Revisiting Heterogeneous Storage Optimization in the Vector-Sum Model

Bojun Huang
Microsoft Research
Beijing, China
bojhuang@microsoft.com

Thomas Moscibroda
Microsoft Research
Beijing, China
moscitho@microsoft.com

ABSTRACT

Large-scale data centers often adopt more than one type of storage device, each with different storage capacity, I/O capability, and cost. Optimizing the performance-to-cost efficiency of such heterogeneous storage systems is of great practical importance (Cap-Ex), and it is a classic problem in computer system design. The Vector-Sum Model (VSM) is a mental model widely-used by system administrators for this task, due to its conceptual simplicity. The model encompasses various commonly-used rules-of-thumb, such as the *five-minute rule* or various *Knapsack*-based heuristics.

In this paper we revisit the vector-sum model and study heterogeneous storage using a new form of *optimization diagrams*. These diagrams give raise to a near-optimal solution to the problem, which subsumes the existing rules-of-thumb used in practice. Our solution also explains that these heuristics are indeed optimal under their respective assumptions, while they become sub-optimal in more general cases. Specifically, our analysis implies that the recent adoption of SSD in data centers may challenge the quality of these commonly-used heuristics, and that our new optimization approach can sustain data center-scale workloads at lower total purchasing cost. Finally, we show that, although the commonly-used I/O metrics of storage are non-additive, we can use regression techniques to transform the metric into an additive form. Experiments using web search production workloads show that the Vector-Sum Model becomes more accurate after the metric transformation.

1. INTRODUCTION

Large-scale data centers often adopt more than one type of storage devices, each with different storage space, I/O capability, cost, and other characteristics. For example, storage hierarchies consisting of SSD and HDD have become typical for a variety of large-scale production workloads in recent years [22]. Given the tremendous cost of building and maintaining data center infrastructure, optimizing the cost and efficiency of such heterogenous storage systems has become an area of great concern and interest in industry. Simplifying, we can state that there are two fundamental problems affecting the efficiency and performance of such large-scale heterogeneous storage systems: i) How much money should we invest on each type of storage (capacity planning), and ii) which data objects should be placed on which storage type (data placement). The over-arching goal is to satisfy the aggregate resource demands of the application workload, while minimizing the total purchasing cost (Cap-EX) of the required storage devices.

Due to the complexity of such large-scale heterogenous storage systems, storage administrators typically have to rely on "rules-of-thumb" to optimize the system. In general, these rules-of-thumb are simple heuristics that are easy to understand and apply, while still giving reasonably good results in practice. Maybe the most famous and widely-employed rules of this kind are the *five minute rule* by Gray and Putzolu [12] and various Knapsack-based heuristics (e.g. [5] [4] [23]). These heuristic rules have been derived reflecting a specific mental model of the target system and workload behavior. Arguably the simplest and most fundamental of these models is the *Vector-Sum Model* (VSM), which can be seen as a first-order approximation of the real system. In VSM, both data and storage are characterized as vectors of their per-object and per-dollar features. The aggregate resource demand (capacity) of a data set (a storage system) is the *sum* of these per-object (per-dollar) feature vectors. A system design is feasible if the aggregate resource capacity of the storage matches the aggregate resource demand of the data assigned to it. As mentioned, the Vector-Sum Model has been widely used by practitioners largely thanks to its conceptual simplicity (e.g. all the aforementioned rules-of-thumbs have been devised under this model, either explicitly or implicitly). On the other hand, there are also longstanding arguments as to the accuracy of the Vector-Sum Model, particularly because of the model's linearity [2].

In this paper we revisit the Vector-Sum Model in terms of both optimization rules and model validation. We analyze the optimality of existing rules-of-thumb under the model. We identify novel near-optimal optimization rules that not only subsume these classic rules in those cases in which the rules are indeed optimal, but also improve on them in more general cases. Moreover, we observe that the ever-expanding scale of modern storage systems may bring new opportunities to improve the modeling accuracy of VSM by statistically *linearizing* the model parameters. Collectively, these results suggest that the Vector-Sum Model could potentially be an abstraction featuring both simplicity and effectiveness for large-scale storage system consolidations. Specifically,

- We analyze the optimization problems under the Vector-Sum Model using a new form of *heterogenous storage optimization diagrams* (Section 3). We identify a near-optimal solution of the problem in these diagrams, which corresponds to the most "compact" solution when such a solution is feasible, and degenerates to the *five-minute rule* and the *knapsack-based rules* in special cases. In particular, this observation suggests that these

widely-used heuristics are indeed optimal in the particular scenarios where they have been typically used and for which they have been designed for. However, our analysis also implies that the optimality of these rules-of-thumbs crucially depends on model parameters. In particular, experimental results show that the adoption of SSD seriously challenges the validity of these classic rules, rendering them no longer optimal. In these cases our new solution can serve as a new and more general rule-of-thumb, yielding substantial cost reduction for cloud-scale production systems. (Section 5.3).

- We also present a regression-based approach to make the I/O related parameters in VSM *statistically additive* (Section 4). Instead of using general-purpose device models, we try to train *workload-specific* models directly using the traces from the target workload. Experimental results on real storage hardware and real-world workloads show an average prediction error of less than 4.2% for HDD and 3.0% for SSD (Section 5.2).

2. BACKGROUND

The Vector-Sum Model formulates the problem of finding the most economic design to host a given set of *data objects* with *storage devices* chosen from a given set of storage types. Depending on the level of system integration, a storage device can be a storage package (e.g. Hard-Disk Drive, Solid-State Drive, DRAM DIMM), or alternatively a RAID group, or even a single storage chip. Similarly, depending on the application, a data object may be either a domain-specific data structure (e.g. string, file segment, database table or row) or a low-level storage management unit such as a page and a block. [1] In this section we briefly review the Vector-Sum Model, as well as two standard heuristic rules derived from the model.

2.1 The Vector-Sum Model

Suppose we want to host N data objects on two different types of storage devices, say, SSD and Hard Disk Drives (HDD). In the Vector-Sum Model (VSM), each data object i is characterized by a feature vector (c_i, b_i) consisting of its size c_i and the expected I/O demand b_i (e.g. throughput, bandwidth, or a function of both). Accordingly, each storage-type is also characterized by a vector of its *per-dollar* resource capacity in the same vector space, denoted by (C_1, B_1) and (C_2, B_2) for the type-1 and type-2 storages, respectively. Common metrics include GB/$, IOPS/$, and so on. Without loss of generality we consider the type-1 storage to be the more economic one in terms of I/O capability (e.g. SSD), while the type-2 storage is more economic in terms of in storage space (e.g. HDD).

To design a storage system using two different types of storage, we need to determine the amount of money invested on each storage-type, as well as decide on which type of storage each data object should be placed. [2] In VSM, the aggregate resource demand of the data set assigned to each storage-type is the *vector-sum* of all feature vectors of the data set. On the other hand, the aggregate resource capacity of a storage type equals the per-dollar-capacity vector of the storage multiplied by the amount of money invested on it. See Figure 1a for an illustration of the model. A system design is *feasible* in VSM if the aggregate capacity of the storage is no less than the aggregate demand of the data, for both storage space and I/O performance. Our goal is to identify the feasible system design having the least purchasing cost.

It is worthwhile distinguishing the Vector-Sum Model thus described from another similar model, the *bin packing* (or vector packing) formulation, in which the resource demand and capacity are also modeled as vector sums but each storage *unit* (rather than a storage type) is considered as a separate "container" which has a fixed resource capacity. In contrast, the Vector-Sum Model ignores the boundary constraints between storage units of the same type as well as the integral constraint in purchasing these storage units. In VSM, we consider each storage-type as a whole, investing arbitrary amount of money on it and only calculating the aggregated resource constraints over all units of this type. This relaxation from bin packing to VSM is clearly suited for today's *large-scale* systems typically found in cloud data centers, in which the number of data objects N can be in the millions or billions. In these systems, the scale of storage units and the whole storage arrays are usually orders of magnitudes larger than the scales of individual data objects and the price of individual storage units, respectively.

The Vector-Sum Model is probably the simplest model (and also among the most widely-used by practitioners) to understand and reason about. Of course, VSM is a first-order approximation of a complex real system, and thus can be challenged for being not entirely accurate. For example, it is a longstanding open problem to accurately characterize the I/O demands of data objects in an *additive* manner [3], as required by VSM. We will discuss the I/O linearization issue in more details later. In fact, we argue that it may be possible to *statistically* linearize the I/O performance given the ever-expanding scale of modern storage systems. On the other hand, despite its conceptual simplicity, it can still be challenging to optimize a large-scale system in the VSM model, since the entirety of the model can easily blow-up to include millions or billions of decision variables. Thus in practice, system administrators typically rely on various "rules-of-thumb" to help in their decision making. In the following we present two of the most well-known and widely-used heuristics. We use the symbols in Table 1 to denote the related model parameters and decision variables.

Table 1: Symbols used in the vector-sum model

(c_i, b_i)	The feature vector of data object i, where c_i and b_i stand for the capacity and bandwidth demands, resp.
(C_1, B_1)	The "per-dollar" feature vector of the type-1 storage
(C_2, B_2)	The "per-dollar" feature vector of the type-2 storage
y_1, y_2	The amount of money spent on the two storage-types
$x_1 \ldots x_N$	The data placement decisions. $x_i = \{1, 2\}$ indicates in which type of storage the data object i is placed.

