2012 IEEE International Conference on Cluster Computing

Replication Based QoS Framework for Flash Arrays

Nihat Altiparmak and Ali Şaman Tosun
Department of Computer Science
University of Texas at San Antonio
San Antonio, TX 78249
{*naltipar,tosun*}*@cs.utsa.edu*

***Abstract*—The increasing popularity of the storage cloud is leading organizations to move their applications and enterprise data into the cloud. It is desirable to move time-critical applications demanding high performance I/O operations. Flash based storage arrays have emerged to address the high performance I/O requirements; however, providing predictable Quality of Service (QoS) for applications with real time data requirements is a challenging open problem. This paper introduces a QoS framework for flash based storage arrays. Our framework provides deterministic and statistical response time guarantees through a combination of techniques including replication, data mining, and online retrieval. We evaluated the framework using synthetic and real-world traces. The QoS performance of the system is compared to the existing high-throughput RAID designs. Numerical results show that under the synthetic traces, QoS performance of the proposed system outperforms the existing high performance RAID designs. Real world traces indicate that the proposed QoS mechanism is tunable to support the guarantees required by various real world applications.**

***Keywords*-storage qos; replication; parallel i/o; design theory**

I. Introduction

With the emergence of storage cloud, companies now provide storage based services for a wide range of applications [1], [4]. Customers can store their data on the cloud with certain reliability and access requirements without buying the storage devices and dealing with the operating cost. Since the organizations are challenged to respond to the changes in a most cost-effective way possible, interest in moving the applications and enterprise data centers to the cloud is expected to increase [23]. Real-time applications running on the cloud and enterprise data centers generally have strict response time requirements. Some of these applications include multimedia streaming with cloud players, business critical applications such as online transaction and query processing, virtual reality, scientific applications with real time storage requirements, and video/game on demand. We believe that these high performance real-time applications running on the cloud or on the enterprise data centers would benefit a lot from a QoS mechanism that can provide deterministic or statistical response time guarantees.

Quality of service (QoS) is a mechanism that is proposed to offer performance guarantees by controlling the distribution of resources. Providing QoS for storage systems is crucial in order to offer some performance guarantees for the data access. The importance of QoS for distributed network storage service is first emphasized by Chuang et al. [17]. They clearly state the superiority of replication over caching to provide QoS guarantees for storage systems. Wei et al. [34] propose a QoS mechanism for distributed real-time databases using replication. They divide data objects into two types, temporal and non-temporal data; and they fully replicate the temporal data in their system. However, the details of the replication scheme such as how the replicas are distributed over the devices, how many replicas should be chosen, and how the retrieval should be performed are not investigated. Hence, we believe that additional studies of providing QoS for storage systems are necessary in order to address these issues.

In this paper, we propose a replication based QoS mechanism for flash based storage arrays. Our QoS mechanism provides deterministic and statistical response time guarantees for data access using simple admission control mechanisms. We evaluate the performance of the proposed framework using DiskSim [16] with synthetic and real world application traces. Numerical results show that under the synthetic traces, QoS performance of our framework outperforms the existing high performance RAID solutions. Real world traces indicate that the proposed QoS framework is tunable to support the guarantees desired by the real world applications by applying data mining techniques, online retrieval of the replicas and adjusting the copy/device amount.

The rest of the paper is organized as follows: In section II, we present the motivation behind this work with the necessary background information. Section III describes the proposed deterministic and statistical QoS frameworks. We adapt the framework to the real world applications in section IV and the performance of the system is evaluated in section V. Finally, we conclude with section VI.

II. Motivation and Background

In this section, we first provide background information on flash arrays. We explain why flash arrays are a better choice for high performance storage systems and providing QoS. Next, we describe declustering, the role of replication in providing QoS, and the important aspects of choosing a suitable replicated declustering scheme for QoS.

A. Flash Arrays

Flash based storage devices are rapidly gaining market share. Flash technology offers non-volatile storage, fast random-access and low power consumption making them ideal. Flashes are already deployed in smart phones, digital cameras and MP3 players. Packing the flash memory in hard disk drive (HDD) form resulted the solid-state drive (SSD) and SSDs have received a lot of attention by the research community recently. Recent improvements in flash density led academia and industry to consider storage arrays entirely based on flash technology. Figure 1 shows a simplified block diagram of a general flash array. Flash array is composed of multiple

978-0-7695-4807-4/12 $26.00 © 2012 IEEE
DOI 10.1109/CLUSTER.2012.53

flash modules connected to a controller enabling parallel retrieval and reliability. Each flash module is composed of multiple flash packages having its own flash module controller (FMC), DRAM and error-correcting code (ECC) mechanism. Some research has already been performed to enhance the reliability and performance of flash arrays [26], [27]. Several products have been launched [5], [6], [7], [9], [10] composed of hundreds of parallel flash modules, offering terabytes of capacity and millions of I/O per second (IOPS).

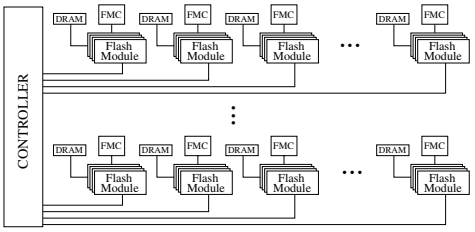


Fig. 1. General structure of a flash array

Although HDD based storage arrays are traditionally used for high performance applications, performance of HDD was limited by 15K revolutions per minute (RPM) disks over years. By reducing the power consumption and increasing the overall throughput, flash based storage arrays have potential to address the challenges faced by storage cloud providers and enterprise data centers for the applications requiring high performance I/O operations. Different than the traditional HDD based storage arrays, flash arrays do not have variable delays caused by mechanical process of accessing disk data such as rotational delay, seek time, head/cylinder switch time and spin-up time. Because of these unpredictable delays, proposing a QoS framework for traditional HDD based storage arrays cannot exceed providing a best effort performance rather than giving response time guarantees. The reasons mentioned above motivate the usage of flash based storage arrays for providing QoS on storage systems.

B. Replicated Declustering for QoS

Multi-device architectures offer the opportunity to exploit I/O parallelism during retrieval. The most crucial part of exploiting I/O parallelism is to develop storage techniques that access the data in parallel. A common approach for efficient parallel I/O is partitioning the data space into disjoint regions (buckets) and allocating the data to multiple devices. This process is called declustering. When users issue a query, data falling into disjoint partitions is retrieved in parallel from multiple devices. Optimally, in a system with N devices, retrieving b buckets requires $\lceil \frac{b}{N} \rceil$ parallel accesses to the storage device.

