Workload Shaping Framework for Storage Systems in Multi Client Environment

Amrita Dange, PG Student, IT Dept., Walchand College of Engg. Sangli, India 
B. S. Shetty, Asst. Professor, IT Dept., Walchand College of Engg. Sangli, India 
Sanjay Maind, AVP, JSoft Solutions Ltd. Bangalore, India

Abstract
Many storage workloads are bursty in nature. If the burst is not handled properly, it may affect the overall system performance causing resource provisioning challenge. In a shared storage environment, each client or application may have different Quality of Service (QoS) requirements. To meet individual client’s requirements, the workload shaping framework is introduced in a multi client environment. Using Response Time Threshold (RTT) decomposition algorithm, the multiple clients are combined using aggregated capacity and then the requests are decomposed into separate queues. This framework is evaluated using several real world storage workload traces. The results show that workload shaping: (i) reduces the server capacity requirements while minimally affecting QoS guarantees, (ii) provides better response time distributions over First Come First Serve (FCFS), a traditional scheduling method, and (iii) decomposition can be used to provide more accurate capacity estimation for several clients in a shared environment.

Keywords: Workload decomposition, Shared environment, Storage system, Quality of service, Resource management.

Introduction
The increasing complexity of handling huge amounts of data, providing high availability despite of hardware or software failures and the economic benefits of resource sharing, are driving storage systems towards a service oriented paradigm, in which personal and corporate clients lease space and access bandwidth on shared storage servers. In a usual setup, Service Level Agreements (SLAs) between the service provider and users require guarantees on throughput for rate-controlled users. The service provider must provide sufficient resources to meet these performance guarantees based on estimation of the resource demands of the individual clients, and the aggregate capacity requirements of the client mix. The runtime system must separate the clients to avoid intervention, and schedule their requests correctly. Without proper resource scheduling and management, some malicious users may send a load flow, resulting in performance degradation of the another well behaved clients. Separation ensures that the effects of a client's bad behavior are restricted to that client only. Discriminated services allow different clients to receive different guarantees [1].

Besides the need for performance separation of the clients, effective resource provisioning can bring economic benefits to the service providers. The performance SLAs typically deliver clients with minimum throughput guarantees, or response time bounds for speed-controlled clients. The server must provide adequate resources to make sure that the clients should obtain their specified performance. The storage workloads are bursty. With traditional methods, the effect of burst can not handled properly, it spreads across well behaved region and may disturb the overall system performance causing performance degradation and resource provisioning challenges [2].

A major challenge in data center operations is the need to deal efficiently with the bursty workloads arising in the network and storage server. A burst is an overload and the randomness of the requests coming to the server. If any explicit mechanism is not present to deal with it,}

![Fig.1. Architecture of Workload Shaping Framework for multi client environment](image)
scheduling the requests. At runtime, it monitors the workload, collects the I/O performance statistics, periodically adjusts the queue length according to request arrival rate. This framework is based on decomposition and recombination of the requests and uses the storage subsystem simulation tool DiskSim [3] to evaluate the performance of the algorithms.

Figure 1 shows the block diagram of workload shaping framework for multiple clients. The framework consists of decomposition algorithm RTT and recombination algorithm Miser. The RTT algorithm aggregates the capacity based on number of clients, combines the multiple workloads into single workload using arrival time of requests and then partitions the workload into two classes. In the recombination phase, the requests of the two classes are combined using Miser algorithm.

### A Survey On Scheduling Of The Workload

The proposed QoS schedulers for storage servers are generally based on Fair Queuing principles. They focus on providing a degree of performance guarantees for multiple clients in a shared environment. WFQ [4], WF2Q [5] and SFQ [6] allocate the disk bandwidth proportionally and divides the server capacity in a specified ratio. Facade [7] utilizes the earliest deadline first (EDF) algorithm to schedule the requests. At runtime, it monitors the workload, collects the I/O performance statistics, periodically adjusts the queue length according to request arrival rate.

pClock [8] monitors the request arrival pattern and checks for its conformance to a leaky bucket model. The throughput target is guaranteed by using fair queuing scheduling. Fahrrad [9] proposes a disk utilization reservation based scheduler for the periodic real time application. It proportionally allocates the disk time to different flows based on their periodic arrival pattern or response time target.

