RZ 3887 (#ZUR1411-045) 03/31/2015 (revised August 2015)
Computer Science 17 pages

Research Report

ExaPlan: Queueing-Based Data Placement and Provisioning for Large Tiered Storage Systems (Revised Version)

Ilias Iliadis, Jens Jelitto, Yusik Kim, Slavisa Sarafijanovic, Vinodh Venkatesan

IBM Research – Zurich
8803 Rüschlikon
Switzerland

Research
Africa • Almaden • Austin • Australia • Brazil • China • Haifa • India • Ireland • Tokyo • Watson • Zurich

ExaPlan: Queueing-Based Data Placement and Provisioning for Large Tiered Storage Systems

Ilias Iliadis, Jens Jelitto, Yusik Kim, Slavisa Sarafijanovic, Vinodh Venkatesan
IBM Research
{ili, jje}@zurich.ibm.com, yusik.kim@nl.ibm.com, {sla,ven}@zurich.ibm.com

***Abstract*—Multi-tiered storage, where each tier comprises one type of storage device, e.g., SSD, HDD, is a commonly used approach to achieve both high performance and cost efficiency in large-scale systems that need to store data with vastly different access characteristics. By aligning the access characteristics of the data to the characteristics of the storage devices, higher performance can be achieved for any given cost. This article presents ExaPlan, a method to determine both the data-to-tier assignment and the number of devices in each tier that minimize the system's mean response time for a given budget and workload. In contrast to other methods that constrain or minimize the system load, ExaPlan directly minimizes the system's mean response time estimated by a queueing model. Minimizing the mean response time is typically intractable as the resulting optimization problem is both non-convex and combinatorial in nature. ExaPlan circumvents this intractability by introducing a parameterized data-placement approach that makes it a highly scalable method that can be easily applied to exascale systems. Through experiments that use parameters from real-world storage systems, such as CERN and LOFAR, it is demonstrated that ExaPlan provides solutions that yield lower mean response times than previous works. It is also capable of determining a data-to-tier assignment both at the level of files and at the level of fixed-size extents. For some of the workloads evaluated, file-level placement exhibited a significant performance improvement over extent-level placement.**

I. Introduction

Today's storage systems are challenged by the ever increasing demand for data storage and retrieval. Their complexity has grown in terms of both the heterogeneity and the number of storage devices required to satisfy user requirements. Multi-tiered storage systems are composed of storage devices with different characteristics in terms of capacity, access time, and sustained throughput. Fast storage devices, such as solid state drives (SSDs) and hard disk drives (HDDs), are more expensive than slow devices, such as magnetic tape drives. On the other hand, slow devices tend to provide higher amounts of storage capacities for a given cost. Consequently, for cost reasons, not all of the data can be stored in fast devices. Clearly, the overall performance of multi-tiered systems can be optimized by storing the least accessed data (cold data) on the slower and cheaper devices, and the most frequently accessed data (hot data) on the faster, albeit more expensive devices [1]. At a scale as large as the storage system planned for the Square Kilometre Array (SKA) [2] radio telescope, where an estimated 1 PB per day will need to be stored, an access-pattern-aware multi-tiered storage system has the potential to dramatically reduce cost while providing high performance compared with using only a single type of device, e.g., hard disk drives. Building and operating a data center with multiple tiers requires assessing the number of devices in each tier (tier *dimensioning* or storage *provisioning*) as well as the way data is placed across the tiers (data *placement*). Data placement with a good match between the access patterns and the device characteristics together with an appropriately chosen device mix for the workload is key to achieving efficiency.

In this article, we present ExaPlan, a queueing-based data placement and provisioning method for large-scaled tiered storage systems. ExaPlan is envisioned to support system administrators and planners by suggesting, for a given budget and expected I/O workload, the number of devices of each type and the data placement that minimizes the system's mean response time.

Most existing approaches addressing data placement and/or provisioning seek to minimize the total system cost while meeting volume and load constraints [3-7]. The load constraint is usually described by enforcing that the number of devices be large enough to collectively provide the service capacity needed for handling the given workload. The load as a measure of congestion can be considered a performance metric as it affects the queueing time. However, using load as the performance metric has limitations as it provides no estimates or guarantees about the actual time to serve user requests. Each of the references cited above has a different way of enforcing the equivalent of "load does not exceed one" in queueing terminology. This allows solutions with the load being arbitrarily close to one, which in turn implies an arbitrarily large mean queueing time for the I/O requests.

We address these limitations by directly minimizing the mean system response time subject to volume, load, and system cost constraints. To the best of our knowledge, the data placement and tier dimensioning issues addressing response time optimization under a cost constraint have not yet been studied. Our work is the first to analyze the multi-tier dimensioning and data placement issue in this context.

A straightforward formulation of the optimization problem with the system's mean response time as the objective function is non-convex and combinatorial; non-convex because of the form of the mean response time function and combinatorial because of the numerous data placement options. This problem is much harder to solve than other mathematical programming approaches that do not account for the queueing effects, such as [7], where minimizing the system cost subject to volume and load constraints results in a mixed integer linear program.

The main contribution of our work is the transformation of this seemingly intractable storage provisioning and data placement problem into a tractable one. We achieve this by

exploiting the property that, for a given data placement, the problem of finding an optimal number of devices in each tier, with the corresponding mean response time, under the constraints considered is convex, for which a closed-form solution can be obtained. To explore the data placement options and determine the one that yields the least mean response time, we use a *black-box optimization* framework. To perform this exploration in a scalable and systematic way, we parameterize data placement and seek an optimal set of parameters through a black-box optimization algorithm. Well-known black-box optimization algorithms include simulated annealing and genetic algorithms. In this case, the "black box" considered takes as input a set of parameters that specifies a data placement option and yields as output the optimal number of devices in each tier and the corresponding mean response time for that data placement.

We use an M/G/1 queueing model for approximating the mean response time of solutions proposed and the *covariance matrix adaptation evolution strategy (CMA-ES)* [8] as the black-box optimization algorithm for finding an optimal data-to-tier placement. We note that ExaPlan is modularized regarding these aspects in the sense that advances in the research fields of stochastic processes and evolutionary algorithms could be integrated into our framework with ease.

In the existing literature, the data units subject to a placement decision are either fixed-size extents [6, 7] or variable-size logical stores (e.g., files) [3-5]. ExaPlan supports data units of both fixed and variable size. This flexibility allowed us to compare the performance of using variable-size units and fixed-size units. For some of the workloads evaluated, file-level placement exhibited a significant performance improvement over extent-level placement. Through experiments that use parameters from real-world storage systems, such as CERN and LOFAR, it is demonstrated that ExaPlan provides solutions that yield lower mean response times than previous works.

The remainder of the paper is organized as follows. Section II provides a survey of the relevant literature on capacity planning and data placement. Section III describes the definitions and notation used in this article and the modeling assumptions. The ExaPlan architecture is described in Section IV. In particular, it presents the classification framework which takes into account device characteristics, namely volume and service capacities, as well as cost. Subsequently, the number of devices for each tier is evaluated analytically such that the mean system response time is minimized under budget constraints. Section V presents experimental results which demonstrate the efficiency of the proposed scheme. Finally, we conclude in Section VI.

II. Related work

A general framework for provisioning a storage system using a utility function that considers various metrics, such as purchase cost, performance, reliability, availability, power, etc., was proposed in [9]. The optimization is performed by using a genetic algorithm that explores the space of candidate system configurations, which is determined by the placement and redundancy of up to 50 datasets over a set of nodes. In contrast, we focus on the performance objective expressed by the system's mean response time by considering a cost budget and other constraints. In particular, we propose a scalable method for systematically exploring the huge space of candidate solutions with up to 1 billion chunks placed over different types of devices.

The data placement issue was considered in [10] and the corresponding data allocation problem was formulated as a knapsack problem, where a total benefit value is to be maximized while respecting volume constraints. This problem was subsequently solved using a greedy algorithm. In [1], the data allocation problem was formulated as a multiple choice knapsack problem which is NP-complete, and therefore solving it in the context of real systems that contain millions of files is infeasible. Instead, an optimal solution was found using a simple heuristic that treats the multiple-tier problem as multiple single-tier problems that can then optimally be solved using dynamic programming. These approaches do not directly consider the queueing of I/O requests and its impact on the request response time.

The use of a queueing model with the objective to minimize the response time by optimally assigning files to a fixed set of disks was considered in [11]. The I/O subsystem, including the CPU, channels, controllers, heads of strings, and disk actuators, was modeled as a memoryless queueing network, from which the optimal proportion of requests assigned to each disk that minimizes the response time can be determined. Then a heuristic is used to assign files to disks so that the optimal proportion found can be realized as closely as possible. Compared to ExaPlan, this is a micro-scale model that implicitly assumes that requests are of equal size and small, and that therefore the response time is dominated by the time a request spends contending for various system resources as it traverses the I/O subsystem. Finding an optimal proportion of requests from the queueing network model is an iterative process whose complexity depends on the size of the network, and therefore it is not expected to scale to the large-scale systems considered by ExaPlan.

The Extent-based Dynamic Tiering (EDT) tool [6] has a configuration adviser (EDT-CA) that uses a fixed utility function to determine a priori the placement of its extents, fixed-size partitions of the volume space, to the various tiers. In EDT-CA, once a placement decision has been made, the number of devices is chosen so that it satisfies the offered load. Similar to the utility function of EDT-CA, ExaPlan also uses a resource cost function to assign data to different tiers. The difference is that we consider a parameterized family of resource cost functions that gives more control over data placement, and then seek an optimal resource cost function within that family. Of the existing works, HybridStore [7] is the method that is the most comparable to ExaPlan with respect to context (tiered storage) and approach (mathematical programming). We provide a detailed overview of the HybridStore approach along with comparison results in Section V.

Other notable approaches for data placement and storage provisioning that minimize cost subject to volume and load constraints start with finding an initial feasible configuration and use iterative heuristics to move to a better solution [3-5].

Many works have considered the problem of grouping files that are likely to be accessed together, so that they can be placed next to each other to improve the access latency

and reduce power consumption (see [12], [13] and references therein). These works are orthogonal to the issues addressed in this article and can be used in conjunction with ExaPlan to improve system performance even further. Similarly, data prefetching has also been widely studied (see [14] and references therein). These methods can be used to prefetch data that is likely to be accessed in the near future to faster tiers. In this article, it is assumed that data is read directly from the tier in which it is stored.

III. Workload and System Model

The notation used herein is detailed in Table I. A workload is represented by a set of tuples

$$\mathcal{W} = \{(Q_j, r_j, v_j) \mid j = 1, \ldots, N\} \,. \tag{1}$$

Each element of $\mathcal{W}$ represents a unit of data and its access characteristics. We refer to this unit of data as a *chunk*, indexed by j, whose volume is denoted by v_j. Requests for data residing within chunk j are assumed to arrive at a rate of r_j. The sizes of requests to chunk j are assumed to be independent and identically distributed, represented by the random variable Q_j. As related sequential requests do not incur more than one seek time on a storage device, we represent these requests by one large request. The degree of sequentiality of requests to a given chunk j is therefore reflected in its mean request size $\mathbf{E}[Q_j]$.

In this article, we only consider those requests that are not cached and have to be directly served by the devices. This is to factor out the effects of caching, which is beyond the scope of this work, when evaluating the performance of the underlying mass storage system. Note also that correlated request arrivals to the same chunk are most likely absorbed by cache at some level of the storage hierarchy. Consequently, the request arrivals that eventually reach the storage devices are assumed independent. Also, in large-scale systems with multiple users, it is likely that the combined workload of all users is a Poisson process. [15]. Consequently, the arrivals are also assumed to be Poisson.

In many systems, for example webservers, the I/O is dominated by read requests. For systems with significant write I/O, it is likely that a dedicated write buffer is used before writing it out when the system is under low I/O load. Therefore, this study exclusively focuses on read I/O, by assuming that write I/O either is negligible handled separately, or that read I/O has priority over write I/O.