Providing QoS guarantees on storage arrays is difficult for many reasons. First, if no assumptions are made about the placement of data on the storage array, in the worst case all the I/O requests can end up being on the same device and require serial retrieval. Second, requirements for I/O requests are dynamic and not easy to predict. Third, providing guarantees in a system with multiple devices and multiple users is challenging. Choosing a suitable declustering scheme for QoS that allows replication of buckets plays an important role to cope with these challenges.

1) Role of Replication: Replication plays a vital role in providing QoS. When replication is used, each bucket is stored on multiple devices and we can choose one of the devices for the retrieval of the bucket. Let k be the number of buckets that can be retrieved from a single device in period T. Using a single copy, the only guarantee we can provide is that any k buckets can be retrieved in time T since all the buckets can be in the same device. Using replication we can increase this number significantly without making any restricting assumptions since we can choose an alternate device for buckets that are on the same device. When c copy replication is used, each bucket is replicated over c devices.

2) Replicated Declustering Schemes: Many replicated declustering schemes were proposed in the literature ranging from random schemes to the schemes based on combinatorial designs. Major schemes include random duplicate allocation (RDA) [29], partitioned allocation [19], dependent periodic allocation [33], orthogonal allocation [20], [30], and design-theoretic allocation [31]. A replicated declustering scheme should have three important features for it to be suitable for storage QoS. First one is the worst case retrieval cost to retrieve b buckets using replication. The scheme should be able to provide deterministic low retrieval cost guarantees. Secondly, the guarantees provided by the scheme should not depend on a specific query type, it should be valid for all type of queries. Lastly, the scheme should have an efficient retrieval algorithm to decide the replica to be used for retrieval of a bucket.

RDA stores a bucket on devices chosen randomly from the set of devices. The retrieval cost of RDA is found to be at most one more than the optimal with high probability [29]; however, the scheme cannot provide any guarantees since it is based on a random selection. In partitioned replication, the set of devices are divided into groups and devices in one group are replicated on other devices in the same group. Although the partitioned allocation performs reasonable for range queries, it performs poorly for arbitrary and connected queries. Dependent periodic allocation allocates a shifted version of the first copy as the second copy. It performs well for the queries including buckets near to each other such as range and connected queries; however, the performance degrades for arbitrary queries. Orthogonal allocations ensure that when the devices that a bucket is stored at are considered as a pair, each pair appears only once in the allocation. It can guarantee a retrieval cost of at most $\lceil \sqrt{b} \rceil$ for an arbitrary query containing b buckets. Design-theoretic allocation assigns buckets to devices using the blocks of a combinatorial design. An $(N, c, 1)$ design for c copy replicated declustering using N devices guarantees that $(c-1)M^2 + cM$ buckets can be retrieved using at most M accesses. The last parameter ensures that each device pair appears only once in the allocation.

3) Replicated Declustering Scheme Chosen: Although both orthogonal and design-theoretic allocations can provide deterministic response time guarantees without depending on

the query type, our choice for replication strategy is design-theoretic allocation for the following reasons. First of all, the guarantees given by the design-theoretic allocation is better than the orthogonal allocation. For $c = 2$, design theoretic allocation can retrieve $(2-1)1^2 + 2*1 = 3$ buckets in $M = 1$ access, 8 buckets in 2 accesses, and 15 buckets in 3 accesses; however, orthogonal allocation requires $\lceil\sqrt{3}\rceil = 2$ accesses for 3 buckets, 3 accesses for 8 buckets, and 4 accesses for 15 buckets. Secondly, finding an orthogonal allocation for higher dimensions of data are challenging and it only supports regular grid structures. On the other hand, depending on the response time requirement of the application, a suitable design providing the requested guarantees can be chosen easily by changing the copy and the device count of the design-theoretic allocation. More information about combinatorial block designs can be found in [18]. Finally, design-theoretic allocation provides an efficient retrieval algorithm. Readers are directed to [32] for an in depth comparison of these replicated declustering schemes and their retrieval strategies.

4) Example of Design-theoretic Allocation: (9,3,1) design of design-theoretic allocation is given in Figure 2. Each column in the figure is a design block. The notation (9,3,1) means we have 9 numbers (0 to 8), each design block has 3 elements and every pair appears together in only 1 design block. 0 and 1 appear together only in the first block. Two different design blocks can have at most one element in common. The numbers in a design block are used to determine the devices a bucket should be stored at. For example, first design block has $(0, 1, 2)$. This means 3 copies of a specific bucket can be stored at devices 0, 1, and 2. Rotations of the design blocks can also be used to assign buckets to devices in order to support more buckets. Rotation of the design block $(0, 1, 2)$ produces the design blocks $(1, 2, 0)$ and $(2, 0, 1)$. (9,3,1) design supports $\frac{N*(N-1)}{c-1} = \frac{9*8}{2} = 36$ buckets with rotations.

block											
0	0	0	0	1	1	1	2	2	2	3	6
1	3	4	5	3	4	5	3	4	5	4	7
2	6	8	7	8	7	6	7	6	8	5	8

Fig. 2. (9,3,1) design

III. QoS Framework

In this section, we discuss the proposed QoS framework in detail. We assume that the storage array has N flash modules (devices) and we divide the time into T sized intervals. Applications issue block requests to the system at the beginning of each interval and the goal is to complete the retrieval of requests before the end of the interval. Next interval starts with a new set of requests. Our goal is to provide deterministic or statistical guarantees to the applications. We use replication to provide the guarantees. In the proposed scheme, applications specify block requests and the admission control algorithm either accepts or rejects/delays some of the request depending on whether it can provide the guarantee. This approach provides deterministic guarantees. By using the properties of the allocation scheme in use, statistical guarantees can also be provided. For statistical QoS, we can guarantee that the probability that the set of requests cannot be retrieved in an interval is less than some threshold ϵ.

