

QoS Mig: Adaptive Rate-Controlled Migration of Bulk Data in Storage Systems

Koustuv Dasgupta Sugata Ghosal Rohit Jain Upendra Sharma Akshat Verma
 {kdasgupta,gsugata,rohithjain,supendra,akshatverma}@in.ibm.com

IBM India Research Lab

Abstract— Logical reorganization of data and requirements of differentiated QoS in information systems necessitate bulk data migration by the underlying storage layer. Such data migration needs to ensure that regular client I/Os are not impacted significantly while migration is in progress. We formalize the data migration problem in a unified admission control framework that captures both the performance requirements of client I/Os and the constraints associated with migration. We propose an adaptive rate-control based data migration methodology, QoS Mig, that achieves the optimal client performance in a differentiated QoS setting, while ensuring that the specified migration constraints are met. QoS Mig uses both long term averages and short term forecasts of client traffic to compute a migration schedule. We present an architecture based on Service Level Enforcement Discipline for Storage (SLEDS) that supports QoS Mig. Our trace-driven experimental study demonstrates that QoS Mig provides significantly better I/O performance as compared to existing migration methodologies.

I. INTRODUCTION

Migration of data between different storage devices is an important technique used in storage systems management for handling situations such as device failures and component upgrades, and for evolving the system to keep pace with increasing space and performance requirements of applications. In differentiated QoS systems, data migration and its variations like data replication, are also used to dynamically adapt to variations in workloads, and alleviate performance issues like hot spots. Data migration is also necessitated by reorganization at the level of information systems like file systems and database systems. In database systems, physical rearrangement of database is performed for various operations such as compaction to reduce fragmentation and clustering of related records or objects on disk to improve disk I/O performance [18], [19]. For large databases, the data migration duration could last several hours, during which client access to the system could be severely affected. Today’s IT environments are expected to provide 24x7 service, so it becomes important to execute migration in a manner that minimizes the impact on client access to the system. At the same time, migration activity could have associated objectives that must also be satisfied. A common objective is to complete migration within a specified deadline. This is often the case when the data

is migrated at a very coarse granularity (like a database table or a storage volume) and concurrent access by clients to data being migrated is not supported by the database system. A more interesting case is when the data is moved at a relatively fine granularity (like a database record or a disk block) and the database system is capable of providing online access to the remaining data. Migration can now be carried out so that the adverse impact on client access is balanced by the fulfillment of the migration objective. For example, migration required for clustering related database records within adjacent disk blocks is likely to improve performance of the transactions that access those records. The decreased I/O performance experienced by transactions during migration should be weighed against this expected gain, and the migration rate be adapted accordingly.

Bulk data migration schemes in practice today take a simplistic approach by scheduling migration either as a low priority maintenance activity, or at the other extreme, at the highest priority without regards to the impact on client access. In this paper, we present QoS Mig, a novel approach to data migration in storage systems that balances the migration objectives with the client I/O performance requirements. This approach provides a flexible approach to data migration that frees up administrators from the unenviable task of computing migration schedules that are not overly disruptive to client traffic and still satisfy the migration objectives. In addition, QoS Mig enables logical reorganization of data to be used more effectively by limiting the impact on concurrent client accesses. This study does not present any new method to determine system configurations that deliver better performance nor does it present techniques for maintaining consistency during migration. Rather, it uses available techniques for them [18].

A. QoS-aware Migration Framework and Contribution

We consider a storage system that provides statistical QoS guarantees. QoS guarantees are provided on a request *stream* defined as an aggregation of I/O requests from an application. The logical grouping of data accessed by the stream is referred to as a *store*, which is backed up by physical space on storage volumes. Typically, QoS guaran-

tees, more formally referred to as service level agreements (SLA), specify an upper bound on the stream request rate and also an obligation on the system to serve a minimum fraction of these requests within a specified latency bound. In this setup, each request is associated with a reward value that the system accrues when the request is serviced within its latency bound. A migration task refers to the movement of a store from one backing physical volume to another volume. We introduce the notion of *migration utility function* to capture the benefit of migration in terms of additional reward that the system will earn after migration is completed. The key idea behind QoSMig is to consider the individual migration requests that constitute the migration task and to *establish the relative importance* of these migration requests vis a vis client I/O requests by analyzing the migration utility and the expected distributions of arrival (and service) times of I/O requests in near-term future. Rewards are assigned to individual migration requests based on the migration utility function and I/O rewards, thus enabling the processing of migration and I/O requests in a unified framework. QoSMig uses an online admission control algorithm to admit requests that maximize the reward earned by the system. Figure 1 presents a functional model of QoSMig that depicts the input-output relationship.

An important feature of QoSMig is that it not only adapts the rate of migration to soak up the spare system bandwidth, but at times of system overload, it has the capability of giving priority to migration requests over I/O requests in order to achieve the migration objective. This has an important benefit that it always results in predictable migration completion deadline given long-term client traffic statistics thus making it extremely useful in today’s service-oriented environments. We describe in this paper how QoSMig is used in the specific and important case of a fixed deadline, as well as in the general case where QoSMig computes a suitable deadline and completes migration within the deadline.

The admission controller used in QoSMig to maximize overall rewards is based on a performance model of the underlying storage system. Numerous researchers have proposed performance models that take into account increasingly complex features of modern storage systems [21], [11]. While QoSMig approach is valid for all types of storage systems (though no new performance models are proposed by this study), to focus on the working of QoS-Mig, we use an empirical performance model of a single disk in our experimental study. The model is constructed by increasing the load (in terms of I/Os per second) and measuring the resulting average response times. To achieve a target average response time, the input workload must be limited to a certain value as given by the model. We call this the disk request processing capacity C (referred to as disk capacity).

The rest of this paper is organized as follows. In Sec. II, we formulate the problem as a reward maximization prob-

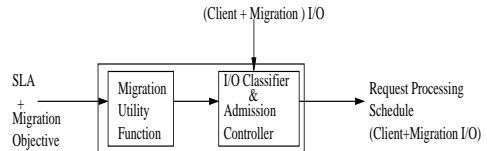


Fig. 1. QoS Mig Functional Model.

lem and propose a scheduling-based approach to solve it. In Sec. III, we describe our methodology for assigning rewards to migration requests in various scenarios, and our admission control and scheduling technique as a solution to the profit maximization problem. Next, in Sec. IV, we discuss some implementation issues relevant to our approach in practical settings. Finally, we describe our experimental setup and results in Sec. V, and conclude with relating our work to current research in Sec. VI.

II. QOSMIG OVERVIEW AND ARCHITECTURE

In this section, we propose a reward maximization based formulation of migration scheduling problem followed by an architecture for incorporating a solution to the problem in QoS based storage systems. We also present a methodology for computing the utility of migration requests relative to the client I/Os in SLA based systems.

The migration problem is formulated as follows. We are given as input a migration task, i.e. a store of size B_m that needs to be moved from a source to a destination, and a set of n application requests to be serviced by the system. Each application request has an associated reward function $R(\delta)$ that gives the reward obtained when the request is served with a latency of δ . The utility of completing migration task in time t is given by migration utility function $U_m(t)$. Intuitively, $U_m(t)$ gives the additional reward earned if migration is completed at time t . We assume that the benefits of migration start accruing only after the entire store is migrated.¹ We denote \mathcal{C} to be the total number of I/O requests (application or migration) that the disk can serve in a unit time.