2.2 The Five Minute Rule

In 1987, Jim Gray and Franco Putzolu [12] proposed a famous rule-of-thumb to guide capacity planning and data placement in database systems consisting of DRAM and hard drive. The rule says that we should place a data object in hard drives if the expected access interval of the object is longer than a certain threshold. At the time this rule was proposed, the threshold roughly equaled 5 minutes for 1KB objects, thus the rule was later dubbed *the five-minute rule*. Intriguingly, the five-minute rule has been reviewed by researchers roughly every ten years since then, and it has been found that the specific threshold of "five minutes" happens to be quite stable even though the parameters of storage devices have changed dramatically [11], and even as completely new storage device like SSD have been employed [10].

In Gray and Putzolu's paper [12], the five-minute rule was de-

[1] For systems enabling data replication (e.g. for load balancing or failure recovery), each copy of the same data record should be treated as a separate data object.

[2] In this paper we assume that a data object is the *atomic* unit of data management, and thus is placed entirely in one storage device.

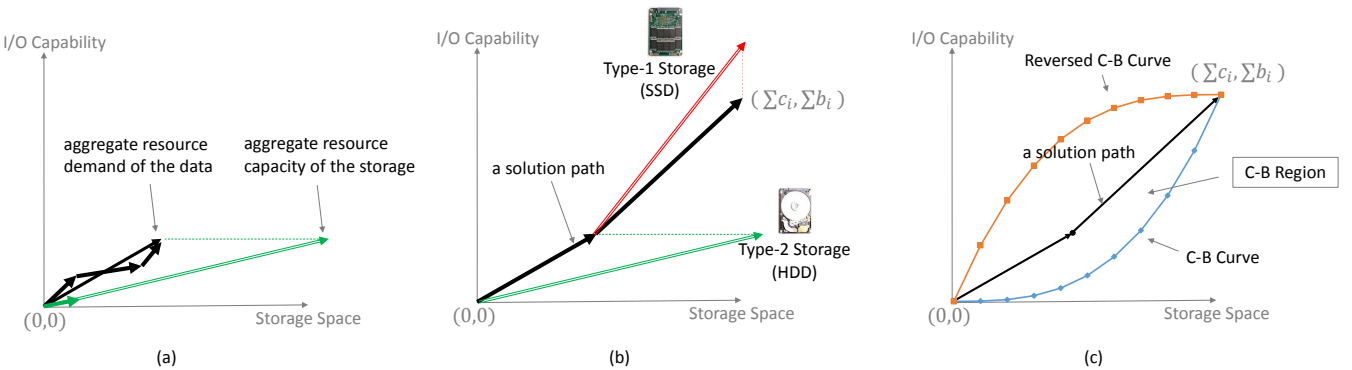

Figure 1: Optimization diagrams for the heterogeneous storage optimization (HSO) problem: (a) shows the diagram when there is only one storage type; (b) shows the diagram for two storage types, and illustrates the notion of solution-path; (c) illustrates the notions of C-B region, C-B curve, and reversed C-B curve in the optimization diagram.

rived exactly from the Vector-Sum Model introduced in the previous subsection. Specifically, they assume that the type-1 storage (i.e. DRAM in their context) is bound by storage space and the type-2 storage (i.e. hard drives) is bound by I/O throughput. Suppose one dollar gives either C_1 bytes of type-1 storage or B_2 IOPS of type-2 storage. Then, for a data object with size c_i and access frequency b_i, the cost to place this object in type-1 storage is $\frac{c_i}{C_1}$, and the cost to place it in type-2 storage is $\frac{b_i}{B_2}$. [3] At the break-even point of this trade-off we have

$$\frac{b_i}{B_2} = \frac{c_i}{C_1}. \tag{1}$$

DEFINITION 1. *(Five-Minute Rule [12]) It is more economic to place data objects with access frequency $b_i > \frac{B_2}{C_1} c_i$ in type-1 storage, and to place objects with $b_i < \frac{B_2}{C_1} c_i$ in type-2 storage.*

Note that the five-minute rule optimizes the system based only on a "context-free" feature of each individual data object: the ratio of its I/O throughput to its size (i.e., b_i/c_i). Later in this paper, we will see that this 30-years-old principle is still valid for the DRAM-HDD hierarchy today. However, in contrast to the observations in [10], we argue that the adoption of SSD in large-scale systems will not only alter the "5-minutes" threshold, but also challenge the basic principle of the rule.

2.3 Knapsack-based Heuristics

Another widely-used rule-of-thumb for heterogeneous storage optimization is the *Knapsack-based rule*. The basic version of the rule (e.g. see [5] [4] [23]) recasts the data placement problem under a given capacity plan as a classic Knapsack Problem: Suppose each data object has size c_i and throughput b_i, we want to direct as much I/O throughput into type-1 storage as possible, subject to a pre-determined size limit C for type-1 storage. Standard knapsack algorithms are then employed, for example the greedy algorithm, which keeps putting in type-1 storage the data objects with the highest b_i/c_i ratio until reaching the space limit C.

Recently, Huang and Xia [14] have applied the Knapsack heuristic in DRAM-SSD hierarchies, and extended the basic Knapsack formulation by co-optimizing the capacity planning at the same time (i.e. to determine the optimal value of the space limit C). Based on the observation that the per-GB price of DRAM is much higher than SSD, they try to minimize the capacity requirement of DRAM C, which is equivalent to maximizing the capacity of SSD for the given data set, subject to the I/O capability of the SSD of that capacity. The optimization leads to a simple but different rule-of-thumb:

DEFINITION 2. *(Knapsack Heuristic [14]) It is more economic to keep placing data objects with smaller $\frac{b_i}{c_i}$ in type-2 storage until the ratio between the aggregate I/O demand and space demand on type-2 storage is over the I/O-to-space ratio of the storage, that is,*

$$\frac{\sum_{\{i \text{ in type-2 storage}\}} b_i}{\sum_{\{i \text{ in type-2 storage}\}} c_i} = \frac{B_2}{C_2}. \tag{2}$$

Then we place all the rest data objects in type-1 storage.

Note that, instead of using the individual ratio b_i/c_i as the five-minute rule does, the knapsack heuristic relies on the cumulative ratio $\sum b_i / \sum c_i$ to compute the threshold.

3. OPTIMALITY ANALYSIS

Given the five-minute rule and the knapsack heuristic, one may naturally ask a series of questions about them: Which one of these two rules-of-thumb should we follow? Is one of them optimal (i.e. also better than any other possible rule)? If not, what is the optimal solution in the Vector-Sum Model? And, fundamentally, can the optimal solution be expressed as a simple rule at all?

In this section, we answer all these questions through a systematic analysis of the Vector-Sum Model. Our analysis is based on *optimization diagrams* of the underlying vector space. In these optimization diagrams, we can identify the existing heuristics, a new near-optimal solution, as well as the relationship between them.

3.1 The Optimization Diagram

In VSM, both storage and data are characterized as vectors in the same vector space. The heterogenous storage optimization diagrams illustrate this vector space, as Figure 1 shows. By definition, in a feasible design the aggregate demand vector of a storage-type must be larger than the sum vector of all data objects assigned to it (Figure 1a).

Figure 1b illustrates the situation when we have two different types of storage. In this case, the data in type-1 and type-2 storages have separate aggregate-demand vectors. The two demand vectors collectively form a *path* from the origin to the point representing the overall resource demand of all the N data objects. Let us call such a two-segmented path, a **solution path**. Given a solution path, we can easily calculate the money needed for each storage type such

[3] The original paper [12] used a slightly different notation from the ones used here [12]: $RI = 1/b_i$, $A\$ = 1/B_2$, $M\$ = 1/C_1$, and $B = c_i$.

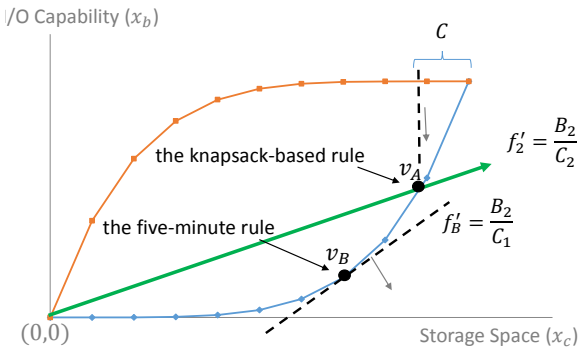

Figure 2: The knapsack heuristic corresponds to the intersection point $\mathbf{v}_A$ on the C-B curve, and the five-minute rule corresponds to the tangent point $\mathbf{v}_B$.

that the capacity vectors of the storage match the demand vectors of the data assigned to it. The cost of a solution path is the sum of the money spent on both storage types, and the goal is to find the solution path with minimal cost.

Given fixed model parameters, all the feasible solution-paths must be located within a certain *convex hull* in the vector space. Figure 1c illustrates the convex hull in the optimization diagram, which is a polygon called *C-B region* in this paper. Accordingly, the two boundary curves of the C-B region are called *C-B curve* and *reversed C-B curve*, respectively. Quantitatively, these geometric objects can be defined as follows:

DEFINITION 3. *Sorting the data objects such that* $\frac{b_1}{c_1} \le \frac{b_2}{c_2} \le \cdots \le \frac{b_N}{c_N}$, *and let point* $V_t = (\sum_{i\le t} c_i, \sum_{i\le t} b_i)$ *denote the sum-vector of the first* t *data objects in the vector space, the* ***C-B curve*** *is the linear interpolation curve of the point set* $\{V_1, V_2, \ldots, V_N\}$. *Similarly, the* ***reversed C-B curve*** *is the linear interpolation curve when data objects are sorted in the descending order of* $\frac{b_i}{c_i}$.