A. Deterministic QoS

Using design-theoretic allocation, $(c-1)M^2 + cM$ buckets can be retrieved using M accesses when c copies of the data is used. In (9,3,1) design, there are 9 devices and 3 copies. We set the value of M to 1. Using the proposed system we can retrieve $(3-1)*1^2 + 3*1 = 5$ requests in 1 access. Assume that there are 3 applications in the system and they request data blocks as shown in Table I. *Application 1* joins at time T_0 and has a request size of 2 block requests per period. *Application 2* has a request size of 2 block requests per period and requests admission at time T_1. Since the total request size includes only *Application 1*, *Application 2* is admitted and total request size is set to 4. *Application 3* requests admission at period T_2 and has a maximum request size of 1 block request per period. Since the maximum request size is 5 based on system parameters and current request size is 4, *Application 3* is admitted and total request size is set to 5, which is the limit. So, no more applications can be admitted until one of the applications leaves the system. Specific block requests the applications make is given in Table I. A triple (a, b, c) denotes the block whose first copy is stored at device a, second copy is stored at device b, and third copy is stored at device c.

TABLE I
I/O REQUESTS

Period	*Application 1*	*Application 2*	*Application 3*
T_0	$(0, 3, 6)$ $(5, 7, 0)$		
T_1	$(0, 4, 8)$	$(8, 0, 4)$ $(7, 0, 5)$	
T_2	$(1, 2, 0)$		$(6, 0, 3)$
T_3	$(1, 4, 7)$	$(1, 3, 8)$ $(0, 5, 7)$	$(0, 1, 2)$

1) Admission Control: Using design-theoretic allocation, admission control is quite simple. Any $S = (c-1)M^2 + cM$ buckets with c copies can be retrieved in an interval T with M parallel accesses to the storage device. The time T is determined depending on the storage device in use. Admission control algorithm checks whether the new request can be satisfied without exceeding this limit. If the request cannot be completed within this interval, it can either be rejected or delayed to the next available interval.

B. Statistical QoS

In order to provide statistical QoS, we use statistical information about the distribution of data blocks to the devices based on the properties of the specific design in use. The goal in statistical QoS is to guarantee that the probability that the set of requests cannot be retrieved in a desired interval is less than some threshold ϵ. In deterministic scheme we assumed that whenever the number of requests are greater than S, retrieval cannot be completed within the interval. However, this is not always true. Let us look at an example. Consider the setting given in Table I. Using this setting any 5 block requests can be retrieved in 1 access. However in some cases even 9 block requests can be retrieved in 1 access. An example is given in Figure 3. 9 block requests

$\{(0,1,2),(1,2,0),(2,0,1),(3,8,1),(4,8,0),(5,7,0),(6,0,3),(7,0,5),(8,1,3)\}$ are non-conflicting and can be retrieved in 1 access from the devices shown in the figure. Using sampling, we can find the optimal retrieval probabilities for different request sizes and improve the model by incorporating these probabilities into the admission control system. By this way, we can accept a greater number of requests than S.

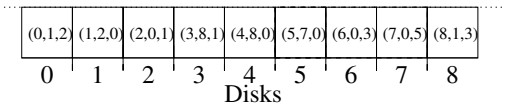


Fig. 3. Example for full retrieval

1) Sampling: Using the properties of our design, we can find the optimal retrieval probabilities of different request sizes by sampling. To find the optimal retrieval probabilities for each request size k, we need to choose k random blocks out of available design blocks and check their retrieval optimality. Repeating this process many times will give us a good estimate about the optimal retrieval probability for the request size of k. Figure 4 shows the optimal retrieval probabilities for the (9,3,1) design. The same design block is allowed to be chosen multiple times for fair results. As expected, the probability takes its lowest values for k that is a multiple of $N = 9$. Precisely, the probability of being optimal for $k = 6$ is 0.99, for $k = 7$ is 0.98, for $k = 8$ is 0.95, and for $k = 9$ is 0.75. In other words, deterministic QoS rejects the 6th request even if there is an optimal retrieval 99% of the time. The probability increases to 1 for $k = 10$ since the optimal retrieval requires $\lceil \frac{b}{N} \rceil = \lceil \frac{10}{9} \rceil = 2$ accesses for 10 buckets. The probability converges to 1 as k increases.

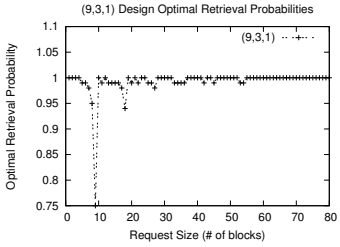


Fig. 4. Optimal retrieval probabilities of (9,3,1) design

2) Admission Control: As in the admission control of the deterministic case, if the number of requests are not greater than the limit S, the requests are admitted immediately. If the number of requests are greater than S, then we use the probabilities found in the previous section. Let P_k denote the probability that the request size k is retrieved optimally. P_k has already been found previously using sampling. Let N_k denote the number of intervals encountered with the request size of k and N_t denote the total number of intervals encountered. N_k and N_t can simply be calculated by keeping $k + 1$ counters. Then, $R_k = \frac{N_k}{N_t}$ will give us the probability of the interval with the request size of k and $(1 - P_k)$ will give us the probability that the interval with the request size of k cannot be retrieved optimally. Let Q denote the probability that the set of accepted requests cannot be retrieved optimally. Admission control algorithm admits the requests of the current interval if $Q = \sum_{i=1}^{max\{k\}} (1 - P_k) * R_k$ is smaller than ϵ. If Q is greater than or equal to ϵ, then the requests are rejected or necessary requests are delayed to the next available interval depending on the user preferences.

C. Retrieval of Requests

Retrieval uses the retrieval algorithm of design-theoretic allocation. Retrieval of requests in Table I is illustrated in Figure 5. Labels on blocks denote the devices they are stored at and a block is placed over device i if it is to be retrieved from device i. Each block is initially mapped to the device on which the first copy of it is stored. If retrieval requires more than 1 access, remapping is used to map the blocks to other devices using the second or the third copy. For the periods T_0, T_1, and T_2; initial mapping requires 1 access and remapping is unnecessary. For T_3 initial mapping requires 2 accesses and remapping is required. The block $(0,1,2)$ is remapped to device 2 since its third copy is stored on device 2. The block $(1,3,8)$ is remapped to device 3 since its second copy is stored on device 3.