Our objective is to schedule the migration task such that the overall reward earned by the system, i.e., the sum of the reward generated from satisfying the application requests and the utility obtained by executing the migration task is maximized. In essence, we are interested in finding the rate of migration $M_r(t)$, at each time t , and migration completion time T_M , that maximizes the total reward. Formally we have,

$$\max_{M_r(t), T_M} [\sum_{i=1}^n R_i(\delta_i) + U_m(T_M)], \quad (1)$$

where δ_i is the latency of the i th application request, $\int_0^{T_M} M_r(t) dt = B_m$, and the total capacity used by migration and I/O requests is no more than available disk capacity C at all times.

¹While this is indeed true for most databases, handling the case where migration benefits start accruing incrementally is an interesting open problem.

Note that, in the case where the migration task has a deadline T_{max} , we need to solve the following optimization problem,

$$\max_{M_r(t), T_m \leq T_{max}} [\sum_{i=1}^n R_i(\delta_i) + U_m(T_m)]. \quad (2)$$

QoSMig works in the following manner. Each migration task is divided into a large number of smaller migration requests. We assign a reward to each of these migration requests in a way that captures the utility function of the migration task. We then process the application requests and migration requests in a unified manner and use an admission control algorithm that strives to maximize the total reward obtained by serving the admitted requests. Note that even though the utility of migration task accrues after its completion, the profit maximizing admission controller is oblivious to it and accrues rewards for the migration requests. It is our proposed reward assignment technique which ensures that an optimal solution to the unified reward maximization problem obtained thus is also an optimal solution for the migration problem. In the following subsections, we describe the QoSMig architecture and detail out the reward and utility models used.

A. Architecture

The main architectural requirement for implementing QoSMig is a controller module that resides in the data path between clients and storage server so that it can regulate the flow of I/O and migration requests. Recently, some controller designs for providing statistical QoS guarantees on storage systems have been proposed [7], [8]. We have chosen to implement QoSMig in SLEDS controller proposed by Chambliss *et al.* in [8]. A schematic architecture of SLEDS with QoSMig enhancements is presented in Figure 2. SLEDS provides statistical performance guarantees on a wide variety of storage systems. The controller logic is implemented inside a gateway module that monitors client I/O streams and performs traffic shaping and policing based on monitored performance metrics. A central server called SLA server stores the client SLAs and collates the monitoring data from the gateways to effect throttling controls in the gateways. SLEDS assumes that the storage system is provisioned to satisfy the client performance target in terms of response time as long as the offered load by the client (in terms of I/Os per second) does not exceed a certain maximum. If some streams suffer QoS violations, it frees up system resources for their use by throttling streams that contend with these streams and are exceeding their maximum load limit. For implementing QoSMig, SLEDS is enhanced by adding a migration utility generator module to the SLA server. This module takes client SLAs and migration objectives like deadline etc. to generate a migration utility function. Migration requests, like I/O requests, are sent to the gateway modules. A migration *reward tagger* module is added to the gateway module that assigns appropriate rewards to individual migration requests before

forwarding them to the *admission controller*. A *predictor* module is also added to the gateway for short-term forecast of the client traffic. SLEDS employs a leaky-bucket traffic shaping policy for regulating flow of I/O requests. For QoSMig, we augment this with an admission control policy that maximizes the objective function as specified in Eqn. 1. The admission controller admits requests from different streams, including migration stream, to maximize the overall reward. The admitted requests are sent to the storage system for processing.

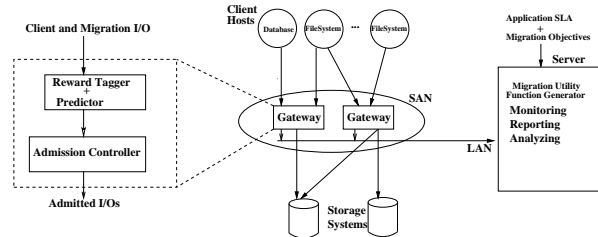


Fig. 2. QoSMig System Architecture. Gateways intercept all I/O requests including migration requests and effect admission control policy according to the SLAs and migration objectives.

B. I/O Reward Model

Each application I/O request r_j is represented as $r < a_j, s_j, \Gamma_j, R_j(\delta) >$, where a_j is the arrival time of the request, s_j is the duration of the request, Γ_j is the stream with which r_j is associated, and $R_j(\delta)$ is the reward generated by the request if it is served with a delay of time δ . Note that the reward for an I/O request is represented as a function of the delay incurred by the request. We consider only those reward functions that are non-increasing with increase in delay. Further, for ease of exposition, we consider step-function reward functions only, i.e., $R_j(\delta) = R$ if $\delta < D_j$ and 0 otherwise. To take an example, for a streaming media workload, a request must be served within a latency bound LB (e.g., frame refresh latency in video replay) and so the deadline $D_j = LB$ and the reward value R could be proportional to the priority of the application. For a workload that has the objective of minimizing average response time a linearly decreasing reward function may be appropriate.

C. Migration Utility Model

The migration utility function $U_m(t)$ is used to denote the utility of completing migration in time t . Migration is typically used as a technique to transition from one system configuration to another that provides improved performance and/or availability. In the latter case, migration utility can often be captured as a step function. For example, disaster recovery mechanisms require asynchronous replication of data to a remote site within a deadline to bound the time lag between the primary and the replica. Similarly, in the case of an anticipated disk failure, it is essential to complete migration before a deadline defined naturally by the mean time to disk failure. The utility can be perceived

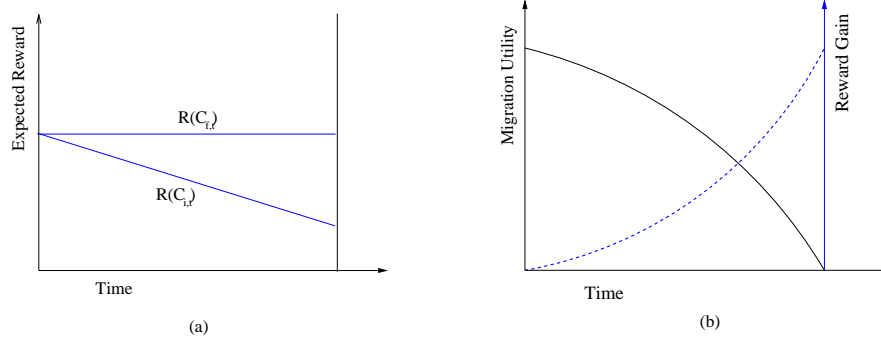


Fig. 3. (a) Reward distribution w.r.t. time for two disk configurations C_i and C_f . (b) The corresponding migration utility is a non-increasing function of time.

as the inverse of the business loss incurred in case the migration deadline is not met. [15] presents an approach for capturing the costs associated with such failures. Thus, migration utility is captured by the following step function,

$$U_m(t) = \begin{cases} \mathcal{U} & \text{if } t \leq T_{max} \\ 0 & \text{otherwise} \end{cases} \quad (3)$$

where T_{max} is the deadline and \mathcal{U} is the business loss incurred by the disk failure.