In network QoS [10], the traffic shaping is used to tailor the workloads to fit QoS-based SLAs. Typically, arriving network traffic is made to conform to a token-bucket model by monitoring the arrivals, and dropping requests that do not conform to the bucket parameters of the SLA. The size-aware schedulers [11] are used to improve performance of Web servers. The shortest remaining processing time (SRPT) scheduler gives preference to jobs with shortest remaining processing times to improve the mean response time of web servers.

### Methodology Of Workload Shaping Framework For Multi Client Environment

The proposed work has following methods:

1. Compute the aggregated capacity based on number of clients.
2. Combine the workloads of multiple clients using their arrival time.
3. Partition the workload into two classes.
5. Recombine the two classes using slack.

The framework consist of 2 algorithms as follows:

- Decomposition Algorithm: RTT
- Recombining Algorithm: Miser

The framework maintains two queues Q1 and Q2. The primary queue Q1 has bounded length to control the response time of requests accepted into it. The overflow queue Q2 holds requests that are not accepted into Q1 because their response time cannot be guaranteed. The requests are partitioned by RTT and the slack value is calculated to schedule requests in Q2 as early as possible [1].

### A. Decomposition Algorithm: RTT

The decomposition algorithm RTT for multi client environment is shown in Algorithm 1. The algorithm combines the workloads of multiple clients, aggregates the capacity based on number of clients and partitions the requests into two classes using the aggregated capacity. If the next incoming request causes the length of the primary queue Q1 (lenQ1) to exceed its maximum length (maxQ1), the request is directed to the overflow queue Q2; else, it is added at the end of the primary queue. The server has a capacity $C$ and the response time bound $r$ for the requests placed in the primary queue [1]. The maximum length of Q1 is calculated as,

$$\text{maxQ1} = C \times r$$

Algorithm 1: RTT Decomposition

Function RTT_Decompose();

1. Begin
2. Compute the aggregated capacity of server based on number of clients.
3. Compute $\text{maxQ1} = C \times r$
4. Combine the requests of multiple clients based on their arrival time.
5. Fetch the Request.
6. If ($\text{lenQ1} \leq \text{maxQ1}$) then
   6.1 Add request to Q1.
6.2 Increment lenQ1.
7. Else add request to Q2.
8. If there are requests then goto step 5.
9. End

B. Recombining Algorithm: Miser
The scheduler uses the free slack to schedule requests in Q2 as early as possible. Algorithm 2 shows the steps followed by Miser on request arrivals and completions.

On a request arrival, the function RTT_Decompose is called to classify the request. If the request is placed in the primary queue, it is allocated a slack value equal to the number of places available in Q1. A request in the overflow queue Q2 is scheduled when the slack value of the requests in Q1 is at least 1 [1].

Algorithm 2: Miser Scheduling

On a request arrival:
1. Begin
2. Call the function RTT_Decompose().
3. If request ri is placed in Q1 then
   3.1 Update ri_slack as ri_slack = [MaxQ1 - lenQ1]
       and
   3.2 minSlack as minSlack = min{minSlack, ri_slack}.
4. End

On a request departure:
1. Begin
2. If (minSlack>=1) and (Q2 is not empty) then
   2.1 Remove request from Q2 in First In First Out (FIFO) order and dispatch it for scheduling.
3. Else remove a request from Q1 in FIFO order and dispatch it for scheduling.
4. If scheduled request ri is from Q1 then
   4.1 Check if ri_slack = minSlack then
      4.1.1 Update minSlack as minSlack = min{j is from Q1} {ri_slack}
   5. Else for all i from Q1
      5.1 Update ri_slack as ri_slack = ri_slack – 1 and
      5.2 Update minSlack as minSlack = minSlack – 1
6. End

Experimental Results
The traces used for the evaluation are taken from two different storage applications: Web Search Engine (WebSearch) and OnLine Transaction Processing (OLTP) application (FinTrans). The traces are obtained from UMass Storage Repository [12]. All of these are block level storage I/O traces. The WebSearch traces are from a popular search engine and consist of user web search requests. The FinTrans traces are generated by financial transactions in an OLTP application running at two large financial institutions.