We define the *workload space* as the three-dimensional space of (q, r, v) where q is the mean request size, r is the request rate, and v is the chunk size. A storage *tier*, indexed by k, is defined as a collection of devices with the same characteristics in terms of seek time, bandwidth, volume capacity, and price. This definition is not to be confused with alternate definitions of the term "tier" used in the context of "hierarchy", as there is no inherent hierarchy in this definition.

The device model considers that a request of size q to a device in tier k incurs a fixed seek time of s_k and a transfer time of q/b_k, such that the total time to serve it is equal to $s_k + q/b_k$. Device cache, if any, is assumed to be accounted for by the seek time and bandwidth specifications of each tier. Note also that the analysis presented can be extended in a straightforward manner to account for variable seek times. If chunk j with attributes (Q_j, r_j, v_j) were to be assigned to tier k, its corresponding *contributed load* to this tier, $\rho_{j,k}$, which indicates the proportion of time a device of tier k would be busy serving requests to chunk j, is given by the quantity $r_j(s_k + \mathbf{E}[Q_j]/b_k)$. Evidently, a chunk cannot be assigned to a device of a tier in which the contributed load exceeds one, as otherwise the mean queueing time would be infinite. Hence, we define the *stability region*, Ψ_k, in the (q, r, v) workload space for each tier k as

$$\Psi_k = \{(q, r, v) : r(s_k + q/b_k) < 1\} \,. \tag{2}$$

Note that, as shown in Figure 1, the stability regions for the various tiers correspond to the areas under the hyperbolas defined in the (q, r) plane by $r(s_k + q/b_k) = 1$.

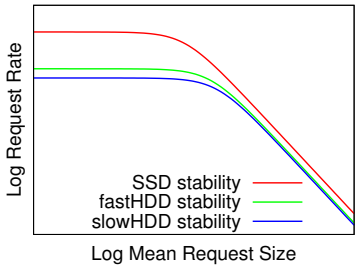


Fig. 1. The stability region of each tier is the area underneath the corresponding curve. A tier may only host chunks that are within its stability region; otherwise the violating chunk will single-handedly saturate the service capacity of the host device within that tier.

TABLE I. Notation

Tier k	$k = 1, \ldots, K$
s_k	device seek time
b_k	device bandwidth
C_k	device capacity
V_k	device cost
γ_k	cost per unit data volume ($= V_k/C_k$)
z_k	number of devices
I_k	index set of chunks assigned
λ_k	request rate ($= \sum_{j \in I_k} r_j$)
$1/\mu_k$	mean service time of requests
σ_k^2	variance of the service time
ρ_k	load ($= \lambda_k/(\mu_k z_k)$)
W_k	waiting time random variable (r.v.)
S_k	service time r.v.
T_k	response time r.v. ($= W_k + S_k$)
α	$= (\alpha_1, \ldots, \alpha_K)$
β	$= (\beta_1, \ldots, \beta_K)$
Chunk j	$j = 1, \ldots, N$
r_j	arrival request rate
Q_j	request size r.v.
v_j	chunk volume
$\rho_{j,k}$	contributed load ($= r_j(s_k + q_j/b_k)$)
Systemwide	
λ	system request rate ($= \sum_k \lambda_k$)
T	system response time r.v. ($= \sum_k \lambda_k T_k/\lambda$)
B	budget

IV. ExaPlan architecture

The overall process of finding the optimal number of devices in each tier and the data placement that minimizes the mean response time is illustrated in Figure 2. It is a black-box optimization framework wherein the black box takes as input a set of parameters that specifies a data placement option and yields as output the optimal number of devices in each tier and the corresponding mean response time for that data placement.

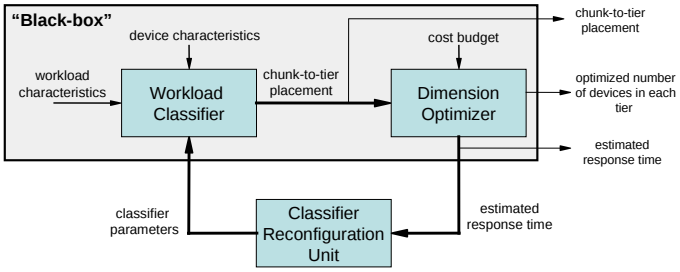


Fig. 2. ExaPlan architecture. Optimization inputs: workload characteristics, device characteristics, cost budget. Outputs: optimized number of devices in each tier, chunk-to-tier placement, estimated response time.

The black box consists of the 'Workload Classifier' and the 'Dimension Optimizer' modules. The Workload Classifier specifies a chunk-to-tier assignment (data placement) based on a set of input parameters by using a parameterized family of resource cost functions that take into account the chunk access and device characteristics. The chunk-to-tier assignment is fed to the Dimension Optimizer module, where an optimal number of devices for each tier along with the resulting mean response time are assessed subject to a cost constraint. The resulting mean response time is then reported to the 'Classifier Reconfiguration Unit', where the next set of classifier parameters is determined according to the selection rule of the black-box optimization algorithm used. This iterative process continues until a specified stop condition is met. We detail the individual components of ExaPlan in this section.

A. Workload Classifier

The role of the Workload Classifier is to provide a chunk-to-tier assignment to the Dimension Optimizer using a family of parameterized resource cost functions $c_k(q, r, v)$, $k = 1, \ldots, K$, referred to as *cost functions* hereafter, whose parameters are provided by the Classifier Reconfiguration Unit. In particular, by denoting the set of parameters provided for tier k by α_k and β_k, a specific choice for the resource cost $c_k(q, r, v)$ incurred by placing a chunk with mean request size q, request rate r, and volume v on tier k is given by

$$c_k(q, r, v) = \beta_k[(1 - \alpha_k)v/C_k + \alpha_k r(s_k + q/b_k)] \ . \quad (3)$$

A chunk with attributes (q, r, v) is assigned to the tier in which it incurs the least resource cost, that is, the tier determined by

$$\arg \min_k c_k(q, r, v) \ . \quad (4)$$

Note that the aim of the cost function is to parameterize the various data placement options and explore them in a scalable and systematic manner regardless of the actual costs of the devices involved. Therefore, the resource cost function does not explicitly consider the device costs V_k. These costs are taken into consideration by the Dimension Optimizer when estimating the optimal number of devices for a given chunk-to-tier assignment (see Section IV-B).

The rationale for considering such a family of cost functions parameterized by (α_k, β_k) is as follows. A single storage device has a volume resource and a service resource. For hard disk drives, the volume resource corresponds to the available space on the disk platter and the service resource corresponds to the read-write head. The resource cost associated with assigning a chunk to a tier must therefore have separate components reflecting the consumption of these respective resources. When considering placing a chunk with workload (q, r, v) on tier k, the volume component of the resource cost incurred should reflect the volume it consumes on the host device (v/C_k). The service component of the resource cost should reflect the proportion of time the device (head) is busy serving requests to that chunk $(r(s_k + q/b_k))$. Observe that these two components can respectively be interpreted as the volume and service utilizations that take values between 0 and 1. The resource cost of placing a chunk to tier k is defined to be proportional to the weighted average of these two components, where the parameter α_k $(0 \leq \alpha_k \leq 1)$ defines the weights. For a given tier k, if the emphasis is placed on the volume component relative to the service component (i.e. small α_k), it may result in a placement that overloads the tier with service requests. As seen from Figure 3, the value of the optimal response time at $\alpha_1* = 0.965$, $\alpha_2* = 0.595$ is $T* = 0.012452$, which is about two orders of magnitude lower than the response time at $\alpha_1 = 0$, $\alpha_2 = 0$. Thus, the α_k parameters provide a mechanism to adjust the balance between volume and service utilization, thereby allowing us to find configurations with better response times. As we eventually intend to assign chunks to the tier with the lowest cost, another natural parameter for the resource cost function is β_k $(\beta_k \geq 0)$, which reflects a relative preference or bias between tiers. The α_k and β_k parameters are adjusted by the black-box optimization algorithm according to the workload and the available budget. By the shorthand notations α and β, we denote the collections $(\alpha_1, \ldots, \alpha_K)$ and $(\beta_1, \ldots, \beta_K)$, respectively, to account for all K tiers.

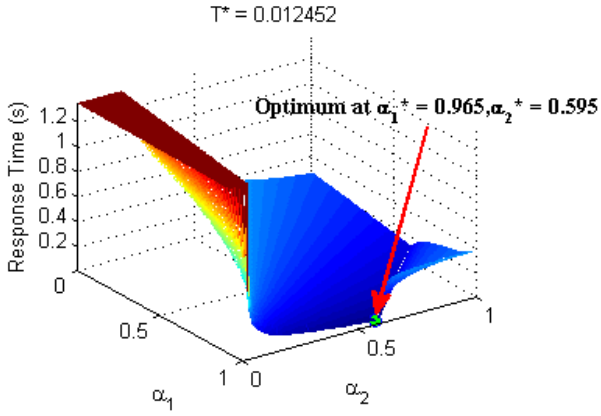


Fig. 3. Impact of the α classifier parameters in the case of a small-scale two-tier system: the mean response time as function of the classifier parameters α_1 and α_2, for a fixed β_1 and β_2.

Note that, by assigning each chunk to the tier in which it incurs the least resource cost, we effectively partition the (q, r, v) space by boundaries determined by the equations

$$c_k(q, r, v) = c_{k'}(q, r, v) \quad (k \neq k') \ . \quad (5)$$

All chunks that fall into a certain partition are assigned to the

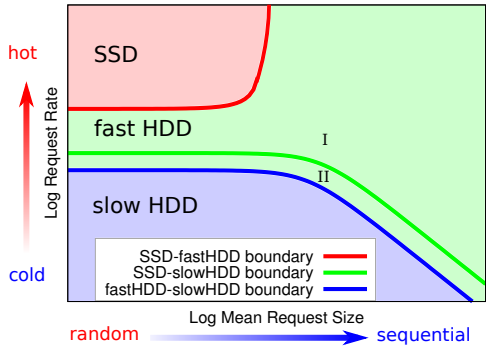


Fig. 4. The partition of the workload space for given boundary parameters α and β when the chunk size v is fixed. Depending on which region (red, green, blue) of the workload space a chunk falls into, it is assigned to the corresponding device (SSD, fast HDD, slow HDD). In the absence of a fast HDD tier, sub-regions I and II of the green region would be assigned to SSD and slow HDD, respectively.

same tier. Thus, the chunk-to-tier assignment can be controlled by modifying these boundaries using different values for the α and β parameters. The parameterization of the data placement allows us to circumvent the computational challenges typically associated with formulating assignment problems using binary decision variables.

An example of how a cross section of the workload space at fixed chunk size v is partitioned using cost functions with given parameters α, β is illustrated in Figure 4. Each point of the workload space is assigned to the tier that yields the lowest cost function value. The resulting regions that are assigned to SSD, fast HDD, and slow HDD are shaded red, green, and blue, respectively. The two sub-regions of the fast HDD (green) partition labeled I and II, respectively, represent the regions that would be assigned to SSD and to slow HDD in the absence of a fast HDD tier. In this example, frequent small (random) requests on the upper-left tend to favor SSD. The eventual placement of chunks must honor the stability regions depicted in Figure 1. Therefore, a chunk is eventually placed on the tier that yields the lowest cost function value among those tiers that satisfy the stability condition (2).

B. Dimension Optimizer

The Dimension Optimizer module of Figure 2 serves as a module for evaluating a chunk-to-tier assignment specified by the α and β parameters. The evaluation of a placement decision involves an internal optimization step, which determines the number of devices for each tier that minimizes the mean response time. In this section, we present the assumptions of the queueing model and present the details of this internal optimization step.