In the former case, utility function for migration is more involved. To illustrate with an example, if the migration is performed for database reorganization to reduce disk fragmentation or to balance workload, the average transaction processing time after migration decreases. For systems, that have to take data offline for reorganization, there is an additional constraint of finishing the migration within a stipulated deadline. Intuitively speaking, if migration is completed at time t , then the benefits of migration start accruing from that point onwards. Hence, the slope of the migration utility curve at time t should capture the potential benefits lost because migration was not completed earlier than t . Hence, we define the slope of $U_m(t)$ as the rate of reward loss, where reward is a measure of the benefit, at t . Hence, again migration curve can be described a general utility w.r.t delay (t). Figure 3(a) shows an example where expected reward before and after migration is denoted by $\mathcal{R}(C_i, t)$ and $\mathcal{R}(C_f, t)$ respectively, where the reward calculations are based on the future client request traffic and expected system performance as predicted by the model. The benefit of migration at time t could be thought of as the additional reward that the system earns in the new configuration i.e. $(\mathcal{R}(C_f, t) - \mathcal{R}(C_i, t))dt$. The reward gain then is summation of all the benefits accrued till time t and can be computed as an integral of the benefit (the dotted line in Figure 3(b)). The corresponding migration utility is simply an inverse of the reward gain and hence a non-increasing function of time t (given by the solid line).

III. QOSMIG: ADAPTIVE RATE-CONTROLLED DATA MIGRATION

We now present our method for assigning rewards to migration requests that ensures that the admission control al-

gorithm solves the problem described in Eqn. 1. Consider the migration of a store S starting at time T_0 , that migrates B_m amount of data (migration requests) from a disk with disk capacity C to another disk. Since the behavior of our proposed method (Migration-aware BSRJF) is complex, for an intuitive understanding of the reward assignment, consider the following simple optimal admission control algorithm OAC . At any given time t , OAC sorts all requests, which have not been rejected, in decreasing order of the reward associated with them. It then selects as many requests as it can without violating the disk capacity constraint. Now consider the reduced scenario where requests arrive at times $T_0 + ks$, $k \in \mathbb{N}$. Also, all requests have duration equal to s . One may note that each independent sub-problem at any time $T_0 + ks$ is a knapsack problem and OAC is the greedy strategy that leads to an optimal solution for this version, where all requests have unit capacity [24]. We discuss general admission control methodology later, but OAC illustrates the intuition behind the reward assignment of migration requests better.

A. Migration With Deadline

We start with the scenario where there is an expected deadline for migration, e.g. 6 hours. In such a case, it is reasonable to assume that migration incurs no penalty as long as the migration completes in time close to the deadline (e.g., a violation by 5 minutes for a migration task of 6 hours). Our aim is to design a reward function for migration requests such that an AC algorithm selects the requests (I/O and migration) in a manner such that the expected deadline is met and the loss in I/O rewards incurred due to migration is minimized.

Let T_{max} denote the expected deadline for migration. The utility of migration is \mathcal{U} if it is completed before the deadline, and 0 otherwise (step-function). Let $C_m = B_m / (T_{max} - T_0)$ denote the average bandwidth required for migration to meet the deadline. Further, let N_m be the total number of migration requests that need to be scheduled, and $N_m(t)$ be the number of migration requests remaining at time t .

We specify the *potential reward* of a migration request to

be R_m s.t.,

$$\lambda \int_{R_m}^{\infty} c_r p_r \leq C - C_m, \quad (4)$$

where c_r is the expected capacity used by requests with reward r , λ is the expected number of requests present at any time given time, and p_r is the probability that a request has reward r .

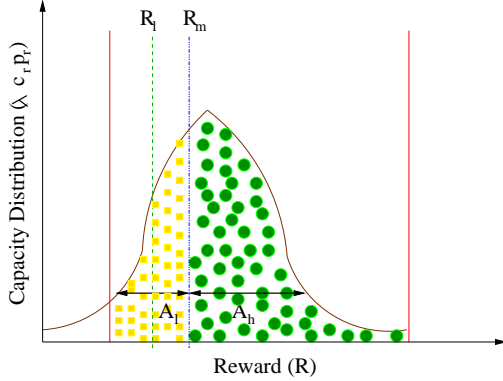


Fig. 4. Capacity Distribution of Requests with Rewards

Eqn. 4 ensures that the disk capacity available after servicing the migration requests is sufficient to service all I/O requests that have rewards higher than the potential reward of the migration requests. The area A_l in Fig. 4 denotes the expected capacity taken by migration requests and the area A_h denotes the expected capacity taken by such high reward requests. We want this to be equal to the capacity available after serving migration requests.

The actual reward $R_m(t)$ for a migration request at time t is given by,

$$R_m(t) = \begin{cases} 0 & \text{if } \frac{U}{N_m(t)} < R_m, \\ R_m & \text{otherwise} \end{cases} \quad (5)$$

where R_m is the potential reward given by Eqn. 4.

The case where $R_m(t)$ is different from R_m refers to a scenario where migration has such a low utility that the deadline can be met only by rejecting I/O requests with higher rewards. In this case, we reject all migration request by setting their rewards to 0.²

With the above reward model for migration and assuming all statistical estimates to be accurate, an optimal admission control (OAC) algorithm would serve all requests that are in the region A_h (in Fig. 4) along with the migration requests. This leads us to the following lemma.

Lemma 1: The expected completion time of migration by the OAC algorithm is T_{max} if the (potential) reward of a migration request is R_m . Also, the algorithm services all requests in A_h and rejects all requests in A_l , thereby maximizing the reward generated by the service provider.

²In practice, we do not expect this to happen since a migration task will be typically associated with a high enough utility.

To ensure that migration finishes strictly before the deadline, migration requests can be assigned the highest priority when the remaining disk bandwidth before deadline is just enough to meet the migration load.

B. Migration with General Utility Functions

As discussed earlier, there are scenarios where migration utility can be represented as a general function w.r.t. the time t taken to complete the migration. In such a scenario, even the time to complete the migration (i.e. deadline) is not explicit. We next present a method that calculates an optimal target deadline for migration to be completed. We also propose a reward model for migration requests in this scenario, such that the AC algorithm maximizes the total reward generated by servicing I/O and migration requests.

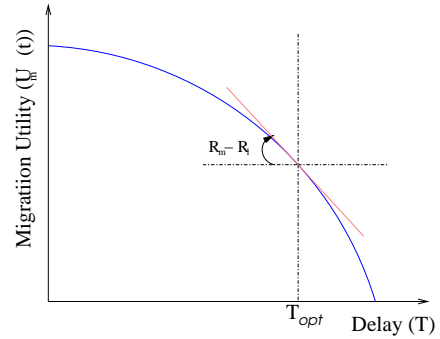


Fig. 5. Migration Utility Function: Optimal Target Deadline Identification

As earlier, we partition the request space into high and low reward requests. Instead of calculating the optimal target deadline rigorously, we present a solution to this optimization problem (with potentially non-convex constraints) and show that the solution is optimal. The following two equations specify the target deadline T_{opt} for migration to complete.

$$\frac{B_m}{T_{opt}} = C_m \quad (6)$$

$$\left| \frac{\delta U_m(t)}{\delta T} \right|_{T_{opt}} = C_m (R_m - R_l), \quad (7)$$

where R_l denotes the average reward of requests that have reward less than R_m . In other words, R_l denotes the average reward of requests in the region A_l in Figure 4.

We have the following result about the deadline T_{opt} thus computed.

Theorem 1: If $U_m(t)$ is a convex function w.r.t time, T_{opt} is the target deadline that maximizes the total expected reward (sum of I/O rewards and migration utility) for an optimal admission control algorithm.

Proof (sketch): We provide an intuitive proof that explains our basic idea. Assume that there is a schedule \mathcal{T} with deadline T' that has total utility larger than any schedule with deadline T_{opt} .