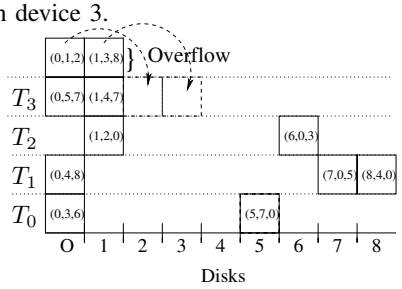


Fig. 5. Retrieval of requests in Table I

Although the retrieval algorithm of design-theoretic allocation guarantees the optimal retrieval for the request sizes not greater than S, it does not guarantee an optimal retrieval for the request sizes greater than S, which may happen in statistical QoS. Finding the optimal retrieval schedule in this case requires solving maximum flow problem. Readers are directed to [14], [15] for details of solving maximum flow algorithm to find the optimal retrieval schedule. For a query size of b, design-theoretic retrieval requires $O(b)$ time and max-flow algorithm requires $O(|b|^3)$ time. Since the design-theoretic retrieval is faster than solving the maximum flow problem, our retrieval algorithm first checks the retrieval optimality using the design-theoretic retrieval, if the access amount is greater than the optimal ($\lceil \frac{b}{N} \rceil$), we solve the maximum flow problem.

IV. Adaptation to the Real World

There are two issues that need to be tackled in order to support real world applications. First of all, we have limited number of design blocks in our designs but generally a storage system has more data blocks (buckets). Therefore, we need an efficient way of matching these data blocks to the design blocks so that the data blocks requested within a close time interval are matched to the different design blocks. This will minimize the possibility of block requests to be retrieved serially from the same device. For this assignment, we are using Frequent Itemset Mining (FIM). The second issue is the unexpected delays caused by the retrieval. Since the retrieval algorithm of design-theoretic allocation assumes the blocks to

be retrieved at the beginning of every interval, requests coming within an interval should be processed at the beginning of the next interval. This requirement introduces an unexpected delay to the response time of the block requests. In order to eliminate this retrieval delay, we propose an online retrieval algorithm.

A. Frequent Itemset Mining

FIM aims to find frequently occurring subsets in a sequence of sets. The original motivation of the problem comes from the need to analyze supermarket customer behavior such that how often items are purchased together. Frequency is determined by a user specified number called $support$. Generally, found patterns by FIM are expressed as association rules such as x *number of customers who bought* $item1$ *also bought* $item2$ for a set size of two or y *number of customers who bought* $item1$ *and* $item2$ *together also bought* $item3$ for a set size of three, where $x, y \geq support$. Since its first introduction in 1993 by Argawal et al. [12], there has been many studies and algorithms proposed for FIM. These algorithms can be categorized as variants of one of three different base algorithms; Apriori [13], Eclat [35], and FP-growth [21].

In our case, since we have limited number of design blocks; i.e. 36 blocks for the (9,3,1) design, we need to match these design blocks to the data blocks of the storage system efficiently. Our intuition for this matching is that the data blocks that are generally requested together (or within a short time interval) should be matched to the different design blocks to increase the chance of parallel retrieval. Although this matching can be done randomly or in a round-robin fashion, FIM is a better choice since it can find the data blocks that are frequently requested together. Therefore, we first investigate the trace of the storage system and determine the data blocks that are requested within a short time interval T. This interval can be determined according to the response time of the storage system in use. Then, we mine this dataset using FIM for set size of 2. Finally, FIM returns the frequent data block pairs that are requested together with their $support$ number showing how many times those data blocks are requested together. Most FIM algorithms also accept minimum support number as a parameter, where we can decrease the mining time by increasing this minimum support. Matching of the design blocks to the data blocks is done by using the information returned by the FIM such that the data blocks requested together are mapped to the different design blocks. The data blocks that are not returned by FIM (previously requested alone or new data blocks that are not encountered in the history) are matched to the design block number returned by $(dataBlockNumber\%numberOfDesignBlocks)$.

B. Online Retrieval

Retrieval algorithm of design-theoretic allocation assumes that block requests arrive at the beginning of each interval. In order to apply this algorithm to the real world scenario, we need to move the block requests that arrive within an interval to the beginning of the next interval. Although retrieval algorithm guarantees optimum retrieval given that the number of requests accepted are within the limit S, this alignment of data blocks will introduce unexpected delays that may violate the guaranteed response time. In order to eliminate this retrieval delay, we developed an online retrieval algorithm that retrieves the blocks as soon as they arrive.

In online retrieval, instead of an interval based approach, a time based approach is used. Except the requests that come exactly at the same time, the requests are retrieved in a first-come, first-served (FCFS) fashion. The requests that come exactly at the same time are retrieved together as previously such that if the retrieval requires more than the optimal number of accesses, necessary remappings are performed. Different than the previous retrieval approach, a block is preferably retrieved from the device having the earliest finish time if no idle device is available. Next, we compare the performance of design-theoretic and online algorithms theoretically.

1) Comparison of Retrieval Algorithms: In this section we develop a theoretical framework to compare interval based approach that uses design theory and online approach using online retrieval algorithm. In the first model, requests arriving within an interval are scheduled at the beginning of the next interval; the second model schedules requests as soon as they arrive. To make a fair comparison we assume that there is no *backlog*. That means requests received in an interval are retrieved at the end of the next interval using online algorithm.

Let k denote the set of requests, let $DTR(k)$ denote the number of accesses required using design-theoretic retrieval algorithm, and let $TDTR(k)$ denote the time to retrieve k requests using design-theoretic algorithm. Let $OLR(k)$ denote the number of accesses required using online algorithm, and let $TOLR(k)$ denote the time to retrieve k requests using online algorithm. Following theorem outlines our comparison.

Theorem 1: If there is no backlog and $OLR(k) = DTR(k)$ then $TOLR(k) \leq TDTR(k)$.

Proof: Since there is no backlog, we need to consider only requests received during the previous interval. If $OLR(k) = DTR(k)$, then starting the requests as soon as they are received results in earlier or equal retrieval time. ■

For some set cardinalities, $OLR(k) = DTR(k)$ and for some sets they may or may not be equal depending on the actual set. The following is for $(9, 3, 1)$ design.

TABLE II
COMPARISON OF RETRIEVAL ALGORITHMS

$\|S\|$	1	2	3	4	5	6
DTR(S)	1	1	1	1	1	2
OLR(S)	1	1	1	1 or 2	1 or 2	2

V. EVALUATION

The first objective of our evaluation is to test whether the proposed QoS mechanism can provide the specified access guarantees. We first analyze how the system performs comparing it to the existing high performance RAID mechanisms by using synthetic workloads and design-theoretic retrieval. After that, we adapt our QoS mechanism to the real world scenario such that requests are retrieved online and limited number of blocks are mapped to the fairly bigger number of data blocks efficiently. Then, we evaluate the performance of the new system under real world workloads. The system is evaluated using trace-driven simulations. Modified version of DiskSim [16] is used as a simulator.