DEFINITION 4. *For any data set, the C-B curve and the reversed C-B curve always intersect at the origin point* $(0,0)$ *of the space and the point* $(\sum_1^N c_i, \sum_1^N b_i)$. *The* ***C-B region*** *is the closed region surrounded by the C-B curve and the reversed C-B curve.*

Since each solution-path contains two segments, which corner at their intersection point, we can fully represent a solution in the optimization diagram with this corner point. For example, Figure 2 identifies the points corresponding to the five-minute rule and the knapsack heuristic. Recall that the basic Knapsack algorithm places objects with higher b_i/c_i into type-1 storage subject to a given space limit C. Such solutions always correspond to a solution-path with corner point *on* the C-B curve. The extended knapsack heuristic in Definition 2 further determines C with Eq. (2). One can verify that the resulting solution corresponds to the *intersection point* of the C-B curve with the straight line of slope B_2/C_2, indicated by the point $\mathbf{v}_A$ in Figure 2. On the other hand, since the five-minute rule (Definition 1) separates data objects between the storage types according to the break-even condition $b_i/c_i = B_2/C_1$, it corresponds to the *tangent point* of the C-B curve with the straight line of slope B_2/C_1, indicated by the point $\mathbf{v}_B$ in Figure 2.

3.2 Finding the Optimal Solution

In the last section, we have seen that the five-minute rule and the knapsack heuristic correspond to two special points on the C-B curve, respectively. In this section we present another simple rule, which is guaranteed to identify the near-optimal solution in the optimization diagram. It turns out that our rule *subsumes* both the five-minute rule and the knapsack heuristic in the sense that it degenerates to these existing rules-of-thumb in different special cases, while it may provide better solutions in more general situations.

Recall that every solution-path of the problem is located within the C-B region of the vector space. But the reverse is not true – there are at most 2^N feasible solution-paths among the infinite possible paths inside the C-B region. It is this discrete nature of the problem that brings the main difficulty to find the *exact* optimal solution efficiently. However, observe that in large-scale storage systems the number of data objects is very large and each data object only consumes a relatively small fraction of the overall resource capacity. Consequently, it is reasonable to assume that we can always group data objects in such a way that the generated solution-path is "reasonably close" to a *arbitrarily* chosen (two-segmented) path in the C-B region. Essentially, the idea is to relax the original high-dimensional MIP formulation by a nonlinear optimization problem in the low-dimensional vector space. [4] Solving the relaxed problem leads to the optimal capacity plan (y_1, y_2) under an "idealized" data placement scheme.

Specifically, we have assumed that the type-1 and type-2 storage types are more economic in I/O capability and storage space, respectively. Formally this means $B_1 > B_2$ and $C_1 < C_2$, and thus we have

$$\frac{B_2}{C_2} < \frac{B_2}{C_1} < \frac{B_1}{C_1} \quad \text{and} \quad \frac{B_2}{C_2} < \frac{B_1}{C_2} < \frac{B_1}{C_1}. \tag{3}$$

Now, draw a straight line through the origin point $(0,0)$ with slope B_2/C_2 in the vector space, as well as a straight line through the point $(\sum_1^N c_i, \sum_1^N b_i)$ with slope B_1/C_1, as Figure 3 shows. We call the two lines the **characteristic lines**, as they characterize the per-dollar capacities of the two storage types (respectively). In general, the two characteristic lines divide the C-B region into four sub-regions (some of which can be empty). The following theorem shows that the optimal solution of the relaxed optimization problem over C-B region is simply the best one among the optimal solutions of four simple "local-optimization problems", each optimizes a different objective function over a specific sub-region. The theorem is proved based on the Mixed Integer Programming (MIP) formulation behind the model. See the Appendix for the complete proof.

THEOREM 1. *Let A, B, C, D denote the four sub-regions of the C-B region divided by the characteristic lines (some of the sub-regions can be empty), as Figure 3 shows. Let* $\mathbf{v}_A$, $\mathbf{v}_B$, $\mathbf{v}_C$, $\mathbf{v}_D$ *be the locally-optimal points in the four sub-regions that optimize (respectively) the objective functions of*

$$\max \quad x_c \quad \textit{over sub-region } A \tag{4}$$
$$\min \quad C_1 x_b - B_2 x_c \quad \textit{over sub-region } B \tag{5}$$
$$\max \quad C_2 x_b - B_1 x_c \quad \textit{over sub-region } C \tag{6}$$
$$\min \quad x_b \quad \textit{over sub-region } D, \tag{7}$$

where $\mathbf{v} = (x_c, x_b)$ *denote a point of the vector space. Further let* $\mathbf{v}^*$ *denote the globally-optimal point among* $\mathbf{v}_A$, $\mathbf{v}_B$, $\mathbf{v}_C$, $\mathbf{v}_D$ *such that the two-segmented path cornered at point* $\mathbf{v}^*$ *has the minimal cost among the paths cornered at these locally-optimal points. Then the path cornered at* $\mathbf{v}^*$ *is the optimal solution-path of the HSO problem, if it is feasible.*

[4] We note that this is different from the standard LP relaxation. The latter only relaxes the *domains* of the decision variables (for removing the integrality gap), while our relaxation reduces the *number* of the variables (for dimension reduction).

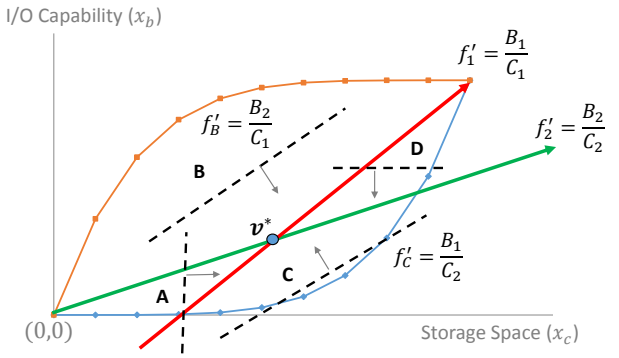

Figure 3: The optimal solution ($\mathbf{v}^*$) when the two characteristic lines intersect inside the C-B region.

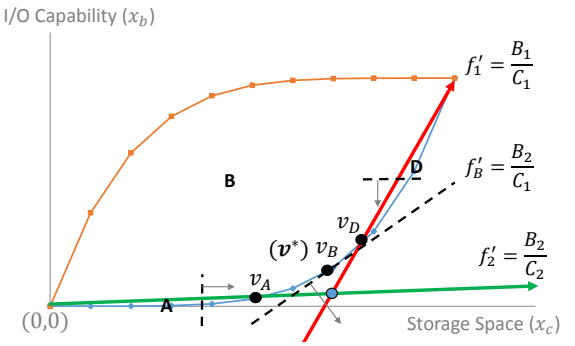

Figure 4: The optimal solution ($\mathbf{v}^*$) when the two characteristic lines intersect below the C-B region.

Interestingly, it turns out that the globally-optimal solution v^* defined in Theorem 1 has intuitive interpretations in the optimization diagram. Specifically, consider the problem in two situations:

When the characteristic lines intersect inside the C-B region. In this case, all of the four sub-regions A, B, C, D are non-empty, as the case in Figure 3. Observe that the objective functions of the four local optimization problems have slopes $f_A' = 1$, $f_B' = \frac{B_2}{C_1}$, $f_C' = \frac{B_1}{C_2}$, and $f_D' = 0$, respectively. Also observe that the two characteristic lines have slopes $f_1' = \frac{B_1}{C_1}$ and $f_2' = \frac{B_2}{C_2}$, respectively. Further incorporating Inequalities (3), we know that the four local-optimization problems must have the same solution in this case, which is the intersection point of the two characteristic lines (see Figure 3). Note that the intersection point of the characteristic lines corresponds to a *compact plan* in which both the type-1 and the type-2 storages are fully utilized in both storage space and I/O capability.

> CONCLUSION 1. *When the characteristic lines intersect inside the C-B region, the optimal heterogeneous storage design is at the intersection point of the characteristic lines in the vector space, which corresponds to the most "compact" solution of the problem.*

Quantitatively, let (y_1^*, y_2^*) denote the capacity plan corresponding to $\mathbf{v}^*$, we have

$$y_1^* = \frac{\gamma_\Sigma - \gamma_2}{\gamma_1 - \gamma_2} \cdot \frac{\sum c_i}{C_1}$$
$$y_2^* = \frac{\gamma_1 - \gamma_\Sigma}{\gamma_1 - \gamma_2} \cdot \frac{\sum c_i}{C_2} \qquad (8)$$
$$\text{where } \gamma_1 = \frac{B_1}{C_1},\ \gamma_2 = \frac{B_2}{C_2},\ \gamma_\Sigma = \frac{\sum b_i}{\sum c_i}$$

Intuitively, $\frac{\sum c_i}{C_1}$ is the cost to place all data objects in type-1 storage, and $\frac{\sum c_i}{C_2}$ is the cost to place all data in type-2 storage. The optimal capacity plan is a linear combination of these two costs, *weighted by* $\gamma_\Sigma - \gamma_2$ and $\gamma_1 - \gamma_\Sigma$. Note that γ_1, γ_2, and γ_Σ are the throughput-to-space ratios of type-1 storage, type-2 storage, and data set, respectively, and the fact that the two characteristic lines intersect inside the C-B region guarantees $\gamma_2 < \gamma_\Sigma < \gamma_1$.

When the characteristic lines intersect outside the C-B region. Without loss of generality we only discuss the cases when the intersection point is below the C-B region, as Figure 4 shows. Such a situation is usually due to a high throughput-to-space ratio $\frac{B_1}{C_1}$ of the type-1 storage and/or due to a low ratio $\frac{B_2}{C_2}$ of the type-2 storage. In particular, this could be a common scenario when using DRAM as type-1 storage, such as in DRAM-HDD or DRAM-SSD hierarchies. In this case, the two characteristic lines divide the C-B region into only three sub-regions A, B, and D because there is no feasible solution in sub-region C at all.