We model read operations by representing each device in the system as a single server queue. Once chunks have been assigned to a tier, a good load-balancing algorithm, e.g., LPT [16], is used to distribute the chunks across the constituent devices of a tier. Therefore, for a large number of chunks, the load is roughly equally distributed across the devices in each tier. Consequently, the corresponding queues in each tier are assumed statistically equivalent, having the same arrival process and the same service time distribution. We therefore model each of the z_k devices within a tier as an independent $M/G/1$ queue. Proposition 1 gives the system's mean response time, $\mathbf{E}[T]$, for a given chunk-to-tier assignment and for a given number of devices in each tier.

Proposition 1: Assume that request arrivals to chunk j are a Poisson process with rate r_j. For a given workload $\mathcal{W} = \{(Q_j, r_j, v_j) \mid j = 1, \ldots, N\}$, device characteristics (s_k, b_k), number of devices for each tier z_k, and a chunk-to-tier placement I_k for all tiers k, the mean response time is approximated by

$$\mathbf{E}[T] \approx \frac{1}{\lambda} \sum_k \left[\frac{(\lambda_k/\mu_k)^2}{z_k - \lambda_k/\mu_k} \cdot \frac{1 + \sigma_k^2 \mu_k^2}{2} + \frac{\lambda_k}{\mu_k} \right] , \quad (6)$$

where

$$\lambda_k = \sum_{j \in I_k} r_j \ , \quad (7)$$

$$\frac{1}{\mu_k} = \sum_{j \in I_k} \left(s_k + \frac{\mathbf{E}[Q_j]}{b_k} \right) \frac{r_j}{\lambda_k} \ , \quad (8)$$

$$\sigma_k^2 = \frac{1}{b_k^2} \left[\sum_{j \in I_k} \mathbf{E}[Q_j]^2 \cdot \frac{r_j}{\lambda_k} - \left(\sum_{j \in I_k} \mathbf{E}[Q_j] \cdot \frac{r_j}{\lambda_k} \right)^2 \right] + \sum_{j \in I_k} \frac{1}{b_k^2} \mathrm{Var}(Q_j) \frac{r_j}{\lambda_k} \ , \quad (9)$$

provided that $z_k > \lambda_k/\mu_k$ for all k. Otherwise, $\mathbf{E}[T] = \infty$.

Proof: See Appendix A.

Note that the $M/G/1$ assumption can be generalized to a $G/G/1$ model using Kingman's formula [17] provided that it is possible to precompute the squared coefficient of variation of the request inter-arrival times for each tier. For each given chunk-to-tier assignment, the arrival process to a tier is a superposition of independent arrival processes to chunks of that tier. For superpositioned general arrival processes, it is fairly straightforward to compute the mean inter-arrival time, but computing the variance of the inter-arrival times of the merged process from the individual inter-arrival times is challenging. There has been work on using renewal approximations of superpositioned general arrival processes to estimate the mean and the variance of the inter-arrival times [18], but using this requires an additional renewal approximation step for each tier of a given chunk assignment and can be time consuming. Nevertheless, we emphasize that this problem can be decoupled from ours, and any advances in this area can be directly applied to our framework.

The Dimension Optimizer minimizes the system-wide mean response time (6) for a given chunk-to-tier assignment, which is specified through the sets I_k of assigned chunk indices for tiers $k = 1, \ldots, K$. Note that for a fixed chunk-to-tier assignment, the mean service time portion of the mean response time of requests is also fixed and does not depend on the number of devices chosen for each tier. Therefore, it suffices to minimize the mean queueing time portion in order to minimize the mean response time given by (6). Indeed, we see that after omitting terms that do not affect the optimal

solution from expression (6), we get the mean queueing time:

$$\sum_k \frac{(\lambda_k/\mu_k)^2}{z_k - \lambda_k/\mu_k} \cdot \frac{1+\sigma_k^2\mu_k^2}{2\,\lambda} \,. \tag{10}$$

Note, however, that this does not imply that ExaPlan provides the same solution if instead of the mean response time, the mean queueing time were minimized. As will be discussed in Section IV-C, this is because at each iteration the Classifier Reconfiguration Unit chooses the α, β parameters, and hence the chunk-to-tier assignment, based on the *mean response time* criterion, and not the *mean queueing time* criterion.

Denoting the coefficient $(1 + \sigma_k^2\mu_k^2)/(2\,\lambda)$ by a_k, we formulate the response-time minimization problem as follows:

$$(P1) \quad \min_{z_k} \quad \sum_k \frac{(\lambda_k/\mu_k)^2 a_k}{z_k - \lambda_k/\mu_k} \tag{11}$$

$$\text{subject to} \quad \sum_k z_k\, V_k \le B \tag{12}$$

$$z_k C_k \ge \sum_{j\in I_k} v_j \quad k = 1,\ldots,K \tag{13}$$

$$z_k \ge \frac{\lambda_k}{\mu_k} \quad k = 1,\ldots,K \,. \tag{14}$$

Constraint (12) is the budget constraint, (13) is the volume constraint, and (14) is the load constraint. As the chunk-to-tier assignments I_k are determined, all coefficients $\lambda_k, \mu_k, \sigma_k^2$ of (10) can be precomputed by using (7), (8) and (9). Note that problem (P1) above has a convex objective function and linear constraints.

Proposition 2: For problem (P1), let A be the set of tier indices for which constraint (13) is binding at an optimal solution. Then the optimal number of devices corresponding to set A is given by

$$z_k^* = \begin{cases} \sum\limits_{j\in I_k} v_j/C_k \,, & \text{if } k \in A \\ \left\lceil \dfrac{\left(B - \sum\limits_{i\in A} H_i - \sum\limits_{i\notin A} \frac{\lambda_i}{\mu_i} C_i\gamma_i\right)\sqrt{a_k}}{\sqrt{C_k\gamma_k}\sum\limits_{i\notin A}\frac{\lambda_i}{\mu_i}\sqrt{a_i C_i \gamma_i}} + 1 \right\rceil \frac{\lambda_k}{\mu_k} \,, & \text{if } k \notin A \end{cases} \tag{15}$$

with

$$H_k \triangleq \sum_{j\in I_k} v_j\gamma_k \,. \tag{16}$$

Proof: See Appendix B.

For a given chunk-to-tier assignment, it is not known a priori which of the volume constraints are binding at an optimal solution, nor is it known whether the problem is feasible. Fortunately in practice, the number of tiers is typically a small number (less than 10). Therefore, it is tractable to enumerate all possibilities for set A and evaluate the corresponding z_k values for each case. The next step is to test if the derived z_k values satisfy the KKT conditions (detailed in Appendix B). As the problem is convex, the KKT conditions are both necessary and sufficient for optimality. Therefore, any set A that satisfies these conditions yields an optimal z_k for the assumed data placement, and the first such set A found is chosen. If there is no set A for which the KKT conditions are satisfied, the problem is infeasible for the given chunk-to-tier assignment.

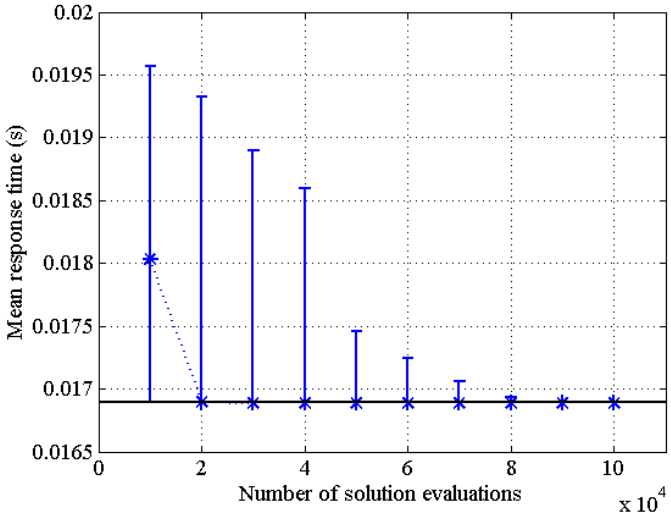


Fig. 5. ExaPlan convergence.

C. Classifier Reconfiguration Unit

The Classifier Reconfiguration Unit chooses the next set of classification parameters α, β to evaluate based on the results of the previous iteration. Functionally, this corresponds to the neighbor selection rule of the black-box optimization algorithm used.

We have implemented and tested several evolutionary and hill-climb algorithms, including simulated annealing, genetic algorithms, gradient descent, and coordinate descent, and concluded that a specific evolutionary algorithm called the *covariance matrix adaptation evolution strategy* (CMA-ES) [8] works best. To gain insight into the classifier parameter optimization problem, we discretized the space of the classifier parameters for a small-scale two-tiered system, found the optimal solution at these discrete points using brute-force search, and visualized the resulting mean response time as function of one or multiple of the parameters. For instance, Figure 6 shows the mean response time as a function of the classifier parameter β_2 after fixing β_1 and optimizing over α_1, α_2. Given the many local minima seen in Figure 6, it is not surprising that pure descent algorithms do not perform well. We have observed from many test cases that there are multiple α, β points that are far away from one another and yield similar objective function values, and for this reason, simulated annealing usually gets stuck at a local optimum.

CMA-ES, which can be seen as a "soft" gradient descent, samples and evaluates candidate solutions from a multivariate normal distribution at each iteration, and updates the mean and covariance matrix to favor the sampling region that was successful in the previous iteration. Thus, if there are several local optima placed far apart, it, unlike simulated annealing, tends to maintain a relatively broad sampling region that encompasses many good-quality local minima until the region to focus on becomes clear. In our specific implementation

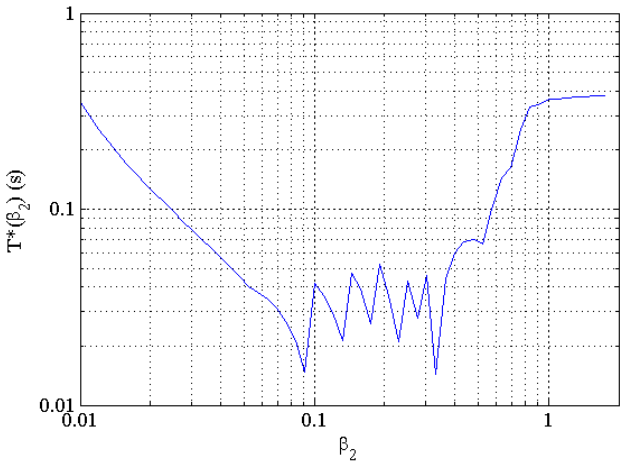


Fig. 6. Impact of the classifier parameters in the case of a small-scale two-tier system: the mean response time as function of the classifier parameter β_2, after optimizing over α_1, α_2 for fixed β_1.

of CMA-ES, we search for an initial feasible (α, β) solution by scaling up both the covariance matrix and the sample size until one is found. Once an initial feasible solution is identified, a standard CMA-ES iteration begins. If there is no improvement of the objective function for a predefined number of consecutive iterations, it jumps to the best solution found so far and starts over with an enlarged sampling region. The total number of objective function evaluations including the search for the initial feasible solution is limited by a predefined parameter that represents the computational budget allowed.

As is the case for most randomized algorithms, there are no known performance guarantees for the CMA-ES algorithm for general objective functions. In fact, there is no guarantee of convergence to a global optimum unless we exhaust the solution space. So the only way to examine the quality of the solution and how fast the solution is found is through empirical results. To demonstrate these points, we used a smaller-scale problem of placing 10000 chunks over two tiers, where it is tractable to find an optimal solution by brute force, and compared it against ExaPlan using 100 different random seeds. It turned out that all seeds succeeded in finding the optimal solution after a median of 21242 objective function evaluations. The best and worst case number of evaluations among the 100 seeds before the optimal solution was found was 1623 and 101435, respectively. Figure 5 shows the median and the 5th and 95th percentile values of the objective function across all 100 seeds as a function of the number of objective function evaluations.

