency obtained by the VM on each host. We observe that when we set $w_{max} = 16$, VMs on hosts 5, 6 and 7 get almost half the IOPS (~42 IOPS) compared to other hosts (~85 IOPS) and their latency (~780 ms) is doubled as compared to others (~360 ms). The latency measured at the hosts (instead of in the VM) is similar at all hosts in both the experiments. The overall latency decreases when one or more hosts are throttled. For example, in the second experiment the overall latency changes from ~470 ms at each host to ~375 ms at each host when $w_{max} = 16$ for hosts 5, 6, 7.

5 Related Work

Much work has been done towards providing quality of service in networks as well as storage systems. Many algorithms for QoS in routers have been proposed [2, 5, 7]. These algorithms are variants of generalized processor sharing and can only be applied in centralized settings where request arrival and release is under control of these algorithms. Stoica et al. [16] have proposed QoS mechanisms based on a stateless core, where only edge routers need to keep per flow state and core routers can provide fair queuing without keeping any state. This again relies at least on edge routers and some light weight support from core routers. In the absence of all these mechanisms, TCP has been serving us quite well both in terms of flow control and congestion avoidance. Commonly used variants of TCP (TCP-Reno, TCP-Vegas) use per flow (source-destination pair) information such as *estimated round trip time (RTT)* and *packet loss* at each host to adapt the overall window size to network conditions. Other variants (RED [6]) have also been proposed that again require some support from routers in terms of congestion signals. FAST-TCP [12] provides a purely latency based approach to improving TCP's throughput in high bandwidth-delay product networks. In this paper we adapt some of the techniques used by TCP and its variants

to do flow control in distributed storage systems. In so doing, we have addressed some of the issues that make it non-trivial to adapt TCP's solution to storage IO.

In case of storage systems, many schemes [3, 9, 10, 13, 18, 21] have been proposed to provide differentiated service to workloads accessing a storage array or a single disk. These techniques handle some storage specific issues but are again centralized and assume full control over release of requests to the disk. In case of distributed storage systems, Yin and Arif [19], Gulati et al. [8] have proposed proportionate bandwidth allocation mechanisms. These mechanisms are proposed for brick based storage systems and require each brick to run the scheduling algorithm independently. In our case, we don't have any control over the firmware on the array and no centralized IO proxy is available. Certain control theoretic approaches such as Triage [14] have also been proposed that observe the utilization of an array periodically and throttle the hosts based on their share of the observed bandwidth. These approaches work at a much coarser granularity as compared to our solution and thus may lead to under-utilization of the array. Also, they rely on a centralized controller to gather statistics and re-assign bandwidth to each host. Our approach only relies on per host measurements and control in order to provide fairness with high utilization. Friendly VMs [20] was proposed recently for fair sharing of CPU and memory, in which VMs monitor the system utilization by keeping track of their virtual time in comparison to real time. Our approach tries to apply similar ideas to distributed storage resource management, by explicitly measuring latency and adjusting issue queue lengths per LUN. Maintaining fairness and efficiency in storage access are more difficult because of well known issues of workload dependent variable capacity.

6 Conclusions and Future Work

In this paper, we studied the problem of providing coarse grained fairness to multiple hosts sharing a single storage system in a distributed manner. We propose to use average latency per host as an indicator for array overload and the limiting of the issue queue length per host as a means of doing flow control. Our initial investigations show that one can distinguish between latency changes due to workload and the higher average latency caused by the overload on the array. Secondly, we show that adjusting issue queue length per host based on the current average latency provides satisfactory fairness among hosts. High utilization is guaranteed by keeping a conservative latency threshold in the system to ensure a lower bound on the queue length per host. Implementation of a complete adaptive flow control mechanism for QoS in distributed storage access is ongoing work and will be presented in a full paper.

Acknowledgements: We would like to thank Carl Waldspurger, Tim Mann, Minwen Ji, Anne Holler, Narasimha

Ragunandana and David Eklov for valuable discussions and feedback.

References

- [1] Iometer. <http://www.iometer.org>.
- [2] J. C. R. Bennett and H. Zhang. *WF²Q: Worst-case fair weighted fair queueing*. In *Proc. of INFOCOM '96*, pages 120–128, March 1996.
- [3] D. D. Chambliss, G. A. Alvarez, P. Pandey, D. Jadav, J. Xu, R. Menon, and T. P. Lee. Performance virtualization for large-scale storage systems. In *Symposium on Reliable Distributed Systems*, pages 109–118, Oct 2003.
- [4] P. M. Chen, E. K. Lee, G. A. Gibson, R. H. Katz, and D. A. Patterson. RAID: High-performance, reliable secondary storage. *ACM Computing Surveys*, 26(2), 1994.
- [5] A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queuing algorithm. *Journal of Internetworking Research and Experience*, 1(1):3–26, September 1990.
- [6] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, 1993.
- [7] P. Goyal, H. M. Vin, and H. Cheng. Start-time fair queuing: A scheduling algorithm for integrated services packet switching networks. Technical Report CS-TR-96-02, UT Austin, January 1996.
- [8] A. Gulati, A. Merchant, and P. Varman. *dClock: Distributed QoS in heterogeneous resource environments*. In *Proc. of ACM PODC (short paper)*, August 2007.
- [9] A. Gulati, A. Merchant, and P. Varman. *pClock: An arrival curve based approach for QoS in shared storage systems*. In *Proc. of ACM SIGMETRICS*, pages 13–24, June 2007.
- [10] L. Huang, G. Peng, and T. cker Chiueh. Multi-dimensional storage virtualization. In *ACM SIGMETRICS*, June 2004.
- [11] I. Ahmad et. al. An Analysis of Disk Performance in VMware ESX Server Virtual Machines. In *IEEE Int. Workshop on Workload Characterization (WWC-6)*, Oct 2003.
- [12] C. Jin, D. Wei, and S. Low. FAST TCP: Motivation, Architecture, Algorithms, Performance. *Proceedings of IEEE INFOCOM*, March 2004.
- [13] W. Jin, J. S. Chase, and J. Kaur. Interposed proportional sharing for a storage service utility. In *ACM SIGMETRICS*, pages 37–48, June 2004.
- [14] M. Karlsson, C. Karamanolis, and X. Zhu. Triage: Performance differentiation for storage systems using adaptive control. *Trans. Storage*, 1(4):457–480, 2005.
- [15] C. Lumb, A. Merchant, and G. Alvarez. Façade: Virtual storage devices with performance guarantees. *USENIX FAST*, March 2003.
- [16] I. Stoica. Stateless core: A scalable approach for quality of service in the Internet, 2000.
- [17] VMware, Inc. *Introduction to VMware Infrastructure*. 2007. <http://www.vmware.com/support/pubs/>.
- [18] M. Wachs, M. Abd-El-Malek, E. Thereska, and G. R. Ganger. Argon: performance insulation for shared storage servers. In *FAST'07*, pages 5–5, 2007.
- [19] Y. Wang and A. Merchant. Proportional-share scheduling for distributed storage systems. In *Proc. of FAST*, Feb 2007.
- [20] Y. Zhang et. al. Friendly virtual machines. In *Proc. of VEE*, June 2005.
- [21] J. Zhang, A. Sivasubramaniam, Q. Wang, A. Riska, and E. Riedel. Storage performance virtualization via throughput and latency control. In *Proc. of MASCOTS*, Sep 2005.