The locally-optimal solutions of the sub-regions A and D correspond to the intersection points of the C-B curve with the two characterizing lines, respectively. For sub-region B, however, the locally-optimal point depends on *where* the objective function of sub-region B is tangent to the C-B region. For example, Figure 4 illustrates a special case in which the tangent point $\mathbf{v}_B$ is in middle of the two intersection points $\mathbf{v}_A$ and $\mathbf{v}_D$. It is obvious that $\mathbf{v}_B$ is locally-optimal in sub-region B in this case. Moreover, because points $\mathbf{v}_A$ and $\mathbf{v}_D$ also belong to the sub-region B (or more accurately, at the boundary of B), $\mathbf{v}_B$ must also be globally optimal over the whole C-B region. In general, it is not hard to see that

> CONCLUSION 2. *When the characteristic lines intersect below the C-B region, the optimal heterogeneous storage design is always* on *the C-B curve. Moreover, it is among the following three points: the two intersection points of the C-B curve with the characteristic lines, or the tangent point of the C-B curve with slope B_2/C_1.*

Now, we apply the analysis results presented above to evaluate the existing rules-of-thumb. By comparing Figure 2 and Figure 4, we immediately see that the five-minute rule and the knapsack heuristic correspond to the locally-optimal solution in sub-regions B and A, respectively. As discussed before, there are indeed cases in which they are also globally optimal, respectively. In these cases, the solution given by Theorem 1 *degenerates* to the five-minute rule and the knapsack heuristic. For example, the five-minute rule ($\mathbf{v}_B$) is optimal in the cases of Figure 4. It is also not hard to see that the knapsack heuristic ($\mathbf{v}_A$) is optimal when the tangent point $\mathbf{v}_B$ is below the intersection point $\mathbf{v}_A$.

On the other hand, Conclusion 1 indicates that when the characteristic lines intersect inside the C-B region the globally optimal point is not on the C-B curve at all (see Figure 3), so none of these two widely-used rules can be optimal. In this case, we should instead use the "compact" solution as Eq.(8) suggests. In fact, combining Conclusions 1 and 2 we can see that, we should follow the five-minute rule or the knapsack-based rules only when such compact solution is infeasible.

Interestingly, experimental results in Section 5 will show that all the three cases discussed above (i.e. the five-minute rule is optimal, the knapsack heuristic is optimal, or none of them is optimal) will occur in heterogeneous systems with common storage devices in

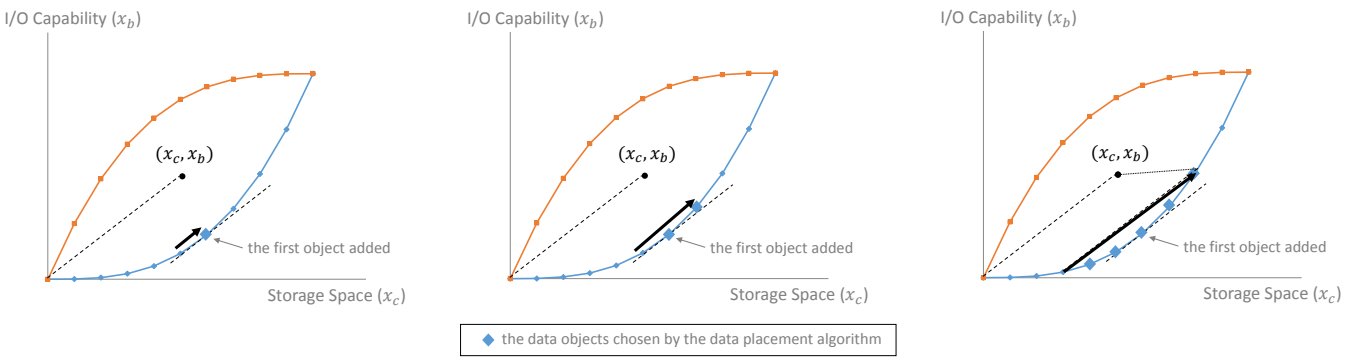

Figure 5: Illustration of the data placement algorithm that tries to approximate the given (x_c, x_b) point.

the real world.

> CONCLUSION 3. *The knapsack-based rules always provide solutions **on** the C-B curve, while the five-minute rule further nails down its solution at the tangent point of the C-B curve with the objective fucntion of sub-region B. The optimality of these rules-of-thumb depends on model parameters, and there are cases where both of them are sub-optimal, as Figure 3 shows.*

3.3 Data Placement

In the last section we have assumed that for any given point $\mathbf{v}^* = (x_c, x_b)$ in the C-B region, we can find a solution-path that is cornered at a point nearby (x_c, x_b). In this section we present a data placement algorithm that finds such a solution-path.

Denote S_2 as the set of data objects placed in type-2 storage, the algorithm works by adding data objects into S_2 in a greedy manner. More specifically, the algorithm expands the set S_2 along the C-B curve, as illustrated by Figure 5. Recall that C-B curve is essentially the concatenation of the vectors of all data objects in the ascending order of b_i/c_i. The first data object added to S_2 is the one whose individual b_i/c_i ratio is the closest to $\frac{x_b}{x_c}$. After that, the algorithm adds data objects into S_2 in such a way that the vectors of the data objects in S_2 always form a connected segment in the C-B curve. In other words, the algorithm always select data objects that are directly *adjacent* to the current segment of S_2.

The specific data object chosen in each round is the one that keeps the I/O-to-capacity ratio of the data set S_2 as close to the targeted ratio as possible. In other words, the algorithm tries to maintain the equation $\frac{\sum_{i\in S_2} b_i}{\sum_{i\in S_2} c_i} \approx \frac{x_b}{x_c}$ throughout the execution. The hope is that the granularity of individual data objects is small enough (compared to (x_c, x_b)) so that we can have a "smooth" expansion of S_2 along the C-B curve. The algorithm terminates when type-2 storage is "full" in some resource dimension. Algorithm 1 gives the psuedo-code of the algorithm thus described.

As special cases, when (x_c, x_b) is on the C-B curve – a case when the characteristic lines intersect below C-B region, as discussed before – Algorithm 1 is equivalent to the greedy algorithm that simply places all the data objects at the left side of the point (x_c, x_b) on the C-B curve to the type-2 storage. This is also consistent with what the five-minute rule and the knapsack heuristic asks to do in these cases (see Definitions 1 and 2).

Algorithm 1: A greedy algorithm for data placement

Input: x_c, x_b
Output: x_i for $i = 1, \ldots, N$

sort data objects in ascending order of b_i/c_i ;
$i^* \leftarrow \arg\min_i |b_i/c_i - x_b/x_c|$;
add i^* to S_2 ;
$(C_S, B_S) \leftarrow (c_{i^*}, b_{i^*})$;
while $C_S < x_c$ **and** $B_S < x_b$ **do**
  if $B/C < x_b/x_c$ **then**
    $i^* \leftarrow \max\{i \in S_2\} + 1$;
  else
    $i^* \leftarrow \min\{i \in S_2\} - 1$;
  add i^* to S_2 ;
  $(C_S, B_S) \leftarrow (C_S + c_{i^*}, B_S + b_{i^*})$;
$x_i \leftarrow 1$ for $i \notin S_2$;
$x_i \leftarrow 2$ for $i \in S_2$;

4. CHARACTERIZING I/O UTILIZATION

In the Vector-Sum Model the resource demand/capacity, if seen as a set function, is required to be *additive* in each dimension. For "benign" resource dimensions like storage space this is by-default true – a 1MB data object plus a 1MB object forms a 2MB data set; a 1TB hard drive plus a 1TB hard drive forms a 2TB disk array.

On the other hand, the actual I/O performance of modern storage devices is known to highly depend on the subtle access pattern in the I/O stream. Generally speaking, it is very challenging to make accurate prediction about whether a storage device would be saturated by a given I/O trace. Indeed, researchers have proposed various nonlinear [27] or even non-close-formed [20] device models to capture the complexity of storages.

However, to linearize the I/O performance of storage, we may not have to make such general and *a-priori* predictions. In this section, we argue that it may be possible to have accurate *a-posterior* judgments on whether subsets of the given workload can saturate the device *if* both the target workload and the target device are available for us to train a *workload-specific model* of the device. The basic idea is to estimate the I/O related parameters (b_i's and B) via polynomial regression based on measurements collected from running the specific trace on the specific hardware. The I/O demands thus learned may not be accurate for individual data objects, but the hope is that when we measure a large group of data objects, the sum of their I/O demands will be *statistically* additive. Notice that what we really want is "only" to predict the critical condition of whether a storage is saturated by a large data set, and it doesn't matter that we don't know exactly *how* unsaturated the storage is

under a light workload.

Our parameter estimation approach is based on the *effective length function* introduced in Section 4.1. Rather than compressing I/O traces into a short vector of workload characterization as many previous approaches do, we only transform each data request in the trace with the effective length function, and use the whole "transformed" trace as input to compute the I/O demands b_i. The workload-specific parameters in the effective length function and the I/O capability of the device B are then trained via standard regression techniques, which is described in Section 4.2.

4.1 The Effective Length Function

To clarify the terminology, we defined an *I/O request* as a tuple of (object_id, action, offset, length). In other words, data object is also the maximal data-access unit. An *I/O trace* is a sequence of I/O requests, each associated with an arrival time-stamp. The *duration* of the trace is the difference between the minimal and maximal arrival time. An *I/O workload* is a stochastic process that characterizes the data-access demands under given user scenario, on given data set, and with given system setting. One I/O trace is considered as a sample of the workload behind.