D. Clustering chunks for scalability

The bottleneck of the solution process is the Workload Classifier module, where the cost function values of each chunk need to be evaluated for every tier in each iteration. If the number of chunks exceeds the order of tens of millions, the run time becomes too long for it to be practically useful. To maintain scalability, we group chunks with similar workload characteristics into clusters and solve the optimization problem on the clusters (treated as "super-chunks"). By doing so, we solve a smaller problem at the cost of executing a one-time clustering step. Even relatively simple clustering algorithms, such as k-means, do not scale well with the number of chunks, suffering from the same scalability problem that we try to solve by clustering. We have evaluated the k-means algorithm on small-scale problems against a simple algorithm that grids the projection of the workload space onto the (q, r) plane, and then clusters chunks that fall into the same grid. We observed that k-means has little to no advantage in terms of the end result over the simple gridding algorithm, but took much more time to run. So we chose to use a simple gridding algorithm with log-spaced separation lines in the (q, r) plane, which results in up to 10^4 clusters.

As many chunks are now represented by a single cluster, we proceed to evaluate the request rate, and the mean and variance of the request size that will represent the cluster. The request rate of the cluster is fairly straightforward to define: the sum of the individual request rates of all of its constituent chunks. The request size distribution of the cluster is defined as the mixture of the constituent chunks' request size distributions, with the mixture probabilities proportional to the respective request rates. By denoting the size of a request to cluster C by Q_C, we use the following expressions to evaluate the mean and variance of Q_C from the request rate and size statistics of its constituent chunks:

$$\mathbf{E}[Q_C] = \sum_{j \in C} q_j p_j \ , \tag{17}$$

$$\mathrm{Var}(Q_C) = \sum_{j \in C} q_j^2 p_j - \left(\sum_{j \in C} q_j p_j \right)^2 + \sum_{j \in C} \mathrm{Var}(Q_j) p_j \ , \tag{18}$$

where $p_j = r_j / \sum_{j \in C} r_j$, q_j is equal to the mean $\mathbf{E}[Q_j]$ of the request size of chunk j, and $\mathrm{Var}(Q_j)$ denotes the corresponding variance. $\mathrm{Var}(Q_C)$ is computed using the variance decomposition formula, as detailed in Appendix C.

E. Intended use cases

ExaPlan can be used for planing a new storage system, assuming the expected workload characteristics are known or can be well estimated. ExaPlan can also be applied on top of either an observed or a projected workload in an existing system to suggest upgrading the number of devices in some or all of the tiers, when either the current or the projected performance is not satisfactory with the current system size. Next, when running an existing multi-tiered system with certain (fixed) number of devices in each tier, ExaPlan could be used for periodically and adaptively changing the data placement based on the observed changes in the workload characteristics. In this case, the Dimension Optimizer does not compute an optimal number of devices, but simply uses the existing number of devices.

V. Experiments

We proceed to assess the performance (mean system response time) of ExaPlan for various price points on a set of synthetic workloads generated with parameters from real workloads whenever available. Price points are defined as the cost-per-GB value of the *stored* data, not to be confused with the cost per GB of the total available storage capacity. For each of the workloads considered, we assess the performance

TABLE II. TIER PARAMETERS

Tier	SSD	HDD 10K	HDD 7.2K
Seek Time (ms)	0.5	4.6	8
Bandwidth (MB/s)	350	200	175
Cost ($/GB)	1.5	0.30	0.10
Cost of Device ($)	600	360	400
Capacity (GB/device)	400	1200	4000

of an all-SSD system with a high price point and compare it with the results of the ExaPlan for a variety of price points. For these workloads, we also illustrate the difference between using files and using fixed-size extents as chunks. Finally, we demonstrate the scalability of ExaPlan as the number of chunks increases. We also show how the performance of ExaPlan compares with that of HybridStore [7] for the various price points and workloads considered. Note that HybridStore is an integrated storage management system that includes a capacity planning and a data placement module called HybridPlan. As ExaPlan only addresses the capacity planning and data placement problem in a static setting, we compare it only with the HybridPlan part of HybridStore.

We set a computational budget for the CMA-ES algorithm so that a solution is found in less than a few minutes, which is comparable to how long it takes for HybridStore to output a solution. The experiments were done on a Linux® cluster that comprises mostly Intel® Xeon® CPUs of different generations. We used a centralized load leveler to submit parallel jobs to evaluate the various data points for the comparison. As it is not a controlled environment, but rather a shared one with jobs sharing common resources often running on heterogeneous machines, it would be unfair to directly compare the raw execution times of the methodologies considered. But typically, we observed that ExaPlan running in single-threaded mode took about twice as long to finish as HybridStore.

We assumed having three storage tiers: SSD eMLC (enterprise multi-level cell), HDD 10K rpm, and HDD 7.2K rpm. The device characteristics of each tier are given in Table II. Cost values assumed correspond to the year 2013. To measure the accuracy of the mean response time estimations evaluated analytically, we implemented a tiered storage system simulator using the OMNEST simulation library, and recorded various statistics.

A. HybridStore extensions

For the experiments, it was necessary to extend Hybridstore so that it can be directly compared with ExaPlan. HybridStore seeks the minimum cost yielding tier dimensions and data placement that meet the imposed service capacity and volume capacity constraints by formulating the problem as a mixed integer linear program. To be able to compare the mean response time yielded by HybridStore with that yielded by ExaPlan at different cost-per-GB price points, and to validate the results through simulation, we made three adjustments to HybridStore, and we refer to this extended version as HybridStore+.

First, we introduce an additional parameter which we call a *load factor*. In contrast to HybridStore that outputs the minimum required budget, ExaPlan takes the budget as an input to the problem and seeks to minimize the mean response time, so it is straightforward to evaluate ExaPlan at various price points. To enable HybridStore to also be evaluated at different price points, we (indirectly) control the response time by strengthening the load constraints, and observing the resulting cost. Analogously to the volume utilization parameter, which is used to control the excess volume desired for individual tiers, we use the load factor to constrain the maximum load allowed for the different tiers. Rewriting HybridStore's service capacity constraint [7, Eq. (3)] in terminology using rates instead of aggregate values during a fixed time horizon, we have the following:

$$\sum_j (\text{total load induced by cluster } j \text{ if placed in tier } k) \times (\text{proportion of chunk in cluster } j \text{ placed in tier } k) / (\text{number of devices in tier } k) \leq \text{load factor}\,. \quad (19)$$

The load factor (≤ 1) is used as a lever to indirectly control the response time by using the relationship between load and response time of (25). In many test cases, we observed that constraint (19) with a load factor equal to 1 is binding at the solution given by HybridStore, resulting in an infinite queueing time. We could avoid these cases by forcing a slack in the original load constraint through the use of load factors. We use the same load factor for all tiers for the experiments because HybridStore does not provide a mechanism to optimize the load for each tier. In fact, the ability to appropriately distribute the load across the tiers to optimize the response time is a novelty of ExaPlan that sets it apart from other solutions. We also experimented with using the built-in volume utilization parameters as a lever to adjust price points. However, this still did not prevent HybridStore from choosing a configuration with binding load constraints, which results in infinite mean response times.

The second extension is the specification of individual chunk-to-tier assignments. In the solution provided by HybridStore, only the proportion of chunks in each cluster that are to be placed in each tier is specified. To evaluate the mean response time of the proposed placement using a simulator, we also need to specify precisely to which tier and to which device each chunk is assigned. We do this by randomly assigning chunks of each cluster to tiers according to the optimal proportions provided by HybridStore. We specify the actual device within a tier to which the chunk is assigned using the same load-balancing algorithm (LPT) as in ExaPlan.

The third extension is the number of clusters used compared with what was used in the HybridStore paper. To account for the wider range of request sizes and rates in the workloads considered, we increased the number of clusters from 33 to 3201, as using 33 clusters gave bad performance results for HybridStore. If we use an excessively large number of clusters, there are many clusters that have only one or two chunks, and we observed artifacts from trying to allocate these few chunks to tiers based on the recommended proportion, which ends up degrading the performance.

In contrast to HybridStore, which gives integer solutions for the number of devices for each tier, ExaPlan gives solutions which are then rounded up. Rounding up a fractional solution is expected to have a negligible effect when trying to design a large scale storage system typically having hundreds of devices per tier. We use the CPLEX solver to implement HybridStore's

mixed integer linear program formulation. Thus, we compare ExaPlan to HybridStore+, which is the HybridStore model with the three extensions listed above.

B. File-level and extent-level workloads

Recall from Section III that the workload is represented in terms of requests to chunks. This means that depending on how chunks are defined, the same access pattern may have different workload representations. We are specifically interested in comparing the case of using files as chunks and using fixed-size extents as chunks. The relationship between file chunks and extent chunks is illustrated in Figure 7. When using files as chunks, a traced request can be considered equivalent to what is specified in an I/O system call, e.g., read X bytes starting at Y bytes from the beginning of file Z. This is typically how a user or an application would specify an I/O request, and thus the response time of such a request may be considered an "end-to-end" response time. We therefore refer to requests when chunks are defined as files as *user requests*. In contrast, requests where chunks are defined as extents are referred to as *modeled requests*. If a file spans multiple extents, e.g., file 1 spans extents 1, 2, and 3 in Figure 7, large sequential user requests to the file may be split into smaller modeled requests that are contained within each of those extents. This is due to the fact that extents set up artificial boundaries in the logical block address (LBA) space that ignore file boundaries, which results in user requests that span multiple extents being split at the extent boundaries. Therefore, when comparing one workload representation with another (e.g., file vs. extent), what constitutes a request in each case has a different interpretation, and consequently, the mean response time of the requests cannot be directly compared.

To compare the performance of using file-level chunks with that of using extent-level chunks, we propose a method to translate the mean response time of modeled requests to a quantity comparable to the mean response time of user requests during the same trace interval. To achieve this, the trace of modeled requests is directly derived from the trace of user requests by splitting it at extent boundaries. Having the two traces covering the same time interval, we assume that the sum of the response times of the trace of modeled requests represents the time it takes to finish the total amount of work generated by the user requests during the same period. Denoting the mean response time of user requests by $\mathbf{E}[T_{UR}]$, the mean response time of the modeled requests by $\mathbf{E}[T_{MR}]$, and the number of user requests and modeled requests during the trace period by n_u and n_m, respectively, we propose the following translation formula

$$\mathbf{E}[T_{UR}] = \mathbf{E}[T_{MR}] \cdot n_m/n_u \ . \tag{20}$$

In all workloads we have considered in the experiments, however, there was no noticeable difference between the mean response times before and after this translation while using extent-level chunks. Therefore, we only show and compare the response time values before translation. This, however, may not hold in general, especially for workloads in which extents are much smaller than the average request size.

Note that this way of comparing the response times does not take into account potentially achievable concurrency by placing certain extents on different devices, nor does it take into account the exact timing of when the split requests are needed, as it is possible that not all parts of a big single request are needed at once. These aspects regarding concurrency as well as other commonly used performance improvement techniques, such as file striping, are beyond the scope of this article. Consequently, we simplify the comparison method by assuming that the split requests are requested at once, but are transferred one after the other.

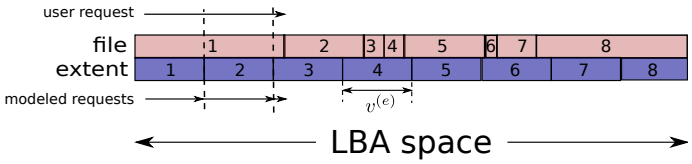


Fig. 7. A file may span multiple extents (file 1), or an extent may span multiple files (extent 4). A user request to file 1 is split into 3 modeled requests at the extent boundaries. Extent size is $v^{(e)}$.

For the experiments, we make the comparison between using file-level chunks and using extent-level chunks, with the number of chunks being the same in both cases. As the execution time of ExaPlan depends primarily on the number of chunks, setting the number of extents equal to the number of files is necessary for a fair comparison. For the CERN workload, which we will describe in further detail in the sections below, we set the size of an extent to be equal to the mean file size of 843 MB. This size is consistent with the extent sizes of 0.5 to 1 GB for IBM® Easy Tier® [19] and EMC® FAST™ VP [20] used in practice.