Two types of experiments are carried out:
(a) Measuring server capacity requirements as a function of the fraction of requests that are guaranteed a response time δ;
(b) Comparison of the response time distribution of recombination algorithm Miser with a non-decompose method FCFS.

Capacity v/s QoS
Because of the unpredictable bursty behavior of real workloads, to avoid resource over provisioning is difficult. This set of experiments shows the tradeoff between the fraction f of the workload that is guaranteed a response time bound δ, and the minimum server capacity Cmin required. For f = 100%, the minimum capacity required for all the requests is given to meet the response time bound. As f is relaxed to 90%, a smaller capacity is also sufficient. The results show that a very small percentage of the workload requires an overwhelming capacity to meet its guarantees.

Table 1: Capacity required for different fractions of WebSearch workload to meet the Response Time Target

<table>
<thead>
<tr>
<th>Response Time Bound (ms)</th>
<th>90</th>
<th>95</th>
<th>99</th>
<th>99.5</th>
<th>99.9</th>
<th>100</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>1650</td>
<td>2050</td>
<td>2770</td>
<td>3060</td>
<td>3820</td>
<td>6500</td>
</tr>
<tr>
<td>20</td>
<td>1175</td>
<td>1320</td>
<td>1703</td>
<td>1890</td>
<td>2308</td>
<td>4538</td>
</tr>
<tr>
<td>50</td>
<td>945</td>
<td>1068</td>
<td>1289</td>
<td>1367</td>
<td>1540</td>
<td>2620</td>
</tr>
<tr>
<td>100</td>
<td>924</td>
<td>1003</td>
<td>1169</td>
<td>1237</td>
<td>1321</td>
<td>1537</td>
</tr>
</tbody>
</table>

Fig. 2. Capacity required for different fractions of WebSearch workload to meet the Response Time Target
Table 1 and 2 shows capacity required for different fractions of WebSearch and FinTrans workload to meet a specified response time target respectively. Response time bounds of 10, 20, 50, 100 ms and f between 90% to 100% of the workload are considered. As can be seen in Table 1, with a 10 ms response time, the response time guarantee ranging from 90% to 100% of the workload requires large capacity increases 4 times (from 1650 to 6500 IOPS) for the WebSearch workload and from Table 2, with a 10 ms response time, the response time guarantee from 90% to 100% of the workload requires capacity increases almost 6.7 times (from 825 to 5500 IOPS) for FinTrans workload. Going from 99% to 100% the capacity required increases by a factor of 2.4 (from 2770 to 6500 IOPS) for WebSearch and by a factor of 4.1 (from 1350 to 5500 IOPS) for FinTrans.

### Table 2: Capacity required for different fractions of FinTrans workload to meet the Response Time Target

<table>
<thead>
<tr>
<th>Response Time Bound (ms)</th>
<th>% of Workload</th>
<th>90</th>
<th>95</th>
<th>99</th>
<th>99.5</th>
<th>99.9</th>
<th>100</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>10</td>
<td>825</td>
<td>1150</td>
<td>1350</td>
<td>1700</td>
<td>2150</td>
<td>5500</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>390</td>
<td>550</td>
<td>700</td>
<td>750</td>
<td>950</td>
<td>2450</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>250</td>
<td>308</td>
<td>421</td>
<td>463</td>
<td>551</td>
<td>1485</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>217</td>
<td>264</td>
<td>332</td>
<td>356</td>
<td>407</td>
<td>653</td>
</tr>
</tbody>
</table>

The results of section 4.1 show that to meet the response time guarantees for a relatively small fraction of the workload requires large server capacity. The effects of the bursts on the response time of the workload is explored in this section. In a data center, scheduling may be done using a fair queuing scheduler or other mechanisms. However, requests of a single client are usually handled using a simple FCFS scheduler. The response time distribution for 10, 20, 50 and 100 ms obtained for the unpartitioned WebSearch workload using FCFS scheduling and for the partitioned WebSearch workload using Miser is shown in Figure 4. Figure 5 shows the response time distribution for the FinTrans workloads for response time bounds of 10 ms, 20 ms, 50 ms and 100 ms respectively.