Given a data-access workload and a storage device, we want to assign the values of b_i's and B such that the storage device will work as a stable system [5] when serving the workload *if and only if*

$$\sum b_i \leq B. \tag{9}$$

For a given trace, let x_{ij} denote the length of the j-th data request to data object i in the trace. It is well known that the data length x_{ij} indeed faithfully reflects the *device utilization* for sequential I/O workloads, but the utilization thus calculated may not hit 100% for non-sequential workloads. In our approach, we assume that each data request can be attributed with a "dark utilization" such that the overall utilization of the trace, when counted in these dark utilizations, always reaches 100% (if the device works under full load). Specifically, let d denote the duration of the trace, we define

$$b_i = \frac{\sum^{j} f(x_{ij})}{d}, \tag{10}$$

where $f(x)$ is called the *Effective Length Function (ELF)*, which maps the raw data length x to its effective length $f(x)$ when counting in the dark utilization. Intuitively, the effective length of a request amounts to the data length that the storage device *could have served* if it were working at its (fixed) performance limit B. Note that $f(x)$ is a *context-free* function, and is essentially a statistical amortization over *all* requests of the same length x in the whole trace. As a special case, for idealized storage device that doesn't have any internal parallelism, the effective length of a request is proportional to the I/O latency of the request. For example, if a device with maximum data rate of 50 KB/ms serves a 1KB request in 2 ms, then the effective length of the request is 100 KB because in the same amount of time the device could have served 100 KB if it were running at the rate 50 KB/ms. Note that real-world storage devices usually exhibit various built-in parallelism (especially for SSD), which may lead to nontrivial effective length function $f(x)$.

The specific form of $f(x)$ depends on the specific characteristics of the storage hardware. Taking hard drive as example, the latency of a disk I/O consists of data-transfer latency, head-seeking time, and rotational latency, where the latter two are independent to the data length of the request [7]. [6] Thus, the effective length function for hard drives could be in the form of

$$f(x) = \lceil x \rceil_{block} + \delta, \tag{11}$$

where $\lceil x_i \rceil_{block}$ is the ceiling function that aligns request length up to the block size of the device, and δ is a constant that amounts to the initial latency of the request. Note that Eq.(11) degenerates to data transfer rate and IOPS when the request length x goes to $+\infty$ and 0, respectively. This is consistent with the standard I/O capability metric for sequential workloads and random workloads. The effective length function thus defined can be seen as a generalization of the two common metrics so as to characterize the I/O capability of storages under mixed workloads.

Despite of its simple form, the ELF function in Eq. (11) turns out to be surprisingly effective in our experiments, as shown later in Section 5. Now we focus on how to computationally estimate the parameters B and δ based on Eq.(10) and (11).

4.2 Statistical Regression

For a trace consisting of n requests and of duration d, by substituting Eq. (11) and (10) into (9) we obtain

$$d = (\frac{1}{B})(\sum^{i,j} \lceil x_{ij} \rceil) + (\frac{\delta}{B})n \tag{12}$$

if the storage device is running under full load when serving the trace. In reality, not every trace saturates the storage device. To work around this, we ignore the time intervals between requests in the given trace, and send the data requests by a *micro-benchmark program* that issues the next request immediately when the device completes one. The micro-benchmark program will generate a new trace, which contains the same set of data requests (i.e., x_{ij}'s and n) with the original trace, but may have a different duration. Note that the micro-benchmark program should be properly multi-threaded so as to push the device to its real limit.

Now, Eq.(12) gives a linear relationship between $\frac{1}{B}$ and $\frac{\delta}{B}$, where B and δ are the model parameters we want to estimate. By running the micro-benchmark program we can calculate a tuple of $(d, \sum \lceil x_{ij} \rceil, n)$, which can serve as one training sample of the estimation. By randomly picking up segments from the given trace we can bootstrap more samples. If the I/O pattern of the whole trace is statistically stable, all these trace segments should share the same values of B and δ. Although in theory two such trace-segments are enough, in practice we usually try much more trace-segments so as to make the result more robust to statistical instability of the trace.

In this way, we have reduced the parameter estimation problem to a classic linear regression problem that asks to estimate the values of $\frac{1}{B}$ and $\frac{\delta}{B}$ from a sequence of $(d, \sum \lceil x_{ij} \rceil, n)$ observations, based on the over-determined linear system of Eq.(12). Standard regression techniques can then be used. For example, we can try to minimize the vector-norm of the relative errors of the durations over all trace segments, between the ones measured from the micro-benchmark program and the ones estimated from Eq.(12). Suppose we have s trace segments, the resulting optimization problem is

$$\begin{aligned} &\min_{\lambda_1,\lambda_2} \left\| \left\{ (\frac{\sum \lceil x_{ij} \rceil}{d})_t \cdot \lambda_1 + (\frac{n}{d})_t \cdot \lambda_2 - 1 \right\}_{t=1\ldots s} \right\|, \\ &B = \frac{1}{\lambda_1} \ , \ \ \delta = \frac{\lambda_2}{\lambda_1} \end{aligned} \tag{13}$$

[5] A queueing system is stable if it has the same average input rate and output rate, which is the boundary condition to avoid a dramatically increased service latency.

[6] More strictly, there could be other performance overheads that contribute to the initial latency observed, such as bus contention or buffer copying. The ELF function in Eq. (11) encompasses all these factors by assuming that all of them are also statistically independent to the raw request length.

Table 2: Parameters of the storage types considered for experimentation

	Price ($)	Capacity (GB)	Data Rate for Read (MB/s)	Capacity/Cost ($GB/\$$)	Bandwidth/Cost ($MBPS/\$$)	γ ($Bandwidth/Capacity$)
SSD (Intel Pro 2500)	120	240	350	2	2.9	1.46
HDD (Seagate ST2000VN001)	100	2000	100	20	1.0	0.05

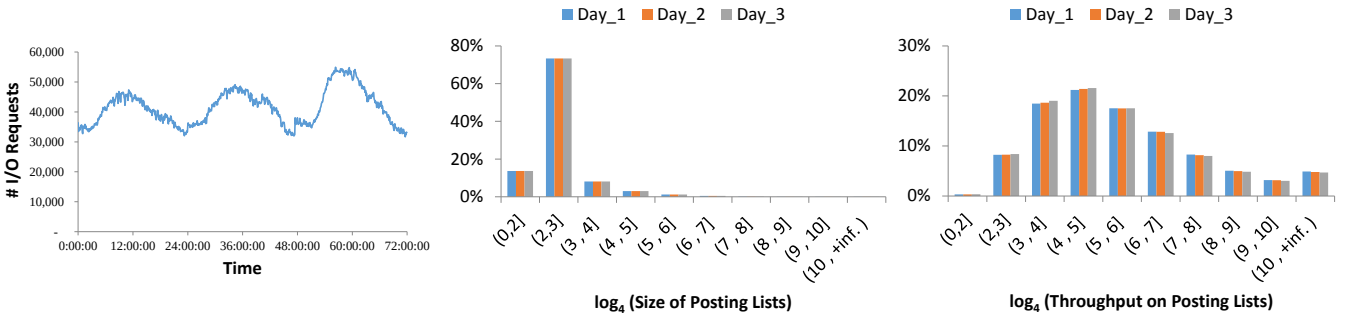

Figure 6: The web search workload. (a) is the curve of I/O throughput per five minutes. (b) and (c) are the histograms of the sizes and I/O throughputs of the data objects (i.e. posting lists in this context).

which can be solved by standard optimization routines [24]. The lower the relative errors we calculate in the test set of trace segments, the more accurate the estimation is.

Finally, we remark that the training process will be the same if we model the effective length function with higher-order polynomial functions. While the experimental results in Section 5 show that the linear function of Eq.(11) can already well capture the cases of HDD and SSD we tested, the additional degrees of freedom in higher-order polynomials may help to characterize some more complex workloads and storage hardware. Also, it is not necessary to characterize I/O capability as an one-dimensional feature. Alternatively, we may learn read I/O capability and write I/O capability separately, in which case the notion of I/O capability is characterized as a vector, and the corresponding heterogeneous storage optimization problem will be solved in high-dimensional vector space.

5. EXPERIMENTS

The evaluations are organized as follows. Section 5.1 describes the workload and storage hardware used to perform the experiments. Then Section 5.2 examines the accuracy of the parameter estimation approach in Section 4. Based on the model parameters thus learned, Section 5.3 verifies how "close" the final solution found by our data placement algorithm is to the "idealized" optimal solution identified by our capacity planning rule. In addition, this section shows that the five-minute rule, the knapsack heuristic, and our optimization rule happen to be optimal in the current DRAM-HDD, DRAM-SSD, and SSD-HDD systems, respectively.

5.1 Experimental Setup

Our research has been triggered by the work to optimize a commercial Web Search system. Web search engines use *inverted index* to select relevant web pages for each search query. An inverted index is essentially a collection of *posting lists*, and each posting list stores the occurrence information of one keyword over all the web pages covered by the index. In the context of web search, each posting list can be seen as a data object. At this moment, globe-scale web search engines cover billions of web pages, from which millions or billions of posting lists (keywords) can be extracted, resulting in inverted indexes of petabyte-level. Moreover, major search engines in the market may need to further replicate the inverted index into multiple copies so as to forking the huge search traffic on it. Hundreds of thousands of storage devices can be deployed in these search engines, and the scale of their systems are still rapidly growing, particularly driven by the development of mobile and social web. In addition to traditional hard drives, DRAM is extensively used in large-scale search engines to store index data at run time, due to its high throughput and low latency. Furthermore, solid-state storages based on NAND Flash memory are also adopted by search engines for better trade-offs between performance and cost [14].