Whether it is better to use files or extents as chunks for a given workload depends on the tradeoff between the efficiency of having a finer granularity of placement for large files and the inefficiency of grouping together several small files with vastly different access characteristics into a single chunk. When there are isolated "hot" regions of a large file that spans multiple extents, given that space on expensive tiers is scarce, the ability to selectively assign only the hot extents to devices of a fast/expensive tier would be preferable over assigning the complete large file to a device of only one of the tiers. Conversely, when multiple small files with vastly different access characteristics are packed into a single extent, the extent will have hot and cold regions, where the option to assign separate constituent files separately to different devices/tiers would be preferable. As we use the same number of extents as the number of files in our experiments, we will generally have both cases. It is then a matter of whether the large files account for most of the workload, that determines which option yields better performance. Another case is when large files are accessed in a uniform fashion, not creating relatively hot regions. There would not be much incentive in this case to use extents to break up a single large file and to assign them separately. In practice, the number of extents need not be equal to the number of files and the user may decide how many extents to use depending on the computational budget available.

C. Workload generation procedure

Denote the file-level workload by $\mathcal{W}^{(f)} = \{(Q_j^{(f)}, r_j^{(f)}, v_j^{(f)}) \mid \ j = 1, \ldots, N^{(f)}\}$, and the extent-level

workload by $\mathcal{W}^{(e)} = \{(Q_j^{(e)}, r_j^{(e)}, v^{(e)}) \mid j = 1, \ldots, N^{(e)}\}$. We set $N^{(f)} = N^{(e)} = 10^6$ for the experiments and use the following procedure to generate the workload using parameters, whenever available, from real-world systems, such as CERN [21] and LOFAR [22]. For CERN, only the file-size distribution was available, but for LOFAR, also the request size and request rate for each file were also available, in addition to the file-size distribution.

1) Generate $v_j^{(f)}$, $j = 1, \ldots, N^{(f)}$, according to the empirical file-size distribution.
2) Set $Q_j^{(f)} = B_j\, v_j^{(f)}$, where B_j is a beta random variable chosen according to the workload (see Sections V-D and V-E).
3) Sample $r_j^{(f)}$ from a log-uniform distribution within the stability region of the fastest device (see Eq. (2)).
4) Using the $\mathcal{W}^{(f)}$ generated by the preceding steps, generate an LBA-level trace, which consists of tuples of the form (time stamp, start address, request size) as follows:

a) Generate time stamps by sampling interarrival times from an exponential distribution with rate $\sum_{j=1}^{N^{(f)}} r_j^{(f)}$ (generate a Poisson arrival process).
b) Sample file j with probability proportional to $r_j^{(f)}$.
c) Sample a request size from $Q_j^{(f)}$.
d) Generate the start address relative to the beginning block of the file according to a chosen distribution, such as uniform or skewed.

5) Partition the LBA space into extents of size $v^{(e)}$, as shown in Figure 7.
6) Translate the LBA-level trace obtained in step 4 into extent-level trace $\mathcal{W}^{(e)}$ by splitting requests that span multiple extent boundaries and keeping the same time stamps.

The same workload can be characterized in terms of user requests at the end of step 3, and in terms of modeled requests at the end of step 6. We compare the two alternatives, file vs. extent, for all workloads considered. We represent the experiments where we used ExaPlan on the file chunk and the extent chunk scenario by ExaPlan(file) and ExaPlan(extent), respectively. The experiments where we used HybridStore+ in the extent chunk scenario are represented as HybridStore+(extent).

The arrival of requests in the file-level workloads used for the experiments is assumed to be a Poisson process. This assumption was made because of the lack of large-scale realistic workloads (or the lack of methods for scaling up smaller-scale workloads in a realistic manner). Furthermore, as discussed in Section III, in large-scale systems with multiple users, it is likely that the combined workload of all users is Poisson. Consequently, the extent-level workloads, which are obtained by splitting the file-level workloads, are also Poisson. As discussed in Section III, we do not consider the effects of cache. Therefore, the workload is assumed to be directly served by the devices without any caching, which would have otherwise absorbed some of the repeated requests.

D. Workload 1: Synthetic Poisson based on CERN file-size distribution

We examine the mean response time and the number of devices for each tier at nine different cost-per-GB price points and compare the results with HybridStore+. Each price point is determined by varying the load factor for HybridStore+, from 0.1 to 0.9, and examining the optimal solution (minimum system cost) at each of the load factors. The corresponding ExaPlan experiment is conducted by using the minimum system cost provided by HybridStore+ as the budget.

We sample the file sizes of step 1 of the workload generation procedure using the file-size distribution of CERN [21] illustrated in Figure 8. For this case, the mean file size is 843 MB, with the largest file being 2.1 TB. The request rates to each file are sampled from a log-uniform distribution between 10^{-6} and 1/50 of the upper limit of the fastest (i.e. SSD) device's stability region at the given mean request size value for the file. We consider two sub-workloads of this case, representing two extremes of the workload spectrum regarding the distribution of the request size and the request-starting block location.

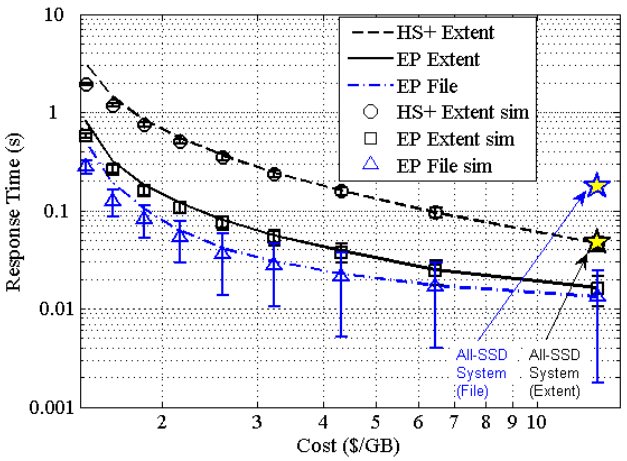
The first case (seq) represents highly sequential workloads where the request size to a file is almost as large as the file itself, with the starting block of the request being uniformly distributed among eligible blocks. This is realized by setting the parameters of the beta distribution (Section V-C, step 2) to $a = 5$ and $b = 1$, where the pdf is proportional to $x^{a-1}(1-x)^{b-1}$. This implies that the average request size to a file is $a/(a+b) = 5/6$ of the file size, thus highly sequential.

The second case (rnd) assumes that request sizes are small relative to the size of the file and that the starting block of the request is skewed towards the beginning part of a file. This is to emulate a type of workload where small random requests are concentrated on isolated "hot" regions of the LBA space. To specify this workload, the parameters of the beta distribution used to describe the request size are set to $a = 1$ and $b = 100$. This implies that the average request size is 1/101 of the file size.

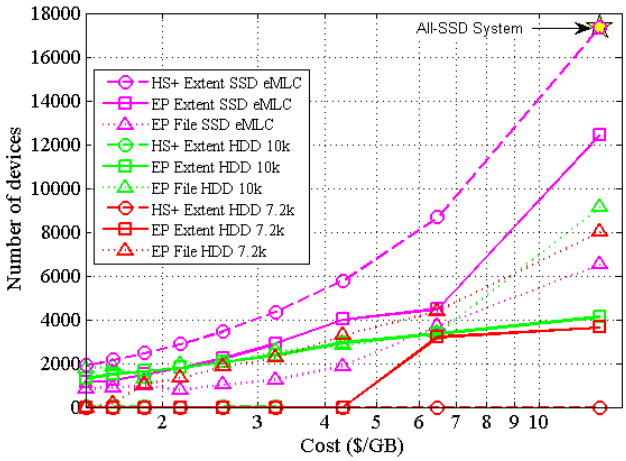
The highest price points in Figures 9(a) and 9(c), are \$12.93/GB and \$9.66/GB, respectively, and are used as the budget for the all-SSD reference system. As can be seen, the mean response times of an all-SSD system with these price points are much higher than the corresponding mean response times obtained by ExaPlan. We also observe that the performance of these all-SSD systems can be achieved at much lower costs by using a mixture of all devices, as suggested by ExaPlan.

From Figures 9(b) and 9(d), we observe that ExaPlan uses fewer SSDs than an all-SSD system (12,430 instead of 17,357 SSDs and 10,925 instead of 12,967 SSDs, for sequential and random workloads, respectively). Owing to the difference in the costs of devices, a number of SSDs can be traded for a higher number of HDDs. It turns out that with ExaPlan many of the chunks with larger request sizes get assigned to a disk tier and the resulting mean response time is decreased compared to an all-SSD system. This is an interesting result that shows that using as many SSDs as the budget allows is not always the best choice from a performance perspective.

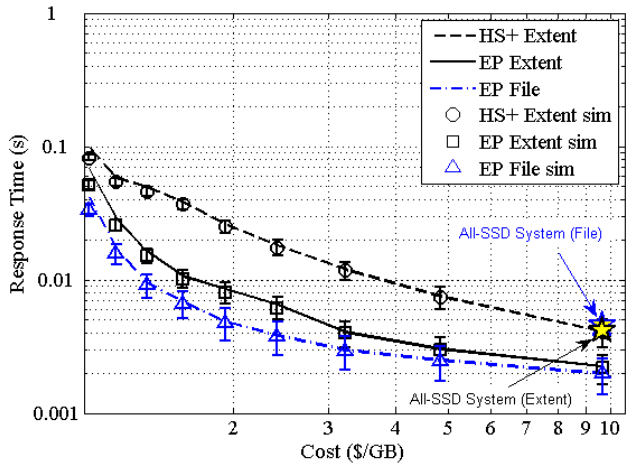
To compare the relative performance between using file-level chunks and using extent-level chunks, we compare ExaPlan(file) with ExaPlan(extent) in Figures 9(a) and 9(c). For both workloads, ExaPlan(file) has significantly smaller mean response time than ExaPlan(extent). We can see that



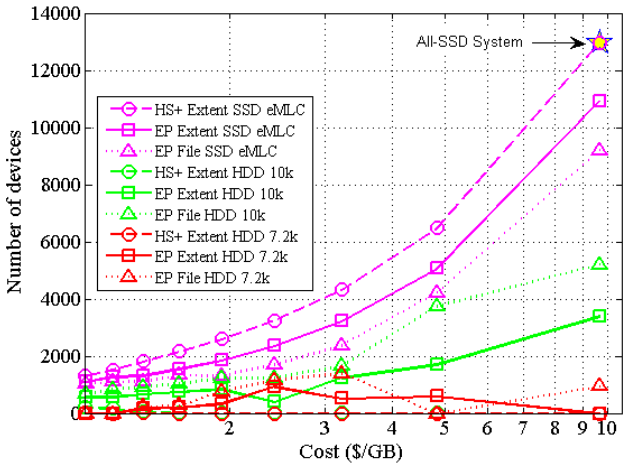
(a) Mean Response Time (seq)



(b) Number of Devices (seq)



(c) Mean Response Time (rnd)



(d) Number of Devices (rnd)

Fig. 9. CERN workload results with sequential (seq) and random (rnd) access patterns. In (a) and (c), lines represent model estimates and markers simulation results at the corresponding price points. The error bars represent the response time standard deviation divided by 30. For each workload, the all-SSD system (with file-level or extent-level data placement) correspond to the highest price point shown in the corresponding figures.