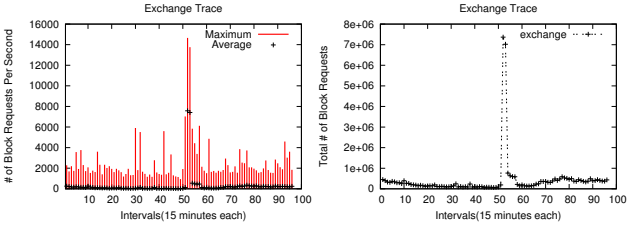
A. Simulator

DiskSim is an efficient, accurate, highly-configurable disk system simulator originally developed to support research into various aspects of storage subsystem architecture. It has been used in a variety of published studies to understand modern storage performance. DiskSim does not support simulation of flash based storage systems; however, there exist an extended version of DiskSim developed by Microsoft Research to provide support for flash simulation [11]. According to the parameters set by Microsoft Research, a single read request (one block=8KB) takes 0.132507 milliseconds.

B. Workloads

In this section, we explain the workloads used in our experiments. We used synthetic workloads for design-theoretic retrieval and real world workloads to show the adaptability of the system to the real world applications.

1) Synthetic Workload Generation: We have developed a trace generation tool that produces ASCII format input trace for DiskSim. It requires the number of devices, interval duration, and the number of blocks to be requested for each interval, and produces the trace by randomly selecting the blocks to be requested from the available design blocks.



(a) Exchange - reads/sec. (b) Exchange - total reads

(c) TPC-E - reads/sec. (d) TPC-E - total reads

Fig. 6. Real world traces

2) Real World Traces: As real world workloads, we use popular multi-device server traces previously used in various storage related studies [11], [24], [25]. They are publicly distributed via the online trace repository provided by SNIA (Storage Networking Industry Association) [3]. The first workload we use is the Exchange workload, which is taken from a server running the Microsoft Exchange 2007 inside Microsoft [22]. It is a mail server for 5000 corporate users consisting of 9 active volumes and about 40 million block read requests. The trace covers a 24-hour weekday period starting at 2:39pm on the 12th December 2007 and it is broken into 15-minute intervals. The second workload we use is TPC-E, which is an online transaction processing (OLTP) benchmark simulating the workload of a brokerage firm [8]. The TPC-E trace covers total of 84 minutes of workload consisting of 13 active volumes and about 101 million block read requests. The trace is taken on 18th October 2007 and broken into 6 parts of 10-16 minutes each. Trace statistics are provided in Figure 6. For each interval, Figure 6(a) and Figure 6(c) show the maximum and average number of read requests per second for the Exchange and the TPC-E traces respectively. Figure 6(b) and 6(d) show the total number of read request made in each interval of the Exchange and the TPC-E traces.

C. Deterministic QoS with Design-theoretic Retrieval

In this section, we compare the QoS performance of the proposed deterministic scheme with the well known high performance RAID mechanisms using synthetic workloads, design-theoretic retrieval, and the (9,3,1) design. Proposed QoS mechanism for (9,3,1) design with the design-theoretic retrieval guarantees the retrieval of 5 data blocks in 1 access (M=1), 14 data blocks in 2 accesses (M=2), and 27 data blocks in 3 accesses (M=3) during the given time interval T. Three different traces are generated by using the synthetic workload generator. The first trace requests 5 data blocks for every 0.133 milliseconds, the second trace requests 14 data blocks for every 0.266 milliseconds and the last trace requests 27 data blocks for every 0.399 milliseconds. All the requests are placed at the beginning of each time interval. Total number of 10000 block requests are made in each trace and the requested blocks are chosen randomly from a pool of 36 blocks. Request size is set as 8KB (one block), which is known to take 0.132507 milliseconds according to the parameters of the simulator. Therefore, every request is expected to be completed within the given time interval T.

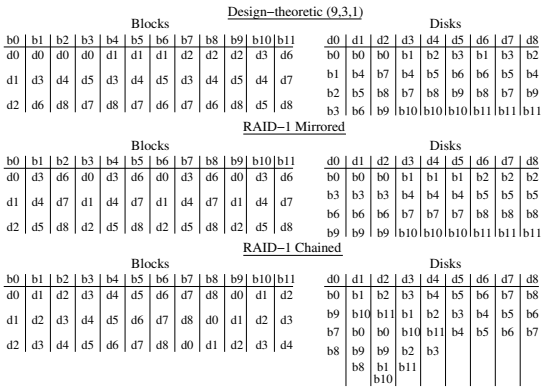


Fig. 7. Allocation schemes

1) Allocation Schemes: For comparison, we use two famous RAID mechanisms that allow replication; *RAID-1 mirrored* and *RAID-1 chained*. The comparison is made with respect to their I/O driver response times, which is defined as the time between sending the I/O request and receiving the corresponding response. In all designs, each data block is replicated over 3 devices and a total of 9 devices are used. All the allocation schemes are shown in Figure 7. *d0, d1, ... , d8* denote the devices and *b0, b1, ... , b11* denote the blocks. For each design, the right chart shows the content of each device and the left chart shows in which devices a specific block is stored at. In RAID-1 mirrored design, 9 devices are configured as 3 groups of mirrored devices. Each device in every group contains the same data such that *d0*, *d1*, and *d2*

TABLE III
COMPARISON OF ALLOCATION SCHEMES: RESPONSE TIMES (MS)

Request Size (blocks)	Interval (ms)	RAID-1 Mirrored			RAID-1 Chained			(9,3,1) Design-theoretic		
		Avg	Std	Max	Avg	Std	Max	Avg	Std	Max
5	**0.133**	0.136	0.024	**0.517**	0.132	0.002	**0.263**	0.132	0.000	**0.132**
14	**0.266**	0.221	0.175	**2.150**	0.187	0.065	**0.524**	0.185	0.064	**0.263**
27	**0.399**	5.006	12.931	**59.771**	0.304	0.170	**1.180**	0.263	0.106	**0.393**