The web search system thus described is a typical example of the large-scale heterogeneous storage systems considered in this paper. For experimentation purpose, we collected an inverted-index partition containing 21.8 millions posting lists. We also collected an I/O trace on the selected index partition of three consecutive days from system logs, which consists of 35.8 millions Read I/O requests. Figure 6(a) shows the I/O curve of the trace over the three days, from which we can see a clear diurnal pattern. Besides, Figure 6(b) and (c) show that the distributions of both data sizes and I/O throughput over the posting lists are quite similar between the three days. These observations imply that we could use the trace of the current day to estimate parameters for the heterogeneous storage (re)design of the next day. The goal is to determine a cost-efficient capacity plan across the HDD and SSD devices listed in Table 2, as well as a placement scheme for the 21.8 millions posting lists that matches the aggregate resource capacities of each storage type, in both storage space and I/O throughput.

5.2 I/O Performance Characterization

First, to verify the effectiveness of our I/O parameter estimation approach, we use the Day-2 trace of the web search workload as the training data to estimate B and δ, and test how well the Eq. (11) can predict the durations of traces in Day-1 and Day-3. Specifically, we randomly select 80 segments from the Day-2 trace, each with the duration of around 5 seconds. Then we run those trace segments on the real storage devices shown in Table 2. Figure 7 and Figure 8 show the plots of the resulting $(d, \sum\lceil x_{ij} \rceil, n)$ tuples generated by the micro-benchmark program, for HDD and SSD respectively. We can see that for both storage devices the data points clearly concentrate around a straight line, which suggests a strong statistical stability

	HDD (ST31000528AS)
B (MB/s)	81.93
δ (bytes)	608504.81
Training Error	6.8% (max) , 3.6% (avg.)
Prediction Error	8.0% (max) , 4.2% (avg.)

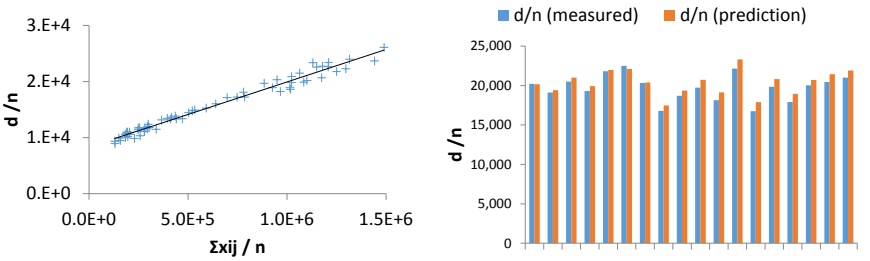

Figure 7: I/O parameter estimation for HDD under the Web Search workload

	SSD (X25-M)
B (MB/s)	233.97
δ (bytes)	3164.24
Training Error	6.1% (max) , 2.5% (avg.)
Prediction Error	8.5% (max) , 3.0% (avg.)

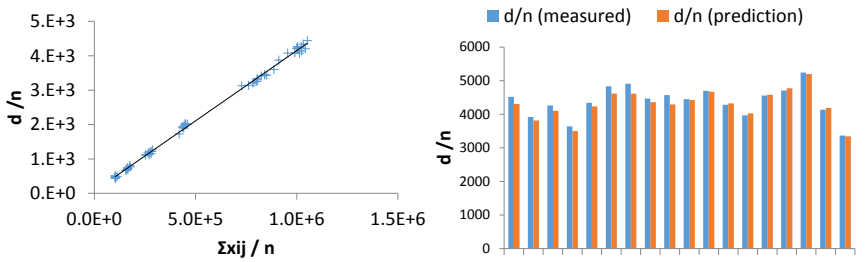

Figure 8: I/O parameter estimation for SSD under the Web Search workload

of the I/O pattern in the trace.

In order to make robust estimations on B and δ, we set the cost function of the regression to be the $\mathscr{L}_\infty$ norm of the relative-error vector shown in Eq.(13), so that the maximal variance over all observations is minimized. We obtain the estimation results as follows.

$$\begin{aligned} f_{hdd}(x) &= \lceil x \rceil_{512} + 608504.81 \\ B_{hdd} &= 81.93 \end{aligned} \tag{14}$$

$$\begin{aligned} f_{ssd}(x) &= \lceil x \rceil_{4096} + 3164.24 \\ B_{ssd} &= 233.97 \end{aligned} . \tag{15}$$

The maximum training errors over all 80 trace-segments is 6.8% for HDD and 6.1% for SSD, with average training errors of 3.6% and 2.5%, respectively.

To evaluate the prediction accuracy of the parameters estimated in Eq.(14) and (15), we randomly choose 18 trace-segments from the Day-1 trace and Day-3 trace. Figure 7 and Figure 8 show the prediction results, as well as the actual average request latency by from the micro-benchmark program. The maximum prediction error over all 18 trace-segments is 8.0% for HDD, and 8.5% for SSD, with average training errors of 4.2% and 3.0%, respectively. The prediction errors in the test set are consistent with the training set.

Notice that $\delta_{hdd} = 608504.81$ implies that the effective length of small disk requests is always close to a constant value around 600 KB, regardless of the real length of the request. This is consistent with the common sense that IOPS is a good metric for traces full of small requests. In terms of SSD, we find $\delta_{ssd} = 3164.24$, which is comparable to the 4KB page size of the SSD, which means the SSD can only show an actual I/O rate of $4096/(3164.24+4096) = 56.4\%$ to its maximum rate for 4KB random reads.

5.3 Different Rules for Different Storages

Given Eq.(14) and (15) trained from the Day-2 traces, we can compute all the model parameters (in particular, the I/O related parameters) needed to optimize the Vector-Sum Model. Figure 9 shows the resulting C-B region for the web search workload, where we can observe a strong convexity of the C-B curve. This indicates that the b_i/c_i ratio varies significantly between posting lists. Also notice that the C-B regions are quite similar between the three consecutive days.

Recall that the only potential loss of our optimization solution comes from the data placement algorithm that tries to approximate the optimal solution-path. So we evaluate the placement error of Algorithm 1 with the web search workload. Specifically, we randomly sampled 605 points (i.e. (x_c, x_b) pairs) in the C-B region (see Figure 9), and measure the *placement error* for each point. The placement error is defined as the larger one between the difference of the planned data size x_c with the size of data actually placed by the algorithm, and the difference of the planned I/O throughput x_b with the throughput actually placed. From Figure 9 we can see that the average placement error over the 605 samples is 1.95%, and only 3.47% of the samples (21 out of 605) have placement error larger than 10%. Moreover, we draw these "bad samples" (i.e. with error $> 10\%$) in the diagram of Figure 9, where we can see that most of them are nearby the reversed C-B curve, corresponding to capacity plans that we are unlikely to make in practice.

We then examined the performance of the five-minute rule and the knapsack-based rule in three different heterogeneous storage settings: the DRAM-HDD, DRAM-SSD, and SSD-HDD combination. Based on Conclusion 3 we can identify the points that these rules correspond to in the optimization diagram, as Figure 10 shows.

Specifically, Figure 10(a) shows the diagram for the DRAM-HDD case, in which the characteristic lines intersect outside the C-B region and the tangent point $\mathbf{v}_{5min}$ is above the intersection point $\mathbf{v}_{sack}$. From Conclusion 2 we know that the five-minute rule is indeed globally optimal in this case. This means that the five-minute rule still works perfectly in the traditional DRAM-HDD hierarchy even after thirty years since it was proposed!

However, we observed that the adoption of SSD could challenge the validity of these traditional wisdoms in new heterogeneous storage settings. For example, Figure 10(b) shows the diagram for the DRAM-SSD case, where we can see that the tangent point $\mathbf{v}_{5min}$ is below the intersection point $\mathbf{v}_{sack}$, meaning that the five-minute

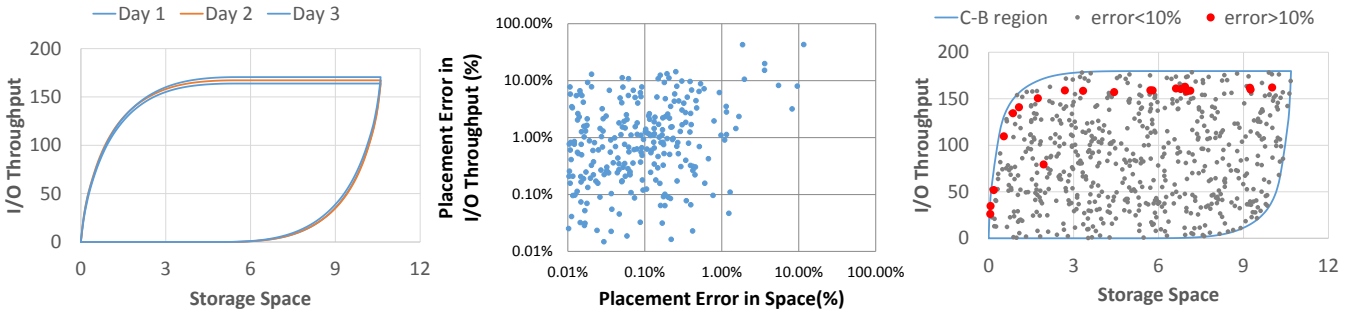

Figure 9: Placement inaccuracy of Algorithm 1 for random capacity plans under the web search workload.