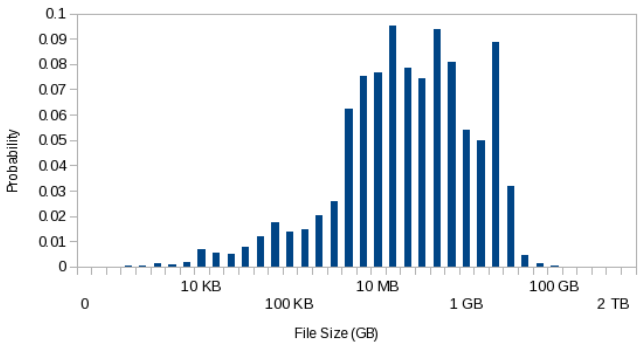
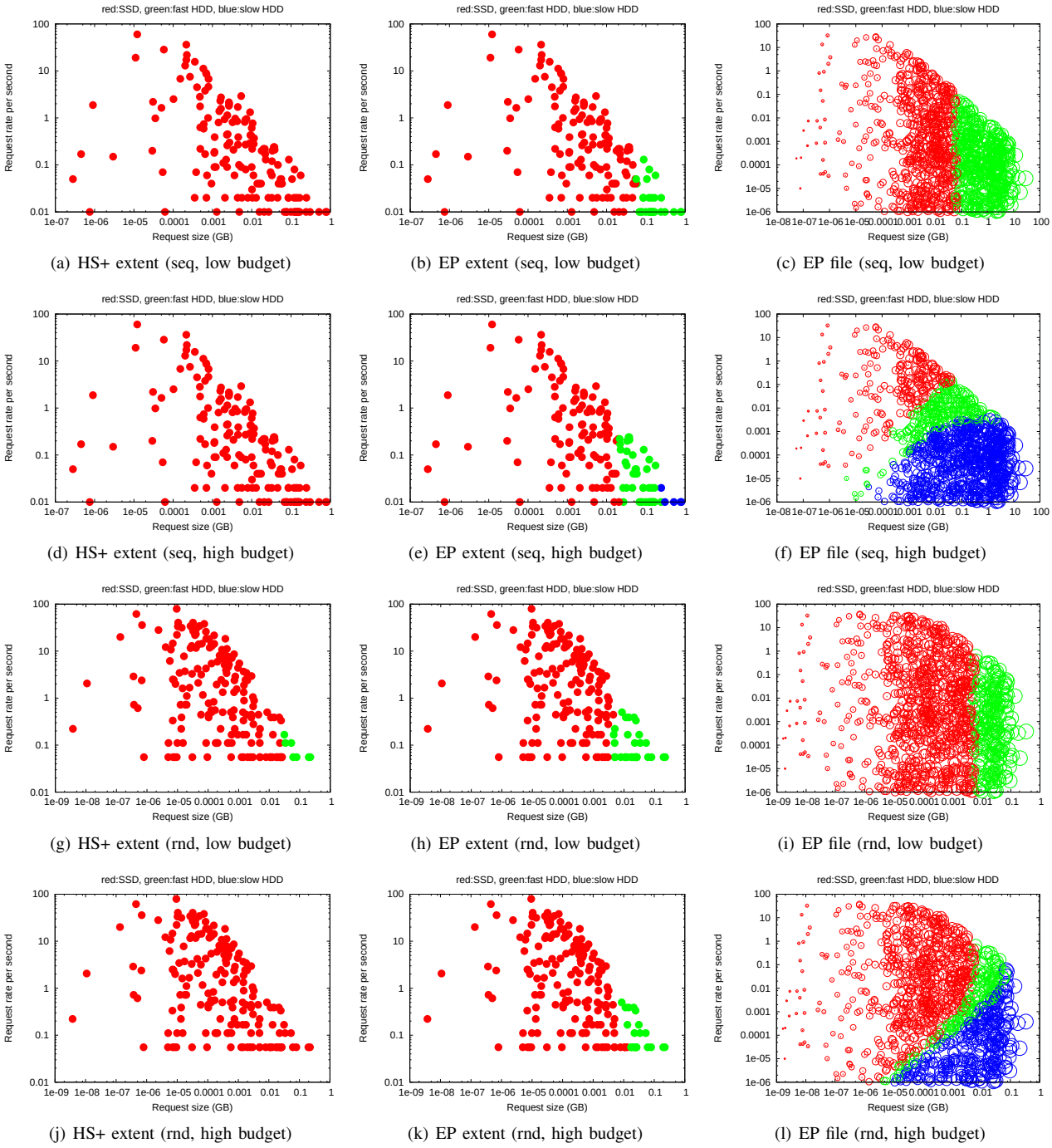


Fig. 8. CERN file-size distribution

the sequential vs. random access patterns represented by the two workloads do not affect this result. Therefore, from the discussion of Section V-B, we could deduce that most of the requests are directed towards the smaller files. Indeed, the stability region from which we sample the request rates, as detailed in Section V-C, gets narrower as the request size increases. Consequently, larger files tend to have smaller request rates than smaller files.

Comparing ExaPlan(extent) with HybridStore+(extent), see Figures 9(a) and 9(c), we observe that for all price points considered, ExaPlan significantly outperforms HybridStore+. This is true for both the estimated and simulated mean response time values. The reason why the simulation results may differ from estimated results in some cases is due to the imperfect load balancing when distributing the chunks assigned to a specific tier to the individual devices of that tier, or to the use of a trace with a duration which is too small compared with the estimated response time. Nevertheless, the simulated values confirm that the estimated values are accurate, albeit not perfect, indicators of performance even when the ideal load-balancing assumption no longer holds.

A notable difference between how ExaPlan and HybridStore+ uses the three available tiers, see Figures 9(b) and 9(d), is that HybridStore+'s device mix is predominantly SSDs, whereas ExaPlan uses a balanced mix among the three tiers. For storing the same volume of data, Figures 9(b) and 9(d)



(a) HS+ extent (seq, low budget)
(b) EP extent (seq, low budget)
(c) EP file (seq, low budget)
(d) HS+ extent (seq, high budget)
(e) EP extent (seq, high budget)
(f) EP file (seq, high budget)
(g) HS+ extent (rnd, low budget)
(h) EP extent (rnd, low budget)
(i) EP file (rnd, low budget)
(j) HS+ extent (rnd, high budget)
(k) EP extent (rnd, high budget)
(l) EP file (rnd, high budget)

Fig. 10. Data placement for CERN experiments for 1000 randomly sampled chunks for the highest and lowest budgets considered in Section V-D. Note that chunks with 0 request rate cannot be plotted on a log-log scale. The diameter of the circles in the file cases represent the size of the file (in log scale).

show that the device mix difference between sequential workload and random workload when using ExaPlan is in accordance with intuition, i.e., more slow HDDs and fewer fast SSDs will be used in the sequential case than the random case.

The data placements for ExaPlan and HybridStore+ are shown in Figure 10 for a few select price points. We note that the cost-per-GB values considered in this experiment are much higher than current market rates for cost per GB of storage. We emphasize that the price points of the experiments are the cost per GB of stored data as opposed to cost per GB of available capacity. In addition to this, as the request rate information was unavailable from CERN, we had to select a rule to generate the request rates for the individual files. The eventual rates selected corresponded to a very load-heavy (as opposed to volume-heavy) workload scenario, and consequently many devices were needed, not for the volume but for the required service capacity. As a result, there was a considerable amount of unused storage capacity, leading to remarkably high cost per GB of stored data.

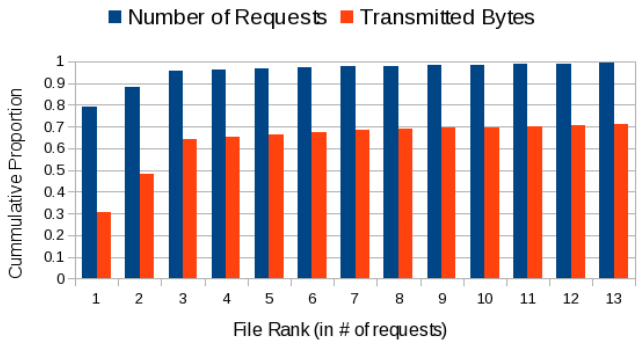


Fig. 11. LOFAR imaging-pipeline file-access characteristics. Cumulative proportion of number of requests and bytes transmitted for the top 13 files.

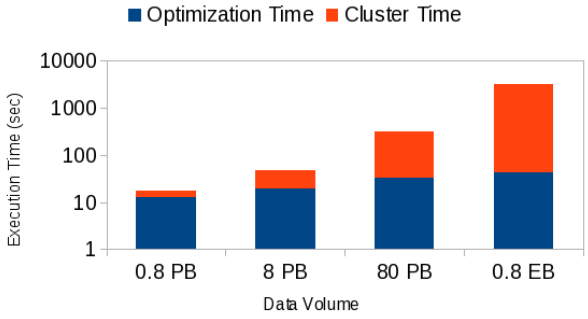


Fig. 14. Execution time of ExaPlan decomposed into cluster time and optimization time for Workload 1 with file-level extents. Scales from 1 million files (0.8 PB) to 1 billion files (0.8 EB).

E. Workload 2: Imaging in astronomy

We traced a full imaging pipeline for the LOFAR radio telescope array [22] on real observation data that resulted in a 113-minute trace involving 1044 files accounting for 15 GB in volume. The total number of requests observed during this period was 20,214,062, resulting in a total of 40.4 GB read and written. We recorded the I/O system calls with strace in a isolated environment in which only the LOFAR imaging pipeline was running. For this workload, we observed that 79.2% of the number of requests, accounting for 30.4% of the bytes transferred, are to the largest file of size 11 GB. Moreover, 1% of the files account for 98.1% of the requests and 69.2% of the total bytes transferred. Further details of the workload are illustrated in Figure 11.

From the trace, we obtained file-level access characteristics, including the mean and variance of request size, and the request rate for each file. By observing the ratio between the request size and the corresponding file size, we were able to estimate the parameters of the beta distribution used to characterize the request size distributions that correspond to the individual files. It turned out that the solution provided by ExaPlan for this workload with a total volume of 15 GB involved small fractions as the number of devices, e.g., 0.01 hard disk drives. Therefore, simulating this workload based on that solution would be problematic as the simulator only accepts integer device numbers, which implies that we would have to round the solution to integers. Instead, we considered a modified scaled-up version of the workload for which the solution obtained involves roughly integer device numbers. More specifically, we scaled up the workload 1000 times, which inflated the volume to 15 TB, by duplicating the characteristics of the 1044 files 1000 times to effectively consider 1,044,000 files. For this inflated workload, we then generated a synthetic trace with Poisson arrivals which we used as input to the simulator.

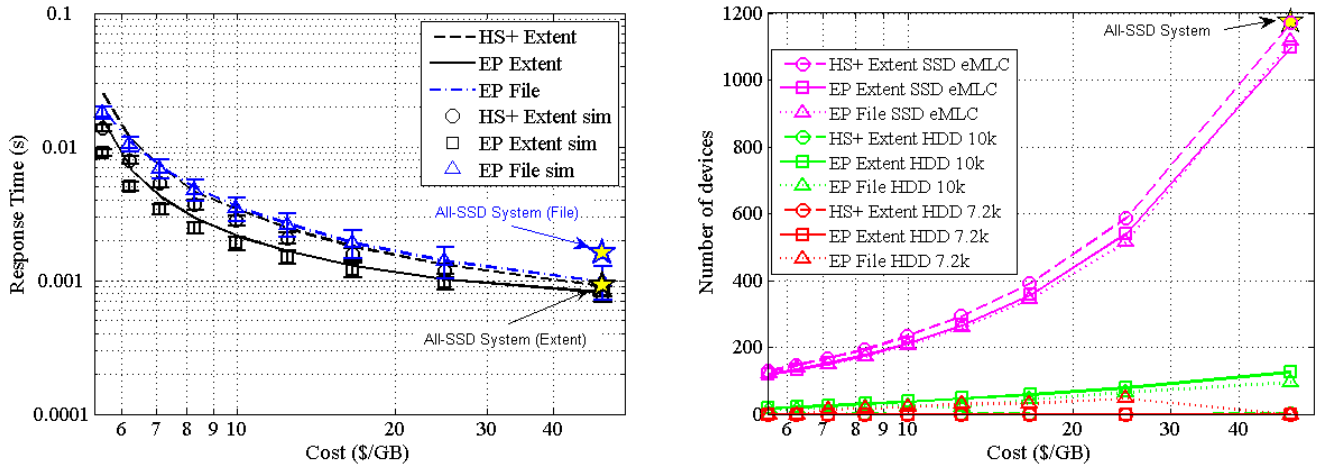
Similar to Workload 1, we note from Figure 12(a) that ExaPlan can offer a significant advantage in terms of cost and performance, when compared with an all-SSD system. As observed earlier, by reducing the number of SSDs, ExaPlan is able to afford a greater number of HDDs (see Figure 12(b)), which helps offload chunks with larger request sizes from SSDs and therefore improves performance. In contrast to Workload 1, Figure 12(a) demonstrates that file-level placement perform worse than extent-level placement. This is attributed to the fact that there is a large dominating file that accounts for the majority of the requests. As discussed in Section V-B, a few large files accounting for most of the requests will make it advantageous to split larger files into extents and assign these extents separately.