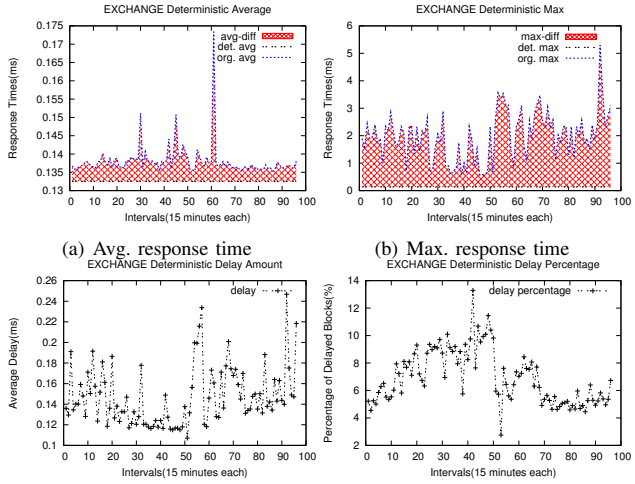
is a group and they are mirrors of each other. In RAID-1 chained design, instead of mirroring the whole content, if the primary copy of a data block is allocated in device *i*, other two copies are allocated in devices $\{(i+1) \mod 9\}$ and $\{(i+2) \mod 9\}$. Although it is not shown in the figure, rotations of the design blocks are also used such that total of 36 data blocks are supported by each design.

2) Simulation Results: In Table III, I/O driver response time averages, standard deviations and maximum response times for all design are given for different M and T values. According to the results, the QoS guarantees are fulfilled by the proposed mechanism such that all the block requests are completed within the given time interval. This can be observed by the maximum response time values given in the table. For the time interval of 0.133 ms, maximum response time of (9,3,1) design is 0.132507 ms. Similarly, for the time intervals of 0.266 ms and 0.399 ms, maximum response times of the proposed scheme are within the limits. However, other designs fail to fulfill the given QoS guarantees. Although for some intervals their response time averages are within the limits, maximum response times show that some requests took longer than the given guarantee. Especially the performance of the RAID-1 mirrored design decreases dramatically as the number of block requests increase since the possibility of the requests ending up with the same device increases.

D. Deterministic QoS with Online Retrieval

In this section, we evaluate the performance of our QoS framework using real world workloads and online retrieval algorithm. For the Exchange trace, we use the (9,3,1) design and for the TPC-E trace, we use the new (13,3,1) design that supports 13 devices. The details of the (13,3,1) design and other designs for different amount of devices can be found in [18]. We compare the performance of the traces with their original stand. For the original stand, every block request is retrieved from the device it is stated in the trace. For this experiment, we first use FIM to map the design blocks to the data blocks and then use design-theoretic allocation in order to distribute the data blocks to the devices according to the design in use. For FIM, we use the trace one previous than the current interval for mining and use the result of this mining in the current interval. Longer history can be used for better matching of the design blocks to the data blocks. In order to make a fair comparison, the requests are aligned to 8KB of block sizes as in DiskSim. By this way, the given guarantee for every request is defined as 0.133 milliseconds, slightly larger than the response time of one block request in DiskSim (0.132507).

1) Simulation Results: Figure 8(a) and 8(b) show the average and the maximum response times for Exchange trace. In both of the figures, the bottom line shows the response time of the deterministic QoS and the top line shows the response time of the original trace. The performance difference is plotted with a filled pattern. The bottom line is 0.132507 ms in every interval. In other words, all the requests are completed within the given QoS guarantees for the deterministic QoS. The top lines are clearly above the given QoS guarantees for both the average and the maximum response times. In order to achieve the deterministic QoS, the admission control mechanism has to either cancel or delay some of the necessary requests that violate the guarantees. Since canceling the requests may effect the running state of applications, we choose the delay option. Figure 8(c) and Figure 8(d) show the average delay amount and the percentage of the delayed requests for the same experiment. Average delay amount for the delayed requests oscillates between 0.1 and 0.25 ms and the percentage of the delayed requests is between 3% to 13%. In average, 7% of the requests are delayed about 0.14 ms.



(a) Avg. response time (b) Max. response time

(c) Avg. delay amount (d) Delay percentage

Fig. 8. Exchange deterministic

Figure 9 shows the results of the same experiment for the TPC-E trace. The first column for each trace interval shows the average/maximum response times of the deterministic QoS, which are both 0.132507 ms for every interval. The second and the third columns show the average and the maximum response times of the original trace. Although the average response time of the original trace is close to the deterministic QoS, it violates the guarantees given by being 0.135145 ms in average. The maximum response time for the original trace clearly exceeds the QoS limits in every interval. The labels on each interval show the percentage of the delayed requests and the average delay amount. Percentage of the delayed requests are about 2 to 3% and average delay amount is about 0.03 ms.

E. Statistical QoS with Online Retrieval

In this section, we show the results of the statistical QoS for the Exchange and the TPC-E workloads using online retrieval.

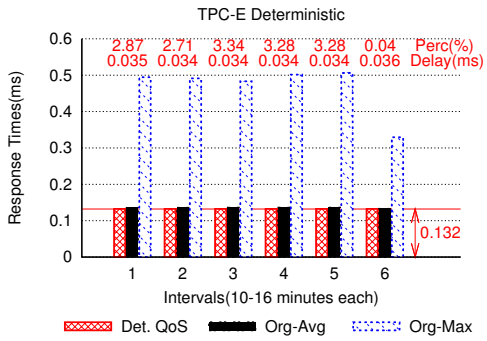
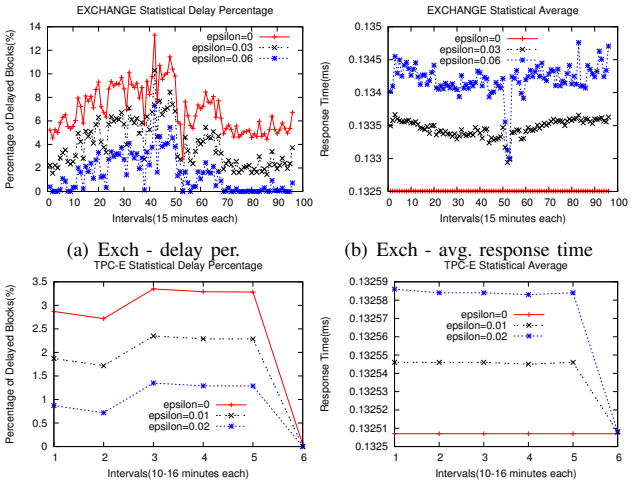


Fig. 9. TPC-E deterministic

We use the same designs and the experimental settings as in Section V-D. In this case, we can control the percentage of the delayed requests using ϵ, which represents the probability that a request cannot be completed within the given limits. Note that for the deterministic case $\epsilon = 0$ and the percentage of the delayed requests is expected to decrease as the ϵ increases. However, this decline in the percentage of the delayed request is also expected to cause an increase in the average response time of the system.