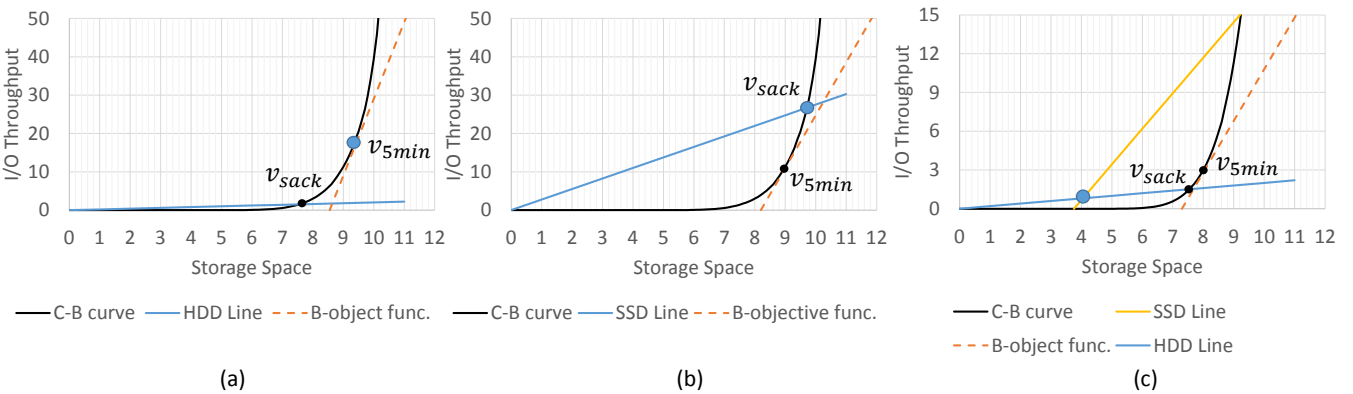

Figure 10: The optimization diagrams for different storage combinations. The blue point denotes the globally-optimal solution in all cases. (a) gives the DRAM-HDD diagram, where the five-minute rule $\mathbf{v}_{5min}$ is optimal. (b) gives the DRAM-SSD diagram, where the knapsack-based rule can be better than the five-minute rule. (c) gives the SSD-HDD diagram, in which case both the five-minute rule and the knapsack-based rule is sub-optimal.

rule is sub-optimal for the DRAM-SSD hierarchy we tested (again based on Conclusion 2). In fact, according to current model parameters, the globally-optimal solution is at point $\mathbf{v}_{sack}$, which has a cost of 16.8% lower than the point $\mathbf{v}_{5min}$ derived from the five-minute rule.

Moreover, Figure 10(c) shows the optimization diagram for the SSD-HDD case, in which the intersection point of characteristic lines is not outside the C-B region at all. From Conclusion 1 we know that in this case the globally-optimal solution is not on the C-B curve, meaning that not only the five-minute rule is sub-optimal, but all the knapsack-based rules are sub-optimal, no matter what the knapsack budget is specified to. Instead, the optimal SSD-HDD design should correspond to the intersection point of the two characteristic lines, in which case we should try to (and it is possible to) saturate both the storage space and the I/O capability of SSD and HDD at the same time.

6. RELATED WORK

The problem of optimizing the performance-to-cost efficiency of heterogeneous storage systems has been extensively studied in computer systems research. In early days, researchers have tried to formulate the systems using microscopic models, such as queuing network models [6] [26] [9] [15], which try to simulate the internal dynamics of the target storage under the given workload. However, it is often difficult to model the modern-day complex systems at this level and to verify the model thus constructed. Moreover, these models usually lead to optimization problems with overwhelming complexity as the number of data objects increases.

Another thread of research formulates heterogeneous storage optimization problems by considering the macroscopic constraints in the target systems. Models of this kind usually consider storage devices as "containers" with certain resource capacity, and consider data objects as items to be packed into these containers. In particular, Anderson et al. [2] [3] proposed an automated optimization tool which essentially models heterogeneous storage optimization as a "nested" bin-packing problem. In their work, components along the pathway of data access in the storage system (such as data channels, controllers) are modeled as containers nested with each other, and the placement of a data object may increase the resource demands of the containers in all layers. In the Vector-Sum Model, we focus on the constraints on the raw storage devices, leaving the optimization of the data pathway as a separate (and more detailed) problem. Moreover, in Anderson et al.'s work the goal is to build independent software advisors for system administrators [2], based on randomized algorithms searching in the configuration space. In contrast, in this paper we are seeking for optimization rules that are not only efficient in terms of computational complexity but also conceptually simple to be applied directly by the system administrators.

Meanwhile, a variety of storage device models have been proposed. "White-box" models can achieve high accuracy by simulating the transient state of the device at run time [8] [17], but it is hard to conduct analytical analysis of such models. On the other hand, researchers have also proposed "black-box" models that take parameterized workload characterizations as input and output performance predictions without simulating the internals of the device. Such models include analytical models [27], probabilis-

tic models [16], table-based models [1], and decision-tree based models [28]. Furthermore, the accuracy of such "absolute" models can be further improved by modeling the *relative fitness* between different storage types [20]. Most of these device models are nonlinear and typically do not admit simple optimization rules. As discussed in our paper, to linearize the I/O performance we may not necessarily generate workload-independent device models. Instead, it could be possible to learn workload-specific, rich-input, and statistically-effective I/O metrics for today's large-scale storage systems.

The Shapley Value model [21] is a related cost allocation approach that assigns additive credits to individuals given their collective outputs. The Shapley value is a nonparametric model that does not depend on a certain regression function as our approach does. However, note that our approach pursues a different additivity from Shapley values: The Shapley Value model asks the credits to be additive over different attributes (such as storage space and I/O capability, in our language), while in our context we want the I/O metric to be additive over different individuals (data objects and storage units).

In this paper we consider cases in which data placement is *static*. Of course, there is a substantial body of work on dynamic data placement, i.e., various caching policies [18] [19]. In these cases, researchers typically assume an particular analytical function (mostly the power function) between the capacity demand and the I/O demand of the workload [13]. For example, by assuming such a characteristic function, Sun et al. [25] have proposed a simple capacity planning rule based on a variant of the Vector-Sum Model for the heterogeneous memory optimization problem. The functions assumed in their works essentially characterize the equilibrium state of the system under a "greedy" dynamic data migration that tries to direct as much I/O traffic to the fast storages as possible. Interestingly, the insights obtained from our analysis suggest that such greedy data migration may not always be optimal – when the two characteristic lines intersect inside the C-B region, it may not be economic to "overload" the fast storage. Instead, the optimal principle of data placement (or data migration) in this case is to balance the capacity and I/O throughput of the data according to the characterizations of the storage types.

7. CONCLUSION

In this paper we have studied the heterogeneous storage optimization problem in its arguably most classic form, the Vector-Sum Model both in terms of optimization rules and parameter estimation. Our findings indicate that while of course not being completely accurate (no model ever is), the ever-expanding scale of modern storage systems is favoring this simple modeling technique in both aspects, at least with regard to an important subset of today's data center production workloads. Notice that our optimization diagram-based approach may also be useful for cloud computing tenants to co-optimize their capacity plan and data placement decisions for the applications storage hierarchy. Finally, it is clear that the Vector-Sum Model is a simple and simplifying abstraction. It is an interesting direction for future work to see whether the model can be made more accurate, while still maintaining the conciseness that gives raise to the insightful heterogeneous storage optimization diagrams.

8. REFERENCES

[1] E. Anderson. Simple table-based modeling of storage devices. Technical report, Technical Report HPL-SSP-2001-04, HP Laboratories, 2001.

[2] E. Anderson, M. Hobbs, K. Keeton, S. Spence, M. Uysal, and A. C. Veitch. Hippodrome: Running circles around storage administration. In *FAST*, volume 2, pages 175–188, 2002.

[3] E. Anderson, S. Spence, R. Swaminathan, M. Kallahalla, and Q. Wang. Quickly finding near-optimal storage designs. *ACM Transactions on Computer Systems (TOCS)*, 23(4):337–374, 2005.

[4] R. Baeza-Yates, A. Gionis, F. Junqueira, V. Murdock, V. Plachouras, and F. Silvestri. The impact of caching on search engines. In *SIGIR*, 2007.

[5] M. Canim, G. A. Mihaila, B. Bhattacharjee, K. A. Ross, and C. A. Lang. An object placement advisor for db2 using solid state storage. *Proc. VLDB Endow.*, 2:1318–1329, August 2009.

[6] C. K. Chow. On optimization of storage hierarchies. *IBM journal of Research and Development*, 18, 1974.

[7] Y. Deng. What is the future of disk drives, death or rebirth? *ACM Comput. Surv.*, 43:23:1–23:27, April 2011.

[8] G. Ganger, B. Worthington, and Y. Patt. The disksim simulation environment version 3.0 reference manual.

[9] R. Geist and K. Trivedi. Optimal design of multilevel storage hierarchies. *IEEE Transactions on Computers*, c-31, 1982.

[10] G. Graefe. The five-minute rule twenty years later, and how flash memory changes the rules. In *Proceedings of the 3rd international workshop on Data management on new hardware*, DaMoN '07, 2007.

[11] J. Gray and G. Graefe. The five-minute rule ten years later, and other computer storage rules of thumb. *SIGMOD Rec.*, 26, December 1997.

[12] J. Gray and F. Putzolu. The 5 minute rule for trading memory for disc accesses and the 10 byte rule for trading memory for cpu time. In *Proceedings of the 1987 ACM SIGMOD international conference on Management of data*, SIGMOD '87, 1987.

[13] A. Hartstein, V. Srinivasan, T. R. Puzak, and P. G. Emma. Cache miss behavior: is it $\sqrt{2}$. *Journal of Instruction-Level Parallelism*, 2008.

[14] B. Huang and Z. Xia. Allocating inverted index into flash memory for search engines. In *Proceedings of the 20th international conference companion on World wide web*, WWW '11, pages 61–62, 2011.