If $T' > T_{opt}$, then \mathcal{T} serves more I/O requests (and fewer migration requests) in time T_{opt} . However, such requests are expected to have reward per unit capacity less than R_m . On the other hand, the loss in migration utility (per unit capacity) by extending migration deadline by unit time is larger than $R_m - R_l$ (due to convexity of $U_m(t)$). Moreover, extending migration leads to rejection of some I/O requests after time T_{opt} . The loss in I/O reward per unit time due to this migration equals R_l . Also, note that if we serve k migration requests in any time, the expected I/O reward loss is larger than kR_m . Hence, the expected increase in total utility by serving an extra I/O request in time T_{opt} is less than the loss in utility by either delaying migration by 1 time unit any time after T_{opt} or serving an extra migration request by rejecting more I/O requests.

Similarly, if $T' < T_{opt}$, it is easy to see that some I/O requests rejected by \mathcal{T} have expected reward greater than or equal to R_m . Hence, serving one of such requests by delaying migration leads to a loss in migration utility (per unit capacity) less than $R_m - R_l$ (due to convexity of U_m). Moreover, the expected I/O reward loss due to such migration after T_{opt} is R_l . Hence, again the expected I/O reward loss by completing migration earlier than T_{opt} is more than the sum of the increase in migration utility and expected additional I/O rewards gained. ■

The marginal loss in utility ($|\delta U_M(t)/\delta t|$) is typically an increasing function and hence T_{opt} is unique (it can be shown that $(R_M - R_l)C_M$ is a decreasing function and hence there is only one intersection point). Once the target deadline is identified, the rest of the procedure is the same as the case with deadline. We associate potential (and actual) reward values for each of the remaining migration requests. Finally, Eqn. 4, 6, 7 are solved using bisection[4]. Since all curves are piecewise linear, we can find a solution quickly.

C. Admission Control Methodology

We now give a concrete formulation of the general reward maximization problem that the Admission Control System solves. Consider an input set of n requests, where each request r_i can be represented as $r_i = \langle a_i, s_i, R_i, D_i, c_i, \Gamma_i \rangle$, where a_i is arrival time, s_i is duration, R_i is reward value, D_i is the deadline, c_i is capacity used (=1 for all requests) and Γ_i is the stream of the request. Note that such requests are both I/O as well as migration requests. Define C as the total capacity of the resource available and T_{tot} as the total time under consideration. In the case of migration with deadline, we can assume this to equal the length of migration or migration deadline for our study. However, in the general setting, an admission controller would be used even if no migration is in progress and T_{tot} would be some suitably defined long interval. The problem is to find a schedule $(x_{i,t})$ of requests such that the overall rewards are maxi-

mized over this period. Formally, we have

$$\begin{aligned} \max \quad & \sum_{i=1}^n R_i \sum_{t=1}^{T_{tot}} x_{i,t} \\ \text{s.t.} \quad & \sum_{i=1}^n c_i p_{i,t} \leq C \quad \forall \text{ time } t; \\ & \sum_{t=1}^{T_{tot}} x_{i,t} \leq 1 \quad \forall \text{ request } i; \\ & x_{i,t} = \begin{cases} 1 & \text{if } r_i \text{ is scheduled at time } t \\ & \text{and } (t + s_i - a_i) < D_i \\ 0 & \text{otherwise} \end{cases} \\ & p_{i,t} = \begin{cases} 1 & \forall t : (t \in (\tau, \tau + s_i - 1) \\ & \text{and } x_{i\tau} = 1) \\ 0 & \text{otherwise} \end{cases} \end{aligned} \quad (8)$$

$\sum_{t=1}^{T_{tot}} x_{i,t} = 0$ implies that the request is rejected. We note that this problem can be modeled as bandwidth allocation problem which is known to be NP-Hard even in a generalized off-line setting [1]. Moreover, the problem needs to be solved in an online setting where the decision of rejecting a request is made without knowledge of the requests which are scheduled to arrive later.

Verma et al. [3] present a provably *optimal* offline algorithm (SRJF) for reward maximization in the scenario where all requests have the same reward and then extend it for the general case to an algorithm (BSRJF) that is *locally optimal*. They also provide online versions of the algorithms. Our QoS aware migration strategy requires us to use any admission control algorithm geared towards reward maximization. Towards this end, we refine the BSRJF algorithm [3] to perform admission control in our setting.

Definition 1: Define the conflict set of request r_i at time t (C_i^t) to be the set of all such requests r_j that have been not been rejected till time t and either (a) $a_i + s_i > a_j$ and $a_i + s_i < a_j + s_j$ or (b) $a_j + s_j > a_i$ and $a_i + s_i > a_j + s_j$. The offline version of BSRJF essentially rejects all such requests r_i for which a C_i^t exists. It finds out all the candidate C_i^t for each r_i and pre-reserves capacity for them. If spare capacity is left after pre-reservation, then r_i is serviced. This leads us to the local optimality condition. In the online version of BSRJF, we compute the expected sum of reward and penalties for such C_i^t candidates, i.e., we compute the sum of the expected rewards and penalties of non-conflicting request sets r_S that arrive later. We compare this sum with the reward of the current request under consideration. If the sum of the expected rewards and the penalties saved exceeds the reward of the current request, we reserve capacity for r_S . This ensures that r_i is serviced only if there is no expected C_i^t that would be rejected later. For further details of the algorithm, we refer the reader to [3].

The BSRJF algorithm can be directly applied for our reward maximization setting with a small change. One may note that BSRJF pre-reserves capacity for requests with high reward potential before accepting a request for service. This pre-reservation needs information about the requests that are expected to arrive in future. [3] use a predictor to generate a short-term forecast of the requests in

order to pre-reserve capacity. In our problem, estimation of expected I/O requests is made using a similar time-series based prediction proposed in [2], [26], [20]. However, the migration requests and their rewards are generated deterministically. To take an example, at the onset of migration, the number of migration requests equal the total capacity of the disk. However, as migration requests get serviced, the number of migration requests that arrive at any given time reduce. Moreover, the rewards of such migration requests may change as the deadline approaches. In order to attribute for this fact, we modify the BSRJF Algorithm to differentiate between I/O and Migration requests. The algorithm computes the expected number of migration requests pending in the following manner. At any time T , it maintains the status of migration (expected number of migration requests completed till time T) and determines the number of pending migration requests and their rewards. Also, in order to estimate the number of migration requests completed at a later time T_j , it assumes that the number of migration requests served between T and T_j equal the average rate of migration needed, i.e., it assumes that at each time $T_k : T < T_k < T_j$, the number of migration requests that would get served equals the long term average rate of migration needed. Hence, in our migration-aware version of BSRJF, future requests are a combination of I/O requests predicted by the predictor and migration requests computed in the manner described above. The rest of the algorithm is same as in [3].

IV. IMPLEMENTATION ISSUES

We now discuss issues we have faced till now in implementing the QoSMig architecture.

A. Disk Capacity Estimation

The admission controller assumes that the disk has a capacity C that denotes the number of I/Os that can be processed by the disk in unit time. However, as opposed to network bandwidth problems, disk capacity is estimated using the knee point of the disk, which depends on the sequentiality of the request streams. Random access streams reduce the knee point whereas sequential streams increase it. To start with, we identify the knee point for the storage device for a traffic that is a mix of the client traffic and migration at an average rate by increasing the client traffic intensity. We denote this knee point for the storage device as the capacity C of the device and use it as a starting estimate. As migration progresses, we use a control-theoretic feedback loop to verify if the used knee point is correct. The disk capacity C used for admission control is then adjusted according to the feedback received.