(a) Exch - delay per. (b) Exch - avg. response time

(c) TPC-E - delay per. (d) TPC-E - avg. response time

Fig. 10. Exchange statistical

1) Simulation Results: Figure 10(a) and 10(c) show the percentage of the delayed requests for the Exchange and the TPC-E workloads for different values of ϵ. As expected, percentage of the delayed requests decreases as the ϵ increases. Figure 10(b) and 10(d) show the average response times of the Exchange and the TPC-E workloads for the same ϵ values. Since we are not delaying some of the conflicting requests depending on the ϵ value, average response times increase as the ϵ increases.

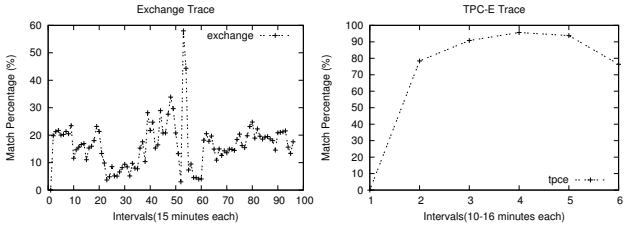
F. FIM Performance

In this section, first we show time and memory requirement of FIM and then the benefit of using FIM for the Exchange and the TPC-E traces. In our experiments, we use *fim apriori-lowmem* [28] implementation since it can deal with large datasets efficiently. Source code of the implementation is publicly available in [2]. Time and memory requirement of *fim apriori-lowmem* for $set\ size = 2$ can be found in Table IV. For this experiment, we used a machine with Intel Xeon E5205 Dual CPU Dual Core processors having total of 4 cores, each core with 1.86 GHz of clock speed, and 16GB of physical memory running on an Ubuntu 10.04 operating system. The program uses one core only. As an interval T, we chose 0.133 ms since it is the response time in our system. We show the results for the trace intervals having the largest and the smallest request sizes of both exchange and TPC-E traces. According to the results for $support = 1$, Exchange takes 1 to 11 seconds to mine with the peak memory usage of 240 to 767 MB and TPC-E takes 1 to 90 seconds to mine with the peak memory usage of 0.3 to 3.4GB. Mining time and the peak memory usage can be reduced by increasing the $support$ such that tpce3 takes 56 seconds to mine with 2.2GB peak memory usage for $support = 3$.

TABLE IV
PERFORMANCE OF FIM

Trace	Requests Size	Support	Peak Memory	Time
exch48	14.3 K	1	240 MB	1.08s
exch52	6.8 M	1	767 MB	11.43s
tpce6	104 K	1	316 MB	1.21s
tpce3	27.6 M	1	3.4 GB	1m30s
tpce3	27.6 M	3	2.2 GB	56.69s

Figure 11(a) and Figure 11(b) show the percentage of blocks that are matched according to the FIM results for Exchange and TPC-E respectively. In FIM, we mined only the trace one previous than the current trace and used this information for the current trace. Therefore, the result is 0 for the first interval. For Exchange, in average 17% of the blocks found mining the previous interval is encountered in the current interval. For TPC-E; in average 87% of the blocks found mining the previous interval is encountered in the current interval.



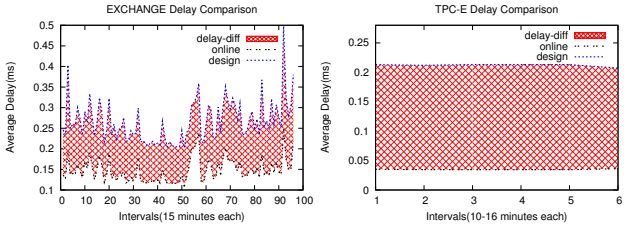
(a) Exchange (b) TPC-E

Fig. 11. Matching performance of FIM results

G. Retrieval Performance

In Section V-D, we showed the average delay amount of the delayed requests for the Exchange and the TPC-E traces using online retrieval. In this section, we use the design-theoretic retrieval for the same settings to compare the performances of these two retrieval algorithms in terms of the average delay amounts they introduce. Figure 12(a) and 12(b) show the average delay amounts caused by the Exchange and the TPC-E traces respectively. In both of the figures, bottom lines show the delay amount introduced by the online retrieval and the top lines show the delay amount introduced by the design-theoretic retrieval. The performance difference of the retrieval algorithms is plotted with a filled pattern. Since the

online retrieval retrieves the requests as soon as they arrive, there is no additional retrieval delay caused for the alignment of the requests as in the design-theoretic retrieval. For the Exchange trace, online retrieval causes 0.12 ms lesser delay in average than the design-theoretic retrieval. For the TPC-E trace, online retrieval causes 0.17 ms lesser delay in average than the design-theoretic retrieval.



(a) Exchange (b) TPC-E

Fig. 12. Delay comparison of retrieval algorithms

VI. Conclusion

Many applications require high performance I/O operations with real time performance guarantees. Flash based storage arrays have emerged as a feasible approach for these applications; however, a simple and practical QoS framework is crucial to provide predictable performance. In this paper, we proposed a novel replication based QoS framework for flash arrays. The proposed framework is simple and provides deterministic or probabilistic response time guarantees using simple admission control algorithms. With extensive experimental results, we show its practicality, superiority over the existing RAID solutions and applicability to the real world scenarios by using real world traces. Utilization of the system can be tuned by adjusting the parameters. We believe proposed framework will lead to better utilization of flash arrays by improving application performance with QoS framework.

VII. Acknowledgments

This work is partially supported by Army Research Office (ARO) Grant W911NF-11-1-0170.