Figure 12(a) demonstrates that ExaPlan(extent) outperforms HybridStore+(extent) for all price points considered for this workload. It is not as pronounced as for Workload 1, but from Figure 12(b), we can see HybridStore+'s inclination towards favoring SSDs by comparing the number of other devices used in the device mix.

The data placements for ExaPlan and HybridStore+ are shown in Figure 13 for a few select price points. As was the case in Workload 1, we also see in this workload that the cost-per-GB values considered are extremely high compared to current market rates on cost per GB of storage. This is due to the fact that the files considered are exclusively files that appeared in the trace that was part of an active pipeline run. These files have relatively high access rates, and as the trace was scaled-up by a factor of 1000 to obtain the workload, the workload is artificially inflated in terms of request rates. The results shown in Figure 12 should be interpreted in terms of what to expect if all data in the system were constantly accessed for long periods of time.

F. Scalability

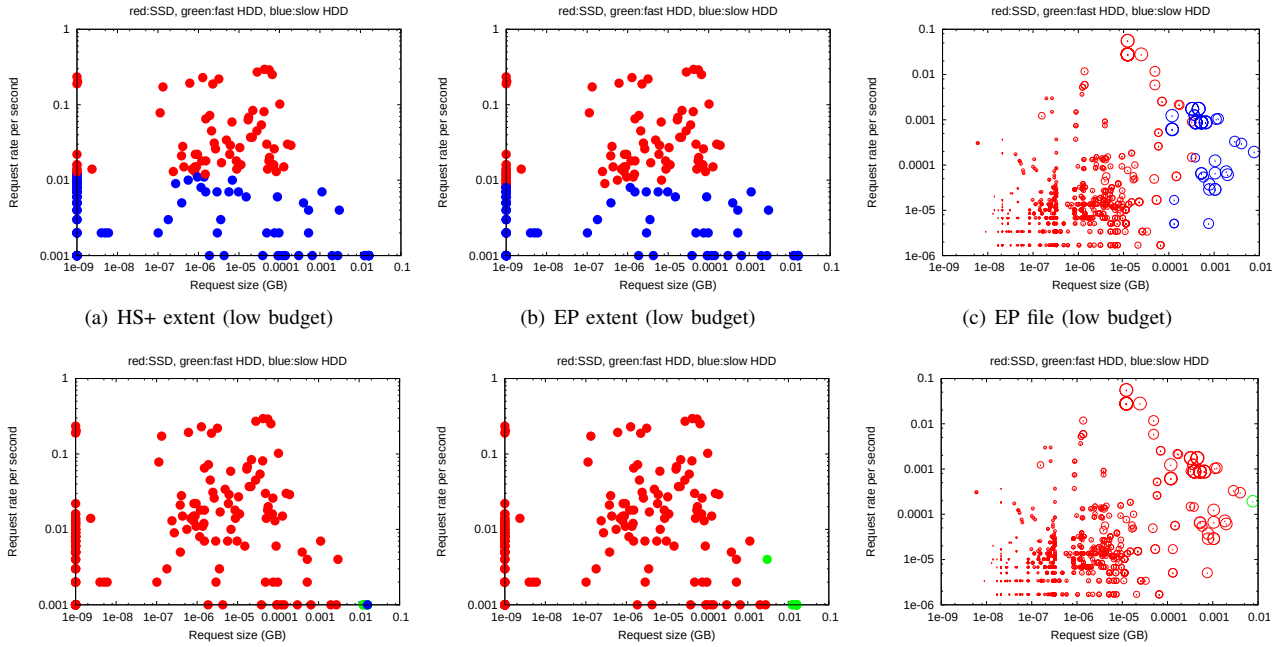
To demonstrate the scalability of ExaPlan, we ran it with the sequential case of Workload 1 using file placements (1 million files) as a baseline and increased the number of files by a factor of 10, 100, and 1000, recording the time each run spent in the clustering and optimization phases. This allowed us to observe how ExaPlan scales from running on a data volume ranging from 0.8 PB to 0.8 EB. This was done in a single threaded mode on an Intel® Xeon® E5-2680 CPU clocked at 2.70 GHz with 4 GB of memory quota. The results are summarized in Figure 14. The optimization time ranges from 12.5 sec for 0.8 PB to 43.4 sec for 0.8 EB. The optimization time depends only on the number of clusters, so we can roughly estimate that the 0.8 EB workload has four times more clusters than the 0.8 PB workload. Using the grid clustering



(a) Mean Response Time (seq)

(b) Number of Devices (seq)

Fig. 12. LOFAR imaging pipeline results. In (a), lines represent model estimates and markers represent simulation values at the corresponding price points. The error bars represent the response time standard deviation divided by 30. For each workload, the all-SSD system (with file-level or extent-level data placement) corresponds to the highest price point shown in the corresponding figures.



(a) HS+ extent (low budget)

(b) EP extent (low budget)

(c) EP file (low budget)

(d) HS+ extent (high budget)

(e) EP extent (high budget)

(f) EP file (high budget)

Fig. 13. Data placement for LOFAR experiments for 1000 randomly sampled chunks for the highest and lowest budgets considered in Section V-E. Note that chunks with 0 request rate cannot be plotted on a log-log scale. The diameter of the circles in the file cases represent the size of the file (in log scale).

algorithm of Section IV-D, the number of clusters is bounded by a user-specified value so the optimization time is bounded for any workload size.

The bulk of the execution time is spent parsing the workload file and clustering the chunks, where each row corresponds to each chunk and its access characteristics. The clustering time increases from 5.2 sec for the 0.8 PB workload to 3127 sec for the 0.8 EB workload. In practice, we envision that the clustering of files can be performed periodically as regular maintenance and kept as part of file metadata so that the next time the data placement needs to be optimized or the storage layout modified, it suffices to perform only the quick optimization part.

VI. Conclusions

Access-pattern-aware multi-tiered storage systems composed of storage devices with different capacity, access time, and sustained throughput characteristics can provide high-performance operation and reduce costs. Assessing the number of devices provided per tier and determining an efficient way of placing vast volumes of data across tiers pose a great challenge. A straightforward formulation of an optimization problem for the task of determining the data-to-tier assignment that minimizes the mean system response time is typically intractable.

This article presented ExaPlan, a method that circumvents these challenges and determines both the data-to-tier assignment and the number of devices in each tier that minimize the system's mean response time for a given budget and workload. It achieves this by parameterizing the data placement that, for fixed parameter values, leads to the derivation of closed-form expressions for the optimal dimension of each tier. It subsequently explores the data-placement parameter space using a black-box optimization module to identify efficient solutions. Experimental results demonstrated that ExaPlan was able to provision exascale systems efficiently by providing device mixes and data placement for a variety of workloads. It was observed that ExaPlan outperforms HybridStore+ in all cases we have tested, and that there is a strong evidence that, for workloads that have data with vastly different access characteristics, file-level data placement is superior to extent-level data placement.

REFERENCES

[1] H. Shi et al., "Optimal disk storage allocation for multi-tier storage system," in APMRC, 2012 Digest, 2012, pp. 1–7.

[2] P. E. Dewdney, SKA1 System Baseline Design. SKA Office, 2013.

[3] E. Anderson et al., "Hippodrome: Running circles around storage administration," in Proc. 1st USENIX Conf. on File and Storage Technologies (FAST '02), 2002.

[4] G. A. Alvarez et al., "Minerva: An automated resource provisioning tool for large-scale storage systems," ACM Trans. Comput. Syst., vol. 19, no. 4, 2001, pp. 483–518.

[5] E. Anderson et al., "Quickly finding near-optimal storage designs," ACM Trans. Comput. Syst., vol. 23, no. 4, 2005, pp. 337–374.

[6] J. Guerra et al., "Cost effective storage using extent based dynamic tiering," in Proc. 9th USENIX Conf. on File and Storage Technologies (FAST'11), 2011.

[7] Y. Kim et al., "Hybridstore: A cost-efficient, high-performance storage system combining SSDs and HDDs," in Proc. 2011 IEEE 19th Annual Int'l Symposium on Modelling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS'11), 2011, pp. 227–236.

[8] N. Hansen and A. Ostermeier, "Adapting arbitrary normal mutation distributions in evolution strategies: the covariance matrix adaptation," in Proc. of IEEE Int'l Conf.on Evolutionary Computation, 1996, pp. 312–317.

[9] J. D. Strunk et al., "Using utility to provision storage systems," in Proc. of the 6th USENIX Conference on File and Storage Technologies (FAST'08), 2008, pp. 21:1–21:16.

[10] L. Lin et al., "Hot random off-loading: A hybrid storage system with dynamic data migration," in Proc. 2011 IEEE 19th Annual Int'l Symposium on Modelling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS'11), 2011, pp. 318–325.

[11] J. Wolf, "The placement optimization program: A practical solution to the disk file assignment problem," SIGMETRICS Perform. Eval. Rev., vol. 17, no. 1, 1989, pp. 1–10.

[12] D. Essary and A. Amer, "Predictive data grouping: Defining the bounds of energy and latency reduction through predictive data grouping and replication," Trans. Storage, vol. 4, no. 1, 2008, pp. 2:1–2:23.

[13] A. Wildani, "The promise of data grouping in large scale storage systems," Ph.D. dissertation, University of California, Santa Cruz, CA, USA, 2013.

[14] K. Brandt et al., "Predicting when not to predict," in Proc. 12th Annual IEEE/ACM Int'l Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS'04), 2004.

[15] A. K. Iyengar, M. S. Squillante, and L. Zhang, "Analysis and characterization of large-scale web server access patterns and performance," World Wide Web, vol. 2, no. 1-2, Jan. 1999, pp. 85–100.

[16] R. L. Graham, "Bounds on multiprocessing timing anomalies," SIAM Journal on Applied Mathematics, vol. 17, no. 2, 1969, pp. 416–429.

[17] J. F. C. Kingman, "The single server queue in heavy traffic," Mathematical Proc. Cambridge Philosophical Society, vol. 57, 1961, pp. 902–904.

[18] B. Balcioglu et al., "Merging and splitting autocorrelated arrival processes and impact on queueing performance," Perform. Eval., vol. 65, no. 9, 2008, pp. 653–669.

[19] N. Clayton and C. Fuente, Planning for Easy Tier with IBM System Storage Storwize V7000 and SAN Volume Controller. IBM Corporation, 2013.

[20] EMC Corporation, White Paper: EMC VNX FAST VP VNX5100, VNX5300, VNX5500, VNX5700, & VNX7500 A Detailed Review. EMC Corporation, 2013.

[21] G. Cancio, "Tape archive challenges when approaching exabyte-scale," 2010, cHEP 2010.

[22] M. P. van Haarlem et al., "LOFAR: The LOw-Frequency ARray," Astronomy&Astrophysics, vol. 556, no. A2, 2013.