B. Discrete Class scenario

The computation of migration reward R_m (Eqn. 4) assumes that the reward distribution is continuous (Fig. 4). However, in many practical scenarios, the reward of a request may equal the priority of the QoS class, and hence,

only a few reward values may exist. As a result, the reward assigned to migration requests R_m may be same as reward of requests of some I/O QoS class³ Q_i and the AC may need to decide between I/O and migration requests that have the same reward.

To obviate this problem, we partition the class Q_i into two sub-classes Q_i^1 and Q_i^2 where the reward of class Q_i^1 is $R_M - \epsilon$ and reward of class Q_i^2 is $R_M + \epsilon$. A request of class Q_i is put into class Q_i^1 with probability $P_i^1 = \frac{C_M - (C - \sum_{j=1}^i c_j)}{c_i}$ and with probability $1 - P_i^1$ into class Q_i^2 , where c_j is the capacity used by requests of class Q_j . i.e., the class is partitioned at the point $C - C_M$ into two classes, one of which has higher reward than migration (falls in area A_h) and another that has lower rewards (falls in area A_l). Hence, the AC does not need to handle the scenario where migration and I/O requests have the same reward. Moreover, it is easy to see that the optimal solution for this modified problem (with one extra class) coincides with the original problem. Hence, all our results about the obtained solution hold after this sub-partitioning as well.

C. Prediction Error Adaptation

In real systems, we need to make our system robust by dealing with prediction errors. For the scenario where migration has a specified deadline, we introduce a factor of $slack(t)^{1-c}$ with the migration rewards for the migration scenario with deadline, where $slack$ is defined as

$$slack(t) = \frac{N_M^t}{T_{max} - t} \quad (9)$$

N_M^t is the number of migration requests remaining at time t and c is the confidence ratio. If the long term distribution values are accurate, $c \rightarrow 1$ and the adaptation factor tends to 1. On the other hand, if the confidence is low, $c \rightarrow 0$ and the algorithm tries to adapt quickly to the current traffic, thus ensuring that the rate of migration is varied quickly and migration completes close to deadline. In the general migration utility case, where no deadlines are specified, the deadlines we calculate are not sacrosanct since the computed deadlines are based on the long term behavior of predicted traffic. Since such predictions may have errors, we periodically recompute the long term traffic parameters and, if they have changed, recompute the target deadline and migration rewards as well, thus making it more robust. Note that for long migration tasks the long term averages may change over time and it is necessary to recompute the migration deadline and rewards independent of any forecasting errors.

V. EXPERIMENTAL EVALUATION OF QoSMIG

In this section, we compare the performance of an adaptive rate-controlled migration approach using QoSMig with

³Classes are sorted in order of reward from high to low.

two commonly used schemes i.e. whole-store and average-rate migration, as well as the recently proposed Aqueeduct[6] approach. Note that, none of the existing schemes use a unified admission control framework that enables rate-controlled migration based on the characteristics of client I/O and migration constraints. We use synthetic workloads and real storage traces to demonstrate the effectiveness of an unified admission control framework in Qos-Mig, such that promised migration deadline guarantees are met while minimally impacting client I/Os.

For the sake of clarity, we next provide a brief description of the competitive algorithms for bulk data migration.

- **Whole-store:** Migration is assigned the highest priority and the total available disk capacity is allocated to migration until it is complete. Note that, this is typically done when the storage or information system is incapable of providing concurrent access to applications during migration. In this situation, it becomes desirable to complete the migration as soon as possible.
- **Average-rate:** An average target rate of migration is computed using the required migration bandwidth and deadline. The rate of migration is then set to this average for the entire duration of the migration. One may note that this approach is a simplified version of our general method and useful only in a scenario where the I/O traffic is steady.
- **Aqueeduct:** In Aqueeduct [6], a control-theoretic approach is used to adapt the rate of migration to minimize the impact on client performance. In this approach, migration is a best effort activity and hence, it is only allocated the capacity remaining after serving all I/O requests.

A. Experimental Setup

Disk Type	Seagate Cheetah 4LP
Cylinders	6582
Sector Size	512 Bytes
Disk Size	4.55 GB
Average Seek	7.7 ms
Rotation Speed	10033 RPM
Transfer Rate (Min/Max)	11.3/16.8 MB/sec
Size of Migration Request	512 KB

TABLE I
CONFIGURATION PARAMETERS

For the performance evaluation, we built a simulation environment that implements the three essential components of the QoS Mig architecture, i.e. the leaky bucket traffic shaper, reward tagger, and admission controller. The simulated system consists of two disks, each of which is modelled using DiskSim [9]. To work in a realistic setting, we use the disk model of Seagate Cheetah 4LP in DiskSim that has been validated against this disk for a variety of workloads. The disk parameters used in the experiments are

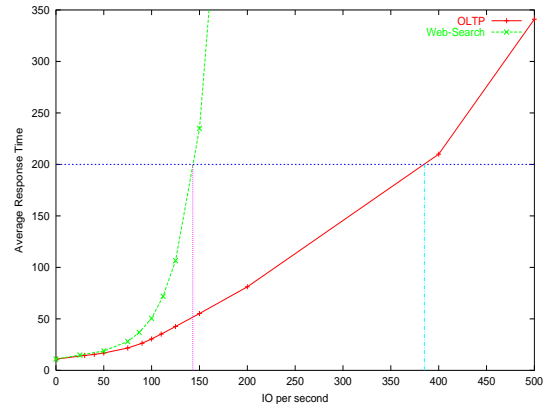


Fig. 6. Estimated knee points for the OLTP and Web search workloads.

summarized in Table I.

Each experiment takes as input a set of client I/O requests and a migration task with associated constraints. Data is migrated from the source disk to the destination disk while the I/O requests are directed only at the source disk. The results reported in this section are for the source disk only. We consider a differentiated QoS setting in which client I/Os belong to three service classes. For ease of understanding, we refer to these classes as gold, silver and bronze. Requests in each service class have a latency requirement of 200ms while the reward associated with requests from the three classes are 20, 10 and 1 units, respectively. We report our results in terms of number of requests serviced from each class and the reward assignment to the requests from the three service classes only highlights the QoS differentiation aspect of our approach. Note that, in this reward model, a request that is dropped by the admission controller has zero reward. In the later subsections, we elaborate on the distributions of the individual QoS classes for the different workloads.

For the simulation study, we interleave the client I/Os with migration requests. The idea is to fragment the migration task into smaller migration requests. We choose the size of each request to be comparable to the size of a single disk track for efficiency [13]. In specific, the size of each migration request is chosen to be 512 KBytes based on the Seagate disk specifications.

1) *Admission Control Methodology:* The I/O requests and the migration requests are submitted to the admission controller which decides on the set of requests to be admitted. Any request not selected by the admission controller are assumed to be dropped. Note that, in a practical setting, the buffer manager in the storage layer typically re-submits the dropped requests, but we have not incorporated this feature in our simulation environment. The admission controller methodology is modified based on the migration strategy being implemented. For the whole-store strategy, the BSRJF algorithm is used with all migration requests accorded a reward higher than the reward of the highest priority I/O request. The average rate migration strategy

is achieved by allocating a proportional fraction of disk bandwidth to migration requests based on the migration deadline. The best-effort migration strategy of Aqueduct is implemented by assigning migration requests a reward lower than the reward of the lowest priority I/O request. Finally, QoS Mig is implemented by using the Migration-Aware BSRJF algorithm in the admission controller.