[15] B. L. Jacob, P. M. Chen, S. R. Silverman, and T. N. Mudge. An analytical model for designing memory hiearchies. *IEEE Transactions on Computers*, 45, October 1996.

[16] T. Kelly, I. Cohen, M. Goldszmidt, and K. Keeton. Inducing models of black-box storage arrays. *HP Laboratories, Palo Alto, CA, Technical Report HPL-2004-108*, 2004.

[17] Y. Kim, B. Tauras, A. Gupta, and B. Urgaonkar. Flashsim: A simulator for nand flash-based solid-state drives. *Advances in System Simulation, International Conference on*, 2009.

[18] D. Lee, J. Choi, J. Kim, S. Noh, S. Min, Y. Cho, and C. Kim. Lrfu: A spectrum of policies that subsumes the least recently used and least frequently used policies. *IEEE Transactions on Computers*, 50, 2001.

[19] N. Megiddo and D. S. Modha. Outperforming lru with an adaptive replacement cache algorithm. *IEEE Transactions on Computers*, 37, 2004.

[20] M. P. Mesnier, M. Wachs, R. R. Sambasivan, A. X. Zheng, and G. R. Ganger. Modeling the relative fitness of storage. *ACM SIGMETRICS Performance Evaluation Review*, 35(1):37–48, 2007.

[21] V. Misra, S. Ioannidis, A. Chaintreau, and L. Massoulié. Incentivizing peer-assisted services: a fluid shapley value approach. In *ACM SIGMETRICS Performance Evaluation Review*, volume 38, pages 215–226. ACM, 2010.

[22] D. Narayanan, E. Thereska, A. Donnelly, S. Elnikety, and A. Rowstron. Migrating server storage to ssds: analysis of tradeoffs. In *Proceedings of the 4th ACM European conference on Computer systems*, pages 145–158. ACM, 2009.

[23] A. Ntoulas and J. Cho. Pruning policies for two-tiered inverted index with correctness guarantee. In *SIGIR*, 2007.

[24] H. Spaeth. *Mathematical Algorithms for Linear Regression*. Academic Press, 1991.

[25] G. Sun, C. J. Hughes, C. Kim, J. Zhao, C. Xu, Y. Xie, and Y.-K. Chen. Moguls: a model to explore the memory hierarchy for bandwidth improvements. In *Proceeding of the 38th annual international symposium on Computer architecture*, ISCA '11, pages 377–388, 2011.

[26] K. S. Trivedi and T. M. Sigmon. Optimal design of linear storage hierarchies. *J. ACM*, 28, April 1981.

[27] M. Uysal, G. A. Alvarez, and A. Merchant. A modular, analytical throughput model for modern disk arrays. In *Modeling, Analysis and Simulation of Computer and Telecommunication Systems, 2001. Proceedings. Ninth International Symposium on*, pages 183–192. IEEE, 2001.

[28] M. Wang, K. Au, A. Ailamaki, A. Brockwell, C. Faloutsos, and G. R. Ganger. Storage device performance prediction with cart models. In *Modeling, Analysis, and Simulation of Computer and Telecommunications Systems, 2004.(MASCOTS 2004). Proceedings. The IEEE Computer Society's 12th Annual International Symposium on*, pages 588–595. IEEE, 2004.

APPENDIX

A. PROOF OF THEOREM 1

PROOF. The key insight is to see that the four sub-regions divided by the characteristic lines correspond to the situations with different resource bottlenecks. In each situation, the original problem can be converted to a local optimization problem that can be easily solved.

Specifically, the Heterogeneous Storage Optimization (HSO) problem under the Vector-Sum Model can be formulated as

$$\begin{array}{ll} \min & y_1+y_2 \\ \text{s.t.} & \sum c_i \cdot \mathbb{1}[x_i=1] \le C_1 \cdot y_1 \\ & \sum c_i \cdot \mathbb{1}[x_i=2] \le C_2 \cdot y_2, \\ & \sum b_i \cdot \mathbb{1}[x_i=1] \le B_1 \cdot y_1 \\ & \sum b_i \cdot \mathbb{1}[x_i=2] \le B_2 \cdot y_2 \end{array} \tag{16}$$

where $\mathbb{1}[e]$ is the *indicator function*, which can be seen as a binary variable that equals 1 if the event e is true, and equals 0 otherwise. We assume that all the model parameters C_1, B_1, C_2, B_2, and $\{c_i, b_i\}_{1\ldots N}$ are known.

Define $x_c \in \mathbb{R}$ and $x_b \in \mathbb{R}$ as the total size and I/O throughput of the data in type-2 storage, respectively. That is, we have

$$x_c = \sum c_i \cdot \mathbb{1}[x_i=2] \;\text{ and }\; x_b = \sum b_i \cdot \mathbb{1}[x_i=2]. \tag{17}$$

A (x_c, x_b) pair is *feasible* if and only if it corresponds to at least one assignment of the $0/1$-vector $\{x_i\}$ (i.e. a data placement scheme). Under the assumption that the feasible solutions are well scattered in the C-B region, and further noticing that

$$\mathbb{1}[x_i=1] + \mathbb{1}[x_i=2] = 1 \;\text{ for any object }\; i, \tag{18}$$

we can recast the original MIP model into the following nonlinear programming problem by substituting Eq. (17) and (18) into Eq. (16). Note that the original model has $N+2$ variables while the new model has only four variables: x_c, x_b, y_1, and y_2.

$$\begin{array}{ll} \min & y_1+y_2 \qquad (19) \\ \text{s.t.} & \sum c_i - C_1 y_1 \le x_c \le C_2 y_2 \qquad (20) \\ & \sum b_i - B_1 y_1 \le x_b \le B_2 y_2 \qquad (21) \\ & (x_b, x_c) \in \text{C-B Region} \end{array}$$

Eq. (20) and Eq. (21) give both upper bounds and lower bounds to x_c and x_b. Observe that at least one of the two upper bounds must be tight in the optimal solution, as y_1 and y_2 are assumed to be real numbers in the model. For example, we have

$$\text{either} \quad x_c = C_2 y_2 \quad \text{or} \quad x_b = B_2 y_2 \tag{22}$$

if the solution (x_c, x_b, y_1, y_2) is optimal. Intuitively, Eq.(22) means the type-2 storage should be bound either in its storage space or in its I/O throughput. In either case we can remove y_2 from the model by Eq.(22). Similarly one of the two lower bounds in Eq.(20) and (21) must be tight, depending on the resource bottleneck of type-1 storage, with which we can also remove y_1 from Eq.(19). In total, there are four possible combinations of the tight bounds (resource bottlenecks). Substituting y_1 and y_2 by x_c and x_b in each case yields four different sub-models:

- Sub-model (A) : both the type-1 storage and type-2 storage are bound in storage space, in which case we have $\sum c_i - C_1 y_1 = x_c$ and $x_c = C_2 y_2$, yielding

$$\begin{array}{ll} \max & (C_2 - C_1) x_c \\ \text{s.t.} & x_b \le \dfrac{B_2}{C_2} x_c \\ & x_b \ge \dfrac{B_1}{C_1} x_c + (\sum b_i - \dfrac{B_1}{C_1} \sum c_i) \\ & (x_b, x_c) \in \text{C-B Region} \end{array} \tag{23}$$

- Sub-model (B) : the type-1 storage is bound in storage space, and the type-2 storage is bound in I/O throughput, in which case we have $\sum c_i - C_1 y_1 = x_c$ and $x_b = B_2 y_2$, yielding

$$\begin{array}{ll} \min & x_b - \dfrac{B_2}{C_1} x_c \\ \text{s.t.} & x_b \ge \dfrac{B_2}{C_2} x_c \\ & x_b \ge \dfrac{B_1}{C_1} x_c + (\sum b_i - \dfrac{B_1}{C_1} \sum c_i) \\ & (x_b, x_c) \in \text{C-B Region} \end{array} \tag{24}$$

- Sub-model (C) : the type-1 storage is bound in throughput, and the type-2 storage is bound in space, in which case we have $\sum b_i - B_1 y_1 = x_b$ and $x_c = C_2 y_2$, yielding

$$\begin{array}{ll} \max & x_b - \dfrac{B_1}{C_2} x_c \\ \text{s.t.} & x_b \le \dfrac{B_2}{C_2} x_c \\ & x_b \le \dfrac{B_1}{C_1} x_c + (\sum b_i - \dfrac{B_1}{C_1} \sum c_i) \\ & (x_b, x_c) \in \text{C-B Region} \end{array} \tag{25}$$

- Sub-model (D) : both the type-1 storage and type-2 storage are bound in I/O throughput, in which case we have $\sum b_i - B_1 y_1 = x_b$ and $x_b = B_2 y_2$, yielding

$$\begin{aligned} \min \quad & (B_1 - B_2) x_b \\ \text{s.t.} \quad & x_b \geq \frac{B_2}{C_2} x_c \\ & x_b \leq \frac{B_1}{C_1} x_c + (\sum b_i - \frac{B_1}{C_1} \sum c_i) \\ & (x_b, x_c) \in \text{C-B Region} \end{aligned} \tag{26}$$

We can see that all the four sub-models are constrained by the same two linear functions of

$$x_b = \frac{B_2}{C_2} x_c \quad \text{and} \quad \sum b_i - x_b = \frac{B_1}{C_1} (\sum c_i - x_c). \tag{27}$$

Notice that the two equations in Eq.(27) correspond exactly to the two characteristic lines in the optimization diagram, which means the solution domain of the four sub-models correspond to the four sub-regions A, B, C, D defined in Theorem 1. We can check that the objective functions of the four sub-models are consistent with the ones in Theorem 1, too. □