APPENDIX A
PROOF OF PROPOSITION 1

The total arrival rate λ_k of all requests to tier k is the sum of the arrival rates of requests for data residing within the chunks in this tier, that is,

$$\lambda_k = \sum_{j \in I_k} r_j \ . \tag{21}$$

We use Kingman's formula [17] with the coefficient of variation of the inter-arrival time set to 1 to approximate the mean waiting time for the $M/G/1$ queue:

$$\mathbf{E}[W_k] \simeq \left(\frac{\rho_k}{1-\rho_k}\right)\left(\frac{1+c_s^2}{2}\right)\mathbf{E}[S_k]. \tag{22}$$

Assuming perfect load balancing within a tier, the request arrival rate at a device in tier k is equal to λ_k/z_k and its mean service time is equal to $1/\mu_k$. The load ρ_k of a single device of tier k is therefore given by

$$\rho_k = \lambda_k/(z_k\mu_k) \ . \tag{23}$$

The approximate mean waiting time of a request served by tier k is

$$\mathbf{E}[W_k] \simeq \frac{\rho_k}{1-\rho_k} \cdot \frac{1}{\mu_k} \cdot \frac{1+c_s^2}{2} \ . \tag{24}$$

Adding the mean service time to (24), the mean response time, $\mathbf{E}[T_k]$, is

$$\mathbf{E}[T_k] \simeq \frac{\rho_k}{1-\rho_k} \cdot \frac{1}{\mu_k} \cdot \frac{1+c_s^2}{2} + \frac{1}{\mu_k} \ . \tag{25}$$

Taking the average of (25) weighted by the request rates λ_k over all tiers k, the mean response time at the system level, $\mathbf{E}[T]$, is given by

$$\begin{aligned}
\mathbf{E}[T] &= \sum_k \frac{\lambda_k}{\lambda}\mathbf{E}[T_k] \\
&\simeq \frac{1}{\lambda}\sum_k \left[\frac{(\lambda_k/\mu_k)^2}{z_k - \lambda_k/\mu_k} \cdot \frac{1+c_s^2}{2} + \frac{\lambda_k}{\mu_k}\right] \\
&= \frac{1}{\lambda}\sum_k \left[\frac{(\lambda_k/\mu_k)^2}{z_k - \lambda_k/\mu_k} \cdot \frac{1+\sigma_k^2\mu_k^2}{2} + \frac{\lambda_k}{\mu_k}\right],
\end{aligned} \tag{26}$$

where $1/\mu_k$ and σ_k^2 are the mean and the variance of the service time distribution of tier k respectively. Denote the service time random variable of tier k by S_k, the chunk

containing the request in service by $J(\in I_k)$, and the mean request size to chunk j $\mathbf{E}[Q_j]$ by q_j. Then

$$P(J=j) = \frac{r_j}{\lambda_k} , \tag{27}$$

$$\mathbf{E}[S_k|J=j] = s_k + \frac{q_j}{b_k} , \tag{28}$$

and

$$\begin{aligned} 1/\mu_k &= \sum_{j\in I_k} \mathbf{E}[S_k|J=j]P(J=j) \\ &= \sum_{j\in I_k} \left(s_k + \frac{q_j}{b_k}\right)\frac{r_j}{\lambda_k} . \end{aligned} \tag{29}$$

Conditioning on the chunk and using the variance decomposition formula

$$\sigma_k^2 = \mathrm{Var}(S_k) = \mathrm{Var}(\mathbf{E}[S_k|J]) + \mathbf{E}[\mathrm{Var}(S_k|J)] , \tag{30}$$

where

$$\begin{aligned} \mathrm{Var}(\mathbf{E}[S_k|J]) &= \mathrm{Var}\left(s_k + \frac{q_J}{b_k}\right) \\ &= \frac{1}{b_k^2}\left[\sum_{j\in I_k} q_j^2 \cdot \frac{r_j}{\lambda_k} - \left(\sum_{j\in I_k} q_j \cdot \frac{r_j}{\lambda_k}\right)^2\right] , \end{aligned} \tag{31}$$

$$\begin{aligned} \mathbf{E}[\mathrm{Var}(S_k|J)] &= \sum_{j\in I_k} \mathrm{Var}\left(s_k + \frac{Q_j}{b_k}\right) P(J=j) \\ &= \sum_{j\in I_k} \frac{1}{b_k^2}\mathrm{Var}(Q_j)\frac{r_j}{\lambda_k} . \end{aligned} \tag{32}$$

Substituting (31) and (32) into (30) yields

$$\begin{aligned} \sigma_k^2 =& \frac{1}{b_k^2}\left[\sum_{j\in I_k} q_j^2 \cdot \frac{r_j}{\lambda_k} - \left(\sum_{j\in I_k} q_j \cdot \frac{r_j}{\lambda_k}\right)^2\right] \\ &+ \sum_{j\in I_k} \frac{1}{b_k^2}\mathrm{Var}(Q_j)\frac{r_j}{\lambda_k} . \end{aligned} \tag{33}$$

Appendix B
Proof of Proposition 2

The objective function (11) is a sum of convex rational functions and the constraint set is a convex polyhedron, which makes the minimization problem convex. Thus, in the formulation, a local minimum implies a global minimum. Let ζ_k, η_k denote the Lagrange multipliers corresponding to the constraints of this minimization problem and consider the Lagrangian function

$$\begin{aligned} &L(z,\omega,\zeta,\eta) = \\ &\sum_k \frac{(\lambda_k/\mu_k)^2 a_k}{z_k - \lambda_k/\mu_k} + \omega\left(\sum_{k=1}^K z_k C_k \gamma_k - B\right) \\ &+ \sum_{k=1}^K \zeta_k \left(\sum_{j\in I_k} v_j - z_k C_k\right) + \sum_{k=1}^K \eta_k \left(\frac{\lambda_k}{\mu_k} - z_k\right) . \end{aligned} \tag{34}$$

At a local minimum point, the following first-order necessary conditions (KKT conditions) need to be satisfied:

$$\frac{\partial L}{\partial z_k} = 0 \quad (k=1,\ldots,K) \tag{35}$$

$$\omega\left(\sum_k z_k C_k \gamma_k - B\right) = 0 \tag{36}$$

$$\zeta_k\left(\sum_{j\in I_k} v_j - z_k C_k\right) = 0 \quad (k=1,\ldots,K) \tag{37}$$

$$\eta_k\left(\frac{\lambda_k}{\mu_k} - z_k\right) = 0 \quad (k=1,\ldots,K) \tag{38}$$

$$\sum_k z_k C_k \gamma_k \le B \tag{39}$$

$$z_k \ge \frac{\sum_{j\in I_k} v_j}{C_k} \quad (k=1,\ldots,K) \tag{40}$$

$$z_k \ge \frac{\lambda_k}{\mu_k} \quad (k=1,\ldots,K) \tag{41}$$

$$\omega \ge 0 \tag{42}$$

$$\zeta_k \ge 0 \quad (k=1,\ldots,K) \tag{43}$$

$$\eta_k \ge 0 \quad (k=1,\ldots,K) \tag{44}$$

To find the optimal number of devices for each tier, it suffices to solve this system of $3K+1$ equations (35, 36, 37, 38) with $3K+1$ unknowns $(\omega, \zeta_1,\ldots,\zeta_K,\eta_1,\ldots,\eta_K,z_1,\ldots,z_K)$ and verify that the solution also satisfies the inequality conditions.

Consider the non-trivial case where $\lambda_k > 0$ for at least one tier k. From (34) and condition (35), it follows that

$$-a_k\left(\frac{\lambda_k}{\mu_k z_k - \lambda_k}\right)^2 + \omega C_k\gamma_k - \zeta_k C_k - \eta_k = 0 . \tag{45}$$

Thus, for $\lambda_k > 0$ and $\omega = 0$, (45) implies that $\zeta_k C_k + \eta_k < 0$, which contradicts (43) and (44). Therefore, we conclude $\omega > 0$ at an optimal point. It is necessary that constraint (14) be non-binding for the objective function to have a finite value. We therefore set $\eta_k = 0$ for all k. Also note that for $k \notin A$, which implies a strict inequality in (13), and considering (37), we get that $\zeta_k = 0$. If $\lambda_k = 0$ was true for $k \notin A$, then from the preceding, (45) would imply that $\omega = 0$, which is false. Thus, $\lambda_k > 0$ for $k \notin A$. Conversely, this means that for a tier with zero request rates, i.e., $\lambda_k = 0$, the volume constraint should always be binding, i.e., $k \in A$.

As we assume that constraint (13) is binding for $k \in A$,

$$z_k^* = \sum_{j\in I_k} \frac{v_j}{C_k} \quad (k\in A). \tag{46}$$

We solve the following system of equations with unknowns $\{z_k, \omega | k \notin A\}$ to find z_k^* for $k \notin A$:

$$-a_k\left(\frac{\lambda_k}{\mu_k z_k - \lambda_k}\right)^2 + \omega C_k\gamma_k = 0 \quad (k\notin A) \tag{47}$$

$$\sum_{k\notin A} z_k C_k \gamma_k = B - \sum_{k\in A}\sum_{j\in I_k} v_j\gamma_k. \tag{48}$$

Equation (47) is a direct result of condition (35) and equation (48) is derived from the result (46) plugged into

$$\sum_{k \notin A} z_k C_k \gamma_k + \sum_{k \in A} z_k^* C_k \gamma_k = B \tag{49}$$

from condition (36) with the previously derived result that $\omega > 0$ at an optimal solution.

Essentially, we are removing the tiers with binding volume constraints from the problem by adjusting the budget and setting the Lagrange multipliers ζ_k to 0 for the remaining tiers $k \notin A$. Assuming that at an optimal solution, $\zeta_k = 0$ for $k \notin A$ indeed holds, and using (16), the optimal number of devices for tier k is given by

$$z_k^* = \begin{cases} \sum\limits_{j \in I_k} v_j / C_k & \text{if } k \in A \\ \left\lfloor \dfrac{\left(B - \sum\limits_{i \in A} H_i - \sum\limits_{i \notin A} \frac{\lambda_i}{\mu_i} C_i \gamma_i \right) \sqrt{a_k}}{\sqrt{C_k \gamma_k} \sum\limits_{i \notin A} \frac{\lambda_i}{\mu_i} \sqrt{a_i C_i \gamma_i}} + 1 \right\rfloor \frac{\lambda_k}{\mu_k} & \text{if } k \notin A. \end{cases} \tag{50}$$

APPENDIX C
REQUEST SIZE VARIANCE FOR A CLUSTER OF CHUNKS

As Q_C is a mixture of the random variables Q_j for $j \in C$, with mixture probabilities $r_j / \sum_{i \in C} r_i$, we obtain $\mathrm{Var}(Q_C)$ by conditioning on chunk J and using the variance decomposition formula.

$$\mathrm{Var}(Q_C) = \mathrm{Var}(\mathbf{E}[Q_C|J]) + \mathbf{E}[\mathrm{Var}(Q_C|J)] \,,$$

where

$$\begin{aligned} \mathrm{Var}(\mathbf{E}[Q_C|J]) &= \sum_{j \in C} \left(\mathbf{E}[Q_C|J=j]\right)^2 \frac{r_j}{\sum_{i \in C} r_i} \\ &\quad - \left\{ \sum_{j \in C} \mathbf{E}[Q_C|J=j] \frac{r_j}{\sum_{i \in C} r_i} \right\}^2 \\ &= \sum_{j \in C} \left(\mathbf{E}[Q_j]\right)^2 \frac{r_j}{\sum_{i \in C} r_i} \\ &\quad - \left\{ \sum_{j \in C} \mathbf{E}[Q_j] \frac{r_j}{\sum_{i \in C} r_i} \right\}^2 , \\ \mathbf{E}[\mathrm{Var}(Q_C|J)] &= \sum_{j \in C} \mathrm{Var}(Q_C|J=j) \frac{r_j}{\sum_{i \in C} r_i} \\ &= \sum_{j \in C} \mathrm{Var}(Q_j) \frac{r_j}{\sum_{i \in C} r_i} \,. \end{aligned}$$