B. Synthetic Workload

In the first set of experiments, we use synthetic workload for generating client I/Os. The input request stream has Poisson arrivals with heavy-tailed service times [2]. Requests are classified into three QoS classes with 20%, 50%, and 30% requests coming from gold, silver and bronze classes, respectively. The client requests are distributed uniformly and at random across the disk. The disk capacity C was empirically measured to be 125 requests/second for this workload. We experimented with a number of settings for I/O and migration loads. Due to lack of space, we report the case where average I/O load is $0.8C$. Figure 7(a) shows the I/O capacity requested by the client application.

We now study the behavior of different migration strategies for a scenario where a migration deadline needs to be computed based on the rewards generated by the system for two different disk configurations. The reward distributions for an initial configuration and a final configuration (obtained by data migration) are shown in Figure 7(b). In a practical setting, the scenario pertains to the case where such migration is required to handle an anticipated increase in traffic intensity (via replication). The final configuration is capable of handling additional requests and the total reward in this configuration increases as the expected traffic intensity increases.

QoS Mig calculates the optimal deadline, using Eqn. 6, to be 820s. Figures 7(c) and 7(d) show the disk capacity used for migration by Aqueduct and QoS Mig, respectively. Aqueduct treats migration as a best effort activity and hence it is allocated only the capacity remaining after serving all client I/Os. Consequently, Aqueduct misses the migration deadline. On the other hand, QoS Mig suitably adapts the rate of data migration depending on the I/O load as well as the migration deadline to be met. At this point, we would like to mention that, the general set of results showed that while under low load conditions (total load $\leq 0.75C$) both Aqueduct and QoS Mig met the deadline, with increasing load conditions Aqueduct failed to meet the migration constraints, rendering it unsuitable for today's time-critical storage systems.

Moreover, we compare the performance of QoS Mig with whole store and average-rate based migration, in terms of the reward (I/O + migration) loss incurred by the corresponding schemes. We summarize our findings in figures 7(e) and 7(f). Figure 7(e) reports the average reward loss at different times of the experiment, while Figure 7(b) shows the cumulative reward loss for the three schemes. Whole

store finishes migration before the deadline speculated by QoS Mig, albeit at the cost of additional I/O reward loss. For the average case, we fix a deadline of 1200s (one greater than one computed by QoS Mig). Interestingly, we note that such an approach incurs least I/O reward loss initially (due to an extended deadline) but is outperformed once QoS Mig completes migration. Hence, it is clear the deadline computed by QoS Mig is indeed the optimal, since finishing migration earlier (whole-store) or later (average-rate) results in significantly higher reward loss.

In the next set of results, we only focus on the more common scenario where a deadline is specified. Since Aqueduct is incapable of meeting deadlines, we compare QoS Mig only with the whole-store and average rate migration strategies. The performance metric we are interested in is the total number of client I/Os that are dropped by the admission controller for the three request classes using the different approaches. In addition to the total number of requests dropped, we analyze our results based on the requests dropped from each of the individual QoS classes.

C. Real Workload

For this study, we used storage traces [23] generated by (i) a Web search engine and (ii) a database running an OLTP workload for a financial service, made available by Storage Performance Council as representative storage workloads for the above applications. The OLTP and Web search traces are collected over a span of 12 hours and 4 hours, respectively. For each of the traces, we use the application identification number of each request to assign it into one of the three QoS classes. The average I/O load of the raw OLTP and Web search traces was found to be 360 requests/second and 600 requests/second, respectively. In the first step, we use the raw trace along with the average migration rate required to meet the deadline, in order to pre-compute appropriate knee points for the two workloads. The disk capacity C is estimated empirically by increasing the I/O load while fixing the migration load at the average rate, until we reach the knee point, i.e. the total load at which the average response time of the disk increases sharply. In our experiments, we treat the point at which the average response time becomes greater than 200ms to be the knee point. We computed the knee points for OLTP and Web search workloads to be 140 and 390 requests/second respectively, as shown in figure 6. The knee points give us the disk capacities C that are used for these experiments.

After having fixed the disk capacities for each of the workloads, we throttle the input traffic such that maximum I/O load is no more than the corresponding disk capacity. In an implementation of QoS Mig architecture, this functionality is provided by the traffic shaper. In figure 8, we show representative samples of the shaped traffic for the OLTP and Web search workloads, respectively. For the OLTP workload, 30%, 50%, and 20% requests arrive from the gold, silver and bronze QoS classes respectively. For

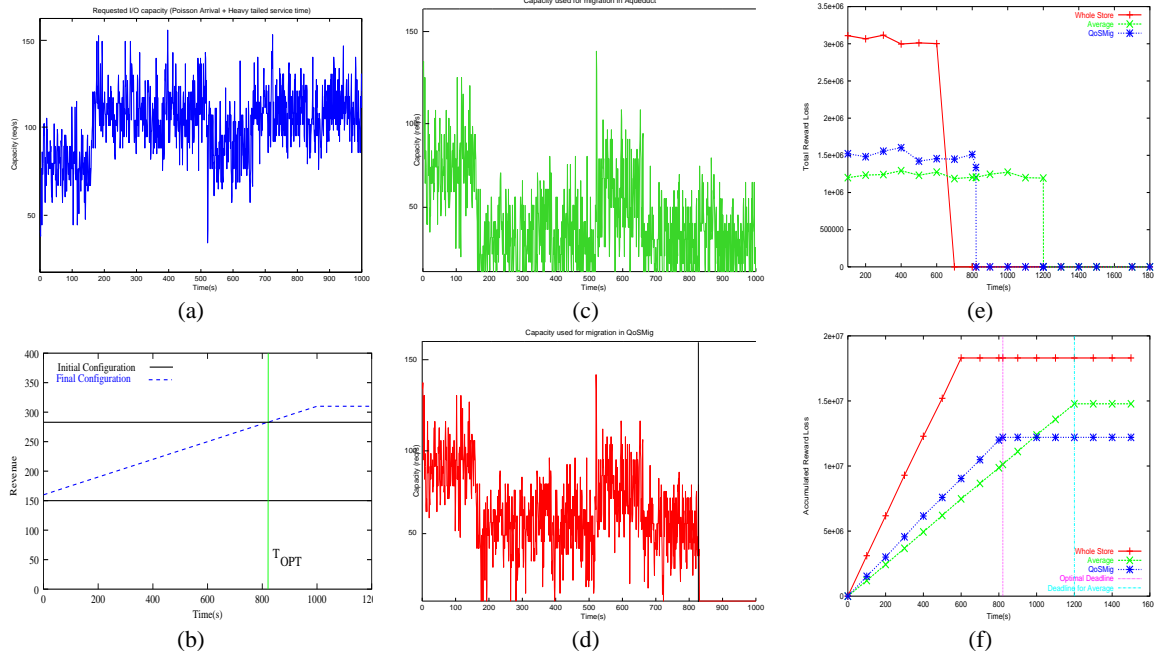


Fig. 7. Performance of different migration strategies for a synthetic I/O workload with Poisson arrivals and heavy-tailed service times. The optimal migration deadline is computed by QoSMig using a reward distribution and leads to significant reduction in reward loss.

the Web search workload, the corresponding percentages are 30%, 40%, and 30%, respectively.