References

[1] Amazon simple storage service (amazon s3). http://aws.amazon.com/s3/.
[2] Frequent itemset mining imp. repository. http://fimi.ua.ac.be/.
[3] Iotta repository. http://iotta.snia.org. Storage Networking Ind. Assoc.
[4] Windows azure. http://www.microsoft.com/windowsazure/.
[5] Sun storage f5100 flash array. http://www.oracle.com/us/043970.pdf, 2009. Oracle Datasheet. Available online (6 pages).
[6] Nimbus data s-class enterprise flash storage systems. http://www.nimbusdata.com/products/Nimbus_S-class_Datasheet.pdf, 2010.
[7] Ramsan-630 flash solid state disk. http://www.ramsan.com/files/download/212, August 2010. Texas Memory Systems White Paper.
[8] Tpc benchmark e. http://tpc.org/tpce/spec/v1.12.0/TPCE-v1.12.0.pdf, June 2010. Standard Specification.
[9] Violin 3200 flash memory array. http://www.violin-memory.com/assets/3200-datasheet.pdf, 2010. Violin 3200 Memory Datasheet.
[10] Violin 6000 flash memory array. http://www.violin-memory.com/assets/Violin_Datasheet_6000.pdf?d=1, 2011. Violin 6000 Memory Datasheet.
[11] Nitin Agrawal, Vijayan Prabhakaran, Ted Wobber, John D. Davis, Mark Manasse, and Rina Panigrahy. Design tradeoffs for ssd performance. In *ATC'08: Usenix Annual Technical Conference*, pages 57–70, Berkeley, CA, USA, 2008. USENIX Association.
[12] Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. Mining association rules between sets of items in large databases. In *SIGMOD '93: Proceedings of the international conference on Management of data*, pages 207–216, New York, NY, USA, 1993. ACM.
[13] Rakesh Agrawal, Heikki Mannila, Ramakrishnan Srikant, Hannu Toivonen, and A. Inkeri Verkamo. Fast discovery of association rules. pages 307–328. American Association for Artificial Intelligence, Menlo Park, CA, USA, 1996.
[14] Nihat Altiparmak and A. Ş. Tosun. Generalized optimal response time retrieval of replicated data from storage arrays, 2012. Technical Report.
[15] Nihat Altiparmak and A. Ş. Tosun. Integrated maximum flow algorithm for optimal response time retrieval of replicated data. In *41st International Conference on Parallel Processing (ICPP 2012)*, Pittsburgh, Pennsylvania, September 2012.
[16] John S. Bucy, Jiri Schindler, Steven W. Schlosser, Gregory R. Ganger, and Contributors. The disksim simulation environment version 4.0 reference manual. Technical report, Carnegie Mellon University Parallel Data Lab, May 2008.
[17] John Chung-I Chuang and Marvin A. Sirbu. Distributed network storage service with quality-of-service guarantees. In *INET*, 1999.
[18] C.J. Colbourn and J.H. Dinitz. *The CRC handbook of combinatorial designs*. The CRC Press series on discrete mathematics and its applications. CRC Press, 1996.
[19] H. Ferhatosmanoglu, A. Ş. Tosun, G. Canahuate, and A Ramachandran. Efficient parallel processing of range queries through replicated declustering. *Journal of Distributed and Parallel Databases*, 20(2):117–147, 2006.
[20] H. Ferhatosmanoglu, A. Ş. Tosun, and A. Ramachandran. Replicated declustering of spatial data. In *ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 125–135, June 2004.
[21] Jiawei Han, Jian Pei, and Yiwen Yin. Mining frequent patterns without candidate generation. In *SIGMOD '00: Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, pages 1–12, New York, NY, USA, 2000. ACM.
[22] S. Kavalanekar, B. Worthington, Qi Zhang, and V. Sharda. Characterization of storage workload traces from production windows servers. In *IEEE International Symposium on Workload Characterization, IISWC 2008.*, pages 119 –128, sept. 2008.
[23] Emerson Liebert. Taking the enterprise data center into the cloud. http://whitepapers.datacenterknowledge.com/content11369.
[24] D. Narayanan, A. Donnelly, E. Thereska, S. Elnikety, and A. Rowston. Everest: Scaling down peak loads through i/o off-loading. In *Operating Systems Design and Implementation*, pages 15–28, 2008.
[25] D. Narayanan, E. Thereska, A. Donnelly, S. Elnikety, and A. Rowston. Migrating server storage to ssds: Analysis and tradeoffs. In *EuroSys 2009*, pages 145–158, April 2009.
[26] Kwanghee Park, Dong-Hwan Lee, Youngjoo Woo, Geunhyung Lee, Ju-Hong Lee, and Deok-Hwan Kim. Reliability and performance enhancement technique for ssd array storage system using raid mechanism. ISCIT'09, pages 140–145, Piscataway, NJ, USA, 2009. IEEE Press.
[27] Ilia Petrov, Guillermo Almeida, Alejandro Buchmann, and Ulirch Grf. Building large storage based on flash disks. In *Proceedings of ADMS 2010, In conjunction with VLDB 2010*, September 2010.
[28] Balázs Rácz, Ferenc Bodon, and Lars Schmidt-Thieme. Benchmarking frequent itemset mining algorithms: from measurement to analysis. In *Proceedings of ACM SIGKDD International Workshop on Open Source Data Mining (OSDM'05)*, pages 36–45, Chicago, IL, USA, August 2005.
[29] P. Sanders, S. Egner, and K. Korst. Fast concurrent access to parallel disks. In *ACM-SIAM Symposium on Discrete Algorithms*11^{th}, 2000.
[30] A. Ş. Tosun. Replicated declustering for arbitrary queries. In *ACM Symposium on Applied Computing*19^{th}, pages 748–753, March 2004.
[31] A. Ş. Tosun. Design theoretic approach to replicated declustering. In *International Conference on Information Technology Coding and Computing*, pages 226–231, April 2005.
[32] A. Ş. Tosun. Analysis and comparison of replicated declustering schemes. *IEEE Transactions on Parallel and Distributed Systems*, 18:1578–1591, November 2007.
[33] A. Ş. Tosun and H. Ferhatosmanoglu. Optimal parallel I/O using replication. In *Proceedings of International Workshops on Parallel Processing (ICPP)*, pages 506–513, Vancouver, Canada, August 2002.
[34] Y. Wei, S. Son, J. Stankovic, and K. Kang. Qos management in replicated real-time databases. In *RTTS*, 2003.
[35] Mohammed J. Zaki. Scalable algorithms for association mining. *IEEE Trans. on Knowl. and Data Eng.*, 12(3):372–390, 2000.