### Table 3: Response Time for different % of WebSearch workload

<table>
<thead>
<tr>
<th>% of Workload</th>
<th>Response Time</th>
<th>Response Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>FCFS</td>
<td>WSF</td>
</tr>
<tr>
<td>90</td>
<td>59.201984</td>
<td>75.099506</td>
</tr>
<tr>
<td>95</td>
<td>63.605785</td>
<td>79.757890</td>
</tr>
<tr>
<td>99</td>
<td>63.841192</td>
<td>75.590864</td>
</tr>
<tr>
<td>99.5</td>
<td>63.021832</td>
<td>74.620705</td>
</tr>
<tr>
<td>99.9</td>
<td>63.021832</td>
<td>74.620705</td>
</tr>
<tr>
<td>100</td>
<td>63.021832</td>
<td>74.620705</td>
</tr>
</tbody>
</table>

Fig. 4. Response Time v/s different % of WebSearch workload

In Figure 4, for a fraction of WebSearch workload f=90%, the non decompose method requires 59.201984 ms response time. Upto f=99.5%, the response time distribution increases and becomes 63.841192 ms. After that it decreases by some small amount and is in steady state for reaching 100%. In contrast, in the partitioned workload, the response time reaches to it's maximum value as 79.757890 ms at f=95%. It goes on decreasing as 74.620705 ms upto f=99.5% and after that it reaches to it’s steady state to serve 100% of the requests.
### Table 4: Response Time for different % of FinTrans workload

<table>
<thead>
<tr>
<th>% of Workload</th>
<th>Response Time FCFS</th>
<th>Response Time WSF</th>
</tr>
</thead>
<tbody>
<tr>
<td>90</td>
<td>10.999436</td>
<td>21.122065</td>
</tr>
<tr>
<td>95</td>
<td>11.030968</td>
<td>22.089949</td>
</tr>
<tr>
<td>99</td>
<td>11.167873</td>
<td>22.476931</td>
</tr>
<tr>
<td>99.5</td>
<td>11.162267</td>
<td>22.455644</td>
</tr>
<tr>
<td>99.9</td>
<td>11.128720</td>
<td>22.376548</td>
</tr>
<tr>
<td>100</td>
<td>11.103742</td>
<td>22.357271</td>
</tr>
</tbody>
</table>

### Fig. 5. Response Time v/s different % of FinTrans workload

In Figure 5, for a fraction of workload \( f = 90\% \), the non-decompose method requires 10.999436 ms response time. Upto \( f = 99\% \), the response time distribution increases and becomes 11.167873 ms. After that it goes on decreasing when \( f \) is reaching to 100%. In the partitioned workload, the response time distribution is same as that of unpartitioned workload. At \( f = 90\% \), the response time is 21.122065 ms and at 99%, it has value 22.476931 ms. After reaching to 99% of workload it decreases and has a value 22.357271 ms.

From the above experiments it can be seen that Miser is better able to guarantee a higher percentage of requests with small deadlines as 10, 20, 50 or 100 ms and with small capacity requirements as compared to FCFS. But the Miser has larger maximum response time than FCFS, because the decomposition based scheduler delays the requests in overflow queue Q2 over the requests in primary queue Q1 leading to larger delays for the overflowing requests in Q2. But the total number of large delay requests is much less than that for FCFS, even though the largest value may be higher.

### Conclusion

In this paper, the problem of resource provisioning and performance degradation in storage servers caused by the burst in many storage workloads is addressed. The burst present in a workload is only a small fraction of the whole workload but if it is not separated then it will affect the well-behaved portions of the workload. Here, a workload shaping framework is presented to address this problem in multi client environment. In this, the several workloads are first combined using aggregated capacity and then classified. While scheduling, the classes are recombined using the free slack. The decomposition algorithm RTT and the recombination Miser algorithm is used for workload shaping, and it is evaluated on several real storage traces. The evaluation results show significant capacity reduction, achieved by relaxing a small fraction of the workload from the response time guarantees. The scheduling framework can also get better response time distributions over FCFS for the same workloads. Finally, the decomposition could be used to provide more accurate capacity estimates for several multiple clients on a shared server, thus improving admission control decisions.

### Acknowledgement

Authors of this paper express the sincere thanks to JSoft Solutions Ltd and WCE Sangli about their guidance and cooperation and to all the authors, whose papers in the area of Workload Scheduling of Storage Systems published in various conference proceedings and journals.

### References