For the next set of results, we keep the client I/O load fixed and increased the migration load as a factor of the disk capacity. Note that, an increase in the migration load captures the effect of shortening the migration deadline. We want to evaluate the performance of the three strategies with varying migration deadlines for the same migration task. Figures 9(a) and 9(b) show the relative performances of the three migration techniques for the OLTP and Web Search workloads, respectively. We make two key observations: (i) QoSMig significantly outperforms whole-store and average-rate in terms of fewer number of client I/Os that are dropped; (ii) both QoSMig and average-rate strive to meet the migration deadline by dropping as many requests from the lowest QoS class and as few requests from the higher QoS classes. However, QoSMig does a much better job of accommodating requests from the silver class when compared to average-rate based migration, and never drops any requests from the gold QoS class. In fact, the improvements are more pronounced for the Web Search workload, where the *only* requests dropped by QoSMig are from the bronze class. We attribute this phenomenon to the more bursty nature of client I/Os for the Web search workload when compared to OLTP. In other words, an adaptive rate-controlled migration strategy can lead to fewer request drops in the storage system.

The next set of results shed light on the performance of the different migration strategies with increase in overall traffic intensity. To this end, we increase the total load while keeping the ratio of I/O to migration load fixed. Fig-

ures 10(a) and 10(b) show the relative performances of the three schemes for the OLTP and Web Search workloads, respectively. Once again QoSMig significantly outperforms whole-store and average-rate in terms of total number of client I/Os dropped. Further, QoSMig achieves improved differentiated QoS by dropping requests *almost exclusively* from the lowest QoS class.

Finally, we verify that the Superior performance of QoSMig, in terms of meeting migration requirements and achieving differentiated QoS, is achieved at the expense of disk utilization. We find that (Figure 11) QoSMig actually achieves better disk utilization than the average-rate migration. In fact, QoSMig was able to achieve a disk utilization upto 8% higher than average-rate migration.

In summary, our experimental results demonstrate the benefits of using an adaptive rate-controlled scheme for bulk data migration. QoSMig not only meets the migration constraints (in this case, migration deadline), but also outperforms strategies like average-rate migration by reducing the impact on application I/Os, and providing better QoS guarantees.

VI. RELATED WORK

Several researchers have studied database and file system reorganization and its relationship with storage I/O performance. However, most of the work has focused on demonstrating either the benefits of reorganization on disk subsystem performance [5], or computing optimum conditions for reorganization so as to minimize the number of I/O requests [16], [19]. Robinson *et al.* [14] have studied reorganization in Log structured File Systems and have concluded that its

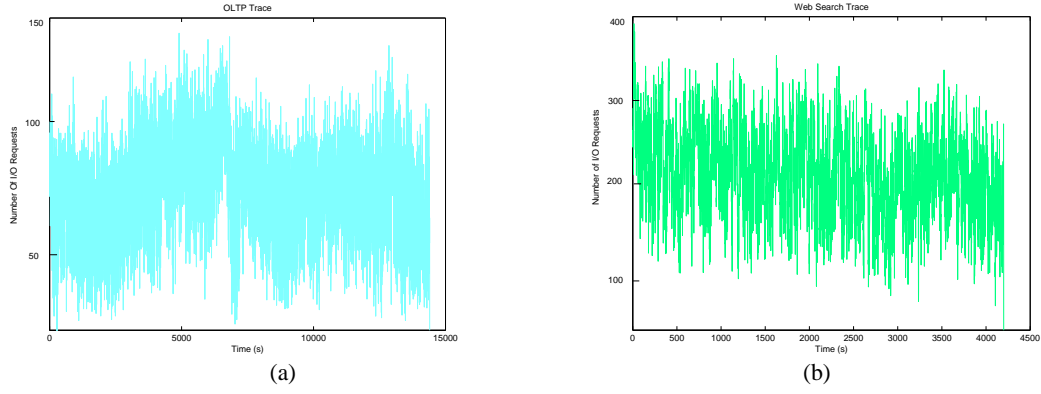
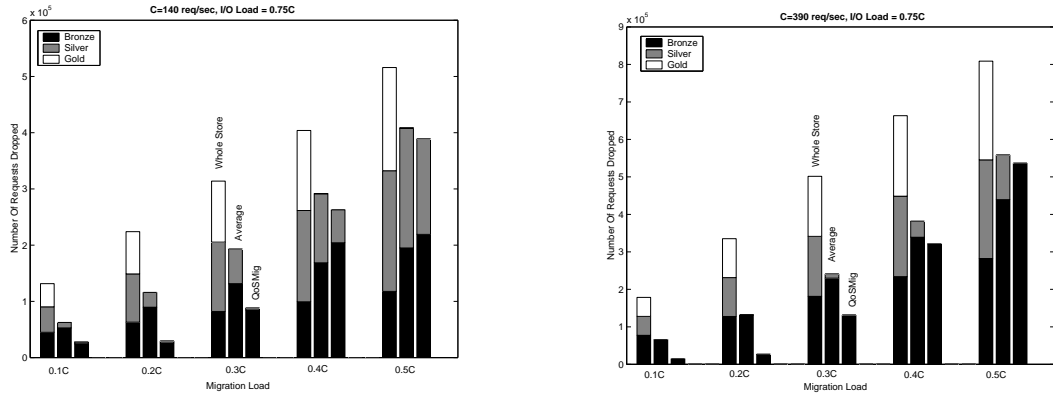
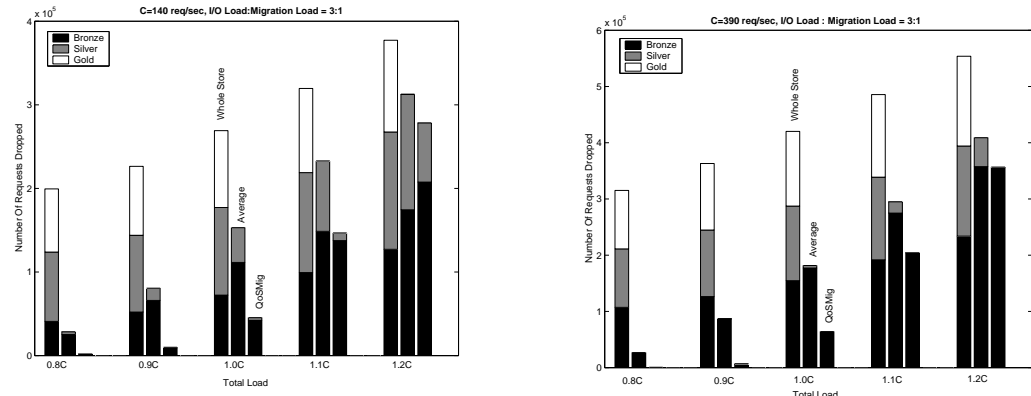


Fig. 8. Shaped traffic distributions for (a) OLTP workload collected over a period of 4 hours, and (b) Web Search workload over a period of 1 hour.



(a) OLTP workload. Total Number of Requests = 2×10^6 (b) Web Search workload. Total Number of Requests = 2×10^6

Fig. 9. Performance comparison of whole-store, average-rate, and QoS Mig based approaches with fixed I/O load and increasing migration load.



(a) OLTP workload. Total Number of Requests = 2×10^6 (b) Web Search workload. Total Number of Requests = 2×10^6

Fig. 10. Performance comparison of whole-store, average-rate, and QoS Mig based migration approaches with increasing I/O and migration load.

overhead on disk subsystem can have significant impact on client performance. However, they do not consider the benefits of reorganization in their study and hence do not provide a scheme for evaluating the trade-offs.

Chambliss *et al.* [8] have proposed a storage systems controller that provides statistical performance guarantees. However, they have not addressed the issue of managing data migration in their work. Lu *et al.* have proposed

Aqueduct [6], a control-theoretic approach to guarantee statistical bounds on impact of data migration on client performance. Aqueduct continuously monitors the client workloads and adapts the migration workload to consume only resources left unused. While this approach performs much better than non-adaptive methods as far as the impact on client performance is concerned, it always considers migration as a low-priority task that is executed in a best-effort

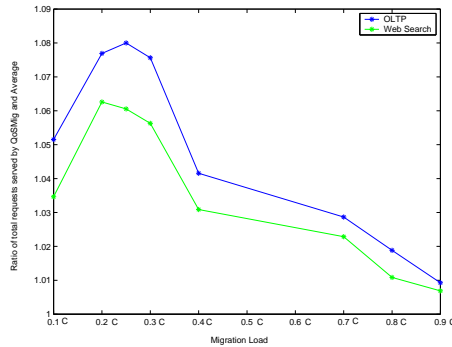


Fig. 11. Improvement in disk throughput using QoS Mig for OLTP and Search traces.

manner with no guaranteed deadline. In loaded systems this could lead to unpredictably long migration duration compromising the benefits of migration.

The reward maximization of service providers has been addressed by several researchers. Most of these address allocation of resources for various request classes with QoS guarantees so that resources are optimally utilized, thereby maximizing the profit of the providers. Among these, Liu *et al.* [27] proposed a multi-class queueing network model for the resource allocation problem in an offline setting. The allocation of servers as discrete resources in a server farm has also been studied by Jayram *et al.* [25], keeping in mind the practical constraint that these need a finite time to be reallocated. Verma *et al.* [3] address the problem of admission control for profit maximization of networked service providers. Marbach *et al.* [22] frame the downlink resource allocation and pricing for wireless networks as a revenue maximization problem. A key idea is to use the profit maximization framework to solve diverse problems (e.g. throughput maximization or differentiated QoS) instead of just maximizing the actual revenue earned. Similarly, our work uses reward maximization to solve the problem of data migration.

VII. CONCLUSION AND FUTURE WORK

We have presented the *QoS Mig* rate-controlled migration methodology that achieves the optimal tradeoff between migration objectives and client performance. The proposed methodology derives its strength from using, on one hand, the long term traffic averages to decide the migration deadline and its relative priority (reward), relative to application requests and, on the other hand, using the short-term traffic forecast to take advantages of burstiness in traffic. *QoS Mig* is especially suited for bursty traffic in conditions where an optimal average migration rate is a significant fraction of the disk throughput (20 – 30% is ideal (Fig. 11)).

An interesting direction for future research is to enhance *QoS Mig* for the scenario where migration utility starts accruing as migration progresses. There are instances where migration may be a very long task and partial migration accrues some benefits, a scenario we have not addressed in this work, and is worth further investigation.

REFERENCES

- [1] A. Bar-Noy, R. Bar-Yehuda, A. Freund, J. (S.) Naor, and B. Schieber. A unified approach to approximating resource allocation and scheduling. In *Proc. ACM Symp. on Theory of Computing*, 2000.
- [2] A.K. Iyengar, M.S. Squillante, and L. Zhang. Analysis and characterization of large-scale Web Server Access Patterns and Performance. In *Proc. Conf. on World Wide Web*, 1999.
- [3] A. Verma, and S. Ghosal. Admission Control for Profit Maximization of Networked Service Providers. In *Proc. International World Wide Web Conference*, 2003.
- [4] Bisection tutorial. At <http://mathworld.wolfram.com/Bisection.html>.
- [5] C.L. Chee, H. Lu, and C.V. Ramamoorthy. Adaptive Prefetching and Storage Reorganization. In *A Log-Structured Storage System*. In *IEEE Trans. on Know. and Data Eng.*, Vol. 10, No. 5, 1998.
- [6] C. Lu, G.A. Alvarez, and J. Wilkes. Aqueduct: Online Data Migration with Performance Guarantees. In *Proc. of USENIX FAST*, 2002.
- [7] C. R. Lumb, A. Merchant, G. A. Alvarez. Facade: virtual storage device with performance guarantees. In *Proc. of USENIX FAST* 2003.
- [8] 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 *Proc. of 22nd Intl. Symp. on Reliable Distributed Systems*, 2003.
- [9] DiskSim Simulation Environment, at www.pdl.cmu.edu/DiskSim/.
- [10] E. Anderson, J. Hall, J.D. Hartline, M. Hobbs, A.R. Karlin, J. Saia, R. Swaminathan, and J. Wilkes. An Experimental Study of Data Migration Algorithms. In *Proc. Workshop on Algorithmic Engineering*, 2001.
- [11] E. Shriver, A. Merchant, and J. Wilkes. An analytic model for disk drives with readahead caches and request reordering. In *Proc. of ACM SIGMETRICS* 1998.
- [12] J. Menon, D. A. Pease, R. Rees, L. Duvanovich, and B. Hillsberg. IBM Storage Tank- A Heterogeneous Scalable SAN File System. In *IBM Systems Journal*, 42(2):250-267, 2003.
- [13] J. Schindler, J. L. Griffin, C.R. Lumb, and G. Ganger. Track-aligned extents: matching access patterns to disk drive characteristics. In *Conference on File and Storage Technologies (FAST)*, 2002.
- [14] J.T. Robinson, and P.A. Franaszek. Analysis of Reorganization Overhead in Log-Structured File Systems. In *Proc. ICDE*, 1994.
- [15] K. Keeton, C. Santos, D. Beyer, J. S. Chase, and J. Wilkes. Designing for Disasters. In *Proc. FAST'04 Conference on File and Storage Technologies*, 2004.
- [16] K. Maruyama and S.E. Smith. Optimal reorganization of distributed space disk files. In *Comm of the ACM*, 1976, 19(11):634-642.
- [17] M.E. Crovella, M.S. Taqqu, and Azer Bestavros. Heavy-tailed probability distributions in the World Wide Web. In *A Practical Guide To Heavy Tails*, pp. 3-26. Chapman & Hall, 1998.
- [18] M.K. Lakhmaraju, R. Rastogi, S. Seshadri, and S. Sudarshan. Online Reorganization in Object Databases. In *Proc. of ACM SIGMOD International Conference on Management of Data*. 2000.
- [19] M.L. Lee, M. Kitsuregawa, B.C. Ooi, K. Tan, and A. Mondal. Towards self-tuning data placement in parallel database systems. In *Proc. ACM SIGMOD*, 2000.
- [20] M.S. Squillante, D.D. Yao, and L. Zhang. Web traffic Modeling and web server performance analysis. In *Proc. IEEE Conf. on Decision and Control*, 1999.
- [21] M. Uysal, G. A. Alvarez, and A. Merchant. A Modular, Analytical Throughput Model for Modern Disk Arrays. In *MASCOTS*, 2001.
- [22] P. Marbach, and R. Berry. Downlink Resource Allocation and Pricing for Wireless Networks. In *Proc. IEEE Infocom*, 2002.
- [23] Storage Trace Repository, at <http://traces.cs.umass.edu/storage/>.
- [24] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. Introduction to Algorithms, 1990.
- [25] T.S. Jayram, T. Kimbrel, R. Krauthgamer, B. Schieber, and M. Sviridenko. Online Server Allocation in a Server Farm via Benefit Task Systems. In *Proc. ACM Symp. on Theory of Computing*, 2001.
- [26] X. Chen, P. Mohapatra, and H. Chen. An Admission Control Scheme for Predictable Server Response Time for Web Accesses. In *Proc. Conf. on World Wide Web*, 2001.
- [27] Z. Liu, M.S. Squillante, and J.L. Wolf. On Maximizing Service-Level Agreement Profits. In *Proc. ACM Conf. on Electronic Commerce*, Orlando, FL, 2001.